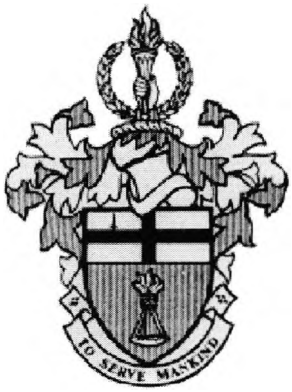

Market Protocols for Computational Clusters and Grids

Jacek Martin Gomoluch

Submission for the Degree of Doctor of Philosophy

City University

Department of Computing

June, 2004

Acknowledgements

I am most indepted to my supervisor, Prof. Michael Schroeder, for his continuous guidance, encouragement, and support during these three years of research.

I also would like to thank Dr. Geraint Wiggins, who has been my internal supervisor in the final months of this research.

I wish to thank my colleagues and fellow suffering students Tshiamo, Reinhold, Penny, Panos, Alex, Eddy, Peter, Rodrigo, Aloysius, and Teddy, who always offered a friendly and supportive environment, and who I very much enjoyed working with. I am also grateful for their fast proof-reading of my thesis.

And finally, I wish to thank my parents for all their help and support throughout this long period of time.

Abstract

Recently, there has been much interest in *Computational Grids* which provide transparent access to large-scale distributed computational resources. One key issue in these open and heterogeneous environments is the efficient allocation of resources. Clients and service providers belong to different organisations and have different priorities, requirements, and goals, making resource management a complex task.

Economic approaches to resource allocation can offer a solution, as they are naturally decentralised, and as decisions about whether to consume or provide resources are taken locally by the clients or service providers. The use of currency offers incentives for service providers to contribute resources, while clients have to act responsibly due to their limited budget. To maximise the benefit of the clients, it is essential to choose an appropriate resource allocation protocol. There exist various economic protocols with different properties, however, their performance in Grid settings has not yet been sufficiently studied.

In this thesis, we review and classify existing work on market protocols in computational clusters and Grids. We then develop a simulation model of an electronic marketplace and evaluate several market protocols for different computational environments, task loads, and optimisation requirements of the clients. We study situations, in which the tasks are independent and arrive randomly. Three scenarios are examined in which the clients have different requirements concerning the execution of their tasks. In two scenarios, the completion times of the tasks need to be minimised. In the first one, all tasks are equally important, whereas in the second one, they have different values for the clients and are weighted accordingly. In the third scenario, tasks have different priorities combined with hard or soft deadlines which need to be met in order to deliver maximum value to the clients.

The resource allocation protocols, which are evaluated, include continuous double auctions (CDA), periodic double auctions (PDA), and a proportional sharing protocol (PSP). Also, several preemptive protocols, with and without task migration, are investigated, as well as protocols, in which the service providers are allowed to set reserve prices. The simulation results reveal that the choice of the protocol should depend on the optimisation requirements of the clients, the number of resources in the system, the heterogeneity of these resources, the amount of load and background load in the system, the local scheduling policy at the resources, and the communication delays. We found that, in most situations, CDA leads to very good results. However, with high heterogeneity and load, it can be outperformed by PDA, PSP, and the preemptive protocols without migration. Also reserve prices can lead to performance improvements. In most cases, the best results are achieved by preemptive protocols which allow migration.

To verify our simulation model and thus our results, we developed a basic Grid computing infrastructure, that is based on the model, and carried out experiments in a local area network. We also demonstrated the effectiveness of our infrastructure for solving real-world problems by deploying a computationally intensive bioinformatics application.

Abbreviations and Acronyms

CDA	Continuous Double Auction Protocol
CORBA	Common Object Request Broker Architecture
CPM	Compute Power Market
CPU	central processing unit
DAG	directed acyclic graph
D'Agents	Dartmouth Agents
EP	embarrassingly parallel
EMP	Electronic Marketplace
ERA	Economic Resource Allocator
FLASH	Flexible Agent System for Heterogeneous Clusters
FIFO	First In First Out
FIPA	Foundation for Intelligent Physical Agents
I/O	input/output
GA	Genetic Algorithms
HBP	Highest Bid Protocol
HTTP	Hypertext Transfer Protocol
JADE	Java Agent Development Framework
JDK	Java Development Kit
Jini	Jini Is Not Initials
JMS	Java Message Service
JXTA	"juxtapose"
LAN	local area network
LJF	Longest Job First
MAJIC	Multiparameter Auctions for Jini Components
MATS	Mobile Agent Teams
MCT	Minimum Completion Time
MFLOPS	mega FLOPS (floating point operations per second)
MPI	Message Passing Interface
MPP	Massively Parallel Processors
NP	Nondeterministic Polynomial
NWS	Network Weather Service
OCEAN	Open Computation Exchange and Arbitration Network
PE-P	Preemptive Protocol (Passive)
PE-A	Preemptive Protocol (Active)
PSA	parameter sweep application
PSP	Proportional Share Protocol
PVM	Parallel Virtual Machine
RP	Random Pushing
RR	Round Robin Protocol
RS	Random Stealing
RMI	Remote Method Invocation
RPC	remote procedure call
SA	Simulated Annealing
SJF	Shortest Job First
TS	Tabu Search
WCT	weighted completion time

Contents

1 Introduction

1.1 Motivation

The last four decades have seen a rapid increase in computing power and the rise and fall of several technologies. In the 1960s, mainframe computers took up a few hundred square feet. Just a decade later, the advent of the minicomputer cut down the size of systems by taking advantage of large-scale integration of semiconductors. During the 1980s, vector computers, and later parallel computers emerged. At about the same time the microcomputer age began, bringing PCs to the desks of the end-users.

As computers became more affordable, the need to link them up gave rise to the first local area networks such as the Ethernet [Metcalfe and Boggs, 1976]. By the mid- to late-1980s, the first network cards began to appear for the PC. Between 1989 and the early 1990s, an emerging wide area network — known as the *Internet* — started becoming the norm at universities in the US, Japan, and Europe. It had evolved from the ARPANET of the US Defense department which was first established in 1969 with just 4 nodes. The growth of the Internet was fueled by a rapid increase in network bandwidth and by the invention of the *World Wide Web* [Bernerns-Lee, 1999] at CERN, in 1989. The Web provided the means for creating and organising documents with hyper-links and accessing them online transparently, irrespective of their location.

The increase in network bandwidth — which has grown twice as fast as the processing power — led to advancements in high performance computing. The availability of powerful PCs, workstations, and high-speed networks as commodity components has resulted in the emergence of computational clusters. In the top 500 list of fastest computers which solve a matrix factorisation problem [Strohmaier *et al.*, accessed in 2003], clusters of PCs or workstations already rank among the top (with seven systems in the top 10). In recent years, the number of cluster systems in the top 500 has grown to more than 200, making them the most common high performance computing architecture.

The Internet can add a new dimension to parallel processing, as comparatively small computing resources such as PCs have the potential to provide vast computing power, when

connected. And yet, many of these resources lie idle for most of the time. Millions of online-PCs are only involved in tasks like word processing or browsing the Internet, which consume very little computing power. The computing resources in many organisations are often severely under-utilised, especially outside of peak business hours.

At the same time, there are many individuals and organisations that have intensive computations to perform but only have limited access to resources that are available to execute them. This disparity in resource utilisation has inspired various projects which connect millions of computers over the Internet to perform computations in areas like drug design [1], biology [2], and astronomy [3]. Such wide-area networks of PCs are, however, only one instance of a much broader vision: to transform the capability and modalities of scientific research by providing transparent, intuitive, timely, and efficient access to distributed, heterogeneous, and dynamic resources. These resources include computational facilities, applications, visualisation, data, and experimental facilities, which are integrated and accessible as a single resource over the Internet - the *Grid* [Foster and Kesselman, 1998].

1.2 The Grid

Inspired by the electrical power grid's pervasiveness, reliability, and ease of use, scientists in the mid-90s began exploring the design and development of an analogous infrastructure called the *computational power Grid* [Foster and Kesselman, 1998]. The vision is to build an environment that enables the "sharing, selection, and aggregation of a wide variety of geographically distributed resources including supercomputers, storage systems, data sources, and specialised devices owned by different organisations for solving large-scale resource-intensive problems in science, engineering, and commerce" [Buyya, 2002]. These efforts are driven by large-scale, resource-intensive scientific applications that require more resources than can be provided in a single administrative domain [Buyya, 2002].

A number of Grid platforms have been developed, such as Globus [Foster and Kesselman, 1997], Unicore [Pallas, accessed in 2003], the Load Sharing Facility [Platform Computing,

[1] Find-A-Drug (http://www.find-a-drug.org), grid.org (http://www.grid.org)
[2] Folding@Home (http:/folding.stanford.edu)
[3] SETI@Home (http://setiathome.berkeley.edu/)

accessed in 2003], and Legion [Natrajan *et al.*, 2001]. Also, major companies including Sun, IBM, and Hewlett Packard have recognised the potential of Grid Computing and provide their own platforms [Sun Microsystems, accessed in 2003e] and services [Hewlett Packard, accessed in 2003; IBM, accessed in 2003]. Common to all Grid infrastructures is the need to provide components which cater for communication infrastructure, data storage and movement, security, task monitoring, and resource management.

Efficiently managing and allocating resources in a Grid is a far more complex task than in a local cluster: Users and service providers are geographically distributed and belong to different organisations, with heterogeneous platforms and varying reliability and availability. Furthermore, the parties involved have different priorities, requirements, and goals, making resource management even harder.

1.3 The Need for Market Protocols

Market-based approaches to resource allocation [Ferguson *et al.*, 1996; Sandholm, 2000; Wellman *et al.*, 2001] can offer a solution to the problems of distributed ownership of resources and the conflicting interests of the users: Resource allocations are determined through the use of economic mechanisms such as *auctions*, in which users place explicit valuations ("bids") on the resources being contended for. The use of currency offers incentives for service providers to contribute resources, while users have to act responsibly due to their limited budget. Hence, resource prices and task priorities are directly related to demand and supply.

To maximise the efficiency of the resource allocations, and thus the benefit to the users, it is essential to choose the best-performing market protocol [4]. There exist various protocols with different properties, however, their performance in Grid scenarios has not yet been sufficiently studied. When market protocols were examined, the experiments were limited to only a few protocols and parameter sets. What has been missing is a systematic comparison of different market protocols which would allow a system designer to choose the most ap-

[4] A *protocol* defines the rules which determine how the participants of the marketplace interact and how the resources are allocated. It does not include the *strategies* of the participants for settings their prices or bids.

propriate protocol for a given situation. In this thesis, we review and classify existing work on market protocols in computational clusters and Grids. We develop a simulation model of an electronic marketplace where CPU time is traded as a resource and evaluate several economic protocols for different computational environments, task loads, and optimisation requirements of the users.

1.4 Scenarios

Our objective is to maximise performance as experienced by the users. We model a marketplace in which users and service providers may belong to different organisations and trade resources in exchange for money. The users have computations which must be performed on certain resources and are willing to pay for the service. An example could be a scientist who wants to conduct a bioinformatics computation and needs several Linux PCs with the Globus Toolkit [Foster and Kesselman, 1997] installed. The service providers have access to idle resources that can execute the computation in question. The system is assumed to be geographically distributed, and therefore communication bandwidth and latency need to be taken into account.

We target the following scenarios:

- A computational cluster, e.g. a lab of PCs at a research institute or company. Users have a limited endowment of artificial money which can be used for buying CPU time.

- An open, geographically distributed computational *Grid* in which the participants belong to different organisations.

In both cases, CPU time is traded at a central electronic marketplace, which the participants can trust, and whose rules must be obeyed [5]. Users submitting computational tasks can choose how much they bid for the execution of each task and may also specify deadlines.

[5]Participants who do not comply could be penalised by the marketplace, e.g. by excluding them from further trade. However, this is not the topic of this thesis.

Concerning the resources, we consider the following cases:

- Time-shared PCs, e.g. running under Linux. Local users may be running applications (such as word-processors or Internet browsers) which consume some of the CPU time and are considered as *background load*. The remaining processing power — which may vary over time — is made available to computational tasks from the electronic marketplace.

- PCs whose processing power is offered at the marketplace only at times when they are completely idle, i.e. when their screensavers are running. If such a policy is used, fewer resources are available to the marketplace. Yet, it may be more acceptable to the local users as it reduces the impact of the incoming tasks on the local applications.

- Space-shared multiprocessor machines on which parallel applications can be executed. Several applications can be run on the machines' processors at any given time. Again, as there are local users, only some of the processing power is made available to the marketplace.

For the scenarios studied in this thesis, we make the assumption of a *managed system*, in which the market is a *tool* to achieve the efficient allocation of resources. Our main focus is on the design of the protocols which are used at the marketplace. The choice of pricing strategies for the Clients and Servers is not the subject of this work: We assume that these strategies — which may or may not be utility-maximising — can be enforced by the system. Money has no value as such, and hence there is no need to deal with resource accounting.

We mainly look at scenarios where PCs are the resources, as these are idle most of their time and can therefore offer large amounts of CPU time to computationally intensive applications. We study situations in which the computational tasks are *independent* [6] and arrive randomly. Since the task arrivals and the state of the system are not known a priori, the scheduling decisions need to be taken *online*. Three scenarios are examined in which the users have different requirements concerning the execution of their tasks. In two scenarios the completion times of the tasks need to be minimised. In the first one, all tasks are

[6]Note that, in this context, we also consider a parallel application to be an independent task because it is allocated to just one multiprocessor machine.

equally important, whereas in the second one, they have different values for the users and are weighted accordingly. In the third scenario, tasks have different priorities combined with hard or soft deadlines which need to be met in order to deliver maximum value to the users. The goal of our simulations is to determine the most suitable resource allocation protocol for each examined situation (w.r.t. the performance metric which is used).

We investigate the impact of parameters, such as the number of resources in the system, the heterogeneity of these resources, the amount of load and background load in the system, the local scheduling policy at the resources, and the communication delays.

1.5 Resource Allocation Protocols

We consider several market protocols, all of which use *auctions* [Kagel, 1995]. In auctions, buyers bid for resources according to a particular auction protocol. An advantage of auctions is that they allow an unknown resource value in a group of agents to be determined. Also, auctions are widely studied, easy to implement, and efficiently computable. A disadvantage is often the communication cost.

A very familiar auction protocol is the *English* auction, in which a seller advertises a resource whose price is gradually increased as the bids come in — and the highest bidder wins. A problem with English auctions is the associated communication cost which we need to avoid in order to maximise performance. For this reason, all our protocols are of *sealed-bid* type. In a sealed-bid auction, all buyers submit sealed bids, and the highest bidder wins. There is only one round of communication — without any time-consuming negotiation.

Furthermore, all our protocols can be classified as *double* auctions: In a double auction, multiple buyers and sellers submit their bids and ask prices, and matches are made by the marketplace. Since there are multiple buyers and sellers in our scenarios, this form of auction is appropriate. A double auction can either be a *continuous* auction, where transactions are carried out immediately whenever bids or offers change, or a *periodic* auction, in which the transactions are carried out only at periodic intervals.

Overall, the following protocols are examined in this thesis:

- The Continuous Double Auctions protocol (CDA), which we expect to result in shorter response times and hence better performance than periodic auctions.
- Several adaptations of CDA in which the tasks can be preempted, both with and without migration.
- Two protocols which allow the service providers to set reserve prices in order to prevent the allocation of the better-performing resources to low priority tasks.
- The Periodic Double Auctions protocol (PDA) in which the transactions are carried out at periodic intervals only.
- The Proportional-Share Protocol (PSP) in which several tasks can execute on a resource, and a task's resource share is proportional to its price bid.
- Several conventional scheduling heuristics which include Round-Robin (RR), First-in-First-Out (FIFO), and Shortest Job First (SJF).

In our simulations we aim to determine which resource allocation protocol is the most appropriate for a given situation. For three scenarios with different user requirements, we compare the performance of the protocols while varying the parameters of the computational environment.

1.6 Verification through Experiments

We verify our simulation model and thus our results by carrying out experiments in a local area network. We determine whether the assumptions we made about communication delays, processing delays, etc. are valid under realistic conditions. For this purpose, we deploy a basic Grid computing infrastructure which we developed as part of the AgentCities deployment grant *CoMAS* (Control and Management of Agents and their Services, iD: ACNET.02.30). This infrastructure is based on the agent platform JADE [Bellifemine *et al.*, 1999] and is an almost exact implementation of the simulation model. We also demonstrate

the effectiveness of our infrastructure for solving real-world problems by using it for the distributed computation of the bioinformatics application PSIMAP [Dafas *et al.*, 2003a].

1.7 Publications

Parts of the experimental work described in this thesis have been published in [Gomoluch and Schroeder, 2003], [Gomoluch and Schroeder, 2004], [Dafas *et al.*, 2003b], and [Dafas *et al.*, 2003a]. An earlier version of the survey on market protocols has appeared in [Gomoluch and Schroeder, 2001b]. In addition, the research undertaken in the course of this PhD has resulted in the publication of [Gomoluch and Schroeder, 2001a], [Gomoluch and Schroeder, 2002] and [Cogan *et al.*, 2001].

1.8 Contributions

In summary, the main contributions of this thesis are:

- A survey and classification of existing approaches to the dynamic allocation of resources in computational clusters and Grids, with an emphasis on market protocols.
- The design of a simulation model of an electronic marketplace for distributed computational resources and of several protocols for the resource allocation. This model is suitable for both computational clusters and Grids.
- The development of a simulation framework for the evaluation of these resource allocation protocols. The framework supports various scenarios, resource types, scheduling policies, and resource allocation protocols. It allows to set any simulation parameter and to measure any statistic, thus enabling the exploration of a large parameter space.
- The evaluation of the market protocols through simulations. We explore the parameter space for three scenarios in which the clients have different requirements concerning the execution of their tasks. We provide guidelines for the choice of protocols in different situations.

- The verification of the simulation results with an experimental Grid computing framework that has been developed for this purpose. We determine in how far the real system behaves as we observed in the simulations. To this end, we deploy it in a cluster of PCs in a local area network. We also demonstrate the effectiveness of this framework for solving real-world problems such as the PSIMAP computation.

1.9 Report Structure

The rest of this thesis is organised as follows.

Chapter 2 presents an overview of application types that can benefit from being distributed and executed in parallel. Chapter 3 covers the properties of the hardware and software infrastructure which need to be considered for the design of a resource allocation protocol. Chapter 4 provides a survey and classification of existing approaches to the *dynamic* allocation of resources in computational clusters and Grids, and mainly looks at market protocols. In chapter 5, we state our research objectives and discuss other work on performance evaluation of market protocols. We show how it relates to this thesis.

Detailed descriptions of our simulation model and the resource allocation protocols investigated in this thesis are given in chapter 6. These are followed by a hierarchical overview of the parameter space explored in our simulations in chapter 7. The results of our simulations for the three scenarios are presented in chapters 8 to 10.

Chapter 11 gives a high-level description of our experimental Grid computing framework. Chapter 12 describes the experiments with this framework which are designed to verify the simulation model. Chapter 13 gives guidelines for the designer of an Electronic Marketplace which are based on the simulation results. Chapter 14 discusses the results of the simulations and experiments, draws conclusions, and gives directions for future work.

An appendix follows which provides further information concerning the implementation of the resource allocation protocols. It also describes our simulation framework, which has been used for the simulations, and shows additional simulation results. Furthermore, it gives more details about the implementation and operation of our experimental Grid computing framework.

2 Overview of Problem Types

Despite ever faster machines, the demand for computing power remains high in application areas like scientific computing and data analysis. Many computational applications are so large, that their execution time on a single machine would not be acceptable to the user. However, for certain applications, the execution time can be reduced by decomposing them into subtasks which can be distributed among several machines.

In order to do this in an appropriate way, it is necessary to know the characteristics of the application and the requirements of the user. In this chapter, we first give an overview of different application types. Then, we discuss the limits of the speedup that can be achieved when an application is executed in parallel. Finally, we introduce several metrics which can be used to assess the performance of a resource allocation protocol.

2.1 Embarrassingly Parallel Application

A simple but very common problem type is termed *"embarrassingly parallel"(EP)* [Fox *et al.*, 1994]. The application consists of a set of independent calculations and can easily be parallelised, since no temporal synchronisation is involved. In practice, modest node-to-node communication will be required though, if only to set up the problem and to accumulate the results. Assuming a high computation to communication ratio, an almost linear speedup (see section 2.5) can be achieved for this type of computation. An example of a task graph representing an embarrassingly parallel application is shown in Figure 1 (left).

Problems which fall into this category include the so-called *Parameter sweep applications (PSAs)* [Casanova *et al.*, 2000], in which many computations of the same type are run with different parameter sets. Each computation can execute *independently*, i.e. without inter-task communication or data-dependencies (task precedences). Examples of parameter sweep applications and other embarrassingly parallel problems can be found in bioinformatics [Casanova *et al.*, 2000; Park *et al.*, 2001], high energy physics, and finance [Fox *et al.*, 1994].

2.2 Parallel Applications with Inter-Task Communication

Another important problem type are applications which consist of several tasks which execute in parallel but which require communication between these tasks. Regarding the communication pattern, this problem type can be subdivided into *synchronous*, *loosely synchronous*, and *asynchronous* problems which will be described in this section [Fox *et al.*, 1994; Fox, 1992].

2.2.1 Synchronous Problems

Synchronous problems are computations on geometrically regular data domains which require synchronisation between the iterations. Examples are matrix computations such as LU decomposition and convolutions such as the Fast Fourier Transform. They are parallelised by simple domain decomposition [Fox *et al.*, 1994; Fox, 1992]. An example of a task graph representing a synchronous problem is shown in Figure 1 (middle).

2.2.2 Asynchronous Problems

Asynchronous problems are characterised by a temporal irregularity which makes parallelisation hard. An important example are event-driven simulations, where events occur in spatially distributed fashion but irregularly in time. Branch-and-bound and other pruned tree algorithms common in artificial intelligence such as computer chess also fall into this category [Fox, 1992]. For the parallelisation the "data parallelism" over the space of events is exploited.

2.2.3 Loosely Synchronous Problems

Loosely synchronous problems are an intermediate case between asynchronous and synchronous problems. They are characterised by iterative calculations on geometrically irregular domains. The computations are parallelised by irregular partitioning of the data domain. The processes are synchronised "every now and then", typically at the end of an iteration or time step in a solution. Examples are irregular mesh finite element problems and inhomogeneous particle dynamics [Fox, 1992].

2.2.4 Scalability of Parallel Applications

Synchronous and loosely synchronous problems parallelise naturally in a fashion that scales to large systems with many nodes. The computations typically divide into communication and calculation phases as given by individual iterations or time steps in a simulation. The efficiency depends on the problem grain size. In many cases an almost linear speedup (see subsection 2.5) can be achieved — and the calculations are not much affected by the synchronisation.

Asynchronous problems are characterised by additional synchronisation overhead, since the division into communication and calculation phases is lacking. The speedup for this type of computations is very problem-dependent. However, large scale parallelisation is possible for a subclass which is referred to as *loosely synchronous complex*. The problem consists of an asynchronous collection of loosely synchronous (or synchronous) modules [Fox *et al.*, 1994; Fox, 1992].

In a survey carried out in 1989, 400 applications from 84 areas have been classified. The survey, which might be outdated now, reports that at most 10 percent of the applications were truly asynchronous, whereas most applications were synchronous (40 percent) or loosely synchronous (36 percent). About 14 percent belonged to the embarrassingly parallel class [Fox *et al.*, 1994].

2.2.5 Flexibility of a Parallel Application

A parallel — but not embarrassingly parallel — application may have different constraints regarding the number of machines it is allocated during its execution. Feitelson *et al.* [Feitelson and Rudolph, 1998; Feitelson *et al.*, 1997] give a classification of applications concerning their flexibility when executed in parallel:

- *Rigid Jobs:* The number of processors assigned to a job is specified externally and cannot be changed by the scheduler.
- *Moldable Jobs:* The number of processors assigned to a job is determined by the system within certain constraints when the job is first activated, and it uses that many

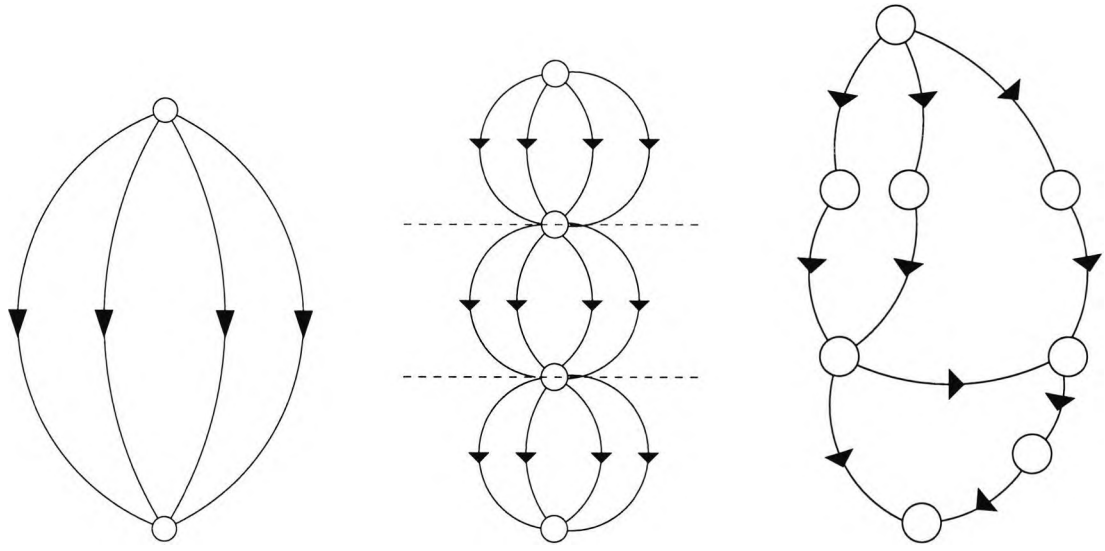

Figure 1: Task graphs for different problem types. Left: Embarrassingly parallel (EP) application. Middle: Synchronous application. Right: Application represented by a directed acyclic graph (DAG).

processors throughout its execution [7].

- *Malleable Jobs:* The number of processors assigned to a job may be changed during the job's execution, as a result of the system giving it additional processors or requiring that the job releases some.

- *Evolving Jobs:* The job goes through different phases which require different numbers of processors. The number of processors allocated may change during the execution in response to the job requesting more processors or relinquishing some. Each job is allocated at least the number of processors it requires at each point in its execution.

2.3 Applications with Subtask Dependencies

In another problem type, which can benefit from being executed in parallel, the application contains *subtasks* with data dependencies among them. It will be described in this section.

[7] According to [Cirne and Berman, 2001b] the majority of tasks in supercomputing centres are moldable — 98%, as the authors found in their survey.

2.3.1 Job Shop Scheduling (Directed Graphs)

In this type of application the subtasks require the results of other subtasks *before* starting their execution, and on completion they pass their results to consecutive subtasks. The application can be modelled with a *directed graph* where a *node* corresponds to a subtask and an *arc* represents a data transfer between two tasks [Maheswaran *et al.*, 1999b]. The graph may or may not contain *cycles*. In the latter case it is called *directed acyclic graph (DAG)*, an example of which is shown in Figure 1 (right). The subtasks may execute on different machines and may even require different machine architectures. This problem type can be found in many areas including bioinformatics [Möller *et al.*, 1999] and defence [Ali *et al.*, 2002]. It is also very common in manufacturing where the machines are usually not computers [Pinedo, 1995].

2.3.2 Divide-and-Conquer Applications

Divide-and-conquer applications consist of subtasks which can spawn further subtasks during their execution. The results of the child tasks are needed by their parent tasks for further processing. As the decomposition of the application happens on-the-fly it is very flexible with respect to the number of machines that are used — which is even allowed to vary during runtime. The application is easily parallelised by letting the programmer annotate potential parallelism in the form of *spawn* and *sync* constructs. Computations that can use this model include geometry procedures, sorting methods, search algorithms, data classification codes, n-body simulations, and data-parallel numerical programs [van Nieuwpoort *et al.*, 2001].

2.4 Other Application Characteristics

In the previous sections, an overview of different application types has been given which can benefit from parallel execution in a distributed environment. This overview, however, may not necessarily cover all aspects of an application that are relevant for the choice of a resource allocation protocol. Further computational characteristics which need to be considered include the following [Braun *et al.*, 1998]:

- *Granularity:* What is the granularity of the application, i.e. its ratio between computation and communication? In the case of *fine-grain parallelism* the tasks execute a small number of instructions between communication cycles. This may lead to a high communication overhead. On the other hand it facilitates load-balancing: a large number of 'small' tasks can easily be distributed.
- *Task Heterogeneity:* Are all tasks of the same size? If not, how greatly and with what properties, e.g. probability distribution, do their execution times vary for a given hardware platform or operating system?
- *Deadline:* Does the application have a deadline? Is it a hard or a soft deadline?
- *Temporal Distribution:* Is the complete set of tasks known *a priori* (*static* application) or do the tasks arrive in a real-time, non-deterministic manner (*dynamic* application)? Or is it a combination of both?
- *Duration:* Is the duration of the tasks known before their execution?
- *Priority:* What is the priority of the application?
- *Memory Requirements:* What are the memory requirements of the application and its subtasks?
- *Quality-of-Service Requirements:* Does the application have specific quality-of-service requirements, such as the level of security?

2.5 Speedup

2.5.1 Definitions

As mentioned before, the aim of parallel execution of a computational task is to minimise the execution time. A measure for the reduction of execution time is the *speedup*. It is defined as the ratio of sequential execution time to parallel execution time of a computation. There is diversity in the definitions of serial execution time which results in different definitions of speedup [Sahni and Thanvantri, 1995]:

In the definition of the *relative speedup*, the serial time is the execution time of the parallel algorithm when run on a single processor of that parallel computer. The relative speedup describes how well the algorithm has been parallelised. Alternatively, the serial time may denote the time taken by the best serial algorithm. This definition is used when an absolute evaluation of the algorithm is required. Since it is not always possible to determine the best serial algorithm, the runtime of the serial algorithm used "in practice" is often taken *(real speedup)*.

2.5.2 The Limit of Speedup: Amdahl's Law

According to Amdahl [Amdahl, 1967], the speedup of parallel execution can never be more than linear. Also, in any computation, there should come a point where further task division creates more overhead than computational speedup and does not lead to a performance improvement. The theoretical limit for the speedup is given by *Amdahl's Law* [Amdahl, 1967]:

For the parallel algorithm, let *seq* be the fraction of sequential operations in the computation, and *par=1-seq* the fraction that is parallelised. The potential speedup S achievable by a parallel computer with N processors performing the computation is:

$$S = \frac{1}{\frac{par}{N} + seq} = \frac{1}{\frac{par}{N} + 1 - par}$$

When the number of processors N is increased, the upper limit for the speedup can be determined as:

$$S_{\max} = \lim_{N \to \infty} \frac{1}{\frac{par}{N} + 1 - par} = \frac{1}{1 - par}$$

In a sense this is not really a limit, since for some applications the parallel fraction increases as the workload is increased *(unbounded parallelism)* [Sahni and Thanvantri, 1995]. In that case the theoretical speedup may be infinite:

$$\lim_{par \to 1} \frac{1}{1 - par} = \infty$$

If we limit the question of maximum attainable speedup to a particular instance of a problem, then speedup is limited for those instances that have a fixed workload associated with them. Two factors limit the attainable speedup [Sahni and Thanvantri, 1995]:

- The *total workload.* If the instance represents a total workload of u units and each processor performs one unit of work in one unit of time, then at most u processors can be gainfully used, and the parallel execution time can be as low as one. The serial execution time will be u, and therefore speedup can not exceed u.

- The *serial component.* If s of the u workload units cannot be parallelised, the parallel run time cannot be reduced below $s+1$. So, the speedup cannot exceed $u/(s+1)$.

However, there exist applications with flexible workload per instance, for which neither of the two factors that limit speedup may apply [Sahni and Thanvantri, 1995].

2.5.3 Loss of Speedup

In practice there are many other reasons which may limit the speedup. The following factors have been identified for task execution on parallel processors [Nguyen *et al.*, 1996]. They are also valid for distributed computing on multiple machines:

- *Idleness:* At times, processors are left idle because of insufficient (*coarse-grain*) parallelism or load imbalance.

- *Communication Overhead:* Communication takes place if an executing task requires access to data that does not currently reside on its machine. Communication overheads appear as the processor stalls while waiting for the data.

- *System Overhead:* Even sequential programs incur system overhead because of events such as page faults, clock interrupts, etc. Such overheads can be more significant for highly parallel programs because these events typically occur on every processor. The asynchronous nature of these events can degrade the performance of tightly-coupled parallel programs.

- *Parallelisation Overhead:* Parallel programs typically incur computational overheads which are not present in sequential programs such as per processor initialisation, work partitioning, and locking and unlocking on entry and exit of a critical section.

According to [Nguyen *et al.*, 1996] the parallelisation overhead and system overhead are typically very small compared to idleness and communication cost.

2.6 Metrics

The *speedup* (see section 2.5) is an important measure for the performance improvement of the parallel execution of a task. However, depending on the type of task and the requirements of the user, various other metrics can be used for the assessment of a resource allocation protocol. In this section an overview will be given.

2.6.1 Performance

At first, some metrics will be discussed which characterise the system performance. These are the completion time, completion rate, user utility, makespan, and throughput.

Completion Time

When considering performance from the application's point of view, the metric involved is often one of minimising individual task completion times [Casavant and Kuhl, 1988]. The *completion time* is defined as the time elapsed from when a task arrives for scheduling to when it completes execution. It includes both the time spent in waiting queues and time spent in execution. A common metric is the *mean completion time* of the tasks of the entire workload. Another metric, the *weighted completion time*, is the weighted sum of the individual task completion times [Feitelson *et al.*, 1997]. The weights may be chosen according to task priorities or durations.

The completion time places greater emphasis on longer tasks, as opposed to short tasks. This is why a normalised metric called *slowdown* [Feitelson and Rudolph, 1998] is sometimes employed. It is defined as the runtime on a loaded system divided by the runtime on a dedicated system. If the tasks have different priorities, the *weighted slowdown* may be used.

Completion Rate

If the workload in the system is high and the tasks to be executed have tight *deadlines*, not all of them may be able to complete on time. In the case of hard deadlines a task will fail if its deadline is missed. In this so-called *deadline scheduling scenario* [Buyya *et al.*, 2000; Fatima, 2000; Takefusa, 2001] an important performance metric will be the *completion rate*, i.e. the ratio of tasks completed on time and all tasks submitted to the system. If the tasks have different priorities, the *weighted completion rate* can be used.

User Utility

In reality, the users of a system may often have tasks with a *soft* deadline, rather than simply wanting to minimise their completion time or to meet *hard* deadlines. This soft deadline may be expressed as a *utility function* whose value depends on the time when the task completes [Chun and Culler, 2002]. At the same time, the value depends on the importance of the task which may be strongly varying: e.g. a company running simulations for pharmaceutical research may attach much more value to its tasks than another user who is running image processing jobs in his spare time. As the utility of a task is sensitive to its delay, it can be represented by a piecewise-linear function of the *slowdown* which is shown in Fig. 2. E.g. the value could remain at its maximum $V_{Task,initial}$ until a given slowdown value sl_1 is reached and then linearly decrease until a slowdown value sl_2 where it becomes zero. In such a scenario, which can be seen as a *generalised deadline scheduling scenario*, the following cases are possible: A task has a *hard* deadline if $sl_1=sl_2$, a *soft* deadline if $sl_1 < sl_2 < \infty$, and no deadline at all if $sl_2=\infty$.

Makespan

When dealing with a *supertask* which consists of many smaller tasks, a typical goal is to minimise a metric called *makespan* [Abraham *et al.*, 2000; Casanova, 2001]. The makespan is defined as the time elapsed between the submission of the *first* task to the system and the completion of the *last* task.

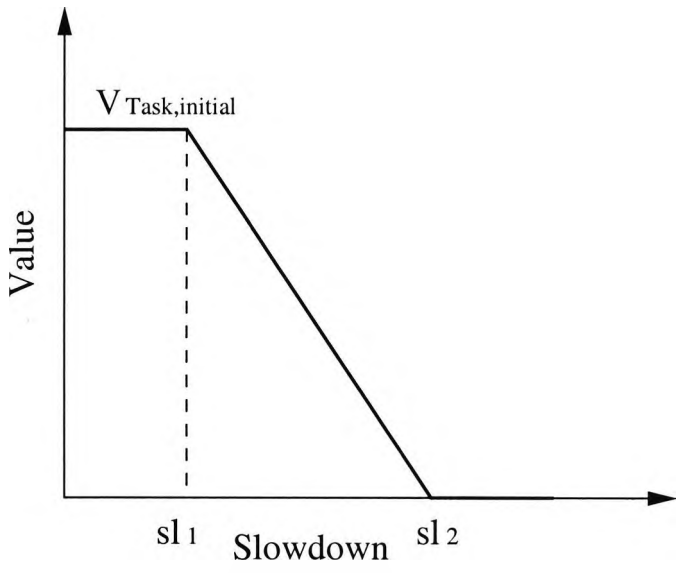

Figure 2: Time-dependent value of tasks.

Throughput

A widely used metric for evaluating the performance from the system's point of view is the *throughput* [Casavant and Kuhl, 1988; Ferguson *et al.*, 1996; IEEE, 1990]. It is defined as the amount of work that can be performed by a computer system or component in a given period of time. Assuming that the performance and capacity of its resources are limited, the aim is usually to maximise the system's throughput.

There is an inherent conflict in trying to optimise both, an individual task's completion time and the system's throughput [Casavant and Kuhl, 1988]. The reason is that throughput is concerned with seeing that *all* tasks are making progress. Therefore an individual task might not obtain the best possible service.

2.6.2 Utilisation

Another important measure is the *utilisation* of the machines in the system. One possible objective is to maximise the utilisation of the system's resources — which is compatible with maximising the throughput.

Another very common goal is to obtain an *even* balance of the load in the system. This, for instance, can be done by minimising the *maximum load* over all machines and over the

whole time [Azar, 1998].

2.6.3 Scalability

The term *scalability* is used to refer to the change in performance of the parallel system as the problem size and machine size or number increase. Intuitively, a parallel system is *scalable* if its performance continues to improve as we 'scale' (i.e. increase) the size of the system and of the problem.

2.6.4 Stability

Stability is a concept used in domains such as the physical sciences where it is regarded as a property of an equilibrium. An equilibrium is stable if, after a small perturbation, the system returns to it 'voluntarily'. For our type of system the notion of an equilibrium is not exactly clear. Intuitively, a system is in equilibrium when the statistical properties of its performance parameters remain stationary for a given variation in the system's external load. Thus, instability occurs when a small perturbation of some system parameters leads to a sharp and persistent deviation in the systems performance indicators [Lee *et al.*, 1998].

2.6.5 Other Metrics

In this thesis we aim to maximise the performance as perceived by the user. Therefore, we mainly use the *performance metrics* that are described in subsection 2.6.1. However, there exist other metrics which can be relevant in some situations:

- *Communication Costs:* When dealing with the amount of traffic which is caused either by the application itself or by the scheduler, the simplest metric is the *number of messages* exchanged. Another one is the distribution of the *message size*.

- *Fairness:* In many systems the aim is to treat all users fairly and therefore ensure that their tasks complete eventually. Fairness can also mean being fair to the resources by balancing the workload so that the utilisation of the machines is approximately the same (see subsection 2.6.2).

- *Cost of Task Re-Assignments:* In pre-emptive resource allocation protocols (see section 4.4), tasks are re-assigned to other machines at execution time in order to improve performance. However, the re-assignments can be very costly themselves and may even degrade performance. Possible metrics are the number of task re-assignments, the amount of data that needs to be transferred for the individual task, the overhead of packaging up the state of the task, and the transfer delay.

- *Economic Efficiency:* One approach to the resource allocation problem is to use computational economies (see subsection 4.3.2), which value resources and aim to achieve an efficient match of demand and supply. In a computational economy, one possible objective is to achieve *economic efficiency*, i.e. the resource allocation mechanism should help to maximise the gains of the participants. In [Levy *et al.*, 2001] the *total welfare* is used as a metric of the allocation efficiency. It is defined as the sum of the buyers' utilities from the services they buy at the market minus the costs of these services for the sellers.

- *Reliability:* When an application is executed in an unreliable environment, a possible metric is the application failure probability [Ferguson *et al.*, 1996].

2.7 Summary

In this chapter, we gave an overview of application types that can benefit from being distributed and executed in parallel. The main emphasis was on the application's communication requirements and dependencies between its subtasks. We then discussed the speedup that can be achieved through the parallel execution. In addition to the theoretical limits, which are stated in Amdahl's law, we named further factors which may reduce it in practice. Finally, we introduced several metrics that can be used to evaluate a resource allocation protocol. In this thesis, we will focus on those metrics which assess performance from the user's point of view.

3 Infrastructure and Middleware

For the design of a resource allocation protocol, the hardware and software infrastructure of the system needs to be considered. This chapter will address the relevant system properties, which include the machine architecture, system control, and network connections. It will also introduce so-called middleware, which is software that enables the communication of applications in a distributed system. An overview of different types of middleware platforms will be provided.

3.1 Machine Architecture

The machine architecture comprises various architectural features such as processor type, processor speed, external I/O bandwidth, memory size, etc. Different machines in the system may or may not be able to execute a particular computational task. This will depend not only on the machine's hardware architecture but also on the operating system and the software that is installed. For a given computation it is also necessary to know how greatly and with which properties the execution times vary across different machines in the system. In the simplest case all machines are the same, and there are no significant variations (*identical machine case*). In the *related machines case* the relations of execution times on different machines are the same for all tasks. If certain machines perform better (or worse) for different types of computation tasks, this is called the *unrelated machine case* [Azar, 1998].

3.2 System Control

The scheduler of a system, and hence the resource allocation protocol, may have different levels of control over its resources. Either it may have complete control over all machines in the system, or it may have to deal with external users which are also consuming resources. In the case of external users a *background load* on the resources will have to be assumed.

Also, one needs to distinguish between *closed* and *open* systems. In an open system the users and resource providers will be able to enter and leave the system over time. In such a system the protocol will have to deal with issues such as user authentication, admission

control, and resource accounting. Another issue which must be considered is whether the number of machines under the system's control is *fixed* or *variable*. The latter will usually be the case for open systems. It is also important to know how tasks — which in this context are usually referred to as *jobs* or *processes* — are scheduled on the individual machines. Can only one task execute at a time, or are the machines *time-shared*? In the latter case, the question is how the CPU time is allocated to the tasks — and whether the resource allocation protocol can influence it. A task's CPU share can be proportional to its priority but also other issues may be considered, e.g. real-time requirements of a task, starvation of low priority tasks, etc. In general, the scheduling of tasks will depend on the operating system — or a dedicated scheduler that is used.

3.3 Network Connections

For certain types of applications, in particular communication-intensive tasks, the network infrastructure plays an important role. A widely used metric for network performance is the *bandwidth* (also referred to as *throughput*): it is defined as the amount of data that can be transferred between two machines per second. *Latency* is another essential measure of network performance. It describes the time passed between the dispatch of a message by the sender and its arrival at the receiver. Normally, it only includes the *network traversal time*. Sometimes, also the *source queueing time* is included in this figure [Chien and Konstantinidou, 1994]. Furthermore, the time of marshalling and unmarshalling the data may be added to it. For the protocol design, it is important to know how the machines in the system are interconnected. Are all machines in a Local Area Network (LAN) or distributed all over the Internet? Does the system consist of a collection of workstation clusters, or are the machines on random geographic locations? The system might also have to deal with information (packet) losses and variations of the performance characteristics over time.

3.4 Middleware

Having briefly discussed the hardware infrastructure, we now introduce the software infrastructure, which is required for any type of distributed computing: *middleware*. Middleware

is a general term for software that connects otherwise separate applications. There exist various middleware platforms with different properties and application domains. These will be discussed in the remaining sections of this chapter. At first, we introduce the most widely used middleware platforms: the message passing libraries *MPI* [Pacheco, 1997] and *PVM* [Geist *et al.*, 1994] for parallel computing, and the distributed object computing platforms *RMI* [Sun Microsystems, accessed in 2003b], *CORBA* [Object Management Group, 1992], *.NET* [Platt and Ballinger, 2002], and *Web Services* [Webopedia.com, accessed in 2004] for distributed systems. Next, we discuss several *mobile agent platforms* [Hohl, accessed in 2003], in which code can autonomously migrate between different hosts.

Recently, middleware has been incorporated in platforms which are dedicated to Grid computing. As an example, we will describe the very popular *Globus Toolkit* [Foster and Kesselman, 1997]. Finally, we discuss other platforms which do not target Grid Computing but can provide a powerful infrastructure, namely Jini [Sun Microsystems, accessed in 2003c], JMS [Sun Microsystems, accessed in 2003a], JXTA [Sun Microsystems, accessed in 2003d] and JADE [Bellifemine *et al.*, 1999].

3.5 Distributed Object Computing

In the early 80s, the Unix operating system with its dedicated programming language C was enhanced by remote procedure calls (RPCs), which made it possible to access remote procedures. Similarly, Java was later extended to cater for remote method invocation (RMI) [Sun Microsystems, accessed in 2003b].

The heterogeneity of many different RPC implementations led to the development of the Common Object Request Broker Architecture (CORBA) [Object Management Group, 1992]. The core of the object management architecture is an object request broker (ORB), which enables an object to request an operation execution from another, possibly remote, object. The objects can be very heterogeneous in the sense that they can be implemented in different programming languages and they can be running on different hardware platforms with different operating systems.

While CORBA has been designed for tightly-coupled applications, *Web Services* [We-

bopedia.com, accessed in 2004] provide a standardised way of integrating loosely-coupled, web-based applications. They use the XML, SOAP, WSDL, and UDDI open standards over an Internet protocol backbone: XML is used for tagging the data, SOAP for transferring it, WSDL for describing services, and UDDI for listing the services that are available.

An increasingly popular framework is Microsoft's *.NET* [Platt and Ballinger, 2002] which is compatible with Web Services and offers a similar functionality as CORBA. Interoperability is achieved by compiling the programming language into intermediate code, a concept similar to the Java bytecode. Unlike in CORBA, the developer does not need to provide interface definitions for the communicating programs.

3.6 Message Passing Libraries (MPI, PVM)

While distributed object computing has been geared towards distributed systems in general, high performance computing on multi-processor platforms has motivated message-passing libraries. Two prominent message-passing libraries are PVM, the parallel virtual machine [Geist *et al.*, 1994], and MPI, the message-passing interface [Pacheco, 1997].

MPI's main motivation is portability of software for massively parallel processors (MPP). Before the standardisation of MPI, software of MPP was not re-usable, since different vendors used proprietary message-passing libraries. To overcome this problem without compromising advantages of proprietary solutions, MPI's main objectives are portability of parallel applications, high performance, and a large set of point-to-point communication routines.

In contrast to programming MPPs with MPI, PVM allows a user to view a network of heterogeneous hosts as a single large parallel computer. PVM takes care of different data formats of the heterogeneous platforms and thus achieves portability in a more general sense than MPI: while MPI caters for portability of software from one platform to another, PVM provides the infrastructure to make different, heterogeneous platforms transparently work together. The network underlying PVM is also re-configurable at run-time, such that hosts can be added and removed and processes notified about the changing configuration.

3.7 Mobile Agents

Mobile agents are objects comprising data and code, which can decide themselves, when to move to another server. This *autonomy* adds a further degree of flexibility: as decisions concerning migration are taken locally by the agents, they can be used to provide a decentralised resource allocation protocol.

Historically, mobile agents are based on work carried out in the 80s on process migration and on distributed object computing [Shoch and Hupp, 1982; Jul *et al.*, 1988; Artsy and Finkel, 1989; Douglas and Ousterhout, 1991]. The combination of the two areas, i.e. to migrate distributed objects was first coined in [Jul *et al.*, 1988]. However, only with the spread of Java, researchers became widely interested in object mobility. Java has been so crucial for the development of mobile agents, as it has been designed as an architecture-independent network-centric programming language, which provides many of the requirements to implement object mobility as a standard feature. With Java's Remote Method Invocation (RMI) it is very simple to use mobile objects. RMI has been developed to support the invocation of methods on remote hosts, but it also caters for mobility, however, indirectly. When local, serialisable objects are passed as parameters to remote methods, RMI copies the object to the target machine. If the object at the source is then destroyed, it has effectively migrated from one machine to another. To support code mobility of classes used by the migrated object, Java provides dynamic class loading [Liang and Bracha, 1998].

Strong vs. Weak Mobility

In general, there are two types of mobility: *strong* mobility and *weak* mobility. In strong mobility, the execution state of the object is preserved during migration, whereas in weak mobility the code is restarted on arrival at the remote destination. Java only supports weak mobility, because it does not allow access to the execution stack. Nonetheless, it has been used by many mobile agent platforms.

Mobile Agent Platforms

There is an increasing number of mobile agent platforms of which a comprehensive overview is given by the *Mobile Agent List* [Hohl, accessed in 2003]. In this section some well-known

Java-based platforms will be briefly described.

Mole [Strasser *et al.*, 1996] was the first Java-based mobile agent framework and was developed at the University of Stuttgart. It provides the notion of *places*, the executing environments, where *user agents* are able to interact. They can communicate with the system resources via the so-called *service-agents* which are always stationary. Mole uses RMI for the communication and supports weak mobility.

Another popular framework is *Aglets* [Lange and Oshima, 1998b] which has been developed by the IBM Tokyo Research Laboratory and aims to facilitate the encoding of complex agent behaviour with moderate effort [Lange and Oshima, 1998a]. It provides enhanced facilities that allow the agents to co-operate with web browsers and Java applets. In addition to RMI, Aglets allows communication via TCP sockets.

Grasshopper [Baeumer *et al.*, accessed in 2003] has been the first mobile agent environment which is compliant to the MASIF standard [Milojicic *et al.*, 1998]. MASIF is based on CORBA and aims to provide inter-operability between different *mobile* agent platforms. Grasshopper also supports the FIPA specifications [FIPA, accessed in 2003], which has been designed to facilitate the *communication* between different agent platforms. The platform uses a proprietary ORB but can also be run using RMI or plain socket connections.

In contrast to the previous platforms, *D'Agents* [Rus *et al.*, 1997] is a Java-based platform which supports *strong* mobility: When an agent moves to another server, the complete state is packaged up, and the agent's execution continues after arrival. However, this facility has come with a high price. In order to support this mechanism, the Java Virtual Machine had to be modified, and the platform will only work with this specialised JVM.

3.8 Grid Computing Platforms: Globus

There exist a number of frameworks which are dedicated to *Grid Computing* [Baker *et al.*, 2001; Foster and Kesselman, 1998]. These include *Globus* [Foster and Kesselman, 1997], *Unicore* [Pallas, accessed in 2003], the *Load Sharing Facility* [Platform Computing, accessed in 2003] and *Legion* [Natrajan *et al.*, 2001]. A very popular one is the Globus Toolkit: It implements parts of the Open Grid Services Architecture (OGSA) standard [Foster *et al.*,

accessed in 2004], which is based on Web Services concepts and technologies. Globus provides a software infrastructure that enables applications to handle distributed computing resources as a single virtual machine. It provides a set of mechanisms for communication, resource discovery, resource allocation and data access, which form a basic infrastructure for a computational Grid. It also has a security infrastructure which allow its deployment in an open environment. The toolkit consists of several components each of which defines an interface and an implementation for various machine architectures and operating systems. A major drawback of the framework is that it has been developed in C language: therefore several versions have to be maintained for different machine architectures and operating systems.

3.9 Other Distributed Computing Platforms

Several other platforms do not target Grid Computing but can still provide a powerful infrastructure for such applications. Some well-known platforms are Jini [Sun Microsystems, accessed in 2003c], JMS [Sun Microsystems, accessed in 2003a], JXTA [Sun Microsystems, accessed in 2003d], and JADE [Bellifemine *et al.*, 1999]. These will be discussed in this section.

Jini

Jini [Sun Microsystems, accessed in 2003c] is Sun Microsystem's proposed architecture for embedded network applications. It provides a *lookup service* that enables devices to plug together to form an impromptu distributed system. Its communication infrastructure is not bound to any specific middleware. However, it is mainly used in combination with RMI. In contrast to mobile agent platforms, Jini does not support active or autonomous objects. However, the use of passive mobile objects in combination with Jini's lookup service provides a powerful infrastructure which is suitable for many distributed applications [Waldo, 2001].

JXTA

JXTA [Sun Microsystems, accessed in 2003d] is an open source project which has been started by Sun. It is a set of open protocols that is supposed to allow any connected device on

the network ranging from cell phones and wireless PDAs to PCs and servers to communicate and collaborate in a peer-to-peer manner. JXTA peers create a virtual network where any peer can interact with other peers and resources directly, even when some of the peers and resources are behind firewalls or are on different network transports.

Java Message Service (JMS)

In contrast to CORBA or RMI, Sun's *Java Message Service (JMS)* [Sun Microsystems, accessed in 2003a] is designed for asynchronous communication in *loosely coupled* systems. It can guarantee the delivery of a message to its recipient. For a user to participate in the system, authentication is required. JMS supports various communication mechanisms including publish-subscribe messaging. It is an open standard that is supported by several vendors, and also open source implementations are available.

Java Agent Development Framework (JADE)

JADE [Bellifemine *et al.*, 1999] is a very popular middleware platform in the agent research community in Europe. It is FIPA-compliant [FIPA, accessed in 2003] and thus can communicate with other FIPA-compliant agent platforms. JADE provides a rich API which enables the developer to build an agent-based distributed system with moderate effort. Among other features it supports agent behaviours, asynchronous messaging, and multiple communication protocols, including RMI, Corba, HTTP, and JMS. It provides mechanisms for resource discovery and several security features. Within a local domain it supports mobile agents via RMI.

3.10 Summary

In this chapter we addressed the different properties of the system's infrastructure which need to be considered when distributing applications for execution on remote resources. These include the machine architecture, the network connections, and the level of control that the system's scheduler has over the resources. We then introduced middleware, which provides the software infrastructure for executing distributed applications. Several middleware platforms have been discussed, including message passing libraries, distributed computing platforms,

mobile agent platforms, and Grid computing frameworks.

So far, we have discussed the relevant aspects of the applications that can benefit from being executed in parallel and of the systems in which they can be deployed. Since the objective of this thesis is to evaluate the performance of market protocols in computational clusters and Grids, the next chapter will provide a survey of various types of resource allocation protocols for such environments.

4 Resource Allocation Protocols

In this chapter a survey of different resource allocation protocols for computational clusters and Grids [8] will be given. The protocols examined here can be classified as *online* or *dynamic* resource allocation protocols, where the decisions concerning the allocation of tasks are taken at run-time. According to the direction of our work, the main focus of this survey will be on market protocols. However, other approaches will also be discussed.

4.1 Resource Allocation Problem

At first, we will briefly describe the *resource allocation problem* which will be tackled. We assume a system with several computational *resources* which are defined by their *speed* and *availability*. Due to background load, the latter may vary over time. Computational *tasks* are arriving to the system and need to be allocated resources. They have different *priorities* — which may or may not be time-dependent. The tasks may either be *independent* or be part of larger *supertasks*. The system is assumed to be geographically distributed, and therefore *communication delays* need to be taken into account.

The problem is how to allocate resources to the tasks so that a given performance metric, such as the mean completion time or the makespan, is optimised. This problem is NP-complete when computed *offline* [9], i.e. when all task arrivals and load variations are known before making the allocation decisions [Garey and Johnson, 1995]. For the *online problem* — where the events in the system are not known in advance — it is not even possible to obtain the optimal allocation. However, good results can be achieved with heuristics which will be described in this chapter.

[8] The differences between clusters and Grids have been briefly discussed in section 1.2: Grids are usually larger than clusters. Users and service providers are geographically distributed and belong to different organisations, with heterogeneous platforms and varying reliability and availability. Furthermore, the parties involved have different priorities, requirements, and goals, making resource management even harder.

[9] Note that the offline-computation of a solution is only of theoretical interest. It is often used to assess the quality of a solution that has been computed *online*, i.e. at runtime.

4.2 Classification of Resource Allocation Protocols

The aim of the *resource allocation protocols* discussed in this thesis is to maximise the benefit of the users who want to execute their tasks on the resources. To achieve this, the allocation decisions need to be based on the utilisation and performance of the machines available and on the requirements of the tasks [10]. There is a large number of different approaches which we classify according to the following criteria:

- *State-based vs Model-based:* Are the allocations based on a current snapshot of the system state (*state-based*) or on a model which predicts the system state (*model-based* or *predictive*)?
- *Preemptive vs Non-Preemptive:* Are tasks assigned to hosts once (*non-preemptive*) and then stay there, or can they migrate if it turns out at a later stage that it is advantageous to leave the machine (*preemptive*)?
- *Cooperative vs Competitive:* Are all parts of the system working towards a common, system-wide goal (*cooperative*) or do self-interested autonomous entities take decisions regarding the use of their resources (*competitive*)?
- *Centralised vs Distributed:* Does the responsibility for the allocation of tasks reside at one single location (*centralised*) or is the decision-making distributed among several machines (*distributed*)?

The following survey is summarised in Figure 3. The table provides additional information, such as the underlying middleware and the organisation of the system.

4.3 State-based, Non-Preemptive Resource Allocation

In state-based approaches to resource allocation, information about the current system state is used to decide at which host to start a task. The quality of this approach depends on the amount of state data available. Gathering the data is expensive but leads to more accurate

[10] Please note that in literature the terms *scheduling* and *load-balancing* are often used when referring to what we mean by *resource allocation*. In this thesis we will use these expressions interchangeably.

decisions. In non-preemptive resource allocation the task cannot be migrated elsewhere once it was launched on a host. Since this form of resource allocation is easy to implement and can lead to good results, it is widely adopted. In this section we review several techniques which are state-based and non-preemptive. After introducing several conventional scheduling heuristics we move on to systems which use market protocols.

4.3.1 Non-Competitive Protocols

Finding the optimal schedule of tasks *online* will often be infeasible because of the computational cost — or because the information about the system is incomplete. For this reason heuristics have been developed which aim to find a sufficiently *good* solution. Heuristics make assumptions about the computational tasks or the system resources. A very simple example is the "Strawman" *Round-Robin* protocol: it assumes that all resources perform equally well and that all tasks to be scheduled are equally important. In this section, several conventional (i.e. non-economic) heuristics will be described.

Shortest Job First (SJF), Min-min

Shortest Job First (SJF) [Chun and Culler, 2002] allocates the shortest task first for which it selects the best resource that is available. This strategy is useful if the goal is to minimise the mean slowdown of the tasks. Though, it also minimises the mean completion time. The idea behind it is that short tasks suffer a larger relative slowdown than longer ones if they are delayed by the same amount of time. Hence, allocating the shortest job first will usually result in a lower mean slowdown of all tasks than e.g. a first-in-first-out (FIFO) protocol. For the *unrelated machine case* a generalised version of Shortest Job First called *Min-min* [Casanova *et al.*, 2000; Maheswaran *et al.*, 1999a] can be used: Min-min tentatively schedules each task to each resource and computes the minimum completion time (MCT). For resources which are unavailable at the time, the calculation of the MCT considers the currently executing and already scheduled tasks. In *Min-min* the task with the *minimum* MCT is scheduled first. Note that in dynamic environments the calculation of the MCT requires estimates of the resource performance and task size, in which case this heuristic becomes a model-based resource allocation protocol, see subsection 4.5.

System / Authors	Protocol	Computation	Middleware	T	M	PE	Org
Streit et al.	SJF, LJF	Ind. tasks	n/a	s	–	–	c
Maheswaran et al.	Min-min, etc.	Ind. tasks	n/a	s	–	–	c
Casanova et al.	Min-min, etc.	PSAs	n/a	s	–	–	c
Abraham et al.	GA, SA, TS	PSAs	n/a	s	–	–	c
v. Nieuwpoort et al.	RS,RP,CHS,CLS,CRS	D & C	Satin	s	–	(✓)	d
Spawn	Second Price Auction	D & C	C, RPC	s	✓	(✓)	d
Ferguson et al.	Auction protocols	Ind. tasks	n/a	s	✓	–	d
ERA	First-Price Auction	Processes	n/a	s	✓	–	d
Chun et al.	First-Price Auction	Ind. tasks	n/a	s	✓	–	c
POPCORN	Auction protocols	Parallel appl.	Applets	s	✓	–	c
CPM	Economic protocols	Parallel appl.	RMI, JXTA	s	✓	–	d
MAJIC	Reverse auction	Serv. requests	Jini, RMI	s	✓	–	c
Dynasty	Brokering	D & C	C	s	✓	(✓)	h
OCEAN	Brokering	Parallel appl.	Java, .NET	s	✓	–	d
G-Commerce	Commodity market	Ind. tasks	n/a	s	✓	–	c
Traveler	Autonomous agents	Parallel appl.	RMI	s	–	✓	d
MATS	Autonomous agents	Parallel appl.	Voyager	s	–	✓	d
FLASH	Autonomous agents	Parallel appl.	Voyager	s	–	✓	d
Keren et al.	Autonomous agents	Parallel appl.	n/a	s	–	✓	d
Bredin et al.	Proportional share	Mobile agents	D'Agents	s	✓	✓	d
Harchol-Balter et al.	Task runtime prediction	Processes	Unix	m	–	✓	c
NWS	Performance prediction	n/a	n/a	m	–	–	n/a
Challenger	Bidding	Ind. tasks	n/a	m	✓	–	d
Nimrod-G	Economic protocols	Globus	PSAs	m	✓	–	h

Abbreviations:
T: type of resource allocation (s=state-based, m=model-based), M: market protocols (✓/–), PE: allows pre-emption/migration (✓/–), Org: organisation (c=centralised, d=distributed, h= hierarchical), ind. tasks: independent tasks, PSAs: parameter sweep applications, D & C: divide-and-conquer applications, serv. requests: service requests.

References:
Streit et al.: [Streit, 2001], Maheswaran et al.: [Maheswaran *et al.*, 1999a], Casanova et al.: [Casanova *et al.*, 2000], Abraham et al.: [Abraham *et al.*, 2000], v. Nieuwpoort et al.: [van Nieuwpoort *et al.*, 2001], Spawn: [Waldspurger *et al.*, 1992], Ferguson et al.: [Ferguson *et al.*, 1996], ERA: [Messer and Wilkinson, 1996], Chun et al.: [Chun and Culler, 2002], POPCORN: [Nisan *et al.*, 1998], CPM: [Buyya and Vazhkudai, 2001], MAJIC: [Levy *et al.*, 2001], Dynasty: [Backschat *et al.*, 1996], OCEAN: [Padala *et al.*, 2003], G-Commerce: [Wolski *et al.*, 2001], TRAVELER :[Wims and Xu, 1999], MATS: [Ghanea-Hercock *et al.*, 1999], FLASH: [Obelöer *et al.*, 1998], Keren et al.: [Keren and Barak, 1998], Bredin et al.: [Bredin *et al.*, 1998], Harchol-Balter et al.: [Harchol-Balter and Downey, 1997], NWS: [Wolski, 1997], Challenger: [Chavez *et al.*, 1997], Nimrod-G: [Abramson *et al.*, 2002].

Figure 3: Overview of systems

Longest Job First (LJF), Max-min

The opposite strategy, Longest Job First (LJF) [Streit, 2001], can be appropriate if the objective is to minimise the *makespan* (see subsection 2.6.1) of a set of tasks which belong to a larger application. The expectation is to overlap long-running tasks with short-running ones. A generalised version of LJF for the unrelated machine case is *Max-min* [Casanova *et al.*, 2000; Maheswaran *et al.*, 1999a]: It schedules the task with the *maximum* MCT first.

Sufferage

Sufferage also aims to minimise the application makespan. The rationale behind this heuristics is that a machine should be assigned to a task that would *suffer* the most if not assigned to it. For each task its *sufferage value* is defined as the difference between its best MCT and its second-best MCT. Tasks with high sufferage value will take precedence. In [Casanova *et al.*, 2000], an extended heuristic called *XSufferage* is proposed which is designed for the allocation of a parameter sweep application in a dynamic Grid environment. It takes advantage of file sharing and, according to the simulation results, achieves better performance and is more tolerant to errors in the estimated execution times.

Search heuristics

There exist several search heuristics which aim to find a near-optimal solution to the resource allocation problem in a limited amount of time. Almost any optimisation goal is possible, such as minimising the completion times of the tasks or the makespan of a super-task. Genetic Algorithms (GA), Simulated Annealing (SA), and Tabu Search (TS) are examples of popular *greedy search* heuristics. They will be briefly described in this section:

- *Genetic Algorithms (GA):* Genetic algorithms (GA) are based on the genetic process of biological organisms and are a popular technique that is used to find near-optimal solutions in optimisation problems. Possible solutions are encoded as *chromosomes*, the set of which is called a *population*. The population is iteratively operated on by the following steps until a stopping criterion is met: In the selection step, some chromosomes are removed and others duplicated based on their *fitness value* — a measure of the quality of the solution. This is followed by a crossover

step where some chromosomes are paired and the corresponding components of the paired chromosomes are exchanged. Finally, the chromosomes are randomly mutated, with the constraint that they still remain valid solutions [Abraham *et al.*, 2000; Braun *et al.*, 1999].

- *Simulated Annealing (SA):* Simulated Annealing (SA) exploits an analogy between the way in which a metal cools and freezes into a minimum energy crystalline structure (the *annealing* process) and the search for a minimum in a more general system. It is an iterative technique that has the ability to avoid becoming trapped at local minima: it uses a procedure which probabilistically allows poorer solutions to be temporarily considered in order to obtain a better search of the solution space. This probability is based on a *system temperature* that decreases for each iteration. As the system temperature *cools*, it becomes more difficult for currently poorer solutions to be accepted [Abraham *et al.*, 2000; Braun *et al.*, 1999].

- *Tabu Search (TS):* Tabu search is a solution space search that keeps track of the regions of the solution space which have already been searched so as not to repeat a search near these areas. It is a meta strategy for guiding known heuristics to overcome local optimality and has become an established optimisation approach that is rapidly spreading to many new fields [Abraham *et al.*, 2000; Braun *et al.*, 1999].

In [Braun *et al.*, 1999] these heuristics are applied to the *offline* scheduling of independent tasks onto heterogeneous resources in a cluster (*unrelated machine case*, see section 3.1). The aim is to minimise the makespan. The observation is that Genetic Algorithms (GA) perform best in most situations and also outperform other heuristics such as *Min-min* and *Max-min*.

The work presented in [Abraham *et al.*, 2000] deals with the problem of how tasks can be allocated *online* to geographically distributed computational resources. For the allocation a global scheduler is used, and the aim is to generate the schedules in a minimum period of time. The authors use simulations to compare the performance of GA, SA, TS, and hybrid random search techniques GA-SA and GA-TS. They find that the GA-SA has better

convergence than GA whereas GA-TS improves its efficiency.

Random Stealing, Random Pushing

Random Stealing (RS) and *Random Pushing (RP)* are well-known load balancing heuristics, used both in shared-memory and distributed-memory systems [van Nieuwpoort *et al.*, 2001]. Both heuristics are distributed in the sense that scheduling decisions are taken locally by the processors. At the beginning of the computation, all jobs of the application are distributed over the processors in the system. (During the execution the number of jobs may rise, e.g. in case of divide-and-conquer computations, see subsection 2.3.2.) Each of the processors maintains a *queue* for jobs which are waiting for execution. In Random Stealing, each processor executes its jobs until the queue becomes empty. Then, the processor attempts to steal a job from a randomly selected peer, repeating steal attempts until it succeeds. This approach minimises communication overhead at the expense of idle time: No communication is performed until a node becomes idle, but then it has to wait for a new job to arrive.

In Random Pushing, a processor, whose queue exceeds a certain length, takes a job from its queue and sends it to a randomly chosen peer. This approach aims to minimise processor idle time because jobs are pushed ahead of time, before they are actually needed, but comes at the expense of additional communication overhead. In [van Nieuwpoort *et al.*, 2001], Random Stealing, Random Pushing, and several improved variants have been evaluated for a range of divide-and-conquer applications (see subsection 2.3.2) in a wide-area setting. These variants included Cluster-aware Hierarchical Stealing (CHS), Cluster-aware Load-based Stealing (CLS), and Cluster-aware Random Stealing (CRS). The applications were implemented on top of the Satin system, a middleware in which the communication mechanisms of Java have been optimised for high-performance computing. In most experiments, Cluster-aware Random Stealing performed better than the other approaches.

4.3.2 Market Protocols

Market-based approaches to resource allocation provide an intuitive way of representing the system state and balancing the workload: they value resources and aim to achieve an efficient match of supply and demand. They satisfy some basic requirements for a Grid setting, as

decisions about whether to consume or provide resources are taken *locally* by the clients or service providers. The use of *currency* provides *incentives* for service providers to contribute resources. It also forces the clients to act responsibly, as, due to their limited budget, they cannot afford to waste resources. Recently, market-based approaches to resource allocation have received much theoretical interest [Ferguson *et al.*, 1996; Sandholm, 2000; Wellman *et al.*, 2001]. They have not only been applied to the allocation of computational resources but also in other fields, e.g. in 'information filtering economies' [Christoffel, 2001; Kephart *et al.*, 2001; Moss, 1999] or in the freight domain [Preist *et al.*, 2001].

Types of Protocols

For transactions between buyers and sellers different pricing mechanisms can be employed. Some systems use only a price, and match offers and bids. Others employ dynamic pricing where the sellers set their prices and may change them at any time, depending on the buyers' demand [Bredin *et al.*, 1998; Kephart *et al.*, 2001]. Another approach are auctions [Kagel, 1995] where buyers bid for resources according to a particular auction protocol.

A very familiar auction protocol is the *English* auction, in which a seller advertises a resource whose price is gradually increased as the bids come in — and the highest bidder wins. In a *Dutch* auction, the auctioneer starts with a high price which is decreased until a buyer is willing to pay it. In a *sealed-bid* auction, there is only one round of communication: All buyers submit sealed bids, and the highest bidder wins. There are different types of sealed bid auctions: In a *first-price* auction, the winning bidder pays a price which is equal to his own bid, whereas in the *second-price* auction, he only has to pay the second highest bid. The idea behind the latter type of auction, which is also called *Vickrey* auction, is that buyers are encouraged to express their true valuations of the resource, thus preventing strategic behaviour.

A different type of auction is the *double* auction which is a many-to-many protocol, and not a 1-to-many protocol as the above. Multiple buyers and sellers submit their bids and ask prices, and matches are made by the marketplace. A double auction can either be a *continuous* auction where transactions are carried out immediately whenever bids or offers

change [11], or a *periodic* auction in which the transactions are carried out only at periodic intervals.

Auctions in Clusters

Spawn [Waldspurger *et al.*, 1992] has been among the first systems which employed market protocols in a computational cluster. It used Vickrey auctions for the allocation of divide-and-conquer applications (see subsection 2.3.2) in a cluster of workstations. Ferguson *et al.* [Ferguson *et al.*, 1996] developed a load-balancing economy for the allocation of independent jobs in a network of processors. These jobs have various preferences on the service they wish to receive: best price, best service time, or a combination of the two. Several auction protocols are examined including English, Dutch, Hybrid, and Sealed Bid auctions.

The ERA system [Messer and Wilkinson, 1996] provides an operating-system level framework for resource allocation in a network of workstations. Its goal is to dynamically maximise performance whilst maintaining fairness between competing processes. A multiple-unit first-price auction is held at each workstation, and agents are used to disseminate resource information to other markets. The auction results in *proportional sharing*: it provides response time proportional to the money paid — and hence to the relative importance of the process. In an experiment, where a matrix multiplication computation was emulated, the framework could be shown to be scalable and low-overhead.

In a more recent work by [Chun and Culler, 2002] a first-price auction is used for the allocation of parallel tasks in a homogeneous computational cluster — which is modelled as a single but divisible resource. In simulations, the performance of the auction is compared to that of conventional protocols such as Shortest Job First (SJF, see subsection 4.3.1). The authors use the time-dependent *user utility* as performance metric (see subsection 2.6.1) which they consider to be a more realistic performance measure than e.g. the task completion time. The results show that using a first-price auction instead of traditional approaches can substantially increase the value delivered to the user.

Spawn, ERA, and the work by Ferguson use architectures in which auctions are held at each machine in the network. These architectures may be suitable for computational clusters

[11] A similar protocol is used at the New York stock exchange.

but will be problematic for a Grid: In a Grid the communication delays are higher. Also, the ownership of resources is distributed, leading to security problems. The architecture used by Chun et al. is also limited to a cluster because the assumption of having a single, homogeneous resource is not valid for a Grid.

Auctions: Grid-wide

POPCORN [Nisan *et al.*, 1998; Regev and Nisan, 1998] has been one of the first market-based systems to target resources on the Internet. It provides a market infrastructure that uses Applets to distribute large applications which can be broken up into independent computations. These include Genetic Algorithms, Simulated Annealing, Brute Force Search, and Code Breaking. A similar approach is used in the Compute Power Market (CPM) [Buyya and Vazhkudai, 2001] which also targets low-end personal computing devices as idle resources of CPU power. The system supports various economic models including the commodity market model, contract-net, and auctions. An important feature is its distributed infrastructure which consists of multiple interacting markets. Both POPCORN and the Compute Power Market use architectures which could be deployed in practice.

Reverse Auctions

MAJIC [Levy *et al.*, 2001] is a marketplace for the allocation of distributed resources which is based on Sun's Jini [Sun Microsystems, accessed in 2003c]. The system consists of a central marketplace with several *buyers* and *sellers*. In contrast to the previous systems it handles multiple parameters in the specification of utilities of the buyers and the costs of each resource. When a buyer sends a service request to the marketplace, the system performs a *reverse auction* where all service providers can participate. The provider, that charges the lowest price, wins. The allocation efficiency of MAJIC is studied both theoretically and experimentally. The authors report that load-balancing is achieved as a by-product.

Brokering

Two other systems, Dynasty [Backschat *et al.*, 1996] and OCEAN [Padala *et al.*, 2003], avoid the communication overhead of auctions. Dynasty allocates the subtasks of a divide-and-conquer application to machines in a computational cluster. It employs a hierarchical

brokering architecture in which the prices of the resources are periodically fixed. In addition, fees for migration and data transport services are introduced, according to the distance between source and destination. The local cluster brokers determine several statistics (e.g. load indices) and pass them up the hierarchy. Also, global knowledge is passed down. The brokers evaluate the qualification of their subbrokers in order to allocate the tasks efficiently.

The aim of OCEAN [Padala *et al.*, 2003] is to provide a scalable market infrastructure where resources like CPU time, associated memory usage, and network bandwidth are the traded commodities. It operates on a peer-to-peer network for which the authors developed efficient matching protocols. A buyer, who is looking for resources, needs to specify resource requirements as well as various constraints on the acceptable sales agreement details, such as price details and method of payment. The buyer's trade proposal is propagated through the network, maximising the number of matches found. After a successful match, the buyer and seller will enter detailed negotiations. OCEAN's peer-to-peer architecture is likely to be more scalable than a centralised system. However, the lack of a single, trusted marketplace leads to additional security problems.

Commodity markets

Another type of market mechanism uses the commodity market model: at the marketplace an equilibrium of demand and supply is iteratively determined by adjusting prices based on the supply and demand functions of the producers and consumers. The authors of G-Commerce [Wolski *et al.*, 2001] apply this approach to computational Grid settings. They study a market of hypothetical resource consumers (users and Grid-aware applications) and resource producers (resource owners who sell their resources on the Grid). In a simulation, the authors consider an independent task scenario in which they compare the performance of commodity markets to auctions. As performance metrics, mainly economic properties like equilibrium and price stability are considered. However, the utilisation of resources and the throughput of tasks are also compared. The authors conclude that commodity markets are a better choice for controlling Grid resources than auctions. An outline of how the examined market model could be implemented in practice is given in [Wolski *et al.*, 2003].

None of the above systems allows *true* task migration. However, Spawn and Dynasty, which deal with the allocation of divide-and-conquer applications (see subsection 2.3.2), allow tasks to send subtasks to remote machines. Preemptive protocols will be discussed in the following subsection.

4.4 State-based, Preemptive Resource Allocation

In state-based, non-preemptive approaches to resource allocation, the task is stuck on one machine once launched. However, if the environment is very dynamic, it may be advantageous for a task to migrate elsewhere. Operating systems researchers already investigated how the allocation of resources can be optimised with mobility [Cabrera, 1986; Harchol-Balter and Downey, 1997]. Mobile agents [Bredin *et al.*, 1998] add a further degree of flexibility: tasks become agents and can decide themselves, when and where to move to, with a global pattern of load-balancing emerging. Also, mobile agents allow migration to remote networks and are therefore suitable for computational Grids.

4.4.1 Non-Competitive Protocols

There exist a number of systems which use a mobile agent based infrastructure [Ghanea-Hercock *et al.*, 1999; Obelöer *et al.*, 1998; Bredin *et al.*, 1998; Bredin *et al.*, 1999; Xu and Wims, 2000; Wims and Xu, 1999; Keren and Barak, 1998]. Three systems, TRAVELER [Xu and Wims, 2000; Wims and Xu, 1999], MATS [Ghanea-Hercock *et al.*, 1999], and FLASH [Obelöer *et al.*, 1998], make specifically use of autonomous mobile agents roaming the net in search of the best host. The resource allocation protocols in these systems can be classified as cooperative.

Autonomous, mobile agents

TRAVELER [Xu and Wims, 2000; Wims and Xu, 1999] allows clients to wrap their parallel applications as mobile agents which are dispatched to a resource broker. The broker forms a parallel virtual machine atop available servers to execute the agents. Instead of hosts asking the broker when they have capacities, the system lets its autonomous agents roam the net

to find servers to run on. The platform is based on RMI but has been extended to support strong mobility of multi-threaded agents. The system's performance has been evaluated for two parallel applications: sorting and LU factorisation problems.

MATS [Ghanea-Hercock *et al.*, 1999] uses a combination of collaborative and mobile agents to compute large parallel applications. The system, which has been evaluated for a standard Genetic Programming problem, distinguishes several agent roles: A *Hive* is responsible for managing user interaction and determining how tasks are to be distributed, while *Queen* agents are run on local servers and control several *Worker* threads. Specialised lightweight mobile agents (*Scouts*) are used for messaging and finding idle computer resources on remote workstations.

In FLASH [Obelöer *et al.*, 1998], mobile agents are used to compute a ray-tracing application in a cluster of workstations. A system agent maintains information of the whole system and passes it to nodes, which keep information about the locally residing mobile agents. The agents migrate through the system searching for free resources. FLASH combines application and system information based load management. Therefore it is able to react efficiently on dynamic background load and avoids unnecessary migration of agents with a short life span.

Additional optimisation of communication

While most of the above systems aim to optimise CPU usage, Keren and Barak [Keren and Barak, 1998] also optimise an agent's location with respect to its communication partners. The system is geared towards improving the overall performance by a dynamic match between the available resources and the execution requirements of the agents. This is accomplished by agents migrating to the hosts where the agents reside with which they communicate. The authors simulated a Matrix computation, for which a 30-40% improvement could be achieved compared to a static placement scheme.

4.4.2 Market Protocols

In [Bredin *et al.*, 1998], a framework is presented, which provides market-based resource control for mobile agents. It is based on the D'Agents platform [Rus *et al.*, 1997] and there-

fore allows strong mobility of the agents. The agents need to travel over certain routes in the network on which they need to consume computational resources. To allocate resources to the agents, the system uses electronic cash, a banking system, and a set of resource managers. In the paper, fixed pricing and dynamic pricing protocols are presented. The authors' focus is on seller-adjusted pricing and sealed-bid second price auction as mechanisms for dynamic pricing. In another publication [Bredin *et al.*, 2001], the authors assume the use of a market protocol and present an algorithm for planning a mobile agent's itinerary. It is supposed to guarantee the agent's optimal completion time. The algorithm is evaluated by discrete event simulations with Swarm [Minar *et al.*, 1996]. In the simulations, a network model is used which is created with a stochastic network topology generator.

4.5 Model-based Resource Allocation

Model-based approaches to resource allocation aim to predict the duration of the tasks and/or the load on the resources — and thus their performance. They are much rarer, as they involve two very challenging problems: how to obtain an initial model and how to adapt the model as time passes.

4.5.1 Non-Competitive Protocols

In the area of operating systems, some researchers explored this approach and used distributions of CPU load and expected process lifetime to decide if and when to migrate tasks [Cabrera, 1986; Harchol-Balter and Downey, 1997].

For computational Grids, the prediction of resource performance has recently received much attention: The Network Weather Service (NWS) [Wolski, 1997] can forecast the load in global computing environments where platforms like Globus [Foster and Kesselman, 1997] are deployed. It takes periodic measurements of deliverable resource performance from distributed network resources and uses numerical models to dynamically generate statistical forecasts of future performance levels. The forecasts are made available to schedulers and other resource management mechanisms at runtime. They allow them to determine the future quality of service for each resource. The authors report that the results of the pre-

dictions are more accurate than those generated from measurements of current conditions alone. The approach, which is based on time-series analysis techniques, performs almost as well as more complex methods. It is lightweight and therefore appropriate for dynamic computational settings [Wolski, 1997].

4.5.2 Market Protocols

A model-based approach is also used in the Challenger system [Chavez *et al.*, 1997] which simulates the allocation of independent tasks. It implements load-balancing with a market approach — however, without money. When a task is created, a *request for bids* containing its priority value and information which can be used to estimate its duration is sent to the agents in the network. These make bids giving the estimated time to complete that task on their machine [12]. Important parameters, which have a major impact on the system performance, are the message delays and errors in estimating the task's completion time. Learning behaviour has been introduced in order to deal with these problems.

Another model-based resource allocation protocol is used by the Nimrod-G Resource Broker [Abramson *et al.*, 2002]. Nimrod-G is a resource management system for scheduling parameter sweep applications (see section 2.1) on globally distributed resources with varying quality of service. The system is an economic-driven environment which supports various market protocols such as the commodity market model, posted pricing, and bargaining. It predicts the future performance of the resources in the system by resource capability measurements and load profiling.

4.6 Summary

In this chapter a survey of different resource allocation protocols for computational clusters and Grids has been given. Due to the topic of this thesis, the main emphasis was on market protocols. However, other approaches have also been discussed. We classified the resource allocation protocols according to several criteria: state-based vs. model-based, preemptive vs. non-preemptive, cooperative vs. competitive, and centralised vs. distributed.

[12] This approach belongs to the class of *bidding* mechanisms [Casavant and Kuhl, 1988].

5 Research Objectives and Related Work

The previous chapter gave an overview of existing work on market protocols in computational clusters and Grids and introduced several conventional scheduling heuristics. Some of the described systems merely try the market approach, whereas others have the objective to improve performance. However, what is missing, is a systematic comparison of different market protocols, which will allow a system designer [13] to choose the most appropriate protocol for a given situation. In this chapter, we state the objectives of this thesis and discuss related work.

Our objectives and assumptions can be summarised as follows:

- We aim to model a marketplace for computational resources which can be applied to both computational clusters and Grids. We assume that the resources are *not* dedicated: Their availability may vary due to *background load*, which is outside the control of the marketplace.

- Our main focus is on independent tasks which arrive randomly and need to be allocated *online*. These applications are very common and easy to execute on geographically distributed infrastructures. However, we also examine cases where tasks arrive in bursts [14].

- As resources we consider PCs which are idle most of their time and therefore can offer large amounts of processing power to computationally intensive applications [15]. As these PCs belong to different users or organisations, there is a need for a marketplace which can provide incentives for participation.

- We consider the market as a *tool* to achieve an efficient allocation of resources. Our main focus is on the design of the protocols which are used at the marketplace. The choice of pricing strategies for the Clients and Servers is not the subject of this work:

[13] The *system designer* is the person who designs the marketplace and decides which resource allocation protocol is used.

[14] This is characteristic for parameter sweep applications (see section 2.1).

[15] In addition to this, we examine a situation where parallel applications are allocated to multiprocessor machines.

We assume a *managed system* which has the power to enforce these strategies. Money has no value as such, and hence there is no need to deal with resource accounting.

- Our objective is to maximise performance from the user's perspective, and therefore, we choose the performance metrics accordingly (see subsection 2.6.1). In the first scenario, our goal is to minimise the average completion time of the task, and in the second one, their weighted completion time. In the third scenario, tasks have hard or soft deadlines. As performance metric, we use the weighted completion rate (for hard deadlines) and the aggregate user utility (for soft deadlines).

- To maximise the performance metrics, the resource allocation protocols need to avoid unnecessary communication. All examined market protocols are double auctions of *sealed-bid*-type ("fire and forget"), i.e. there is no complex negotiation with multiple rounds. They exploit techniques such as continuous auctions, periodic auctions, proportional sharing, preemption, migration, and the use of reserve prices by the Servers. In addition to the market protocols, several conventional scheduling heuristics are examined.

- We want to be able to determine the advantages and disadvantages of the protocols for different situations which are characteristic for computational clusters and Grids. This requires a comprehensive exploration of the parameter space. We investigate the impact of parameters, such as the number of resources in the system, the heterogeneity of these resources, the amount of load and background load in the system, the local scheduling policy at the resources, and the communication delays.

We are aware of only few other efforts to evaluate the performance of different market protocols for computational clusters or Grids. They will be described in this chapter.

5.1 POPCORN

The authors of POPCORN [Nisan *et al.*, 1998; Regev and Nisan, 1998] (see subsection 4.3.2) use simulations to compare several market protocols for the allocation of computational resources in a globally distributed system. In the simulations, the arrivals of buyers and sellers

are modelled by Poisson processes. The buyers have valuations for the computation of their tasks, which are time-dependent and are expressed by a decay function. In our thesis, a similar scenario will be investigated.

The examined protocols are of sealed-bid-type: They require only a single round of communication and are efficiently computable. However, only few protocols are studied, which include Vickrey Auctions, Double Auctions, and Clearinghouse Double Auctions. The latter auction type is similar to the technique which [Wolski *et al.*, 2001] refer to as the *Commodity Market Model* (see subsection 4.3.2). The main difference to our work is that these auctions are only carried out at periodic time intervals, rather than continuously. As will be shown, these *periodic auctions* often result in poor performance.

In their simulations, the authors use other performance metrics for the assessment of the protocols. The focus is on economic aspects of the system, like price stability and social efficiency, i.e. the generated welfare of the agents — which is not the concern of our work. The results show that the Clearinghouse Double Auction protocol leads to the highest economic efficiency and the best price stability. However, the experiments are limited to a few parameter sets, i.e. there is no systematic exploration of the parameter space.

5.2 G-Commerce

The work presented in [Wolski *et al.*, 2001] investigates computational economies for controlling resource allocation in computational Grid settings ('G-Commerce', see subsection 4.3.2). The resources are traded at a central marketplace in which several *consumers* and *producers* participate. Simulations are used to compare the performance of commodity markets to that of Vickrey auctions. As in the work by [Nisan *et al.*, 1998], all transactions are carried out at periodic intervals, rather than immediately.

As performance metrics, the authors examine price stability, utilisation of resources, and throughput of jobs. The results indicate that commodity markets perform better than Vickrey auctions. One interesting aspect of the examined scenario is that the producers (i.e. service providers) set their prices based on past revenue. We believe that this technique can help to improve performance in certain situations, and we will therefore implement it in two of our

protocols (CDA-RES and HBP-RES, see subsections 6.5.2 and 6.5.7, respectively).

5.3 Work by Ferguson et al.

In the work reported in [Ferguson *et al.*, 1996], several auction protocols (English, Dutch, Hybrid, Sealed-Bid) are compared to non-economic strategies. In contrast to our work, the studied system is a cluster of processing nodes connected by a point-to-point network, and not a geographically distributed infrastructure with a central marketplace. At each of the processing nodes, tasks are generated which migrate to other nodes and seek CPU time. In the system, auctions are held at each processing node, rather than centrally. We believe that, due to its overhead, such an approach would be hard to implement in a computational Grid. In the simulations, a performance metric called *waiting time* is used. It corresponds to the task completion time, that we also use as metric in one of our scenarios. Its main limitation is that it does not consider the different priorities of tasks. According to the authors, the results show that the auctions can achieve better performance levels than non-economic algorithms.

5.4 Work by Chun et al.

The authors of [Chun and Culler, 2002] use simulations to examine the performance of market protocols in a computational cluster. In the studied scenario, the cluster is modelled as a single, divisible resource consisting of identical processors, which is also fully dedicated, i.e. without any background load. The author's goal is to maximise the value delivered to the users: Users are modelled as having a utility function for each task, which measures its value as a function of its slowdown (see subsection 2.6.1). A task's value declines over time — which can be considered a *soft deadline*.

As performance metric, the authors use the aggregate utility of all tasks. We will use a similar performance metric in one of our scenarios, in which the tasks have time-dependent priorities (see chapter 10).

While the main focus of our work is on independent tasks or embarrassingly parallel applications, [Chun and Culler, 2002] study both, sequential and highly parallel workloads. In the latter case, tasks are assumed to be *rigid*, i.e. they require a specified number of pro-

cessors which cannot be changed (see subsection 2.2.5). We also study a case with parallel workload. However, our tasks are assumed to be *moldable*, which according to [Cirne and Berman, 2001b] is more common than rigid tasks.

The authors study several market protocols and conventional scheduling heuristics, all of which carry out the transactions only at periodic intervals. They examine first-price auctions in which the bids for the tasks are static and also first-price auctions where the price bids decrease according to the tasks slowdown [16]. The performance of the auction protocols is compared to that of commonly used conventional scheduling heuristics, such as Shortest-Job-First (SJF) (see subsection 4.3.1) and PRIO-FIFO. In PRIO-FIFO, the scheduler uses a set of FIFO queues with different priorities (see subsection 6.5.12 for more details).

The results show that, in comparison to traditional approaches like SJF or PRIO-FIFO, the first-price auction with time-dependent user valuations can substantially increase the value delivered to the user. The gain is higher for parallel load than for sequential load. However, in comparison to the first-price auction with static bids, improvements can only be observed for highly parallel workloads. The authors also found that preemption does not add significant value.

5.5 Work by Bredin et al.

In [Bredin *et al.*, 2001; Bredin, 2001], a scenario is studied which is not typical for a computational Grid, but which could become relevant in the future. The authors simulate a system that provides market-based resource control for mobile agents (also see subsection 4.4.2). The agents need to travel over certain routes in a network, on which they consume different types of computational resources, for which they have to pay. The agents have different monetary endowments, which reflect their priorities, and their goal is to complete their routes as fast as possible.

A proportional sharing protocol is proposed in which the rate, at which a job is processed, is proportional to the agent's price bid. Each agent uses a strategy in which its price bid is

[16] We adapted this first-price auction with time-dependent price bids for our scenario with multiple resources which requires a double auction (CDA-TDB, see subsection 6.5.3).

proportional to the congestion at a resource, i.e. the number of other agents already executing there. This protocol is compared to several traditional heuristics, including First-Come-First-Served (FCFS), Shortest-Remaining-Processing-Time (SRPT), and an equal-share policy. In SRPT, the shortest job is executed first (as in SJF), and an agent chooses a resource based on how many agents are already there. According to the authors, SRPT is locally optimal in the sense that it minimises the average waiting time at the resources.

In the simulations, both undersubscribed and oversubscribed systems are examined, and also the effects of network delays and errors of the job size estimation are investigated. The authors find that the cost of the prioritisation is about 8% of the overall performance. However, the market protocol allows to run important jobs even when the system is oversubscribed, and can also operate under uncertainty and network delay.

It must be noted that the mean completion time of the agents, which is used as performance metric in the simulations, does not reflect the priorities of the tasks. Hence, the comparison is not fair to the market protocol.

5.6 Work by Kim et al.

In [Kim *et al.*, 2003] a scenario is studied, in which independent tasks with different priorities are generated by a Poisson process and are allocated in a cluster of eight machines. There are different types of resources in the system, on which each task will have different execution times (*unrelated machine case*, see section 3.1). The value of a task's execution depends on its completion time — which is, again, similar to one of our scenarios where tasks have time-dependent priorities (see chapter 10). The authors limit their experiments to an oversubscribed system, where there is not enough capacity to execute all tasks.

Several heuristics are compared, all of which operate in batch mode, i.e. the transactions are carried out only at periodic intervals. These heuristics include *Max-Min*, *Min-Min* (see subsection 4.3.1), and several variants which take into account the priorities of the tasks, such as *Slack Sufferage* and *Max-Max*. Slack Sufferage uses the Sufferage concept (see subsection 4.3.1), but also takes into account task priorities and deadlines. Max-Max is based on Min-Min but prioritises tasks with higher value. In the simulations, several situations are

investigated: tasks with loose or tight deadlines, high and low heterogeneity of resources. In most of the experiments, Max-Max and Slack Sufferage provide the best results.

We believe that the studied scenario (unrelated machine case) is not very common, and that the *related machine case* (see subsection 3.1) will be a close approximation of reality in most situations. Also, we find it unrealistic that each task can be submitted with a set of estimates of completion times at the different resources. For these reasons, our thesis will focus on the related machine case.

5.7 Summary

In this chapter, we stated our research objectives and discussed related work on performance evaluation of market protocols. We pointed out the limitations of these approaches. If market protocols were examined for computational Grids, the experiments were limited to just a few protocols and parameter sets — a comprehensive exploration of the parameter space was missing. Furthermore, none of the work considered continuous auctions or proportional sharing. Task preemption and migration were only examined by [Chun and Culler, 2002] and [Ferguson *et al.*, 1996], respectively, and only for computational clusters. In two other cases, the authors studied economic properties of the market rather than measuring the performance as perceived by the user.

In the next chapter, we will introduce our simulation model of an electronic marketplace for computational resources.

6 Simulation Model

6.1 Introduction

For our evaluation of the market protocols, we decided to use discrete-event simulations, as they allow us to explore various scenarios and to use arbitrary values for parameters like message delays, processing delays, arrival times, etc. We use *continuous* — rather than discrete — simulation time: There are no discrete time-steps, and hence no restrictions to the granularity of the simulations. In this chapter, we present the model that we use for our simulations. A description of the simulation framework, which implements this model, can be found in chapter B of the appendix. We start by giving a detailed characterisation of the actors in the system and the underlying communication model. Next, we describe the sequence of interactions in the system which all the examined resource allocation protocols have in common. Finally, we introduce the protocols that are studied in this thesis.

6.2 Model Description

Our model represents an electronic marketplace for distributed computational resources. As shown in Figure 4 there are three main actors in the model, which are assumed to be distributed over the Internet: the Clients, the Servers and the Electronic Marketplace (EMP). Clients generate tasks which require computational resources for their execution. The Servers provide these resources: they advertise and sell them at the Electronic Marketplace. The accounts of the Clients and Servers are located at a Bank. In this section a detailed description of the model, its actors, and the underlying assumptions will be given. A summary of the main variables of the actors is shown in Figure 5.

6.2.1 Clients

There is a constant number of Clients N_{Client} in the system [17], each provided with an initial amount of money ('endowment') M. Each Client generates tasks at a rate which is modelled

[17] In this thesis, we only consider the *stationary* case, i.e. system properties such as the number of Clients and Servers or the task arrival rate do not change over time — or they change so slowly that this cannot be noticed.

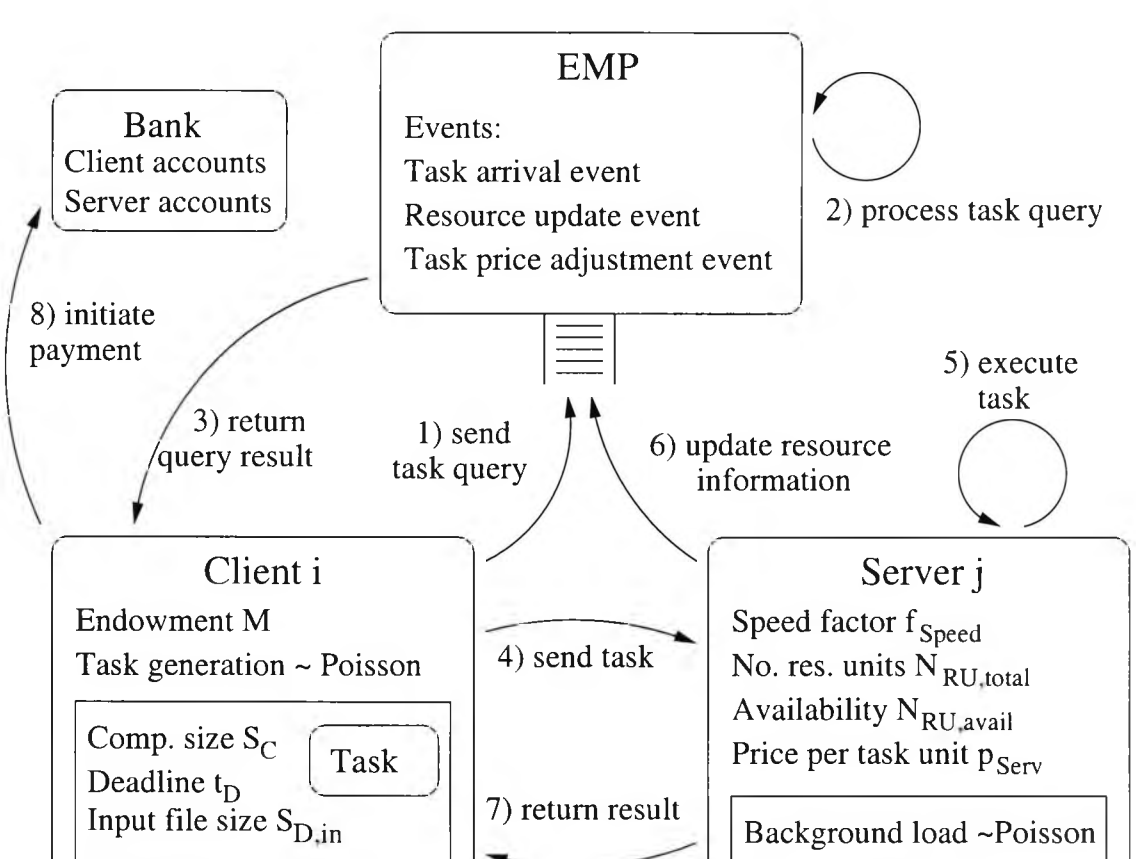

Figure 4: Model of the marketplace

by a Poisson arrival process. Poisson processes have been chosen, since they are suitable for describing user session arrivals on the Internet [Floyd and Paxson, 2001]. Also, they are used in models describing task arrivals in supercomputing centres [Downey, 1997]. A Poisson arrival process has an exponential inter-arrival distribution. Its density function is given by $f(\tau) = \lambda \cdot e^{-\lambda\tau}$, where λ is the inverse of the mean time between two task creations.

6.2.2 Tasks

A task which is created by a Client is characterised by the size of its computation S_C (in task units), the sizes of the input and output files $S_{D,in}$ and $S_{D,out}$ (in bytes), and — in the market

This is realistic if the task execution times are relatively short in comparison to the variations of the system properties.

Entity	Parameter	Symbol	Distribution
Client	Number of Clients	N_{Client}	–
	Endowment ("money")	M	–
	Task inter-arrival time	τ	exponential
Task	Computation size	S_C	-
	Weight	w_{Task}	uniform
	Deadline	t_D	–
	Price bid	p_{Task}	–
	Input file size	$S_{D,in}$	–
	Output file size	$S_{D,out}$	–
Server	Number of Servers	N_{Serv}	–
	Speed factor	f_{Speed}	uniform
	Number of resource units	$N_{RU,total}$	–
	Availability (res. units)	$N_{RU,avail}$	–
	Res. units allocated (task)	$N_{RU,alloc}$	–
	BG task res. units	$N_{RU,BG}$	–
	BG task comp. size	$S_{C,BG}$	–
	Price per task unit	p_{Serv}	–
	Eff. exec. speed (total)	$s_{Eff,total}$	–
	Eff. exec. speed (task)	s_{Eff}	–
Network	Communication delay	T_{Comm}	lognormal

Figure 5: Summary of the main variables

protocols — its price bid p_{Task}. It may also have a deadline t_D or a weight w_{Task}. We assume that the execution of a task can be spread over an arbitrarily large fraction of a resource. We only consider the case that there are no dependencies among the tasks. Also, the task's memory requirements are not considered in our model.

Computation size distributions

The computation size S_C of a task (also called 'runtime') can have different probability distributions. These depend on the type of workload used. We use a probability distribution which is based on workload logs collected from large-scale systems in production use. We consider a model which uses a loguniform distribution for the job runtime [Cirne and Berman, 2001a; Downey, 1997]. A distribution of a random variable X is loguniform if $P[X < x] \sim \log(x)$.

6.2.3 Servers

There is a constant number of Servers N_{Serv} in the system, which provide CPU resources to the Clients and charge them according to the protocol of the marketplace.

We assume that each Server has a resource consisting of one or more resource units (RU). This allows us to model either time-shared resources (e.g. a stand-alone machine where a resource unit corresponds to a time share of its CPU) or space-shared resources (e.g. a multiprocessor machine where a resource unit corresponds to a processor). We further assume that a task can execute on several resource units in parallel and that each unit can be split and allocated to different tasks.

To enable modelling of resources with different speed, we introduce a speed factor f_{Speed}, which is equal to 1.0 on a reference machine. On a reference machine, one resource unit will need one unit of simulation time (e.g. one second) to execute one task unit. Thus, if a constant number of resource units $N_{RU,alloc}$ is allocated to a task with size S_C, the duration of its execution is given by $\frac{S_C}{N_{RU,alloc} \cdot f_{Speed}}$.

Background Load

We also introduce *background load* on the Servers' resources, i.e. a load generated by tasks which are outside the control of the EMP. We assume a Poisson distribution for the arrivals of background tasks, each of which has the same computation size, $S_{C,BG}$, and the same number of resource units allocated, $N_{RU,BG}$. If a background task arrives at a time when no (or not enough) resource units are available, it is put into a queue. Background tasks waiting in this queue are started immediately when resources become available. This guarantees that the (average) resource share of background tasks is the same for any load of incoming tasks.

Scheduling policy

There are different ways of scheduling resource units to tasks which have been allocated by the EMP. We consider the following cases ($N_{RU,alloc}$: number of resource units allocated, $N_{RU,avail}$: number of resource units available):

1. allocate all available resource units of the Server, $N_{RU,avail}$, to the task. The number of allocated units, $N_{RU,alloc}$, does not change throughout the task's execution. This type of policy may be used for *moldable* parallel tasks which are executed on a space-shared resource (see subsection 2.2.5).

2. allocate all available resource units of the Server, $N_{RU,avail}$, to the task. The number of allocated units, $N_{RU,alloc}$, may increase or decrease during the task's execution due to changes in the background load, which is given priority. Such a policy may be used on a single-processor, time-shared machine which is not dedicated to the execution of incoming tasks — e.g. a PC in a university lab.

3. allocate all available resource units of the Server, $N_{RU,avail}$, to the task. The number of allocated units, $N_{RU,alloc}$, may increase or decrease during the task's execution due to changes in the background load, which is given priority. Also, the task can be preempted by an another task with higher priority. In this case it may either suspend its execution or migrate to another resource.

4. allocate a fraction of the currently available resource, $N_{RU,avail}$, which is proportional to the task's price bid $p_{Task,i}$. The allocated share, $N_{RU,alloc}$, may increase or decrease during the task's execution due to arrivals and departures of other tasks or changes in the background load, which is given priority. It is given by $N_{RU,alloc} = \frac{p_{Task,i}}{\sum_i p_{Task,i}} N_{RU,avail}$, where $\sum p_{Task,i}$ is the sum of all price bids at the Server, including the task's bid itself. In section A.1 this scheduling policy is described in more detail.

6.2.4 Electronic Marketplace (EMP)

The Electronic Marketplace (EMP) provides facilities for the Servers to advertise their resources. The parameters to be published include the number of available resource units, $N_{RU,avail}$, the price per task unit, p_{Serv}, and the resource's speed, f_{Speed}. For the Clients it provides means to search for a suitable resource and negotiate the price.

6.2.5 Communication Model

As the actors in our systems are distributed over the Internet, communication delays need to be taken into account. According to experimental results in [Schroeder and Boro, 2001], the communication delay T_{Comm} on a network link i can be considered to be a lognormally distributed random variable. This observation is supported by [Floyd and Paxson, 2001].

In our simulation model, the communication delay for a data transfer is determined by the latency and bandwidth of the network link and by the size of the transmitted data.

We assume that the mean μ_{Comm} and the standard deviation σ_{Comm} of the probability distribution of T_{Comm} are unique for a given network link i and that they also depend on the size of the data S_D to be transferred. The communication delay T_{Comm} on the network link i is lognormally distributed with mean $\mu_{Comm,i} = f_{\mu,i}(S_D)$ and standard deviation $\sigma_{Comm,i} = f_{\sigma,i}(S_D)$.

The observations by [Schroeder and Boro, 2001] indicate a linear relation between the mean communication delay μ_{Comm} and the size of the data S_D: Hence, for our lognormal distribution $LN(\mu_{Comm,i}|\sigma_{Comm,i})$ we assume $\mu_{Comm,i} = A_{\mu,i} \cdot S_D + B_{\mu,i}$. The factors $A_{\mu,i}$ and $B_{\mu,i}$ are constants. $A_{\mu,i}$ corresponds to the inverse of the bandwidth (or throughput) and $B_{\mu,i}$ to the network latency. Similarly, in our simulation model, the standard deviation $\sigma_{Comm,i}$ is modelled using constant factors $C_{\sigma,i}$ and $D_{\sigma,i}$. It is given by $\sigma_{Comm,i} = C_{\sigma,i} \cdot S_D + D_{\sigma,i}$.

Furthermore, we assume that the load of data transfers within our system is already considered in the probability distributions used. In the simulations presented here, we use a network topology where all actors are on different nodes, where all nodes are connected to each other, and where all network links are equal.

6.3 Interactions in the system

Before discussing the resource allocation protocols that are examined in this thesis, we will describe the sequence of interactions in the system which these protocols have in common. The steps (1-8) are shown in Figure 4.

Server: Registration of resources

Before any interactions at the EMP can take place, the Servers need to register their resource offers. These include the following parameters: the number of resource units available $N_{RU,avail}$, the speed factor f_{Speed}, and the price per task unit p_{Serv}.

Client: Task creation and query at the EMP

Tasks are created using an exponential distribution for the inter-arrival time τ. For each task,

two objects are generated. The Task Query object contains the necessary information for a query to the EMP: the computation size S_C, deadline t_D, price bid per task unit p_{Task}, task ID, and a reference to the Client. On creation of a task, the Task Query object is sent to the EMP (step 1) and remains there until an appropriate resource is found (step 2). The Task Data object represents the input parameters of the task. In case of a successful query, it will later be sent from the Client to the Server [18].

EMP: Process task query

Each task query which arrives at the EMP is processed immediately. If a suitable resource is available and the task's price bid is high enough, the resource is reserved and the query result is sent back to the Client (step 3). The resource is considered unavailable until the task completes its execution at the Server [19]. If no match is found, the task will wait at the EMP until a suitable resource becomes available or the task's deadline has passed. Also, for the market protocols a mechanism is provided which we refer to as *task price adjustment*: It enables Task Query objects to linearly increase their price bid p_{Task} at regular time intervals in order to be served eventually [20]. This aspect of the protocol is only used in some of the simulations presented in this thesis. The procedures for the main loop of the EMP and for the task arrival event are shown in Figure 69 and 70, respectively. The procedure for the *task price adjustment event* is described in subsection A.2.5 in the appendix.

Please note that, in order to simplify the matching of tasks to resources, only one incoming task query or resource update is processed at a time. In our model these events are serialised, as is indicated by an input queue for the EMP in Figure 4. Hence, in a 'task arrival event' exactly one task is matched to N resources, and in a 'resource update event' exactly one resource is matched to N tasks.

[18] There is an additional step in the process if resource scheduling policy 1 is used: When the Client receives a query result, it will contact the Server to check whether the required number of resource units is still available. If this is the case, the Server will reserve it, update the information at the EMP, and request the Task Data object to be sent by the Client. If not, the Client will have to send the Task Query object to the EMP again.

[19] An exception to this is the PSP protocol where several tasks can share a resource. Also, in the preemptive protocols, the resource is considered available to higher-priority tasks.

[20] In this case the Task Query object needs to contain an initial minimum price $p_{Task,min}$, a maximum price $p_{Task,max}$, and a negotiation time T_{Neg}.

Server: Task execution

After receiving a query result from the EMP, the Client sends the Task Data object to the Server (step 4). The Server executes the task on the number of resource units $N_{RU,alloc}$ which have been allocated by the EMP (step 5). Hence, the effective execution speed is given by $N_{RU,alloc} \cdot f_{Speed}$. Note that in resource scheduling policies 2–4, $N_{RU,alloc}$ can vary during the task's execution. Thus, the duration of the execution is not known a priori. On completion, the resource information at the EMP is updated (step 6) and the result of the task is sent to the Client (step 7). If it arrives before the deadline, a bank transfer from the Client's to the Server's account is initiated (step 8). Otherwise, the Server is penalised and receives nothing. Note that the accounting is not relevant to the results reported in this thesis.

6.4 Assumption: Managed System

We make the assumption of a *managed system*, in which the market is a *tool* to achieve the efficient allocation of resources. Our main focus is on the design of the protocols which are used at the marketplace. The choice of pricing strategies for the Clients and Servers is not the subject of this work: We assume that these strategies can be enforced by the system. Money has no value as such, and hence there is no need to deal with resource accounting.

If the Servers belong to the same organisation(s) as the Clients, it will not be necessary to incentivise the Servers to participate in the marketplace. Therefore, it will not be a problem to enforce their pricing strategy. If the Servers are self-interested and utility-maximising, it may still be possible to enforce a pricing strategy. However, it will be necessary to incentivise them so that they participate in the marketplace. They could be compensated in a different way, e.g. by receiving a flat fee on a monthly basis, depending on the capacity and availability of their resources.

To enforce the bidding strategy of the Clients, it will be necessary to introduce a *proxy* which sets the task price bids and submits the Clients' queries to the marketplace [21]. The price bids set by the proxy could be based on different parameters, such as the importance of

[21] For our experiments it is not relevant whether this proxy is located at the Client's site or at the Electronic Marketplace.

the Clients and their past usage of the resources.

It must be noted that, for our performance evaluation, it is not relevant *how* the price bids are determined by the proxy. What counts is the resulting probability distribution of the bids and the outcome of the experiment (w.r.t. to the performance metric used).

6.5 Protocol descriptions

In this section we introduce the resource allocation protocols that are studied in this thesis. As market protocols we examine the Continuous Double Auction Protocol (CDA), CDA with reserve prices, CDA with time-dependent price bids, the Proportional Share Protocol (PSP), the Highest Bid Protocol (HBP), HBP with thresholds, HBP with reserve prices, the Preemptive Protocol (PE), and the Periodic Double Auction Protocol (PDA). The non-economic protocols include the Round-Robin Protocol (RR), the First-in-First-Out Protocol (FIFO), PRIO-FIFO, and the Shortest Job First Protocol (SJF).

Note that this thesis is limited to the case where CPU time is the only type of resource, and where only one resource is needed for the execution of a (sub)task. This is realistic for many computationally-intensive applications, such as the PSIMAP application which is examined in chapter 12. However, there exist other scenarios, in which the applications require bundles of different types of resources for their execution (e.g. CPU time, memory, storage, etc.). This *combinatorial case* is subject of future work and will require different types of protocols.

6.5.1 Continuous Double Auction Protocol (CDA)

The aim of the Continuous Double Auction Protocol (CDA) [Kagel, 1995] is to allocate the best possible resource to an arriving task and to prioritise tasks according to their price bid. CDA has been chosen, because the studied scenario requires a *double* auction, i.e. a many-to-many protocol (and not 1-to-many protocols like English or Vickrey auctions). It is a *continuous* auction where transactions are carried out immediately whenever bids or offers change. For our scenario, it is likely to outperform protocols where the transactions are only carried out at periodic intervals. In such a protocol, an arriving task would have to wait for

the next auction — which we need to avoid in order to minimise response times. Like the other market protocols examined in this thesis, CDA is *greedy* in the sense that a task is assigned the best possible resource that is available at a given time.

When a Task Query object arrives at the EMP, the protocol searches all available resource offers and returns the first occurrence of the *best* match (see Figure 71). In our experiments, '*best*' means the fastest resource which satisfies the task's constraints, i.e. which meets all requirements for the task's execution (the required size and price of the resource and its capability to meet the task's deadline). If no match is found, the Task Query object is stored in a queue. When a resource becomes available ('*resource update event*') and several tasks are waiting, the one with the highest price bid, p_{Task}, is processed first. The pseudo code for this *resource update event* is given in Figure 72.

6.5.2 CDA with Reserve Prices (CDA-RES)

In the 'normal' Continuous Double Auction protocol, each Server, which is available, has to accept any price bid of a task. However, in some situations it might be advantageous to allow the Servers to use *reserve prices*, i.e. minimum values for the task bids to be accepted. As will be shown by the simulations, reserve prices can help to ensure that the better-performing resources are available to high-priority tasks (by keeping away low-priority tasks). In the CDA-RES protocol, the Servers use reserve prices which are based on the Server's average rate of revenue measured in the past (at periodic intervals). The idea is that this might help to ensure that the better resources are available to high priority tasks by keeping out low priority tasks. The disadvantage of this approach is that resources will be wasted. Also, it is possible that some low priority tasks may not be allocated any resource at all. To solve the latter problem, we introduced *price discounts*: a Server determines its reserve price by calculating the average revenue in the past and deducting a fixed price discount from it. By choosing an appropriate value for this price discount, there will always be resources in the system which the low priority tasks can afford. Another way to avoid the problem of tasks never getting executed is to allow them to increase their prices while waiting at the EMP (*task price adjustments*, see subsection A.2.5 in the appendix).

6.5.3 CDA with Time-Dependent Bids (CDA-TDB)

This protocol is an extension of CDA that is designed for the scheduling of tasks whose values depend on their timely completion [22]. The EMP aims to maximise a time-dependent *user utility* which has been defined in subsection 2.6.1. Hence, it needs to deal with time-dependent task price bids p_{Task}.

Whenever a resource offer becomes available, the price bids of all tasks waiting at the EMP need to be determined, and the task with the highest bid will be allocated to that resource. A task's price bid corresponds to its expected value to the Client, when executed on that resource. This value is determined by the task's slowdown, as described in subsection 2.6.1. As a result, this value decreases while the Task Query object is waiting at the EMP. The main drawback of this protocol is its computational complexity. It is therefore unsuitable for a situation in which the number of Task Query objects or Server offers at the EMP is high.

6.5.4 Proportional Share Protocol (PSP)

In contrast to the other protocols described in this thesis, the Proportional Share Protocol (PSP) allows several tasks to execute on a Server at a time. This protocol uses the resource scheduling policy 4 (see subsection 6.2.3). The amount of resources allocated to a task depends on its price bid, $p_{Task,i}$, in relation to the sum of price bids $\sum p_{Task,i}$ of all tasks executing on that Server, including the bid of the task itself [23]. The reason for examining PSP is that similar protocols have been proposed for the scheduling of tasks in computational clusters [Chun and Culler, 2000; Messer and Wilkinson, 1996; Sherwani *et al.*, 2002; Waldspurger, 1995]. PSP can improve on CDA for certain situations like high network latency and high resource heterogeneity.

When a Task Query object arrives at the EMP, all resource offers are checked in order to find the resource which is the fastest to execute the task and which meets the task's constraints {size, price, deadline} (see Figure 73). The effective execution speed, $s_{Eff,i}$, is given

[22] Its allocation decisions are identical to CDA's if the task price bids p_{Task} at the EMP do not change over time.

[23] The task's resource share is proportional to its price bid, hence the name *Proportional Share Protocol*.

by $s_{Eff,total} = f_{Speed} \cdot N_{RU,avail} \cdot \frac{p_{Task,i}}{\sum p_{Task,i}}$, where $N_{RU,avail}$ is the number of resource units that are available, i.e. not occupied by background tasks. Note that, due to arrivals or departures of tasks and background tasks, this speed may vary during the task's execution. If no match is found, the task will have to wait until the next resource update event (see Figure 74) or a task price adjustment event. Concerning the scheduling of tasks at the Server, a detailed description is given in section A.1.

6.5.5 Highest Bid Protocol (HBP)

One problem with the CDA protocol is that, once a task has been allocated to a Server, no other task can execute there until it completes. Hence, higher priority tasks may have to wait or use a resource with lower speed, leading to a lower overall performance of the system. With the PSP protocol, higher priority tasks may still be allocated but will not take up the whole resource. Several tasks may have to execute at a resource in parallel, resulting in them all being delayed. For this reason, we introduce the Highest Bid Protocol (HBP). It allows a task with a higher price bid to *suspend* a task with a lower bid that is currently being executed on a Server. The suspended task(s) will resume execution once the higher priority task has completed [24].

When a Task Query object arrives at the EMP, all resource offers are checked in order to find the resource which is the fastest to execute the task and which meets the task's constraints. Only resources are considered, for which the task's price bid is high enough to suspend other tasks already executing there.

6.5.6 HBP with Threshold (HBP-T)

HBP-T is a version of the HBP protocol which uses a threshold that we call the *bid improvement factor*, IF_{bid}. The threshold determines, how many times higher the bid of a task must be, in order to *suspend* another task. The idea behind this protocol is to limit the number of tasks being suspended in order to reduce their completion times.

[24] Note that the cost of suspending and resuming tasks is not considered in our simulation model, because it is assumed to be small in comparison to task execution times and communication delays. It is subject of future work.

6.5.7 HBP with Reserve Prices (HBP-RES)

This version of the HBP protocol uses reserve prices which are determined in the same way as in the CDA-RES protocol (see subsection 6.5.2). The idea is to prevent low priority tasks from using the better resources — which may help to avoid that they will be suspended during their execution.

6.5.8 Preemptive Protocol (PE)

In the Highest Bid Protocol (HBP), low priority tasks may be suspended by higher priority tasks and will have to wait for them to complete — even if resources are available at other Servers. For this reason we introduce a protocol which allows migration of tasks to other Servers [25]. In this thesis, we refer to it as the Preemptive Protocol (PE). We distinguish different versions of the protocol with regard to when preemption can take place:

1. *Preemption-Passive (PE-P)*: A task can only migrate if it is suspended by another task with a higher price bid. It will send a Task Query object to the EMP in order to obtain a new resource and resume execution.

2. *Preemption-Active (PE-A):* A task will migrate whenever any other resource becomes available which can execute it at a higher speed. In this case,'available' means that any task executing on that other resource has a lower price bid than the querying task. The migration of a task may be triggered by the completion of tasks or background tasks executing on other resources or by the start of a background task at its current resource. In such an event, the Server of the task with the highest price bid will be informed by the EMP, and the task will migrate.

3. *Preemption with Thresholds (PE-T)*: A task will migrate to a resource only under conditions specified by two thresholds, the speed improvement factor, IF_{speed}, and the bid improvement factor, IF_{bid}: firstly, the execution speed at the new resource must be at least IF_{speed}-times faster, and secondly, the task's price bid must be IF_{bid}-times higher

[25] A survey of related work dealing with preemptive resource allocation protocols can be found in section 4.4.

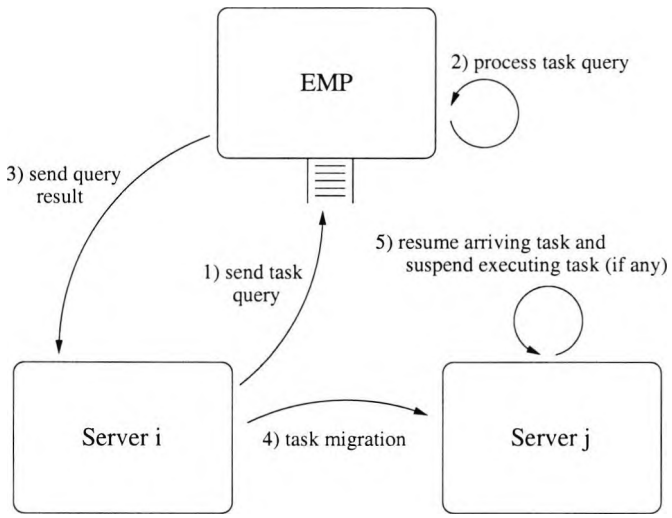

Figure 6: Cases 1 and 2 of the interactions in the PE protocols. Case 1 can occur in both, PE-P and PE-A. Case 2 can only occur in PE-A.

than that of the task executing on that resource. These conditions are designed to reduce the number of migrations, as the associated cost may be higher than the gain. Note, that these thresholds can be applied to PE-P and PE-A.

Preemptions can be triggered by different types of events. We distinguish three cases which will be described in the following paragraph: Case 1 can occur in both, PE-P and PE-A, whereas the cases 2 and 3 can only occur in PE-A. The interactions of the cases 1 and 2 are illustrated in Figure 6, and those of case 3 in Figure 7.

Case 1

1. Due to the arrival of a task to Server *i*, the currently executing task is suspended. The suspended task sends a query to the EMP.

2. The query is processed by the EMP. It is determined whether there exists any Server whose executing task has a smaller price bid than the querying task (from Server *i*). If there is more than one candidate, the best (i.e. fastest) one is selected for migration. If there is no match, the query will wait until a Server becomes available.

3. As soon as a match is found (e.g. Server *j*), the EMP informs Server *i* about it.

4. The querying task migrates from Server *i* to the new Server (Server *j*).

5. The migrating task resumes its execution at Server *j*. If a task has been is executing Server *j*, it is suspended. It sends a query to the EMP, etc.

Case 2

1. The performance of Server *i* decreases (due to the start of a background task). The executing task sends a query to the EMP, while continuing its execution.

2. The query is processed by the EMP. It is determined whether there exists any Server which is (currently) faster than Server *i* and whose executing task has a smaller price bid than the querying task (from Server *i*). If there is more than one candidate, the best (i.e. fastest) one is selected for migration. If no match is found, the query is cancelled.

3. If a match is found (e.g. Server *j*), the EMP informs Server *i* about it.

4. Unless already completed, the querying task migrates from Server *i* to the new Server (Server *j*). A task which may have previously been suspended at Server *i* will now be resumed.

5. The migrating task resumes its execution at Server *j*. If a task has been is executing there, it is suspended. It sends a query to the EMP, etc.

Case 3

1. After the completion of a task or background task at Server *i* the resource information at the EMP is updated.

2. This 'resource update event' is processed by the EMP. Before considering any Task Objects that may be waiting at the EMP, it is determined whether there exist any Servers whose tasks can migrate to Server *i*. A task can migrate to Server *i* if its Server has a lower effective speed than Server *i and* if it has a higher price bid than the task executing there. If there is more than one such Server, the EMP selects the one which executes the task with the highest bid.

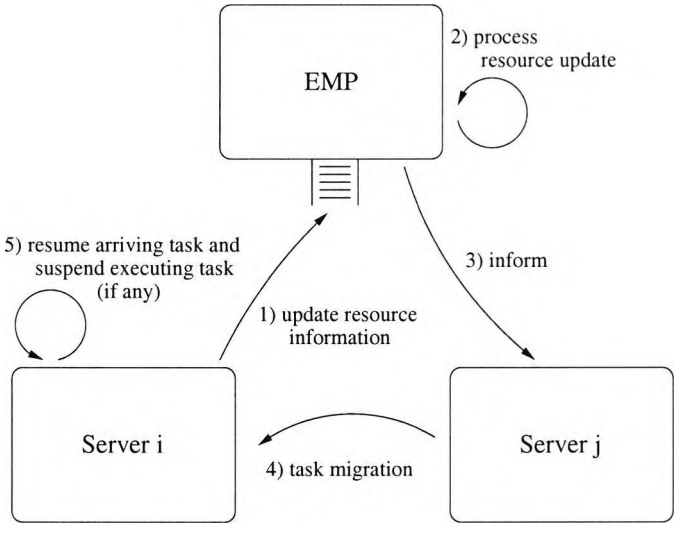

Figure 7: Case 3 of the interactions in the PE protocols. This case can occur in PE-A but not in PE-P.

3. As soon as a match is found, the EMP informs that Server (in our case: Server j).

4. The task, which is executing at Server j, migrates to Server i. A task which may have previously been suspended at Server j will now be resumed.

5. The migrating task resumes its execution at Server i. If a task has been is executing at Server i, it is suspended. It sends a query to the EMP, etc.

6.5.9 Periodic Double Auction Protocol (PDA)

In the Continuous Double Auction Protocol, transactions are carried out immediately, whenever tasks arrive at the EMP or resources become available. The idea behind this approach is to reduce task waiting times. In the Periodic Double Auction Protocol (PDA), however, price bids and Server offers are allowed to accumulate, and the transactions are carried out only at periodic time intervals. In these transactions, preference is given to the tasks with the highest price bids which will then select the *best* resource offers. The idea behind this approach is that the higher priority tasks may be allocated better resources than in the CDA protocol. A disadvantage is that some resources remain idle during the transaction intervals, and that tasks have to wait longer at the EMP.

6.5.10 Round-Robin Protocol (RR)

The Round-Robin Protocol (RR) does not use any pricing: It processes the incoming task queries on a first-come-first-served basis. They are matched with the *next* available resource offer which meets the task's constraints — but which is usually not the *best*. For this purpose an iterator is used which cycles through the list of resource offers (see Figure 75). RR is far simpler than the market protocols because it does not use information about load or speed of the resources for the allocation decisions. However, it is still adequate for certain situations. A more detailed description of the procedures used at the EMP, which includes their pseudo code, is given in subsection A.2.4.

6.5.11 First In First Out (FIFO)

Unlike in the market protocols, FIFO does not use any pricing: Task queries are processed on a first-come-first-served basis. However, like CDA, the FIFO protocol is also *greedy*, as tasks are allocated the *best*, i.e. fastest, resource that is available at the time.

6.5.12 PRIO-FIFO

Supercomputing centres often use a set of FIFO queues with different priorities. As in [Chun and Culler, 2002], we will refer to such a protocol as PRIO-FIFO. The Client which submits a task to the system can assign a priority to it. This task is then added to the queue which corresponds to its level of priority. In each queue of the system, the waiting tasks are prioritised in the order of their arrival. When allocating tasks to free resources, the EMP chooses the earliest task in the highest priority queue which is non-empty. A Client is charged for the execution of its tasks according to their level of priority. Effectively, this protocol operates in the same way as CDA. The only difference is that it only allows coarse-grained assignment of priorities to tasks, as the number of FIFO queues is finite. From the system's point of view, one problem is, that it may be difficult to set the charging rates for the different queues in a way which will result in maximum gain for the users.

Since PRIO-FIFO is a mix of CDA and FIFO, its performance is likely to lie between the

two [26]. For this reason, the performance of PRIO-FIFO is not examined in this thesis.

6.5.13 Shortest Job First (SJF)

Shortest Job First (SJF) is a *greedy* protocol which prioritises the shortest task. The idea behind this strategy is that short tasks suffer a larger relative slowdown than longer ones if they are delayed by the same amount of time. Hence, allocating the shortest job first will usually result in a lower mean slowdown of all tasks than a FIFO protocol. Furthermore, since a long task may take up as much CPU time as several smaller tasks, SJF may result in a lower mean completion time.

6.6 Model Discussion and Related Work

Our simulation model is very flexible as it allows the modelling of various cluster and Grid infrastructures [27]. To our knowledge, the use of a *single* model for clusters and Grids is unique and has not been reported elsewhere. We consider this model to be realistic because a computational cluster can be seen as a special case of a Grid, which has little heterogeneity, small communication delays, and which is small in size.

There exist other models and frameworks for the simulation of computational Grids. Their purpose is either to assess different scheduling protocols in a 'clean room environment' or to evaluate the performance of middleware or application software. The differences of these approaches to our work will be briefly discussed in this section.

SimGrid [Casanova, 2001] provides the basic functionality for the study of scheduling algorithms for parallel applications in distributed environments. In contrast to our work, which is about the allocation of independent tasks competing for resources, SimGrid considers the allocation of resources to a single, large application. Also, it uses load traces for the simulations whereas we use a statistical model, allowing us to adjust parameters arbitrarily and draw more general conclusions from our experiments.

Bricks [Aida *et al.*, 2000; Takefusa, 2001] is a simulator for client-server style global

[26] We made this observation in several experiments.
[27] This will be demonstrated in the following chapter.

computing systems, which allows the evaluation of scheduling algorithms and scheduler components. Much emphasis is put on providing realistic models for the network traffic: The authors experimented with queuing systems and self-similar load traces. They validated their simulation model by experiments on a global computing testbed using NAS benchmarks. As an example of the simulator's capabilities, a deadline scheduling algorithm for Grid resources has been examined. In contrast to Bricks, our work uses a statistic distribution for the communication delays which is sufficiently realistic for the scenarios examined and also computationally less intensive.

The GridSim Toolkit [Buyya and Murshed, 2002] is a general-purpose simulator for performance evaluation on the Grid. It supports the modelling and simulation of heterogeneous Grid resources, users and application models. It has been used to simulate Nimrod-G (see subsection 4.5.2) and the cluster scheduler Libra [Sherwani *et al.*, 2002]. In contrast to GridSim, our system targets the simulation of an electronic marketplace, which is an information service and 'broker' at the same time. Also, our system uses different load models, communication models, and resource scheduling protocols.

6.7 Summary

This chapter introduced the simulation model that will be used for the performance evaluation of the market protocols. A detailed description of the actors in the system and the underlying communication model has been given. Next, the interactions in the system which all the examined resource allocation protocols have in common have been described. We presented the protocols, which will be studied in this thesis. Finally, we compared our simulation model to related work.

7 Simulations: Overview

7.1 Introduction

In this chapter, we describe the general setup of the simulations and give an overview of the parameter space that is explored in this thesis (see Figure 8).

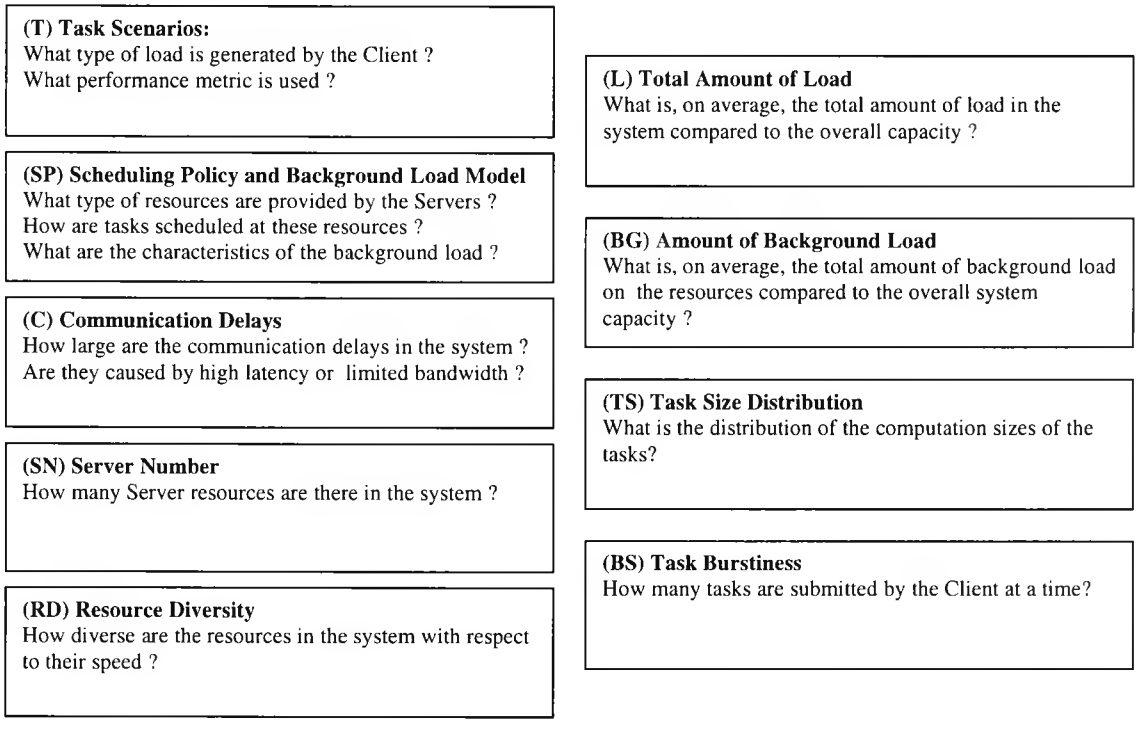

Figure 8: Overview of the parameter space to be explored

7.1.1 General Setup

All simulations are carried out with the discrete-event simulation framework, which is described in chapter B of the appendix. The total length of each simulation run is set to 1300 time units. During this time, tasks are randomly generated by the Client. During the first 100 time units no measurements are made. This is to ensure that the system reaches a steady state. After this initial period, the number of tasks which is statistically expected to be generated during an interval of 1000 time units is considered in the result. To allow these tasks to complete, an additional final margin of 200 time units is provided. To ensure that the results

are statistically significant, we carry out each measurement 40 times with different random seeds. For each point in our diagrams, the error bars of the 95% confidence interval of the mean are given [28]. The confidence interval is determined by the method described in section B.8 of the appendix. As will be shown, it is very small in most of the experiments. Hence, the duration and number of simulation runs are sufficiently large. In the simulated scenarios, it does not matter how many Clients there are in the system, because the arriving tasks are independent and are generated by a Poisson process. For this reason, we use only one Client for the task generation.

7.1.2 Task Scenarios

We distinguish several scenarios concerning the task load generated by the Client. These are characterised by how the generated tasks are prioritised. They require different metrics to evaluate the performance of the resource allocation protocols. Concerning the resource pricing, the Servers will accept any task price bid ≥ 0 [29]. We examine the following cases:

(T1) Tasks with the same priority

All tasks in the system have the same priority (1.0) and are assigned the same price bid $p_{Task,i} = 1.0$. The average load in the system is lower than its overall capacity. As a performance metric for evaluating the protocols we use the mean of the completion times (see subsection 2.6.1) of all tasks that are measured during the experiment. The aim is to minimise this metric.

(T2) Tasks with different priorities

Tasks have different priorities (*weights*) which reflect their value to the Client. For the weights of a task w_{Task} we use a uniform distribution $[0.0, 2.0]$ [30]. In order to allow com-

[28] In some experiments, e.g. Figure 15 (right), the simulations were very time-consuming, and therefore only one simulation run has been carried out for each data point. In these cases, the confidence intervals are not given.

[29] Exceptions are the protocols CDA-RES and HBP-RES, in which the Servers use reserve prices.

[30] There are two reasons for choosing a uniform distribution: Firstly, no empirical figures are available for the importance of tasks. Secondly, it has a high standard deviation. Thus, other distributions of task weights are likely to lead to results which lie between two extremes: those obtained with a uniform distribution, and those obtained in a scenario with identical task priorities. Our observations indicate that this is the case.

parisons with the T1 scenario, this distribution has to be chosen such that it has the mean value 1.0. In the market protocols the price bid p_{Task} of a task is equal to its weight. As a performance metric we use the *weighted completion time (WCT)* which is defined as the mean of the completion times of the tasks multiplied by their weights (also see subsection 2.6.1).

(T3) Tasks with different, time-dependent priorities

In the third scenario, the value of a task to the user depends on when it completes its execution ([Chun and Culler, 2002], see subsection 2.6.1). We express this value by *deadlines* and distinguish two cases: tasks with *hard* deadlines and tasks with *soft* deadlines. In the case where tasks have hard deadlines, their execution will only benefit the Client if completed on time. As a performance metric we use the weighted completion rate (WCR), which we define as the sum of weights of the tasks completed before the deadline divided by the sum of all task weights. Again, for the weights of the tasks, we use a uniform distribution $[0.0, 2.0]$. In the case where tasks have soft deadlines, a task's value (per task unit) is expressed as a piecewise-linear function of the *slowdown* (see subsection 2.6.1). Initially, this value — which is proportional to the task's price bid — is set to its maximum $V_{Task,initial}$, and remains there until the slowdown value sl_1 is reached. Then, it linearly decreases, and at the slowdown value sl_2 it becomes zero. As a performance metric we use the average of the values delivered by the tasks to the Client, i.e. the average of the task values (per task unit) multiplied by the task sizes. For the initial value a task, $V_{Task,initial}$, we use a uniform distribution $[0.0, 2.0]$.

7.1.3 Scheduling Policy and Background Load Model

The next examined parameter defines the scheduling policy and background load of the Server resources. These depend on the machine type, operating system, and the way the machine is deployed. The parameters covered here include the number of resource units of a Server, $N_{RU,total}$, the (average) Server speed factor, f_{Speed}, the background task size, $S_{C,BG}$, and the number of resource units allocated to each background task, $N_{RU,BG}$.

(SP1) PC with fine-grained background load

The aim of this parameter set is to model a time-shared PC where the background load generated by a local user only needs a fraction of the CPU time of the machine, i.e. only some *resource units*. The remaining resource units are allocated to tasks arriving from the EMP. The background load is given priority. Hence, the amount of resources allocated to an incoming task can vary during its execution — the task may even be suspended. As parameters we use $N_{RU,total} = 10$, $f_{Speed} = 0.1$, $S_{C,BG} = 1.0$, and $N_{RU,BG} = 1$ [31]. Note that, depending on the protocol that is used, this setup corresponds to the resource scheduling policies 2–4 which are described in subsection 6.2.3.

(SP1V) PC: Variation of the background task size

We use the same parameters as in parameter set SP1, except that the background task size, $S_{C,BG}$, is varied.

(SP2) PC in screensaver mode

In this parameter set, each background task started by the local user will take up the whole resource, i.e. all resource units — and may suspend tasks currently executing there. Such a situation is characteristic for a PC which is only made available to incoming tasks if the screensaver is active (and hence nobody is using it at the time). The parameters are the same as in parameter set SP1, except that $N_{RU,BG} = N_{RU,total} = 10$.

(SP2A) PC in screensaver mode: Larger background tasks

Here the same parameters are used as in parameter set SP2, except that the background task size is larger ($S_{C,BG}$=10).

(SP2V) PC in screensaver mode: Variation of the background task size

Now the same parameters are used as in parameter set SP2, except that the background task size, $S_{C,BG}$, is varied.

(SP3) Multi-processor machine where background tasks use some processors

The parameters are the same as in parameter set SP1. However, the scheduling policy and the

[31] In this parameter set, only 1/10 of a resource is allocated to each background task. In parameter set SP2 we study the impact of allocating the whole resource to it, i.e. $N_{RU,BG}$=10.

type of resource that is modelled are different. We assume a space-shared multi-processor system, where each processor is modelled as a resource unit ($N_{RU,total}$ = 10). Each incoming task is executed in parallel on as many resource units, N_{RU}, as are available at the time of its arrival. This resource share remains constant throughout the execution [32]. The task cannot be preempted by background load or by other tasks. This setup corresponds to resource scheduling policy 1 in subsection 6.2.3.

(SP4) Multi-processor machine where background tasks use the whole resource

Here the scheduling policy and resource type are the same as in parameter set SP3. The difference is that, when a background task arrives, it uses all units of the resource, i.e. $N_{RU,BG}$ = $N_{RU,total}$ = 10.

7.1.4 Communication Delays

The communication delay is another factor which may affect the performance of a resource allocation protocol, in particular for a globally distributed environment. We use the communication model described in subsection 6.2.5 and consider the following cases:

(C1) Negligible communication delay

The communication delay T_{Comm} is neglected. Latency is assumed to be infinitely small, and bandwidth infinitely high. This approximation can be valid for a local cluster of resources.

(C2V) Variation of the network latency

Network latency is introduced, while the bandwidth is still infinite. The mean of the communication delay $\mu_{Comm} = B_{\mu,i}$ is varied, and its standard deviation is set to 50% of μ_{Comm} [33]. This setup will be realistic if the communicating agents are geographically distributed and the exchanged messages are small in size, i.e. there are no larger data transfers.

(C3V) Variation of the transmitted data size

Now network latency is neglected while the bandwidth of the network is considered finite. Parameter $A_{\mu,i}$ is set to 10^{-6}, resulting in a mean bandwidth of 1 MB per time unit. Parameter

[32] Tasks are considered *moldable*, see subsection 2.2.5.

[33] This value is arbitrarily chosen. However, our experiments have shown that setting it to a different value, such as 10% or 90%, did not have much impact on the results.

$C_{\sigma,i}$, which determines the standard deviation, is set to 50% of $A_{\mu,i}$. The size of the input data of the tasks $S_{D,in}$ is varied in order to see what happens if bandwidth is the determining factor of the communication delay [34].

7.1.5 Number of Servers

For the number of Servers in the system, N_{Serv}, we consider the following three cases:

(SN1) Computational Cluster: The number of resources in the system is rather small. We choose $N_{Serv} = 32$.

(SN2) Computational Grid: The number of resources in the system is large, which may be the case for a computational Grid, or for a large, local cluster. We choose $N_{Serv} = 256$.

(SNV) Variation of the Server number: Here we vary the Server number and see the impact on the result.

7.1.6 Resource Diversity

Concerning diversity of resources we consider the following situations:

(RD1) Identical resources: All resources have the same speed factor f_{Speed}=0.1. This is likely to be the case in a local cluster of resources.

(RD2) Heterogeneous resources: We assume heterogeneous resources, which we consider realistic for a wide-area computational Grid. For the server speed factor, f_{Speed}, we use a uniform distribution $[f_{Speed,min}, f_{Speed,max}]$ where $f_{Speed,min} = f_{Speed,av}/2N_{Serv}$, $f_{Speed,max} = 2 \cdot f_{Speed,av} - f_{Speed,min}$,and $f_{Speed,av} = 0.1$ [35].

(RDV) Variation of resource heterogeneity: In this parameter set, different degrees of heterogeneity are examined by varying the minimum speed factor $f_{Speed,min}$ between 0 and $f_{Speed,av} = 0.1$. The maximum speed factor is calculated by $f_{Speed,max} = 2 \cdot f_{Speed,av} - f_{Speed,min}$. This choice of speed factors results in a constant total capacity of the system.

[34] Note that the result would be the same if the data size was constant and the bandwidth was varied.

[35] Again, the reason for using the uniform distribution is that we have no empirical data available. Also, it provides a high variance, so that it can be regarded as an extreme case.

7.1.7 Total Amount of Load

The load in the system is generated by Poisson processes at the Clients and the Servers, which create tasks and background tasks, respectively. We define the load factor, l_{total}, as the ratio of average total load in the system and the overall system capacity [36]. This ratio can be adjusted by choosing the mean inter-arrival times of these Poisson processes accordingly. We examine the following cases:

(L1) The average total load is set to 90% of the overall system capacity: $l_{total} = 0.9$. At this load level, resources may temporarily become scarce.

(LV) The average total load is varied: $l_{total} = [0.0 - 1.0]$.

7.1.8 Amount of Background Load

The overall amount of background load in the system is determined by the mean inter-arrival times of background tasks at the Servers. We define the background load factor, l_{BG}, as the ratio of the average background load in the system and the overall system capacity. In order to examine how the level of background load affects the performance of the resource allocation protocols, we examine the following situations:

(BG0) There is no background load in the system: $l_{BG} = 0$.

(BG1) The background load is 25% of the average total load: $l_{BG} = 0.25 \cdot l_{total}$.

(BG2) The background load is 50% of the average total load: $l_{BG} = 0.5 \cdot l_{total}$.

(BG3) The background load is 75% of the average total load: $l_{BG} = 0.75 \cdot l_{total}$.

7.1.9 Task Size Distribution

Concerning the computation size S_C of the tasks in the system, we examine two cases:

(TS1) Identical task sizes: All tasks have the same size S_C=1.0.

(TS2) Loguniform distribution of task sizes: We use a loguniform distribution of the task

[36]The term 'load in the system' refers to the load of tasks that have been generated but have not yet completed execution. These tasks may be executing or waiting to be executed.

size [37]. We choose the distribution such that its average value is 1.0, and the ratio of its maximum to its minimum is 10.

7.1.10 Task Burstiness

As the final parameter we consider the task burst size, *BS*, i.e. the number of tasks that a Client submits to the system at a time:

(BS1) *BS*=1.

(BS2) *BS*=10.

(BSV) *BS* is varied.

7.2 Realistic System Infrastructures

In the previous section, an overview of the parameter space has been given, which will be explored in our experiments. To give a structure to these experiments and simplify their description, we now define several realistic system infrastructures which will be studied. As our first infrastructure, we examine a *PC Cluster*, which has a small number of identical resources, and where communication delays are negligible. The resources are assumed to be single-processor PCs, and therefore the scheduling policies (SP1) and (SP2) are applicable. The parameter space covered by this infrastructure is given by {*, SP1-2, C1, SN1, RD1, *, *, *, *} [38]. The second, more general infrastructure that is explored in our simulations will be referred to as *PC Grid*. In this system, PCs of different speeds are distributed over the Internet. The number of machines can be higher than in a cluster, and communication delays may no longer be negligible. Concerning the scheduling policy, the same assumptions are made as for the PC Cluster. The corresponding parameter space is given by {*, SP1-2, *, *, *, *, *, *, *}. We also briefly examine two other system infrastructures: the *Supercomputing Cluster* and *Supercomputing Grid*. These correspond to the PC Cluster and PC Grid infrastructures, respectively, except that the resource scheduling policies (SP1) and (SP2) are now replaced by (SP3) and (SP4). This means, that machines have multiple processors, for which

[37] This is motivated by a workload model for supercomputing centres (see subsection 6.2.2).

[38] The '*' stands for 'any value'.

space-sharing is used, i.e. different processors can be allocated to different users. Each task can spread its execution over several processors. The parameter space for the Supercomputing Cluster is given by {*, SP3-4, C1, SN1, RD1, *, *, *, *}, and for the Supercomputing Grid by {*, SP3-4, *, *, *, *, *, *, *}.

7.3 Summary

This chapter provided the structure for our simulations and also demonstrated the flexibility of our simulation model which can cover various cluster and Grid scenarios. We described the experimental setup and the parameter space which will be explored in our simulations. We defined the relevant parameters, which include the type of scenario, the scheduling policy at the resources, the communication delays, the number of resources in the system, resource heterogeneity, the amount of load and background load, the task size distribution, and the task burstiness. Finally, we defined the different types of system infrastructures which will be examined.

8 Tasks with the Same Priority

This chapter provides a systematic performance comparison of three resource allocation protocols for a scenario in which all tasks have the same priority ($w_{Task} = 1.0$) and are assigned the same price bid $p_{Task,i} = 1.0$. The following protocols are examined: the Continuous Double Auction Protocol (CDA), the Proportional Share Protocol (PSP), and Round-Robin (RR) [39]. As a performance metric we use the mean of the completion times of all tasks that are measured during the experiment. An important objective is to examine the sensitivity of the results to different parameters. This will help to determine how general our results are. Also, it will limit the parameter space that needs to be studied in further experiments.

We start with the *PC Cluster* infrastructure, and then move on to the *PC Grid* infrastructure. The results for the *Supercomputing Cluster* and the *Supercomputing Grid* infrastructures are in many cases similar and can be found in chapter C of the appendix.

8.1 PC Cluster

8.1.1 No Background Load

The first experiment is defined by the parameters {T1, SP1, C1, SN1, RD1, LV, BG0, TS1, BS1}. This means that we have a system with N_{Serv}= 32 identical Servers without background load. The Servers have the resource size $N_{RU,total}$ = 10 and speed factor f_{Speed} = 1.0. All tasks have the same size S_C=1.0 and their burst size is 1. Communication delays are neglected. The average total amount of load in the system is varied between 0 and 100% of the system capacity.

Figure 9 shows the mean completion time for the three protocols. As expected, all three protocols degrade, when load is increased. Since there is no difference between the resources, RR and CDA perform equally well. PSP performs worse because it allocates tasks to Servers which are already busy. Thus, it delays tasks which are already executing. At 90% load its mean completion time is 19% higher.

[39] We do not examine FIFO, PRIO-FIFO, and CDA-TDB, because they would provide the same result as CDA. HBP is not examined either, because it cannot lead to any improvement in a situation in which all tasks have the same priority.

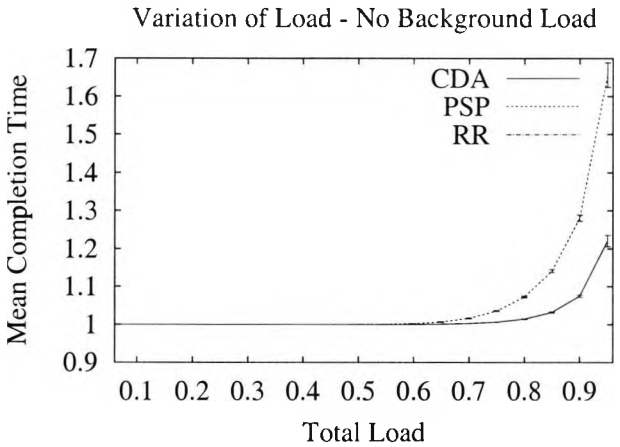

Figure 9: Tasks with the same priority, variation of load: Since the resources are identical and have no background load, there is no difference between CDA and RR.

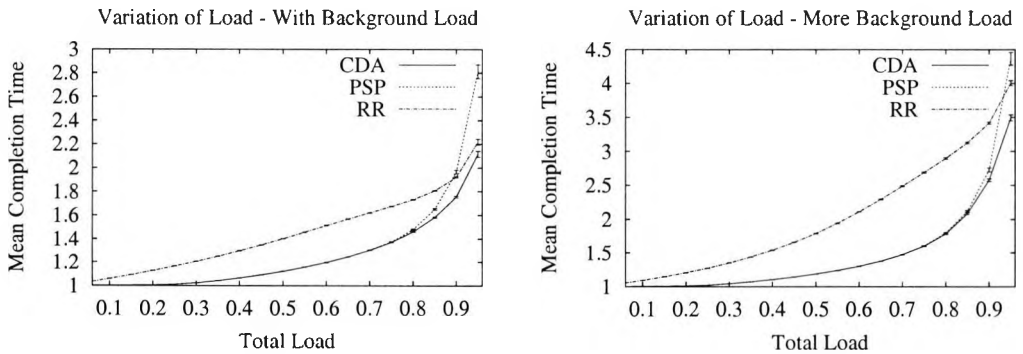

Figure 10: Left: When background load is introduced, the performance of all protocols degrades. Now CDA performs best for all loads. RR is worst for low and medium load, and PSP for high load. Right: When the amount of background load is even higher, the three protocols degrade even more, but their order does not change.

8.1.2 Different Amounts of Background Load

These results change when background load is introduced. We examine a case where half of the load in the system is background load (BG2). The background tasks have the computation size $S_{C,BG} = 1.0$, and are allocated $N_{RU,BG} = 1$ resource units at a time.

As shown in Figure 10 (left), CDA provides the best results for the whole range of loads. RR performs worse because resources are allocated arbitrarily, whereas CDA selects the fastest available resource. The difference is particularly high at a load of about 60%, where RR's completion time is 26% higher than CDA's. PSP performs almost as well as CDA as

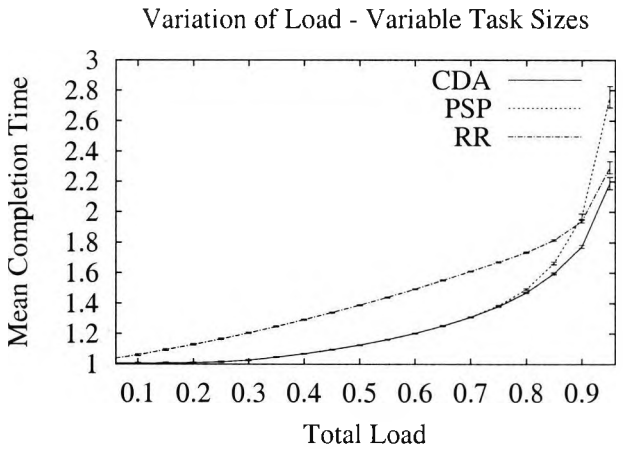

Figure 11: Variation of load: Tasks with variable sizes lead to almost the same results as identical tasks.

long as the load is low, because it also selects the fastest available resources for the tasks. However, as in the previous experiment, it degrades when the load is increased.

The overall performance of the system strongly depends on the amount of background load on the Servers, as is shown in Figure 10 (right). In that experiment, the total load in the system is varied, with the difference that now three quarters of it is background load (BG3). If the total load is high, all protocols degrade, and their completion times are more than 50% higher than before.

Conclusion: When background load is introduced, CDA performs best. For low load, the differences between the three protocols are small. For moderate load, RR performs worst, and for high load, PSP does. When the share of the background load is increased, all three protocols degrade in a similar way.

8.1.3 Variable Task Sizes

Next, we examine the effect of having the Client submitting tasks of different sizes to the system. We use the same parameters as in Figure 10 (left), except that the task computation size S_C now has a loguniform distribution (TS2). The mean of S_C is not changed ($S_{C,mean} =$

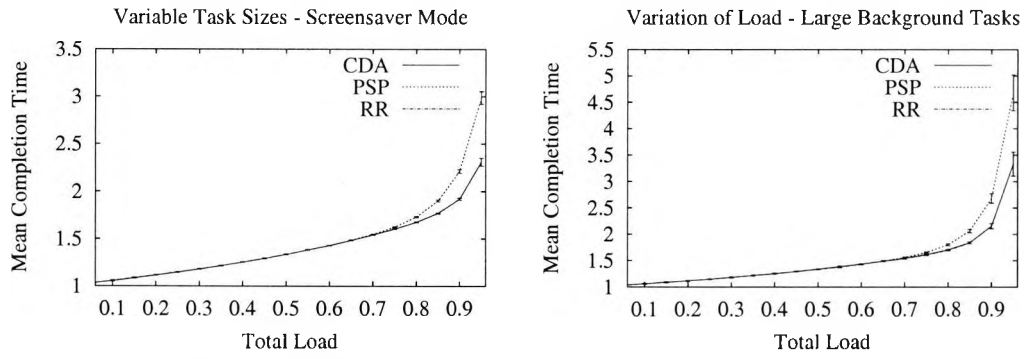

Figure 12: Left: Variation of load in screensaver mode, where CDA and RR perform equally well. Right: Screensaver mode with larger background tasks: Performance degrades for all protocols as the tasks are suspended for a longer time.

1.0) [40]. Figure 11 shows that the task size distribution has little effect on the mean completion time. These findings are not surprising, as tasks of different sizes are treated equally by the examined protocols.

Conclusion: The distribution of task sizes has very little impact on the performance of the protocols. A loguniform distribution of task sizes led to almost the same results as identical task sizes. Hence, the task size distribution is a factor which can be neglected in future experiments — at least for these three protocols.

8.1.4 Granularity of Background Load

The granularity of the background load is a parameter which *does* affect the mean completion time. In the experiment in Figure 12 (left) we use the same parameters as in Figure 11 (left), except that now each background task takes up all resource units of the Server, i.e. $N_{RU,BG}$=10 (*screensaver mode*, SP2). Not surprisingly, RR now performs equally well as CDA, as all resources that are offered at the EMP now execute tasks at the same speed. Apart from this difference, the results do not change much in comparison to Figure 10 (left). However, if the size of the background task is increased to $S_{C,BG}$ = 10.0 — as done in Figure 12 (right) — all three protocols degrade: In particular, with PSP, the completion time at 90%

[40] We will not vary the mean of the task computation size, S_C, as this would have the same effect as changing background task size, $S_{C,BG}$, in the opposite direction. This will be done later in Figure 13 (right).

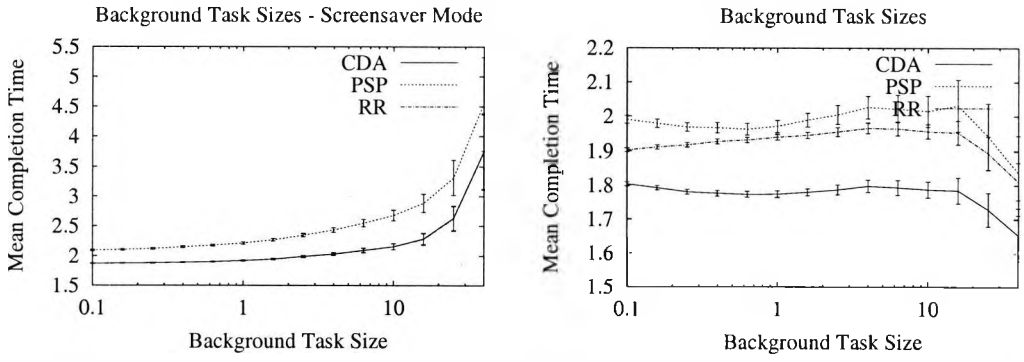

Figure 13: Left: Variation of the background task size in screensaver mode. Right: Variation of the background task size for $N_{RU,BG}$=1.

load is now 21% higher than in Figure 12 (left).

A wider range of values for the background task $S_{C,BG}$ is studied in Figure 13 (left). The experimental parameters are {T1, SP2V, C1, SN1, RD1, L1, BG2, TS2, BS1}, i.e. we examine the *screensaver mode*, where the average total load is set to 90% and the background load to 45%. For $S_{C,BG}$=0.1 the average completion time of the three protocols is only slightly lower than for $S_{C,BG}$=1.0. For higher $S_{C,BG}$, however, a strong degradation can be observed (95% for CDA at $S_{C,BG}$=40). This can be explained by the fact that tasks, which are suspended by background load, now have to wait for a longer time before they can resume execution. However, the results are still very stable over two orders of magnitude ($S_{C,BG} = [0.1, 10.0]$). Carrying out this experiment with $N_{RU,BG}$=1 (SP1V) does not result in any degradation, as shown in Figure 13 (right). The reason for this is that tasks do not get completely suspended by the finer-grained background tasks. The results even seem to improve for very high $S_{C,BG}$. This, however, can be explained by the fact that the share of a resource, which is allocated to a task, is more likely to remain stable during its execution, as there are fewer arrivals of background tasks. Note that for the same reason the results of the experiment become less accurate, as can be seen by the larger confidence intervals of the measurements.

Conclusion:

The background task granularity has a considerable impact on the performance of the examined protocols: In *screensaver mode*, where a resource is either completely available or

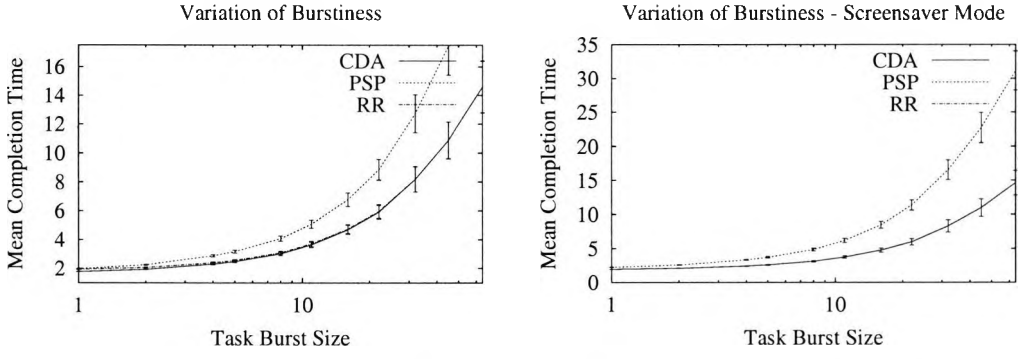

Figure 14: Left: Variation of the task burst size. Right: Variation of the task burst size in screensaver mode.

unavailable, the results of the three protocols are very different from the previously considered cases. RR and CDA perform equally well, whereas PSP's performance is worse.

The size of the background tasks has little impact on the performance of the protocols — except in screensaver mode, where the mean completion time slowly rises with increased background task size. All three protocols are affected in a similar way.

8.1.5 Task Burstiness

Next, we examine the effect of changing the task burst size, *BS*. As parameters of the experiment we choose {T1, SP1, C1, SN1, RD1, L1, BG2, TS2, BSV}. These are the same as in Figure 10, except that now the total load is fixed at 90%, while the burst size is varied. The results in Figure 14 (left) show that with increased *BS* also the mean completion time increases. This is not surprising: If the burst size is high in comparison to the number of resources in the system, it becomes less likely that all the tasks of a burst can be executed immediately. However, for low values of *BS* (≤ 5), the completion time is relatively stable. Note that with increased burst size the gap between RR and CDA decreases. The reason is that the choice of resources becomes smaller when *BS* is higher. Very similar observations could be made for the screensaver mode (SP2), for which the results are shown in Figure 14 (right). The degradation appears to depend on the ratio of task burst size and number of Servers in the system. The results of an experiment where the task burst size was varied for 256 Servers in the system is shown in Figure 15 (SN2). In comparison to Figure 14 (left),

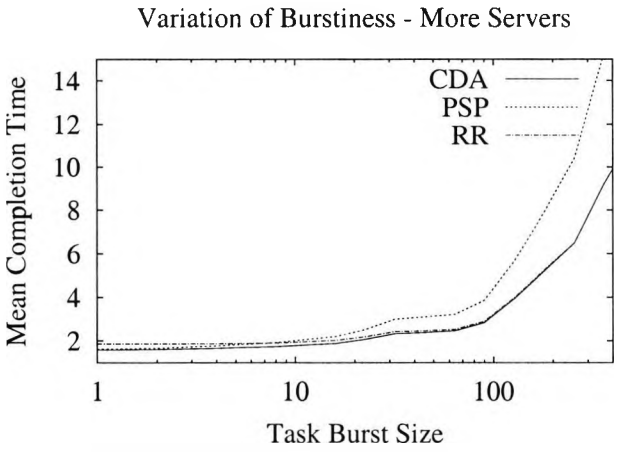

Figure 15: Variation of the task burst size when there are 256 Servers in the system. Note that there is less degradation than for 32 Servers.

the sharp increase of the completion time now occurs at a higher burst size.

Conclusion: The higher the number of tasks per burst, the higher the mean completion time. PSP suffers most, whereas RR and CDA perform equally well. However, the degradation of the protocols is small as long as the burst size is small in comparison to the number of resources in the system.

8.2 PC Grid

Next, we study *PC Grid*-type infrastructures which are characterised by higher resource heterogeneity, higher number of resources, and higher communication delays than computational clusters. To see the impact of each of these parameters, we start off with parameters typical for a cluster setting and change them one by one.

8.2.1 Resource Heterogeneity

At first, we examine the impact of having different degrees of heterogeneity of resources in the system (RDV). In Figure 16 (left) we use the parameter set {T1, SP1, C1, SN1, RDV, L1, BG2, TS2, BS1}: We have 32 resources in the system, and the average total load is 90%, half of which is background load. Task sizes have a loguniform distribution, and the

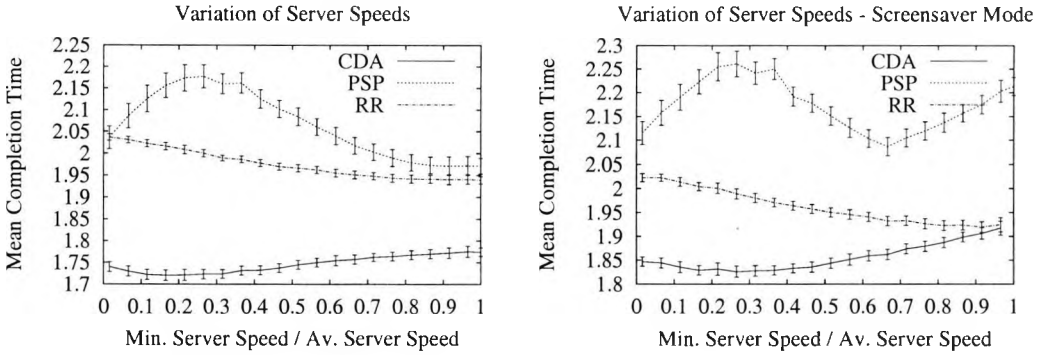

Figure 16: Left: Variation of the resource heterogeneity. Right: Variation of the resource heterogeneity in screensaver mode.

task burst size is 1. The number of resource units per background task is set to $N_{RU,BG} = 1$, and communication delays are neglected. For the speed factor f_{Speed} we use a uniform distribution, in which the minimum $f_{Speed,min}$ is varied between 0 and $f_{Speed,av} = 0.1$. The maximum speed factor is calculated by $f_{Speed,max} = 2 \cdot f_{Speed,av} - f_{Speed,min}$. Thus, in the diagram, we get maximum resource heterogeneity on the left, and identical resources on the right.

The performance of CDA slightly improves when heterogeneity is increased. At a ratio of $f_{Speed,min}/f_{Speed,av} = 0.01562$, its mean completion time is 2% lower than for identical resources. RR degrades slightly, by about 5%, because it does not consider speed for the choice of resources. PSP degrades by 9% when $f_{Speed,min}/f_{Speed,av} = 0.3$, but by just 3% for a value of = 0.01562. This result indicates that proportional sharing does not always lead to improvements, even when resources are heterogeneous. However, as will later be shown in Figure 17 (left), such a poor performance of PSP can only occur at high loads, when many tasks use a resource in parallel.

In screensaver mode, RR performs in a similar way as in the previous experiment. The results are shown in (Figure 16 (right)). CDA improves more than before. The reason is that speed is now the only criterion for choosing a resource: background load now does not play any role in the allocation decision — a resource is either available or not available. The results of PSP are now very irregular. Overall, an improvement can be observed for all values of $f_{Speed,min}/f_{Speed,av}$, except in the range 0.2 — 0.4.

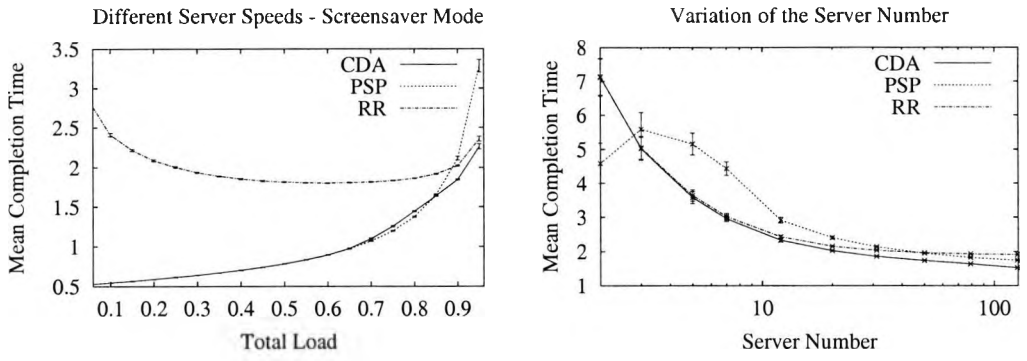

Figure 17: Screensaver mode. Left: Variation of load when resources are heterogeneous. RR performs poorly for low load, while PSP outperforms CDA for moderate load. Right: Variation of the Server number when resources are heterogeneous.

Conclusion: As heterogeneity of resources in the system is increased, we observe a small improvement for CDA, a small degradation for RR, and large fluctuations for PSP. Note that these observations are limited to the case that the total load in the system is set to 90%. The results for different amounts of load will be given in the next section.

8.2.2 Resource Heterogeneity and Different Amounts of Load

Next, we examine different amounts of load in the system (LV) when resource heterogeneity is high (RD2, i.e. $f_{Speed,min} = 0.0015625$). Figure 17 (left) shows the result for the screensaver mode. For low load, CDA and PSP perform much better than for identical resources as the tasks are very likely to execute on faster resources.

What at first looks surprising is that RR's performance is worse for low load than for moderate or high load. The reason for this is that RR chooses resources indifferently. With low load, the availability of slow resources will be the same as that of fast ones. For high load, slow resources are more likely to be in use because tasks take longer to complete. Hence, faster resources will be used more often, resulting in better average performance.

Another important result is that PSP performs slightly better than CDA for a total system load of 65% to 85%. It appears that in this range it can be advantageous to execute several tasks in parallel on fast resources — and slowing them all down — rather than allocating some tasks to slow resources.

Conclusion: Whether a protocol improves (or degrades) with increased resource heterogeneity depends very much on the amount of load in the system: For very heterogeneous resources and moderate amounts of load, PSP can outperform CDA. RR performs much worse than the other protocols, if load is low or moderate.

8.2.3 Different Server Numbers

The aim of this experiment is to see the effect of the Server number on the completion time. In Figure 17 (right) the number of Servers N_{Serv} in the system is varied while the load and background load ratios are kept constant. We examine the screensaver mode (SP2) and assume heterogeneous resources (RD2). Hence, the parameters are given by {T1, SP2, C1, SNV, RD2, L1, BG2, TS2, BS1}.

For all protocols, the mean completion time goes down as N_{Serv} is increased. The reason is that with increased size of the market a shortage of resources is less likely, because the overall amount of resources offered is more stable. CDA performs best for almost all examined values. For a high Server number, PSP approaches its performance, because it becomes less likely that several tasks execute on a resource at the same time. RR performs worse: For N_{Serv}=126 its completion time is 25% higher than CDA's. This can be explained by its indifferent allocation of resources. However, for low N_{Serv}, RR's performance is almost the same as CDA's.

An anomaly of PSP can be observed for N_{Serv}=2, where its mean completion time is lower than for 3 or 4 Servers. This can be explained by the distribution of Server speeds (RD2). For N_{Serv}=2 it appears to be advantageous to concurrently execute several tasks on the faster resource, rather than using the slower resource.

Conclusion: For a high Server number and heterogeneous resources, all protocols improve. PSP's mean completion time approaches that of CDA, whereas that of RR remains at a higher level.

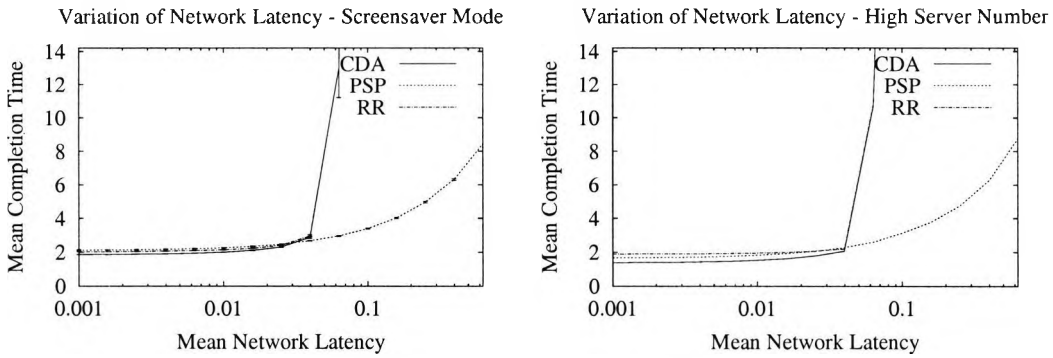

Figure 18: Screensaver mode. Left: Variation of the network latency. CDA and RR degrade strongly while PSP performs much better. Right: Variation of network latency when there are 256 Servers in the system.

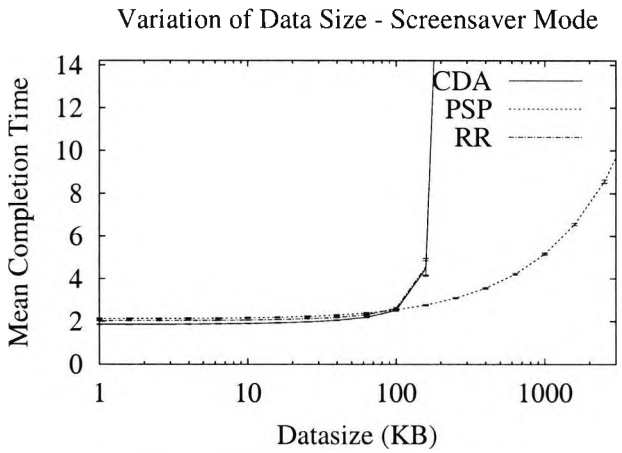

Figure 19: Screensaver mode: Variation of the size of the input data of tasks, when bandwidth is limited. It has a similar effect on the overall performance as varying the network latency.

8.2.4 Communication Delays

So far, the communication delay between Clients, Servers, and the EMP has been neglected. In Figure 18 (left) we investigate how the performance of the protocols degrades when network latency is introduced. The simulation parameters of this experiment are given by {T1, SP2, C2V, SN1, RD2, L1, BG2, TS2, BS1}. For the latency, we use a lognormal distribution, of which the mean is varied (C2V). A sharp rise of the mean completion time can be observed for RR and for CDA when the latency is increased over 0.04 [41]. This is because resources,

[41] Note that the reason for the 'angle' in the slope is that only few data points are used in that experiment. With more data points the curve would be smoother.

which are released, need to be advertised at the EMP before they can be used again. They remain idle during the communication delays, leading to a shortage of resources. For PSP, however, the rise is much slower. The reason is that the tasks are allocated immediately after arrival without waiting for resources to become available. As shown in Figure 18 (right), the results do not change much when the number of resources in the system is increased from 32 to 256 (SN2).

Next, we investigate a case where the determining factor for the communication delays is the network bandwidth rather than the network latency. In Figure 19 the size of the input data of the tasks is varied (C3V). The main difference to the previous experiments is that now the delays only occur when tasks are sent from the Client to a Server. However, as the results show, this difference does not affect the relative performance of the protocols.

Conclusion: When communication delays are large in comparison to the size of the tasks, all protocols will degrade strongly. However, PSP can cope much better with the delays than RR and CDA: its degradation is much slower. This observation has been made for two cases: one, where all messages were treated in the same way, and the other one, where only the transfers of larger input data were delayed.

8.3 Summary

Distributed computing environments are diverse in their nature, ranging from local clusters of PCs to geographically distributed networks of heterogeneous resources (*Grids*). Typically, parameters like the number of resources, resource heterogeneity, and communication delays are low in a cluster, but high in a Grid.

In this chapter, we explored various realistic situations by varying these and other relevant parameters. We examined the *PC Cluster* and *PC Grid* infrastructures. For these, we compared the performance of three protocols: the Continuous Double Auction Protocol (CDA), the Proportional Share Protocol (PSP), and the Round Robin Protocol (RR). We examined the sensitivity of our results to different parameters and identified those parameters which have considerable impact on the results.

In addition to the *PC Cluster* and *PC Grid* infrastructures, we explored the *Supercomputing Cluster* and *Supercomputing Grid*. The results from these experiments are in many cases similar to those of the PC infrastructures and can be found in chapter C of the appendix. An overview of all experiments and their parameters is given in Figure 20. Our guidelines for the system designer are summarised in chapter 13.

Infrastructure	Figure	Parameters
PC Cluster	9	T1, SP1, C1, SN1, RD1, LV, BG0, TS1, BS1
PC Cluster	10(left)	T1, SP1, C1, SN1, RD1, LV, BG2, TS1, BS1
PC Cluster	10(right)	T1, SP1, C1, SN1, RD1, LV, BG3, TS1, BS1
PC Cluster	11	T1, SP1, C1, SN1, RD1, LV, BG2, TS2, BS1
PC Cluster	12(left)	T1, SP2, C1, SN1, RD1, LV, BG2, TS2, BS1
PC Cluster	12(right)	T1, SP2A, C1, SN1, RD1, LV, BG2, TS2, BS1
PC Cluster	13(left)	T1, SP2V, C1, SN1, RD1, L1, BG2, TS2, BS1
PC Cluster	13(right)	T1, SP1V, C1, SN1, RD1, L1, BG2, TS2, BS1
PC Cluster	14(left)	T1, SP1, C1, SN1, RD1, L1, BG2, TS2, BSV
PC Cluster	14(right)	T1, SP2, C1, SN1, RD1, L1, BG2, TS2, BSV
PC Cluster	15	T1, SP1, C1, SN2, RD1, L1, BG2, TS2, BSV
PC Grid	16(left)	T1, SP1, C1, SN1, RDV, L1, BG2, TS2, BS1
PC Grid	16(right)	T1, SP2, C1, SN1, RDV, L1, BG2, TS2, BS1
PC Grid	17(left)	T1, SP2, C1, SN1, RD2, LV, BG2, TS2, BS1
PC Grid	17(right)	T1, SP2, C1, SNV, RD2, L1, BG2, TS2, BS1
PC Grid	18(left)	T1, SP2, C2V, SN1, RD2, L1, BG2, TS2, BS1
PC Grid	18(right)	T1, SP2, C3V, SN1, RD2, L1, BG2, TS2, BS1
PC Grid	19	T1, SP2, C2V, SN2, RD2, L1, BG2, TS2, BS1
SC Cluster	79(left)	T1, SP3, C1, SN1, RD1, LV, BG2, TS2, BS1
SC Cluster	79(right)	T1, SP3, C1, SN1, RD1, LV, BG3, TS2, BS1
SC Cluster	80(left)	T1, SP4, C1, SN1, RD1, LV, BG2, TS2, BS1
SC Grid	80(right)	T1, SP3, C1, SN1, RD2, LV, BG2, TS2, BS1
SC Grid	81(left)	T1, SP3, C1, SNV, RD2, L1, BG2, TS2, BS1
SC Grid	81(right)	T1, SP3, C2V, SN1, RD2, L1, BG2, TS2, BS1
SC Grid	82	T1, SP3, C1, SN1, RD2, L1, BG2, TS2, BSV

Figure 20: Tasks with the same priority: Overview of experiments.

In almost all situations CDA outperforms the two other protocols. PSP performed better only for moderate loads combined with high resource heterogeneity. It also degraded less than the two other protocols when communication delays were high.

Concerning the sensitivity of our results to different parameters, our main findings are:

- The distribution of task sizes has very little impact on the performance of the protocols and will not be examined in further experiments.

- The higher the number of tasks per burst, the higher the mean completion time. However, the degradation of the protocols is small as long as the burst size is small in comparison to the number of resources in the system. In our further investigations we will focus on this case as it is likely to be found in Grid settings: We will use the burst size 1 for most of our experiments.

- An increase of background load in the system generally leads to a degradation of performance. It affects all protocols in a similar way.

- The granularity of background load has a considerable impact on performance. Therefore, both the *screensaver mode* and finer background load will be examined in further experiments.

- Resource heterogeneity is a factor which affects the examined protocols in different ways: Depending on the amount of load in the system, it may improve CDA and PSP, whereas RR will degrade.

- An increased number of resources in the system generally leads to performance improvements for all three protocols. However, for CDA and PSP this improvement is larger than for RR.

- When communication delays are introduced, CDA and RR will degrade more than PSP.

This scenario is relatively simple: Since all tasks have the same priority, the only useful feature of the market protocols is their greedy behaviour, i.e. the selection of the fastest resource that is available at the time. Hence, if resources become scarce, CDA and RR perform about equally well. We also note that, in this scenario, the generated tasks have no deadlines or other constraints and that the average load generated is less than 100% of the systems capacity. Therefore, all tasks get executed eventually, and the average load on the Servers is equal to the average load that is generated.

In the following chapters, scenarios will be considered where tasks have different priorities or deadlines. We will introduce further resource allocation protocols, which can provide

improvements in some of the examined situations. These protocols use features such as the suspension or migration of tasks, reserve prices, periodic auctions, time-dependent price bids, etc. As the impact of the different parameters has been examined in this chapter, we will be able to reduce the amount of experiments that are needed.

9 Tasks with Different Priorities

This chapter provides a comprehensive performance comparison of different resource allocation protocols for a scenario in which tasks have different priorities (*weights*). For these weights, w_{Task}, which reflect their value to the Client, we use a uniform distribution $[0.0, 2.0]$. In the market protocols, the price bid of a task p_{Task} is equal to its weight. As performance metric we use the *weighted completion time (WCT)* which is defined as the mean of the completion times of the tasks multiplied by their weights (see also subsection 2.6.1).

Now that tasks have different priorities, the market protocols are expected to result in additional performance improvements in comparison to conventional protocols. For this reason, we will present a more comprehensive study where several market protocols are compared. These include the Continuous Double Auction Protocol (CDA), the Proportional Share Protocol (PSP), the Highest Bid Protocol (HBP), and the two preemptive protocols, PE-P and PE-A. Also, First-In-First-Out (FIFO) and the Round Robin Protocol (RR) will be examined. In addition to these, we will evaluate some other, more specialised protocols for some selected situations in order to determine whether they can lead to any improvement at all. These are the Periodic Double Auction Protocol (PDA), Shortest-Job-First (SJF), two protocols which use reserve prices for the Servers (CDA-RES and HBP-RES), and two protocols which use thresholds for the preemptions (HBP-T and PE-T). We start off with the *PC Cluster* infrastructure, and then move on to the *PC Grid* infrastructure.

9.1 PC Cluster

9.1.1 No Background Load

In the first experiment, we consider a setup with identical resources and no background load. It is defined by the parameters {T2, SP1, C1, SN1, RD1, LV, BG0, TS2, BS1}. This means, that we have a system with 32 Servers, each with the resource size $N_{RU,total} = 10$ and speed factor $f_{Speed} = 0.1$. Tasks sizes have a loguniform distribution with mean 1.0 (TS2), the task burst size is 1, and communication delays are neglected. In the experiment, the total amount of load in the system is varied (LV).

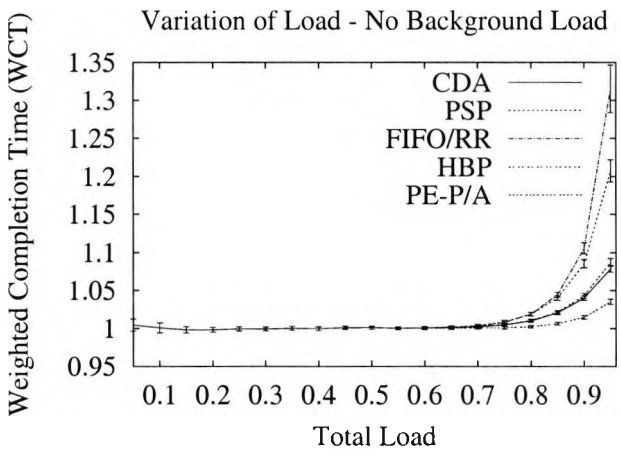

Figure 21: Tasks with different priorities: Variation of load, without background load.

Figure 21 shows the weighted completion time (WCT) for the examined protocols. For low to moderate load of up to 70%, all compared protocols perform about equally well, whereas for high load, the best performance is achieved by the two preemptive protocols, PE-P and PE-A. At 90% load, their WCT is about 4% lower than that for CDA. HBP performs slightly worse than CDA: at 90% load, its WCT is only about 0.8% higher: The delay of the low priority tasks, that are suspended, appears to outweigh the gain of the higher priority tasks. Not surprisingly, PSP performs even worse than HBP, as the concurrent execution of several tasks at the same resource delays the execution of *all* allocated tasks. The worst result is observed for RR and FIFO, because these two protocols do not prioritise tasks according to their weights. Since there is no difference between the resources, their results are identical.

Conclusion: For low to moderate load there is little difference between the examined protocols. For high loads, neither proportional-sharing nor task suspension provides any improvement over CDA. Only the two preemptive protocols outperform CDA by a small margin.

9.1.2 Background Load: Screensaver Mode

Next, we consider the case in which half of the total load in the system is background load (BG2). Background tasks have the computation size $S_{C,BG}$ = 1.0, and are allocated $N_{RU,BG}$ = 10 resource units at a time (SP2, *screensaver mode*). The results for CDA, SJF, PE-P, and

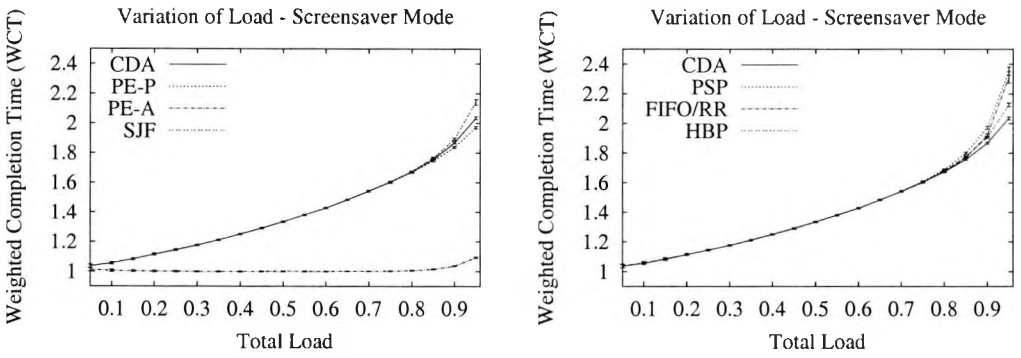

Figure 22: Variation of load, screensaver mode.

PE-A are shown in Figure 22 (left): There is hardly any difference between the protocols for loads between 0 and 75%. The only exception is the PE-A protocol, whose performance is much better. Its WCT remains close to 1.0 until 85% load, and then slightly increases to 1.1 at 95% load. The reason is that PE-A is able to reschedule a task when a background task is started at the resource — which would otherwise result in the task's suspension.

The result for PE-P at 95% load is only 3% better than that for CDA, and the result for SJF is 5% worse. HBP performs about equally well as SJF (see Figure 22 (right)). As there are still no differences between the resources in the system, RR and FIFO perform equally well: At 95% load, their WCT is 14% higher than CDA's. The poorest performance is observed for PSP, whose WCT at 95% load is now 16% higher than CDA's.

We also explored the use of reserve prices by the Servers (i.e. the CDA-RES protocol). In Figure 23, we compared the performance of CDA-RES to that of 'normal' CDA (i.e. where the Server reserve prices are set to zero). We considered different price discounts [42] for the reserve prices, however, no improvements were observed. Again, this can be explained by the fact that there are no differences in performance between the available resources. We also examined the PDA protocol, and it did not lead to any improvements either. Its results for different time intervals δt between the transactions are given in Figure 83 (left) of the appendix.

[42]Note that the price discount is the value which is deducted from a Server's average pay rate measured in the past. The resulting value is the Server's reserve price.

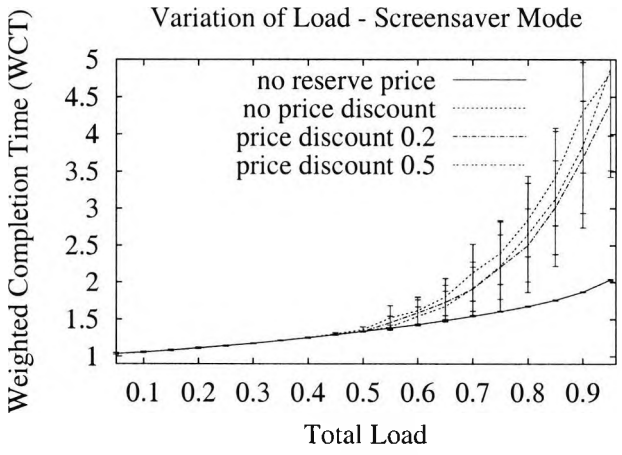

Figure 23: Variation of load, screensaver mode: CDA-RES with different price discounts.

Conclusion:

With coarse-grained background load (i.e. in screensaver mode), the differences between the protocols are small. The only exception is the PE-A protocol, which results in a considerable improvement in comparison to all other protocols. Using reserve prices (CDA-RES) or periodic auctions (PDA) does not lead to any performance improvements.

9.1.3 Fine-Grained Background Load

We also examine the protocols for fine-grained background load (SP1). The results for CDA, FIFO, SJF, PE-P, and PE-A are shown in Figure 24 (left). Now, both preemptive protocols perform considerably better than CDA: At 95% load, PE-P improves the WCT by 17%. As PE-A is able to react to changes in the background load, its WCT is even 23% lower than CDA's. For FIFO and SJF, there is hardly any difference to CDA for up to 90% load. However, at 95% load, FIFO's WCT is 12% higher than CDA's, and that of SJF is 4% higher.

Other protocols perform even worse (see Figure 24 (right)): At 95% load, PSP's and RR's WCTs are 14% and 17% higher, respectively. By far the poorest performance is observed for HBP: at 95% load its WCT is more than twice as high as CDA's. The reason is that, due to the differences in the background load of the resources, low priority tasks are more likely to be suspended by high priority tasks. Their delay results in a strong degradation of the overall performance.

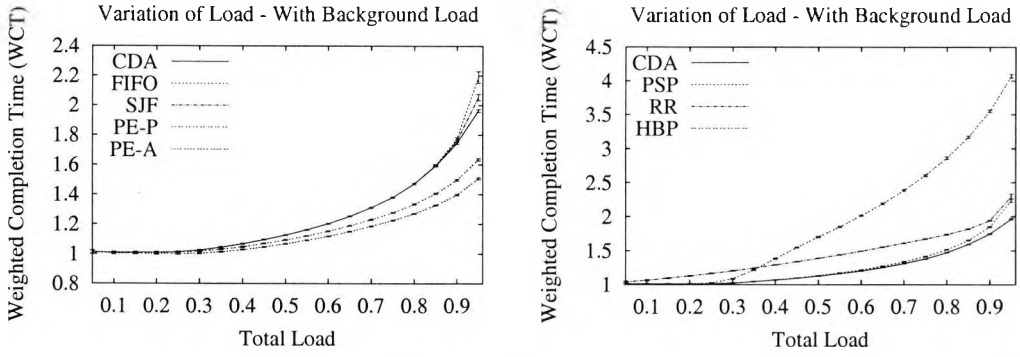

Figure 24: Tasks with different priorities: Variation of load, with fine-grained background load.

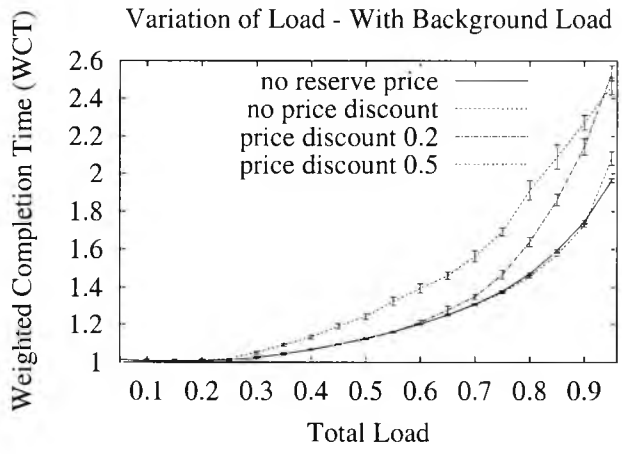

Figure 25: Variation of load, with fine-grained background load: CDA-RES with different price discounts.

The above results change slightly when the share of the background load is increased. These results are presented in Figure 84 of the appendix.

We also investigated the CDA-RES protocol (Figure 25). We found that, in general, CDA without reserve prices performs better. However, for the price discount of 0.5 — which corresponds to 50% of the average price bids of the tasks — a small improvement can be observed for moderate load (i.e. 65% — 90%).

Finally, we considered the PDA protocol but no improvements were observed. The results are given in Figure 83 (right) of the appendix.

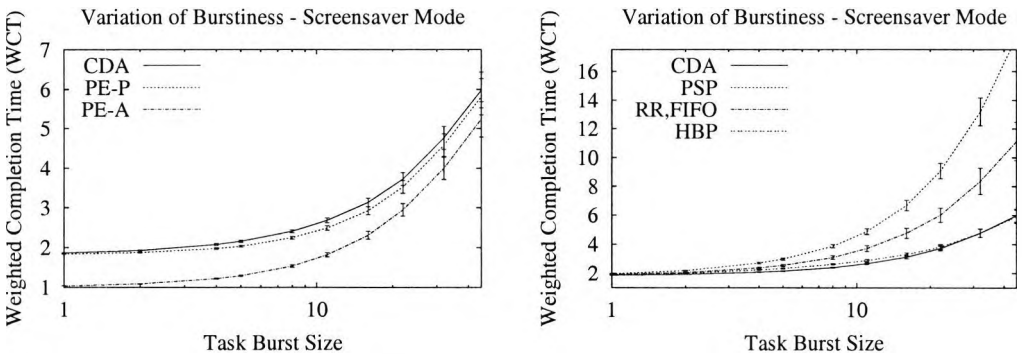

Figure 26: Variation of burstiness, screensaver mode.

Conclusion:

For fine-grained background load, PE-A clearly outperforms PE-P, and both protocols perform much better than all other protocols. The poorest result has been observed for HBP, which strongly degrades compared to a situation without background load. Prioritising shorter tasks, as in SJF, leads to small improvements when compared to FIFO but still results in poorer performance than CDA. Only marginal improvements could be observed for CDA-RES — and these were only achieved by using large price discounts.

9.1.4 Task Burstiness

Next, we examine the effect of changing the task burst size, *BS*. As parameters of the experiment we choose {T2, SP2, C1, SN1, RD1, L1, BG2, TS2, BSV}, i.e. the resources operate in screensaver mode, and the total load in the system is fixed at 90%. As shown in Figure 26 (left), the differences between CDA and the preemptive protocols become smaller with increased burst size. Interestingly, HBP now approaches CDA's performance.

In contrast to the scenario where tasks have the same priorities (Figure 14), Round-Robin and FIFO now degrade much stronger than CDA. This is because tasks are not prioritised by these protocols. For $BS = 45$, their WCT is 86% higher than for CDA (see Figure 26 (right)). The performance of PSP is even poorer: Its WCT is three times as high as for CDA.

Conclusion:

We varied the task burst size for the screensaver scenario and found that, with increased burst size, Round-Robin and FIFO now degrade much stronger than CDA. The preemptive protocols still perform best, but the gap to CDA is becoming smaller. The HBP protocol now performs equally well as CDA.

9.2 PC Grid

After having studied the *PC Cluster*, we now move on to the *PC Grid*-type infrastructures, which are characterised by higher resource heterogeneity, higher number of resources, and higher communication delays.

9.2.1 Resource Heterogeneity: Screensaver Mode

Initially, we examine the impact of having different degrees of heterogeneity for the resources in the system (RDV). We use the parameter set {T2, S2, C1, SN1, RDV, L1, BG2, TS2, BS1}: We have 32 resources in the system, and the average total load is 90%, half of which is background load. Task sizes have a loguniform distribution, and the task burst size is 1. The number of resource units per background task is set to $N_{RU,BG} = 10$, and the communication delays are set to zero.

Figure 27 (left) shows that the performance of CDA slightly improves when heterogeneity is increased. At a ratio of $f_{Speed,min}/f_{Speed,av} = 0.01562$, its WCT is 3% lower than for identical resources. RR degrades by about 5.6%, because it does not consider speed for the choice of resources. PSP improves by about 13% and now event outperforms CDA.

As shown in Figure 27 (right), the WCT of the HBP protocol is about twice as high as for the other protocols. Also, there is a strong decrease of the WCT for completely homogeneous resources, i.e for the case $f_{Speed,min} = f_{Speed,av}$. It can be explained by the fact that the HBP protocol gives preference to idle resources when several resources with *identical* performance are available, resulting in fewer delays of the executing tasks. A small improvement can be observed for the FIFO protocol, whose WCT at $f_{Speed,min}/f_{Speed,av} = 0.01562$ is 2.5% higher than CDA's. The performance of the two preemptive protocols improves con-

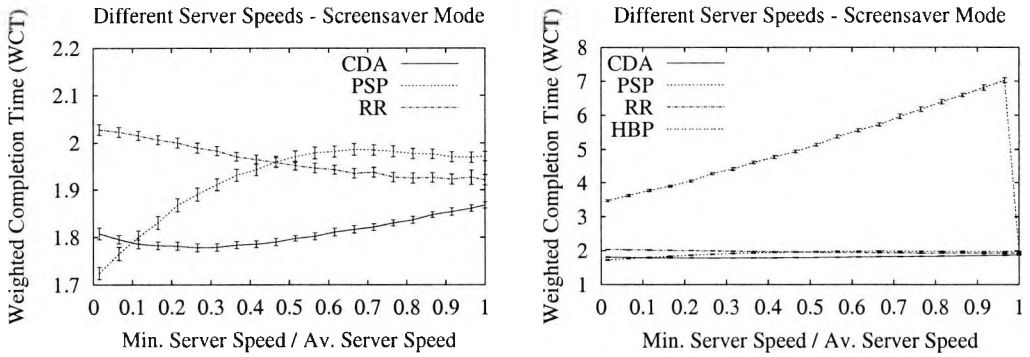

Figure 27: PC Grid: Variation of resource heterogeneity, screensaver mode. Left: Results for CDA, PSP, and RR. Right: Results for the HBP protocol.

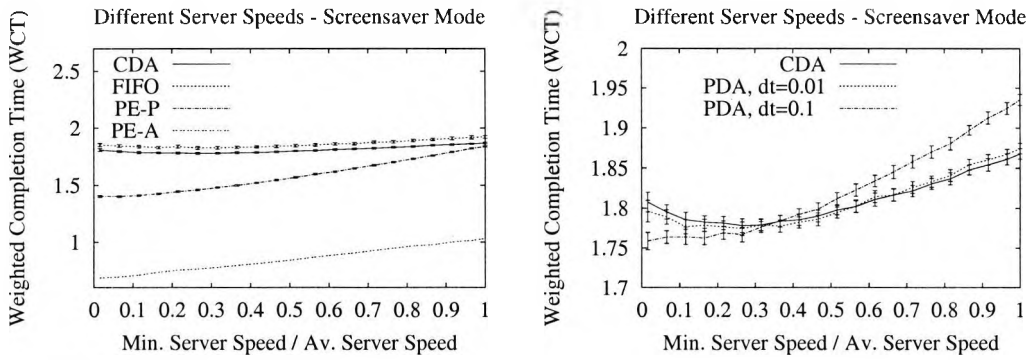

Figure 28: PC Grid: Variation of resource heterogeneity, screensaver mode. Left: Results for FIFO, PE-P, and PE-A in comparison to CDA. Right: Results for PDA with different intervals between the transactions.

siderably as heterogeneity is increased (see Figure 28 (left)). PE-P's WCT decreases by 24% and that of PE-A by about 34%.

We examined the effect of Server reserve prices for both, CDA and HBP, but could not find any improvements. The results can be found in Figure 85 of the appendix.

A more interesting observation has been made for the PDA protocol. In Figure 28 (right), the results for PDA with different time intervals δt between the transactions are compared to CDA's results. A reduction of the WCT was obtained when $\delta t = 0.1$, though only for $f_{Speed,min}/f_{Speed,av} < 0.35$. At $f_{Speed,min}/f_{Speed,av} = 0.01562$, the WCT was 3% lower than for CDA. For $\delta t = 0.01$, only marginal improvements were observed — but for a larger range of ratios $f_{Speed,min}/f_{Speed,av}$.

Conclusion:

When resource heterogeneity is increased in screensaver mode, considerable performance improvements are observed for PSP and the two preemptive protocols, and small improvements for CDA and FIFO. In this setup, PSP even outperforms CDA. A strong degradation, however, occurs for Round-Robin and HBP. We also found, that periodic auctions (PDA) can outperform continuous auctions when resource heterogeneity is high — provided that a suitable value is chosen for the time interval between the transactions. No improvements were achieved by using CDA-RES or HBP-RES.

9.2.2 Resource Heterogeneity: Fine-Grained Background Load

Next, we examine different degrees of resource heterogeneity for fine-grained background load (SP1). The results for CDA, PSP, and RR are shown in Figure 29 (left). They are similar to those observed in screensaver mode (Figure 27 (left)). The main difference is that PSP now performs better than Round-Robin for all degrees of heterogeneity. For HBP, FIFO, PE-P, and PE-A, the observations from the previous experiment can also be confirmed (see Figure 86 in the appendix).

We also examined the CDA-RES protocol, for which the results are shown in Figure 29 (right). In all cases, the performance improves with increased heterogeneity. However, only for the price discount 0.5, a better performance can be achieved than for CDA. The gap widens and amounts to about 9% at $f_{Speed,min}/f_{Speed,av} = 0.01562$. The reason is that the faster resources have higher prices and therefore are not available to low priority tasks. As a result, they are more likely to be available to high priority tasks, leading to improved performance — which is even better than for the PSP protocol.

Conclusion: When the resource heterogeneity is varied, the results are very similar to those obtained in screensaver mode. With increased heterogeneity, all examined protocols, except Round-Robin, improve their performance. The largest gains could be observed for PSP and the two preemptive protocols. A considerable improvement was also observed for CDA-RES, which delivers a better result than PSP.

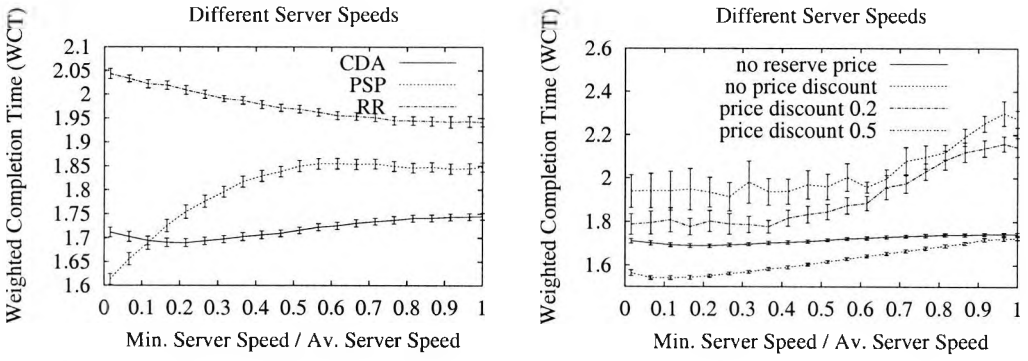

Figure 29: PC Grid: Variation of resource heterogeneity for fine-grained background load. Left: Results for CDA, PSP, and RR. Right: CDA-RES with different price discounts.

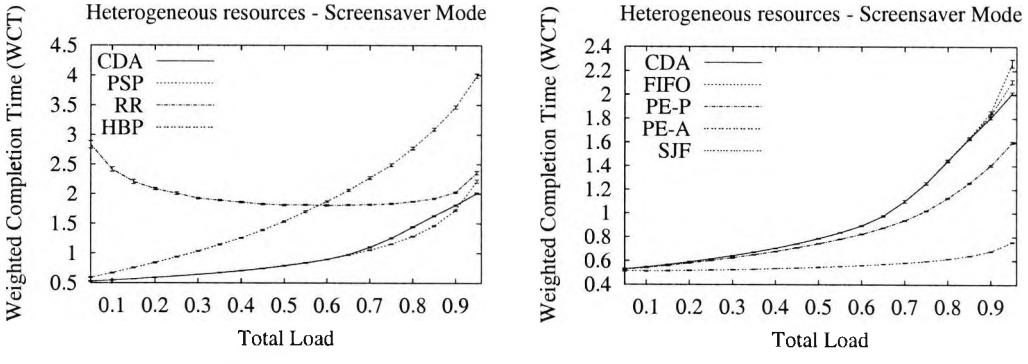

Figure 30: Screensaver mode: Variation of load for heterogeneous resources.

9.2.3 Variation of Load: Screensaver Mode

As could be observed in subsections 9.2.1 and 9.2.2, some of the examined protocols can lead to considerable performance improvements if the heterogeneity of resources is high. However, in all these experiments, the average load in the system has been set to 90%. In this section we will examine a situation where the resource heterogeneity is high and will compare the protocols for different amounts of load. In the experiments, we use the parameter set {T2, SP2, C1, SN1, RD2, LV, BG2, TS2, BS1}. Due to the poor performance of HBP in the previous experiments, we now also examine the HBP-T and HBP-RES protocols.

The results for CDA, PSP, RR, and HBP are shown in Figure 30 (left). For RR, the same observations as for the scenario described in chapter 8 (see Figure 17 (left)) have been made. HBP's performance is also very poor compared to CDA: Its WCT is even higher than that for RR, when the load in the system is greater than 60%. Among these four protocols, CDA

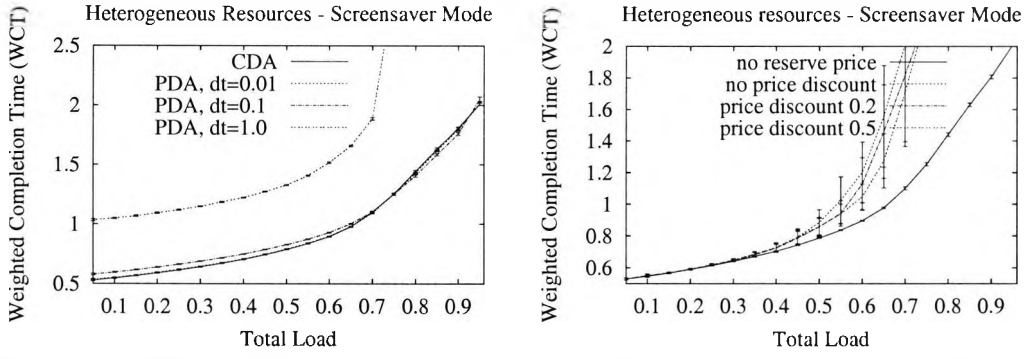

Figure 31: Variation of load for heterogeneous resources. Left: Results for PDA with different intervals between the transactions. Right: CDA-RES with different price discounts.

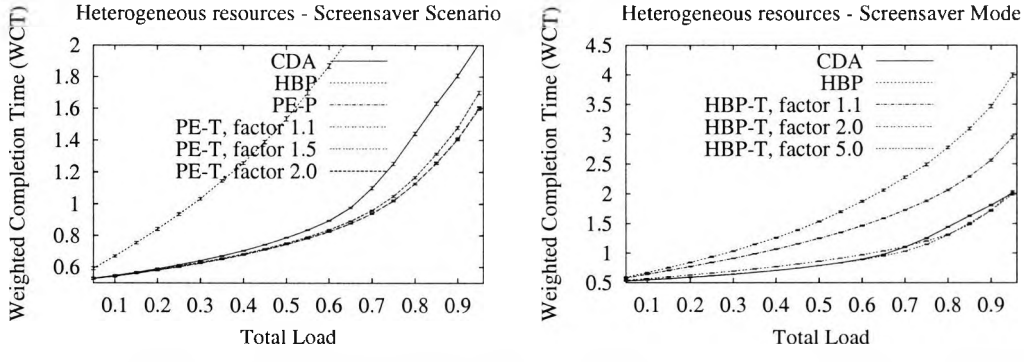

Figure 32: Variation of load for heterogeneous resources. Left: PE protocol with different thresholds (PE-T). Right: HBP protocol with different thresholds (HBP-T).

performs best, except in the range of 65% to 90% load, where it is outperformed by PSP. At 80% load, PSP is 11% better than CDA. FIFO performs about equally well as CDA for up to 90% load, as is shown in Figure 30 (right). Still, the two preemptive protocols provide the best results for the whole range of values. The gap to CDA widens as load is increased. Note that the WCT for PE-A is considerably lower than for PE-P.

With PDA, only marginal improvements in comparison to CDA were achieved, as is shown in Figure 31 (left). CDA-RES does not lead to any improvement (see Figure 31 (right)). The same is the case for CDA-RES in combination with *task price adjustment* [43]. The results with the negotiation times 1.0 and 10.0 are given in Figure 89 of the appendix.

The HBP protocol with reserve prices (HBP-RES) can improve performance in compar-

[43]This version of CDA-RES will be described in subsection 9.2.4. The mechanism for the task price adjustment and the notion of negotiation time will be described in subsection A.2.5 of the appendix.

ison to the 'normal' HBP protocol but still provides poorer results than the other protocols. The results for different price discounts are shown in Figure 90 in the appendix.

We also examined the PE-T protocol, for which we enabled only 'passive' preemption (as in PE-P). The *speed improvement factor* was set to a very high value. This prevented a task from preempting other, lower priority tasks — unless it had no resource at all. We examined different values for the *bid improvement factor*, i.e. the threshold which determines how much higher the bid of a task must be in order to *preempt* another task. We found that PE-T is never better than PE-P but performs better than CDA and HBP for all the examined bid improvement factors (see Figure 32 (left)).

Finally, we considered HBP-T, a version of the HBP protocol which uses a *bid improvement factor*. We examined different values for this factor and compared the results to those obtained by CDA and HBP. We found that HBP-T always outperforms HBP. Furthermore, with the values 2.0 and 5.0, it is also better than CDA — if the amount of load in the system is high. The results are shown in Figure 32 (right).

Conclusion:

Most of the observations that were made for heterogeneous resources, could now be confirmed for a wider range of loads in the system. PSP performs better than CDA for 65% to 90% load. PDA, however, provides little improvement, and CDA-RES results in no improvement at all. The same applies to HBP-RES. We also examined the HBP-T protocol and found that it always outperformed HBP. With an appropriate choice of threshold, its WCT was also lower than CDA's.

9.2.4 Variation of Load: Fine-Grained Background Load

The setup which was studied in the previous section will now be examined for fine-grained background load (SP1): The experiment is defined by the parameter set {T2, SP1, C1, SN1, RD2, LV1, BG2, TS2, BS1}. For CDA, PSP, RR, HBP, FIFO, PE-P, and PE-A the results are almost the same as before and are given in Figure 91 of the appendix.

The results for the PDA protocol are shown in Figure 33 (left). Compared to CDA, there is a visible improvement for $\delta t = 0.1$, when the load in the system is higher than 75%.

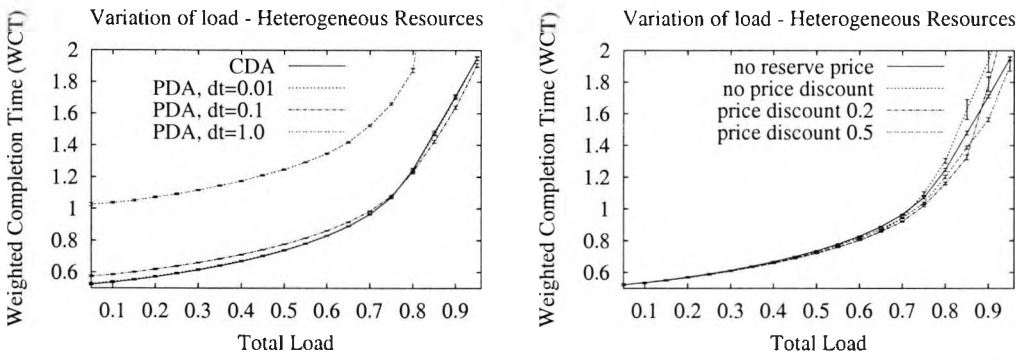

Figure 33: Variation of load for heterogeneous resources. Left: Results for PDA with different intervals between the transactions. Right: CDA-RES with different price discounts.

We also examined CDA-RES with different price discounts, and found that for almost the whole range of loads it can outperform 'normal' CDA (see Figure 33 (right)). However, the price discount has to be chosen carefully: For low load, 'no price discount' leads to the best performance, for moderate load, the best results are achieved with price discount 0.2, and for high load, price discount 0.5 is best.

One risk of using CDA-RES is that low priority tasks may never be executed. This may be the case if the price discount is low and load is high. To avoid this problem, we extended CDA-RES, so that it allows tasks to adjust their prices while waiting at the EMP (see subsection A.2.5 in the appendix). A task will start with an initial bid $p_{Task,min}$, which is linearly increased up to its maximum bid $p_{Task,max}$. These price adjustments at the EMP are carried out at periodic intervals, for which we used $\tau = 0.1$. For each task, we choose the initial bid $p_{Task,min}$ such that it is proportional to its weight. The maximum bid $p_{Task,max}$ is the same for all tasks and corresponds to the bid of the highest priority task. We considered different *negotiation times* T_{Neg} between the initial and the maximum price bid of a task. The results for negotiation time 1.0 are given in Figure 34 (left), and those for negotiation time 10.0 in Figure 34 (right). For moderate loads, performance was best with the price discount 0.2 — with both negotiation times. For high loads, the best result was achieved with price discount 0.5 and negotiation time 10.0.

Most of the experiments, which have been described in this subsection, have also been carried out for a situation without any background load on the resources. The results are

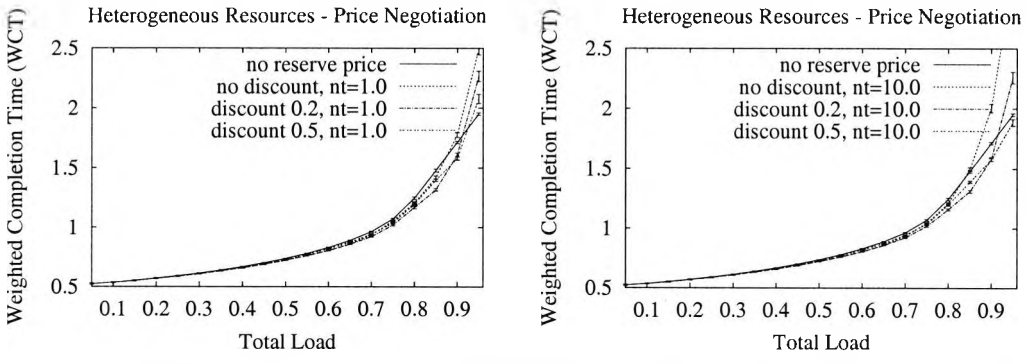

Figure 34: Heterogeneous resources: CDA-RES with different price discounts. Left: Task negotiation time 1.0. Right: Task negotiation time 10.0.

similar in most cases and are given in the Figures 87 and 88 of the appendix.

Conclusion:

With fine-grained background load, not much has changed for most of the examined protocols. However, in contrast to the screensaver mode, PDA performs better than CDA for loads higher than 75%. CDA-RES improves the performance for almost any amount of load — as long as an appropriate value for the price discount is chosen. Using reserve prices *and* task price adjustments can also perform better than 'normal' CDA. At the same time it ensures that even low priority tasks will eventually be executed.

9.2.5 Different Server Numbers

The aim of this experiment is to see the effect of the Server number on the weighted completion time. We examine the screensaver mode and set the average load in the system to 90%. Hence, the parameters of the experiment are given by {T2, SP2, C1, SNV, RD2, L1, BG2, TS2, BS1}.

The results for CDA, PSP, RR, and HBP are shown in Figure 35 (left). With increased number of Servers, the performance of these protocols improves. An exception is HBP, which degrades, because, with higher number of Servers, tasks are more likely to be preempted and delayed. As already observed in Figure 17 (right), Round-Robin's WCT is higher than for most other protocols. PSP outperforms CDA for more than 20 Servers. How-

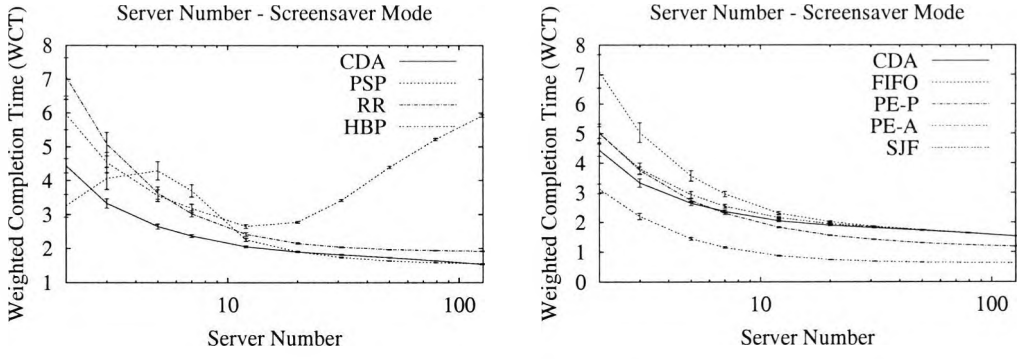

Figure 35: Variation of the Server number, screensaver mode.

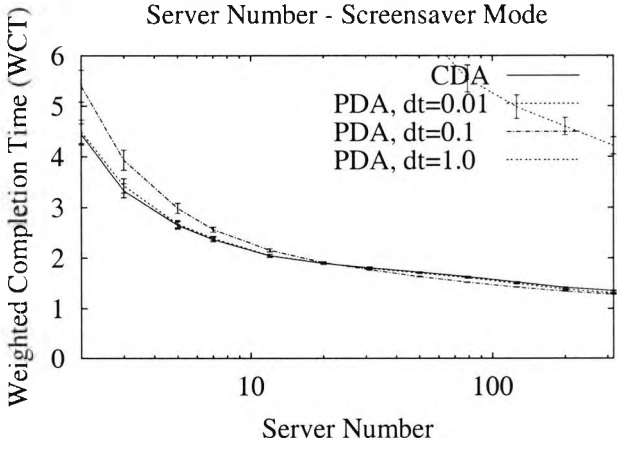

Figure 36: Variation of the Server number, screensaver mode: Results for PDA with different intervals between the transactions.

ever, the result of the two protocols are almost the same when there are more than 100 Servers in the system. The reason is that a shortage of resources becomes less likely, and therefore, tasks will not have to share a resource. FIFO and SJF also lead to the same results as CDA when the number of Servers is increased (see Figure 35 (right)). Since tasks rarely have to wait at the EMP, it does not matter any more, in what way they are prioritised. Again, the two preemptive protocols perform best: With 126 Servers in the system, the WCT for PE-P is 23% lower than CDA's, and for PE-A, it is even 59% lower. Interestingly, PDA is the only non-preemptive protocol which outperforms CDA for a high number of Servers (see Figure 36). The gap between the two protocols is largest for about 100 Servers, where, with $\delta t = 0.1$, PDA's WCT is about 7% lower than CDA's.

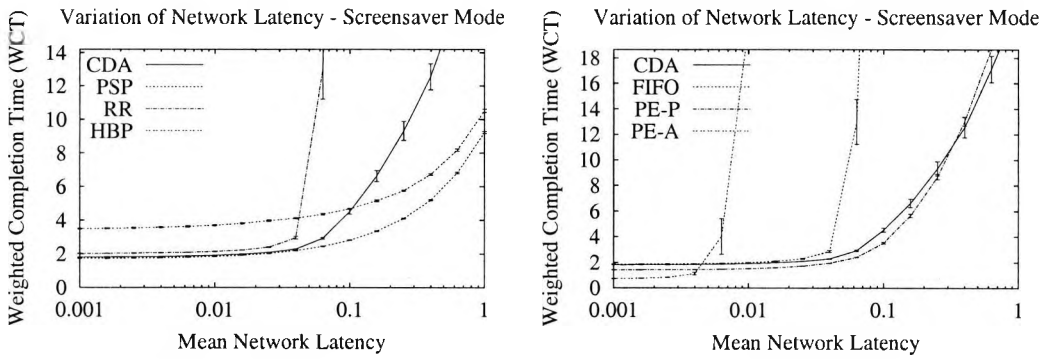

Figure 37: Variation of the communication delay, screensaver mode.

Conclusion: We varied the Server number in screensaver mode and found that the results of all protocols — except HBP — improve with increased number of Servers. The protocols PSP, FIFO, SJF, and CDA perform about equally well. PDA however outperforms CDA, even for a high number of Servers.

9.2.6 Communication Delays

We also examine a variation of the communication delays, as has been done in subsection 8.2.4. The simulation parameters of this experiment are given by {T2, SP2, C2V, SN1, RD2, L1, BG2, TS2, BS1}. For the latency, we use a lognormal distribution, of which the mean is varied (C2V).

As shown in Figure 37 (left), RR and CDA degrade more rapidly than PSP — for the same reason as in subsection 8.2.4 (Figure 18 (left)). However, the increase is now slower for CDA than for RR: This is because of its prioritisation of tasks and selection of the best possible resources. The result for HBP is similar to that of PSP. The reason is that, on arrival of a high priority task, a resource is allocated immediately, without waiting for it to become available. Hence, resources do not remain idle during the communication delays.

For FIFO (Figure 37 (right)), a similar degradation can be observed as for Round-Robin, because it uses the same protocol for prioritising tasks. PE-P performs better than CDA for a mean latency of up to 0.2, beyond which it increases faster. This can be explained by the additional communication delays which are caused by the preemptions. For PE-A, a very sharp rise of the WCT can already be observed for a mean latency of less than 0.01. This is

because preemptions are more likely to occur than in PE-P and increase the delays.

Conclusion: For a high network latency, PSP and HBP degrade less than the other protocols. The poorest performance can be observed for PE-A: This is caused by the additional communication delays during the preemptions. For moderate and high communication delays, PE-P's performance is similar to that of CDA — in spite of the additional communication delays.

9.3 Summary

In this chapter, we investigated the allocation of tasks with different priorities and used the weighted completion time (WCT) as performance metric. In this section, we give an overview of the parameters used in each experiment (see Figure 38). Also, we summarise our findings for each protocol. Our guidelines for the system designer will be given in chapter 13.

Continuous Double Auction Protocol (CDA)

Among the protocols, that do not use preemption, CDA usually provides the best performance — or, at least, is very close to the best. However, most other protocols perform equally well, if the load is low or the number of resources in the system is very high. Differences occur at higher loads, and are usually small in screensaver mode.

CDA with Reserve Prices (CDA-RES)

If there are differences in speed or load among the resources, CDA-RES can lead to better performance than 'normal' CDA. The reason is that reserve prices can help to express these differences: The reserve prices exclude low priority tasks from using the well-performing resources so that they remain available to high priority tasks, which may arrive at a later time. However, performance gains could only be observed when large price discounts were used. No improvements have been achieved in screensaver mode: It appears that the performance of CDA-RES degrades, if there are large variations of the background load at the resources. One risk of CDA-RES is that some low priority tasks may never be executed. However, this problem can be dealt with by gradually increasing their price bids ('task price adjustment').

Infrastructure	Figure	Parameters
PC Cluster	21	T2, SP1, C1, SN1, RD1, LV, BG0, TS2, BS1
PC Cluster	22	T2, SP2, C1, SN1, RD1, LV, BG2, TS2, BS1
PC Cluster	83 (left)	T2, SP2, C1, SN1, RD1, LV, BG2, TS2, BS1
PC Cluster	23	T2, SP2, C1, SN1, RD1, LV, BG2, TS2, BS1
PC Cluster	24	T2, SP1, C1, SN1, RD1, LV, BG2, TS2, BS1
PC Cluster	84	T2, SP1, C1, SN1, RD1, LV, BG3, TS2, BS1
PC Cluster	25	T2, SP1, C1, SN1, RD1, LV, BG2, TS2, BS1
PC Cluster	83 (right)	T2, SP1, C1, SN1, RD1, LV, BG2, TS2, BS1
PC Cluster	26	T2, SP2, C1, SN1, RD1, L1, BG2, TS2, BSV
PC Grid	27	T2, SP2, C1, SN1, RDV, L1, BG2, TS2, BS1
PC Grid	28	T2, SP2, C1, SN1, RDV, L1, BG2, TS2, BS1
PC Grid	85	T2, SP2, C1, SN1, RDV, L1, BG2, TS2, BS1
PC Grid	29	T2, SP1, C1, SN1, RDV, L1, BG2, TS2, BS1
PC Grid	86	T2, SP1, C1, SN1, RDV, L1, BG2, TS2, BS1
PC Grid	30	T2, SP2, C1, SN1, RD2, LV, BG2, TS2, BS1
PC Grid	31	T2, SP2, C1, SN1, RD2, LV, BG2, TS2, BS1
PC Grid	32	T2, SP2, C1, SN1, RD2, LV, BG2, TS2, BS1
PC Grid	89	T2, SP2, C1, SN1, RD2, LV, BG2, TS2, BS1
PC Grid	90	T2, SP2, C1, SN1, RD2, LV, BG2, TS2, BS1
PC Grid	91	T2, SP1, C1, SN1, RD2, LV, BG2, TS2, BS1
PC Grid	33	T2, SP1, C1, SN1, RD2, LV, BG2, TS2, BS1
PC Grid	34	T2, SP1, C1, SN1, RD2, LV, BG2, TS2, BS1
PC Grid	87	T2, SP1, C1, SN1, RD2, LV, BG0, TS2, BS1
PC Grid	88	T2, SP1, C1, SN1, RD2, LV, BG0, TS2, BS1
PC Grid	35	T2, SP2, C1, SNV, RD2, L1, BG2, TS2, BS1
PC Grid	36	T2, SP2, C1, SNV, RD2, L1, BG2, TS2, BS1
PC Grid	37	T2, SP2, C2V, SN1, RD2, L1, BG2, TS2, BS1

Figure 38: Tasks with different priorities: Overview of experiments.

We found that this approach can lead to better results than normal CDA.

Interestingly, the performance improvements by CDA-RES can be achieved, even though the resources remain idle for considerable amounts of time. It appears that this 'waste' of processing power is outweighed by the gain for the high priority tasks — and overall, the users can benefit from the opportunistic pricing of the Servers.

Proportional Share Protocol (PSP)

PSP will perform worse than CDA if the resources in the system are identical, as is the case in a PC Cluster. It degrades stronger than CDA for higher loads: This is due to the concurrent execution of several tasks on one resource. The differences become even larger when the burstiness of the tasks is increased. However, with a high heterogeneity of resources

and moderate-to-high loads, PSP outperforms CDA. This performance improvement can be attributed to two factors: Firstly, its preemptive behaviour enables arriving high priority tasks to take much of the resource share from already executing low priority tasks — effectively preempting them. Secondly, it might be better to allocate two tasks to one fast resource, rather than to a fast and a slow one. PSP also outperforms the other protocols in situations with high communication delays: It degrades less, as it does not need to check the availability of the resources.

Highest Bid Protocol (HBP)

In all our experiments, HBP leads to poorer performance than CDA. For heterogeneous resources, its results are worse than for identical resources. It appears that the delay of the suspended low-priority tasks outweighs the gain of the high priority tasks. In contrast to all other protocols, HBP's WCT increases with higher number of resources. A strong degradation has also been observed for a large task burst size. However, when communication delays are introduced, HBP degrades less than most other protocols.

HBP with Threshold (HBP-T)

Due to the poor performance of HBP, we introduced the HBP-T protocol, which uses a threshold for the preemptions of low priority tasks. We examined it for heterogenous resources operated in screensaver mode and found that it outperformed HBP for all amounts of load. Also, if an appropriate threshold was chosen, its results were better than CDA's. This shows that preemption *without* migration can lead to performance improvements — however, there needs to be a limit for the preemptions. A disadvantage of HBP and HBP-T is that low priority tasks may be starved. A solution could be to increase the priority of the suspended tasks over time, or to limit the number of times a task can be preempted (as done by [Chun and Culler, 2002]).

HBP with Reserve Prices (HBP-RES)

HBP-RES aims to reduce the number of preemptions through the use of reserve prices and has been examined for heterogeneous resources in screensaver mode. We found that HBP-RES can perform better than HBP for low and moderate loads. However, in comparison to

CDA, no improvements were observed.

Preemptive Protocol (PE)

In a PC Cluster without background load, the results for PE-P and PE-A are only marginally better than CDA's. In screensaver mode, PE-P's WCT is only slightly lower than CDA's, whereas PE-A's WCT is much lower than for any other protocol. With fine-grained background load, PE-P clearly outperforms CDA — and PE-A's performance is even better.

In a PC Grid, PE-P performs clearly better than CDA, and PE-A is much better than all other protocols. The differences remain large even for a high number of resources. The only disadvantage of using task migration becomes evident when communication delays are introduced: PE-P's degradation is similar to that of CDA, and PE-A's degradation is the worst. Another weakness of our PE-P and PE-A protocols is that each migration may trigger another migration. This will often result in a chain reaction which can be a huge burden on the central marketplace — especially if the system is large [44]. A possible solution to this problem could be limiting the number of preemptions, as done by the PE-T protocol, which has briefly been examined. Its performance will be lower than for PE-P or PE-A in ideal situations without communication or processing delays, but it is likely to be more scalable.

Periodic Double Auction Protocol (PDA)

In the PC Cluster infrastructure, PDA did not lead to any improvements in comparison to CDA. It appears, that the degree of heterogeneity was not sufficient in this setup. However, in the PC Grid scenario, a considerable improvement was observed for moderate and high loads, when a suitable interval for the time between the transactions was chosen. The improvement was smaller in screensaver mode, where the large variations of the background load seem to affect the result. For a high number of resources, PDA also performed better than CDA, with the maximum difference at 100 resources in the system. The reason why PDA can, in some cases, lead to improvements over CDA, is that, during the transaction period, several tasks and resources are accumulated, and better matches can be made.

[44] Note that our simulation model does not consider the cost of processing incoming requests at the marketplace or the cost of packaging up the execution state of tasks for migration.

First In First Out (FIFO)

FIFO uses a greedy strategy, and therefore leads to almost the same results as CDA when the load is low or the number of resources is high. However, it does not take into account the priority of the tasks, and therefore performs worse when there is considerable competition among them — which is the case when the load in the system is high. FIFO's performance also degrades more than CDA's when communication delays are introduced.

Round-Robin Protocol (RR)

Among the examined protocols, Round-Robin leads, in most cases, to the poorest performance. It degrades with increased heterogeneity, especially if the load in the system is very small. It improves with increased Server number, but not as much as the other protocols. However, Round-Robin is computationally less expensive. Therefore, it can be appropriate for a system with identical resources and low or moderate load — and, in screensaver mode, even with high load. In such a setup, which may often be found in a computational cluster, its results will be almost as good as CDA's.

Shortest Job First (SJF)

In the examined situations, SJF's performance was always better than FIFO's and worse than CDA's. However, it must be noted, that the outcome of this comparison with CDA may depend on the probability distributions used for the task priorities and task sizes.

10 Tasks with Time-Dependent Priorities

In reality, the users of a system will often have tasks with deadlines rather than simply wanting to minimise their completion times. This chapter provides a comprehensive performance comparison of different resource allocation protocols for a scenario in which tasks have deadlines and different weights. For the weights, we use a uniform distribution $[0.0, 2.0]$. In this scenario, the system is de-facto *oversubscribed* because there are not enough resources which are sufficiently fast to execute all tasks on time.

Concerning the deadlines, we distinguish two cases: In the first case, tasks have *hard* deadlines, i.e. their execution will only benefit the Client if completed on time. In our experiments, hard deadlines are expressed by different values for the *deadline factor* which we define as the maximum slowdown a task may suffer without missing the deadline. As performance metric we use the weighted completion rate (WCR), which is the sum of weights of the tasks completed before the deadline divided by the sum of all task weights. In the second case, tasks have *soft* deadlines: A task's value (per task unit) is expressed as a piecewise-linear function of its slowdown (see subsection 2.6.1). As a performance metric, we use the *aggregate user utility* which we define as the average of the values delivered by the tasks to the Client, i.e. the average of task values multiplied by the task sizes. We express soft deadlines by different values of the slowdown factors sl_1 and sl_2 (see subsection 2.6.1).

10.1 Examined Parameter Space

We want to investigate situations with different types of deadlines and therefore need to limit the range of other parameters. Concerning background load, we examine the *screensaver mode* (SP2), which we consider to be more realistic than fine-grained background load (SP1). However, we also study situations without any background load, for which — in chapter 9 — we made (qualitatively) similar observations as with fine-grained background load.

In addition to the protocols which were compared in chapter 9, we now also investigate CDA-TDB which is specifically designed for situations with soft deadlines [45]. Note that, in

[45] CDA-TDB will not be examined for situations with hard deadlines as it would operate in the same way as CDA.

the experiments with soft deadlines, the time-dependent values of the tasks are only considered by the CDA-TDB protocol, whereas all other protocols use the initial price bids when allocating tasks, i.e. they are 'blind' to the soft deadlines. Round-Robin and HBP-RES are not examined due to their poor performance in previous experiments. For PDA, we only consider $\delta t = 0.1$ for the time-interval between the transactions, as this led to the best results in the previous chapter. For the same reason, we always choose the 'bid improvement factor' 5 for the HBP-T protocol and the 'price discount' 0.5 for the CDA-RES protocol. In addition to this, we examine CDA-RES for the case that no discount is used for the reserve prices. We start with the PC Cluster infrastructure and then move on to the PC Grid infrastructure.

10.2 PC Cluster

Concerning the deadlines, we examine the following cases for the PC Cluster infrastructure: We first consider situations where tasks have hard deadlines. We examine both 'tight' and 'loose' deadlines for which we use the deadline factors 1.1 and 1.5, respectively [46]. Next, we examine three different cases where tasks have soft deadlines which are expressed by the parameters sl_1 and sl_2 [47]. In all three cases, the parameter sl_1 is set to 1.1 for all tasks, whereas for sl_2, we use different uniform distributions. The reason for choosing uniform distributions for sl_2 is that we want to examine situations in which tasks have different delay tolerances. In the first case with 'tight' deadlines this distribution is given by $[1.1, 1.5]$ (we refer to this case as 'Soft Deadline I'). In the second case ('Soft Deadline II') we use $[1.1, 3.0]$, and in the third case ('Soft Deadline III') $[1.1, 10.0]$.

10.2.1 Hard Deadlines

In the first experiment, we examine a situation with tight deadlines, i.e. the deadline factor is set to 1.1. The other parameters of the experiment are given by {SP2, C1, SN1, RD1, LV, BG2, TS2, BS1}. This means, that we have a system with N_{Serv}= 32 Servers which are

[46] We consider the deadline factor 1.1 to be a tight deadline, because, for a deadline factor of less than 1.0, no task would be able to complete on time — even when executed on an unloaded machine.

[47] Note that the slowdown factor sl_1 determines the maximum slowdown that a task can suffer without any loss in value. Beyond the slowdown sl_2, the value of the task becomes zero (see subsection 2.6.1 for more details).

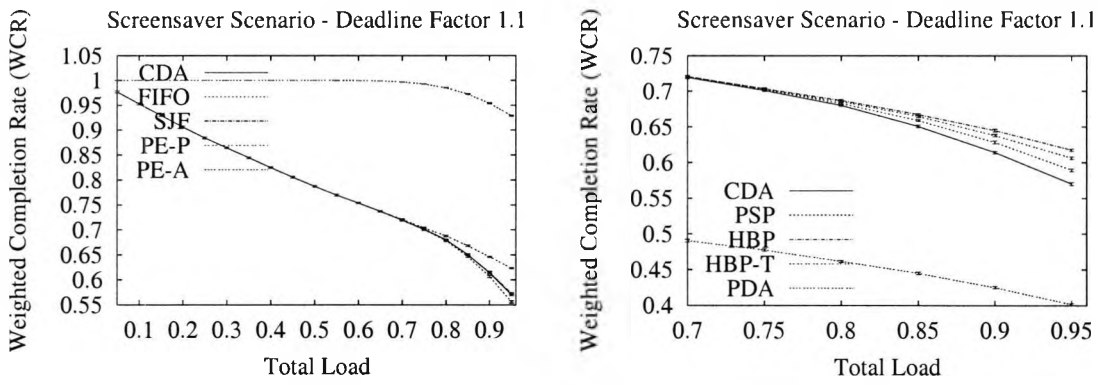

Figure 39: Variation of load, screensaver mode: Tight deadlines (deadline factor 1.1).

operating in *screensaver mode* (SP2), each with the resource size $N_{RU,total}$ = 10 and speed factor f_{Speed} = 1.0. Tasks sizes have a loguniform distribution with mean 1.0 (TS2), the task burst size is 1, and communication delays are neglected. In the experiment, the total amount of load in the system is varied (LV).

The results for CDA, FIFO, SJF, PE-P, and PE-A are shown in Figure 39 (left): There is hardly any difference between the protocols for loads between 0 and 75%. The only exception is the PE-A protocol, whose performance is much better — as was already the case in chapter 9. Its WCR remains close to 1.0 until 85% load, and then slightly decreases to 0.93 at 95% load. The reason for PE-A's good performance is its ability to reschedule a task when a background task is started at the resource — which would otherwise result in the task's suspension. CDA's WCR declines to about 0.57 at 95% load. PE-P performs better, and also the results of SJF are marginally better than for CDA. A possible reason is that shorter tasks are likely to suffer more from being suspended by background load than longer tasks. Therefore, it makes sense to prioritise them.

Interestingly, CDA is now also outperformed by PSP and HBP (see Figure 39 (right)). At 95% load, HBP is 5% better than CDA and 1% worse than PE-P. PSP's relatively good performance may appear surprising since the resources in the system are identical. It can be explained by the fact that PSP allows arriving high priority tasks to take some of the resources being used by the other executing tasks — thus, effectively preempting them. PDA's performance is the poorest of all protocols that are compared. This is because of the delays

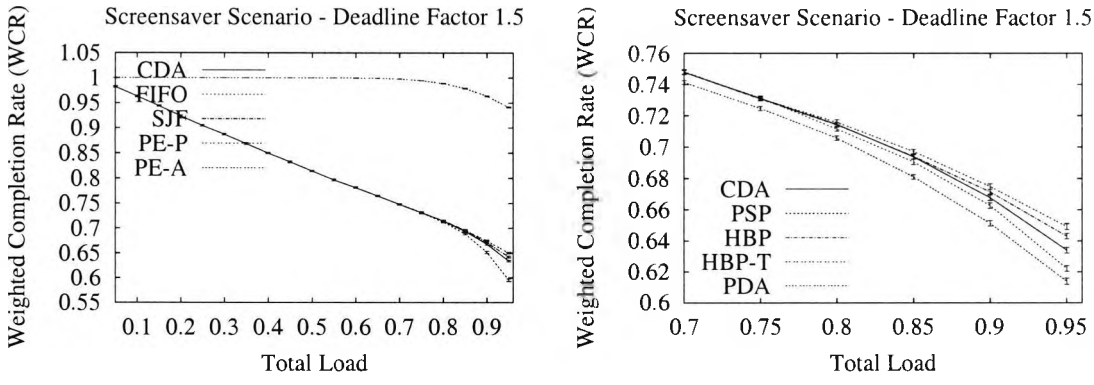

Figure 40: Variation of load, screensaver mode: Loose deadlines (deadline factor 1.5).

which the tasks suffer during the time intervals between the transactions at the EMP. HBP-T is 6% better than CDA but still 2% worse than 'normal' HBP.

We also carried out this experiment without any background load in the system (BG0). As expected, the degradation of the protocols is much smaller than in screensaver mode. We found that, performance-wise, the order of the protocols remains unchanged, except that SJF now performs slightly worse than CDA. Also, due to the lack of background load, PE-P and PE-A are identical. The results are shown in Figure 92 of the appendix.

Next, we relaxed the deadline (deadline factor 1.5), and carried out the experiment in screensaver mode (SP2). As shown in Figure 40, the performance of the protocols improves only slightly in comparison to the experiment with the deadline factor 1.1. The reason could be that, whenever a task is suspended by background load, it is likely to miss its deadline anyway. As before, PE-A's WCR is much higher than for the other protocols. Also, PE-P and SJF both outperform CDA if load is high.

HBP-T provides better results than HBP, which also performs better than CDA (Figure 40 (right)). PDA's WCR is still lower than for CDA. However, the difference is much smaller than in the previous experiments, because tasks are less likely to miss their deadlines while waiting for the next transaction at the EMP. In contrast to the previous experiments, PSP now leads to a poorer performance than CDA but still performs better than PDA and FIFO.

Conclusion:

With hard deadlines, PE-A's results are much better than for the other protocols. How-

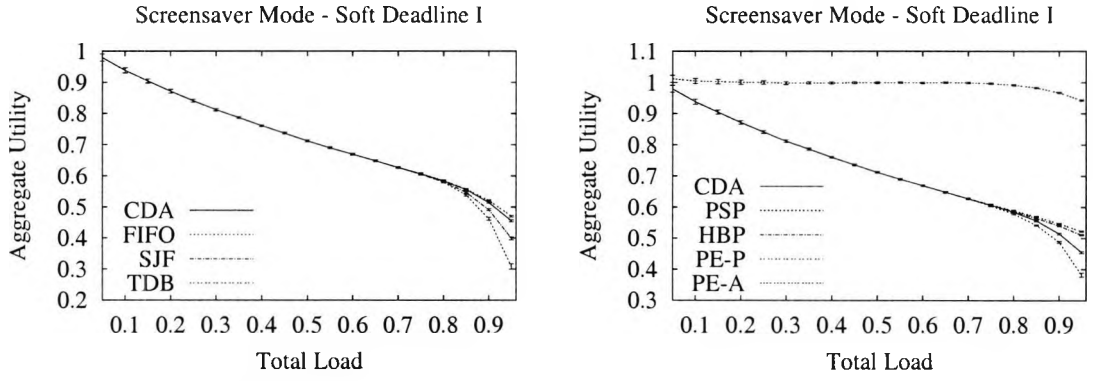

Figure 41: Variation of load, screensaver mode: Soft deadlines I (tight).

ever, in contrast to the experiments in chapter 9, HBP now performs better than CDA and, for tight deadlines, also better than HBP-T (with the bid improvement factor 5). CDA is outperformed by PSP when deadlines are tight, and by SJF, when there is background load at the resources (which are operated in screensaver mode).

10.2.2 Soft Deadlines

In this section, we examine three situations with soft deadlines for the tasks. The parameter sl_1 is always set to 1.1. In the first experiment, the resources operate in screensaver mode, and half of the load is background load. The tasks have 'tight' deadlines, and therefore the distribution of sl_2 is given by $[1.1, 1.5]$ ('Soft Deadline I'). The results of the protocols are given in Figure 41. As in the experiments with hard deadlines, PE-A's performance is the best by far. It is followed by PE-P and HBP, which both outperform CDA. SJF and PSP now both lead to poorer results than CDA. The protocol CDA-TDB, which is designed to deal with soft deadlines, performs better than CDA but is still worse than HBP.

For 'moderate' deadlines ('Soft Deadline II'), the results of the protocols improve a bit, however their order does not change (see Figure 93 in the appendix). If deadlines are relaxed even more ('Soft Deadline III', see Figure 94 in the appendix), CDA-TDB provides about the same results as normal CDA, and PSP's performance is not much better than FIFO's. HBP's result will now be slightly worse than CDA's.

Conclusion:

With soft deadlines, similar observations have been made as with hard deadlines. Now, in all examined cases, PSP and SJF perform worse than CDA. CDA-TDB — which has been specifically designed for soft deadlines – leads to improvements over CDA only in situations, where deadlines are tight. The same is the case for HBP, which outperforms CDA-TDB.

10.3 PC Grid

Due to the heterogeneity of resources in the *PC Grid infrastructure*, tasks may be able to execute faster than in the *PC Cluster*. For this reason, we will investigate situations where the deadlines are tighter than before. Concerning *hard* deadlines, we will examine three situations in which tasks have the deadline factors 0.6, 1.1, and 1.5. For the situations with *soft* deadlines, we set the slowdown factor sl_1 to 0.6 [48]. We will study three cases where different uniform distributions are used for sl_2. In the first case with 'tight' deadlines, this distribution is given by $[0.6, 1.1]$ (we refer to this case as 'Soft Deadline IV'). In the second case ('Soft Deadline V') we use $[0.6, 3.0]$, and in the third case ('Soft Deadline VI') $[0.6, 10.0]$.

10.3.1 Variation of Load: Hard Deadlines

In the first experiment, we examine a situation with tight deadlines, i.e. the deadline factor is set to 0.6. The other parameters of the experiment are given by {SP2, C1, SN1, RD2, LV, BG2, TS2, BS1}. This means that we have a system with N_{Serv}= 32 Servers which are highly heterogeneous (RD2) and operate in *screensaver mode* (SP2). In the experiment, the total amount of load in the system is varied (LV), and half of this load is background load (BG2). The results are shown in Figure 42.

For low loads, the best results are achieved by CDA, SJF, FIFO, and PSP, whereas for high loads, PE-A provides the highest WCR, and is followed by PE-P (Figure 42 (left)). SJF performs slightly better than CDA. HBP-T leads to a better result than HBP, CDA, and PSP, as is shown in Figure 42 (right), whereas PDA provides a much poorer performance. We also

[48] Note that, for a deadline factor of less than 0.5, no task would be able to execute on time, even when executed on the fastest resource.

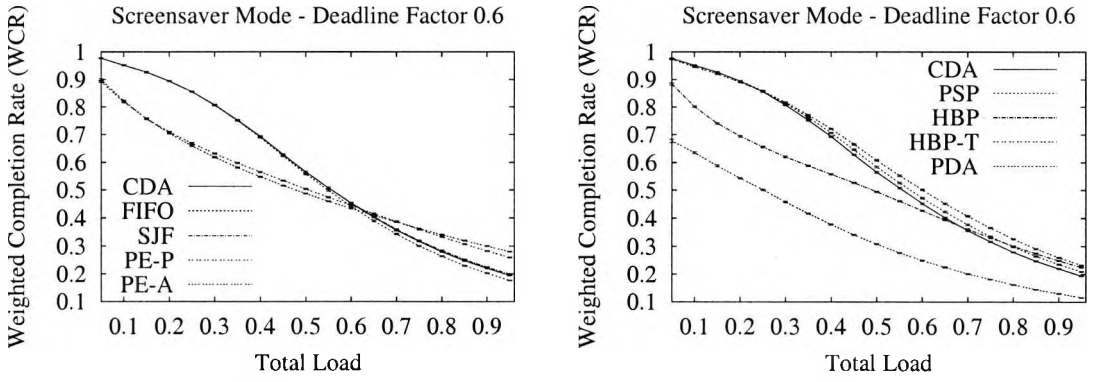

Figure 42: PC Grid: Variation of load, screensaver mode. Tight deadlines (deadline factor 0.6).

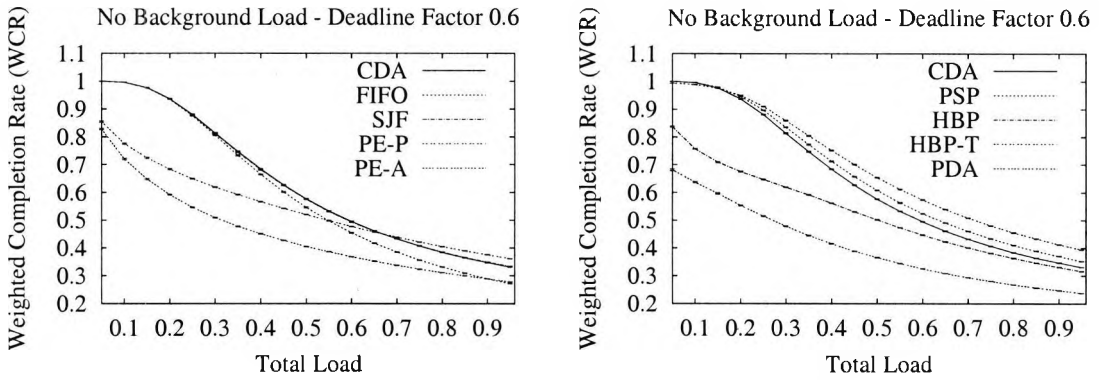

Figure 43: PC Grid: Variation of load, no background load. Tight deadlines (deadline factor 0.6).

examined the performance of the CDA-RES protocol but could not find any improvements in comparison to CDA. This has also been the case for all other situations where the resources operated in *screensaver mode*.

Next, we examine a situation without any background load (Figure 43 (left)). Again, for low load, the preemptive protocols are clearly outperformed by the non-preemptive ones, and PE-A's WCR is now the lowest. Even for very high load, PE-A's results are very poor and only marginally better than FIFO's. At 95% load, PE-P performs better than most protocols, and CDA's and SJF's WCRs are about 8% lower. HBP-T provides even better results, for all amounts of loads (see Figure 43 (right)), whereas those of PDA are the poorest. PSP's WCR at 95% load is 11% lower than HBP-T's, and that of HBP about 20%. The highest WCR for all amounts of load is achieved by CDA-RES. This has been observed for the case that no

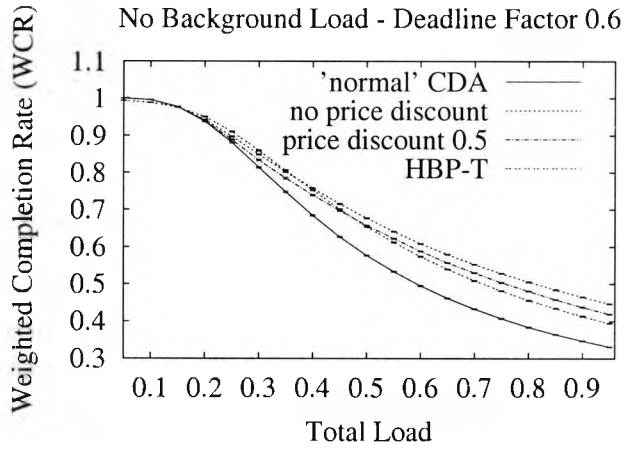

Figure 44: Variation of load, no background load: Results for CDA-RES with different price discounts. Again, tight deadlines are used (deadline factor 0.6).

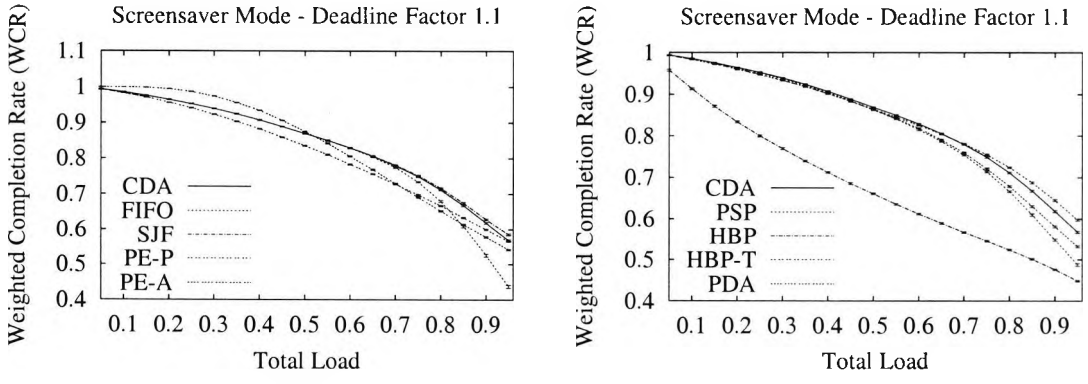

Figure 45: Variation of load, screensaver mode: Moderate deadlines (deadline factor 1.1).

price discounts are used (see Figure 44).

In the next experiment, whose results are shown in Figure 45, we use the same parameters as in Figure 42, except that now the tasks have 'moderate' deadlines (deadline factor 1.1). In contrast to the previous experiments, PE-A performs best when load is low. At 95% load, SJF leads to better results than CDA. PE-P performs about equally well as CDA, and PE-A is slightly worse. As shown in Figure 45 (right), HBP's results are the worst for (almost) all amounts of load. PSP now also perform worse than CDA, and the WCR of HBP is even lower. HBP-T leads to the best results when load is high, whereas PDA still performs worse than CDA.

When the deadlines are relaxed even more (deadline factor 1.5), PE-A performs best for

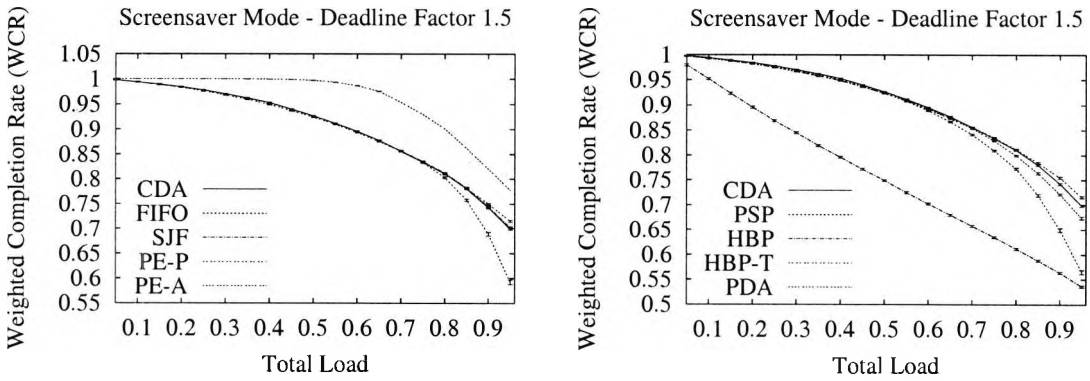

Figure 46: Variation of load, screensaver mode: Loose deadlines (deadline factor 1.5).

all amounts of load (see Figure 46 (left)). PE-P's WCR is much lower than PE-A's, and at 95% load, it outperforms both SJF and CDA. HBP-T's performance (shown in Figure 46 (right)) is comparable to that of PE-P, whereas PSP and HBP now have a lower WCR than FIFO. As before, PDA's WCR is lower than CDA's but the difference is now smaller and only occurs for higher loads.

Conclusion:

In a PC Grid, the choice of protocol will depend not only on the deadline but also on the amount of load in the system. With tight deadlines and low loads, HBP-T is best, and the non-preemptive protocols perform almost equally well. If load is increased, PE-A will perform best, except for a situation where there is no background load in the system, in which case CDA-RES is best. For moderate deadlines, PE-A provides the best results when load is low, whereas HBP-T is best if load is high. With loose deadlines, PE-A's WCR is much higher than for the other protocols, and HBP's WCR is the lowest.

10.3.2 Variation of Load: Soft Deadlines

Now we study situations in which tasks have soft deadlines. The parameter sl_1 is always set to 0.6. In the first experiment, the resources operate in screensaver mode, and half of the load is background load. The tasks have 'tight' deadlines, and therefore the distribution of sl_2 is given by $[0.6, 1.1]$ ('Soft Deadline IV').

PE-A (Figure 47 (left)) provides by far the highest user utility for all amounts of load

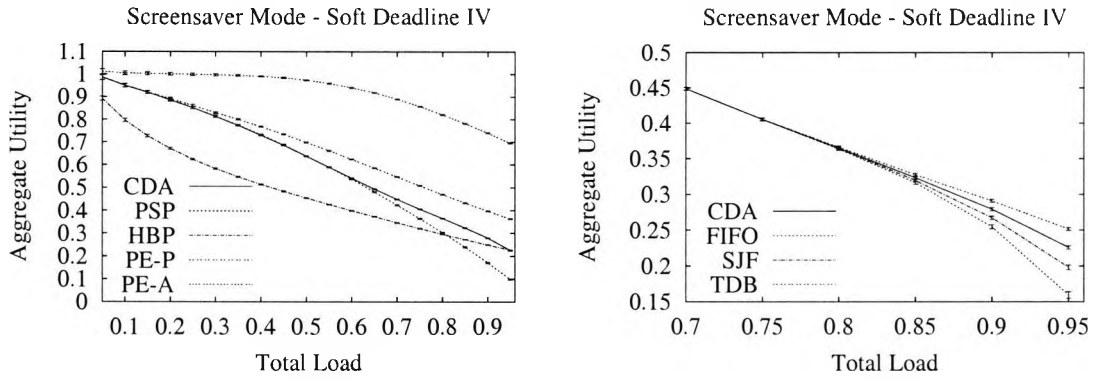

Figure 47: Variation of load, screensaver mode: Soft deadlines IV (tight). Note the different scaling in the right figure.

and is followed by PE-P. For low loads, all other protocols (except HBP) perform about equally well as PE-P, whereas for high loads their results are much worse. The poorest performance for high load has been observed for PSP. As shown in Figure 47 (right), CDA-TDB's performance is better than for all other non-preemptive protocols. However, even at 95% load its user utility is only slightly higher than CDA's. The results of SJF and FIFO are both below CDA's.

Similar observations have been made for the situations with moderate and loose deadlines ('Soft Deadline V' and 'Soft Deadline VI'). The results are given in the Figures 95 and 96 of the appendix.

Conclusion:

In contrast to the situation with hard deadlines, PE-A and PE-P now clearly outperform the other protocols. CDA-TDB does not offer much improvement in comparison to CDA. It performs considerably worse than PE-A and PE-P, even though these protocols do only consider static priorities. The poorest performance has been observed for PSP, HBP, and FIFO.

10.3.3 Different Server Numbers: Hard Deadlines

The aim of this experiment is to see the effect of the Server number on the weighted completion rate. We start with the case that tasks have tight deadlines (i.e. deadline factor = 0.6).

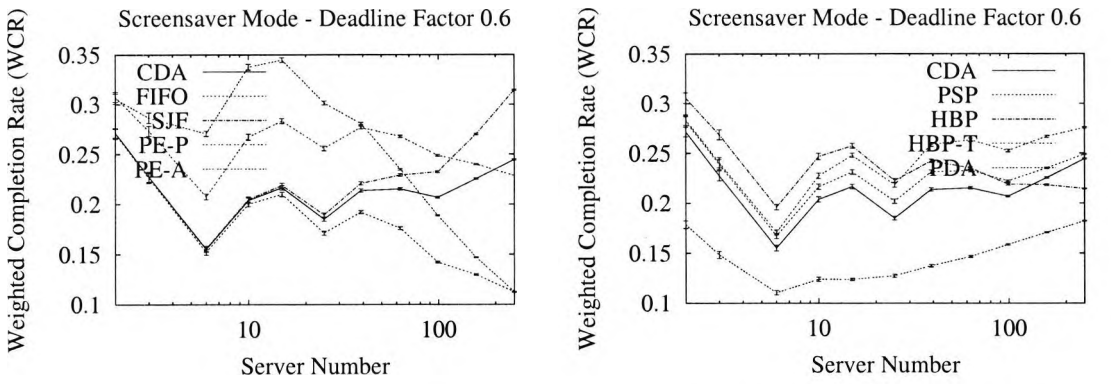

Figure 48: Variation of the Server number, screensaver mode: Tight deadlines (deadline factor 0.6).

We examine the screensaver mode and set the average load in the system to 90%. Hence, the remaining parameters of the experiment are given by {SP2, C1, SNV, RD2, L1, BG2, TS2, BS1}.

The results are shown in Figure 48. We found that, with increased number of resources, the performance of SJF, HBP-T, CDA, PSP, and PDA improves. Interestingly, SJF provides by far the highest WCR when there are 251 Servers in the system. It is followed by HBP-T, PSP, and CDA. The latter two perform about equally well. PDA also improves but still has a lower WCR than CDA. A degradation can be observed for HBP, PE-P, FIFO, and PE-A. For 251 Servers, PE-A's WCR is the lowest.

We made similar observations for a situation where there is no background load in the system. The only exception is CDA-RES, which provides better results than any other protocol. The results are shown in Figure 97 and 98 of the appendix.

We also examined a situation with moderate deadlines (deadline factor 1.1). The results are shown in Figure 99 of the appendix. For a high number of Servers, HBP-T's result is best. It is followed by CDA and SJF. PE-P's WCR remains almost constant as the Server number is increased. The same is the case for PSP — however at a lower level. FIFO improves and approaches the performance of PE-P, whereas for HBP and PE-A, a strong degradation can be observed. In a situation with loose deadlines (deadline factor 1.5, see Figure 49), clearly the best results can be achieved with PE-A. HBP-T and PDA are next, and perform about equally well. For 251 Servers, the protocols CDA, SJF, FIFO, and PE-P all provide about the

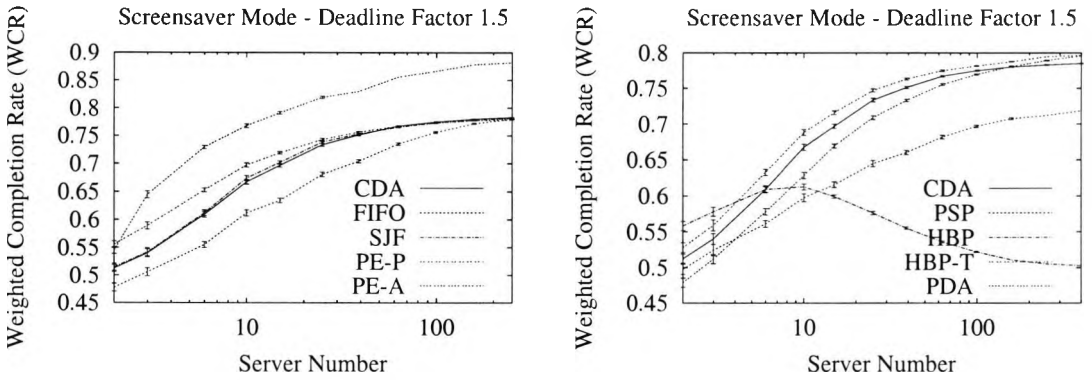

Figure 49: Variation of the Server number, screensaver mode: Loose deadlines (deadline factor 1.5).

same WCR, which is about 1% lower than for HBP-T. Although PSP improves with higher Server number, its WCR still remains considerably lower than CDA's. By far the poorest performance has been observed for HBP — which is also the only protocol, whose WCR decreases for a higher Server number.

Conclusion:

With 90% load in the system and a high number of Servers, SJF and HBP-T provide the best performance, if deadlines are tight. With moderate deadlines, HBP-T and CDA perform best, whereas with loose deadlines, PE-A clearly outperforms the other protocols and is followed by HBP-T and PDA.

10.3.4 Different Server Numbers: Soft Deadlines

Next, we study situations in which tasks have soft deadlines, starting with the case 'Soft Deadline IV' which is examined in Figure 50. With increased number of Servers, the results of all protocols (except HBP) improve. For 251 Servers, PE-A's performance is far better than for the other protocols. It is followed by PE-P whose result is about 45% lower. CDA, FIFO, SJF, and TDB lead to the same user utility, which, for 251 Servers, is 60% lower than for PE-A. The lowest WCRs have been observed for HBP and PSP.

For situations with moderate and loose deadlines, similar observations have been made. The results can be found in Figure 100 and 101 in the appendix.

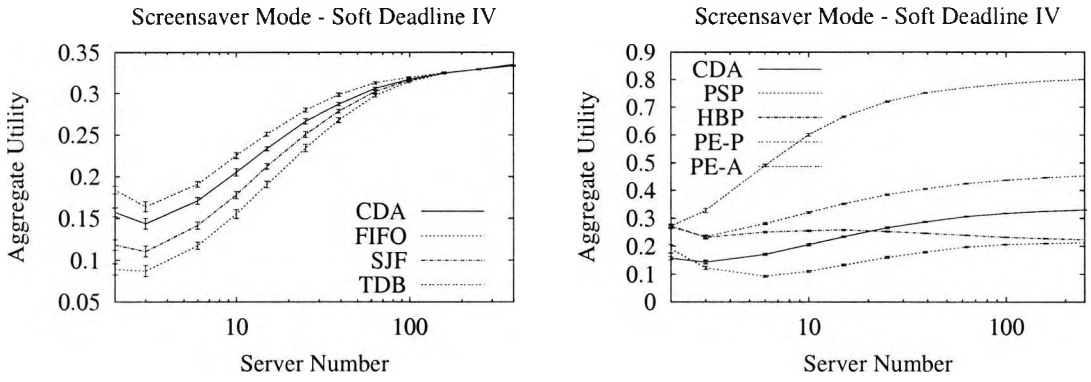

Figure 50: Variation of the Server number, screensaver mode: Soft deadlines IV (tight).

Conclusion:

With soft deadlines and a high number of Servers, similar observations have been made in all the examined situations (i.e. tight, moderate, and loose deadlines): PE-A performs best and is followed by PE-P. The protocols CDA, FIFO, SJF, and TDB are next and lead to about the same performance, whereas the poorest results are observed for PSP and HBP.

10.3.5 Communication Delays

Finally, we investigate situations where the communication delays are varied. We only consider some of the protocols and focus on a situation with hard deadlines. The other simulation parameters are given by {SP2, C2V, SN1, RD2, L1, BG2, TS2, BS1}. This means that there are 32 Servers in the system, and that the average load is set to 90%. Half of this load is background load, and the resources operate in screensaver mode. For the network latency, we use a lognormal distribution of which the mean is varied (C2V).

The results for tight deadlines (deadline factor 0.6) are shown in Figure 51. As expected, the WCR of all protocols decreases to zero. However, the best performance for moderate latencies (about 0.005 to 0.05 time units) can be observed for the PE-P protocol (see Figure 51 (right)). PSP's and SJF's WCRs now decrease faster than CDA's, and the same is the case for PE-A. For a situation without background load, PE-P also leads to the best results (see Figure 102 in the appendix).

When deadlines are relaxed ('moderate deadlines', i.e. deadline factor 1.1), the best re-

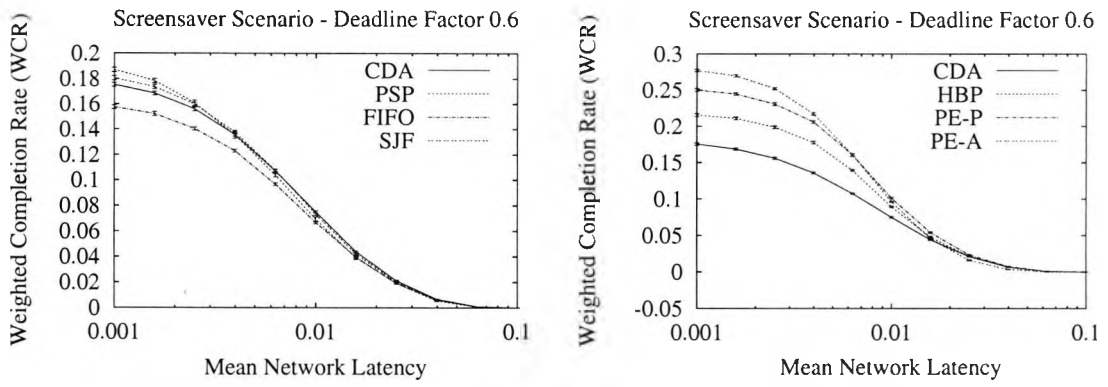

Figure 51: Variation of the communication delay, screensaver mode: Tight deadlines (deadline factor 0.6).

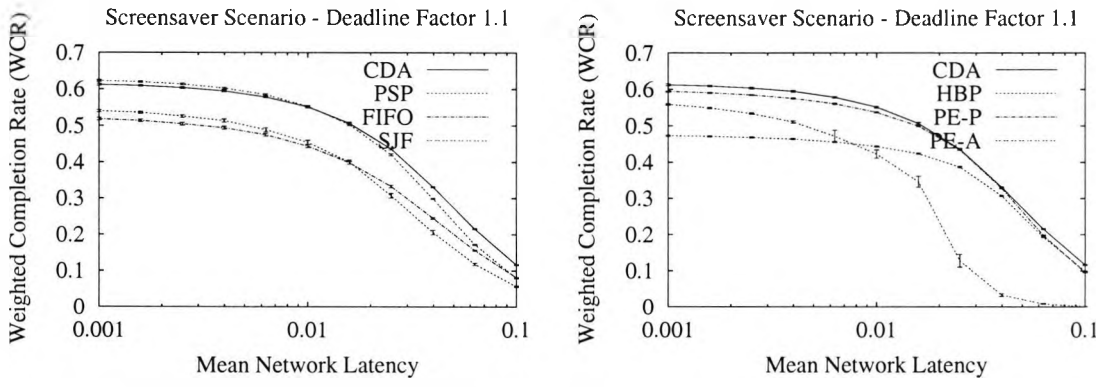

Figure 52: Variation of the communication delay, screensaver mode: Moderate deadlines (deadline factor 1.1).

sults will be obtained with CDA (see Figure 52). The fastest degradation can be observed for PE-A (Figure 52 (right)). The reason why PE-A now degrades faster is that, with moderate deadlines, there are more tasks in the system than with tight deadlines. Hence, there are more (time-consuming) preemptions, and tasks are less likely to complete.

Similar observations have been made for a situation with loose deadlines (see Figure 103 in the appendix).

Conclusion:

We compared several protocols for situations where the communication delays are varied. We found that, for tight deadlines, PE-P performs best, whereas for moderate and loose deadlines, CDA leads to the best results.

10.4 Summary

In this chapter, we investigated the allocation of tasks with time-dependent priorities. As a performance metric, we used the weighted completion rate (WCR) (for hard deadlines) and the aggregate user utility (for soft deadlines). In this section, we give an overview of the parameters used in each experiment (see Figure 53). Also, we summarise our findings for each protocol. Our guidelines for the system designer will be given in chapter 13.

Infrastructure	Figure	Deadline	Parameters
PC Cluster	39	hard / tight	SP2, C1, SN1, RD1, LV, BG2, TS2, BS1
PC Cluster	92	hard / tight	SP2, C1, SN1, RD1, LV, BG0, TS2, BS1
PC Cluster	40	hard / loose	SP2, C1, SN1, RD1, LV, BG2, TS2, BS1
PC Cluster	41	soft / tight	SP2, C1, SN1, RD1, LV, BG2, TS2, BS1
PC Cluster	93	soft / moderate	SP2, C1, SN1, RD1, LV, BG2, TS2, BS1
PC Cluster	94	soft / loose	SP2, C1, SN1, RD1, LV, BG2, TS2, BS1
PC Grid	42	hard / tight	SP2, C1, SN1, RD2, LV, BG2, TS2, BS1
PC Grid	43	hard / tight	SP2, C1, SN1, RD2, LV, BG0, TS2, BS1
PC Grid	44	hard / tight	SP2, C1, SN1, RD2, LV, BG0, TS2, BS1
PC Grid	45	hard / moderate	SP2, C1, SN1, RD2, LV, BG2, TS2, BS1
PC Grid	46	hard / loose	SP2, C1, SN1, RD2, LV, BG2, TS2, BS1
PC Grid	47	soft / tight	SP2, C1, SN1, RD2, LV, BG2, TS2, BS1
PC Grid	95	soft / moderate	SP2, C1, SN1, RD2, LV, BG2, TS2, BS1
PC Grid	96	soft / loose	SP2, C1, SN1, RD2, LV, BG2, TS2, BS1
PC Grid	48	hard / tight	SP2, C1, SNV, RD2, L1, BG2, TS2, BS1
PC Grid	97	hard / tight	SP2, C1, SNV, RD2, L1, BG0, TS2, BS1
PC Grid	98	hard / tight	SP2, C1, SNV, RD2, L1, BG0, TS2, BS1
PC Grid	99	hard / moderate	SP2, C1, SNV, RD2, L1, BG2, TS2, BS1
PC Grid	49	hard / loose	SP2, C1, SNV, RD2, L1, BG2, TS2, BS1
PC Grid	50	soft / tight	SP2, C1, SNV, RD2, L1, BG2, TS2, BS1
PC Grid	100	soft / moderate	SP2, C1, SNV, RD2, L1, BG2, TS2, BS1
PC Grid	101	soft / loose	SP2, C1, SNV, RD2, L1, BG2, TS2, BS1
PC Grid	51	hard / tight	SP2, C2V, SN1, RD2, L1, BG2, TS2, BS1
PC Grid	102	hard / tight	SP2, C2V, SN1, RD2, L1, BG0, TS2, BS1
PC Grid	52	hard / moderate	SP2, C2V, SN1, RD2, L1, BG2, TS2, BS1
PC Grid	103	hard / loose	SP2, C2V, SN1, RD2, L1, BG2, TS2, BS1

Figure 53: Tasks with time-dependent priorities: Overview of experiments.

Continuous Double Auction Protocol (CDA)

In the investigated scenarios, CDA's results are among the best when compared to other non-preemptive protocols. For the PC Cluster, there are usually not many differences between most of the protocols for up to 70% load. In a PC Grid, CDA outperforms the preemptive protocols if deadlines are tight and loads are low. Like in the scenario in chapter 9, CDA

also improves when the number of Servers in the system is increased. When communication delays are increased, the degradation of CDA is similar to that of most other protocols.

CDA with Reserve Prices (CDA-RES)

CDA-RES has only been examined for the PC Grid, as it cannot lead to improvements in the PC Cluster. Again, we found that, when resources operate in screensaver mode, CDA-RES never leads to improvements over CDA. However, for a situation with tight deadlines and without background load, we found that it performs better than any other protocol. Its best performance is achieved if no discounts are used for the reserve prices. A possible reason for this good performance is that, with tight deadlines, only a small fraction of the tasks can be executed on time. Hence, the reserve price helps to ensure that the high priority tasks get executed.

CDA with Time-Dependent Bids (CDA-TDB)

In situations with soft deadlines, CDA-TDB is marginally better than normal CDA. It only makes a difference if load is high and deadlines are tight. We also found that, for a high number of Servers in the system, CDA-TDB's advantage over CDA disappears. This is in line with the observations of [Chun and Culler, 2002], who used a similar technique for the allocation of parallel tasks in computational clusters (see section 5.4): These indicate that large improvements can only be achieved for highly parallel load, and not for sequential load. However, for our scenario, we could not confirm the authors' finding that 'preemption does not add significant value', as CDA-TDB is clearly outperformed by the preemptive protocols.

Proportional Share Protocol (PSP)

In a PC Cluster, PSP now outperforms CDA, provided that the tasks have hard and tight deadlines. The same is the case for the PC Grid, where the difference to CDA increases with the load. This observation, however, has not been made for situations with soft deadlines, in which PSP led to a poor performance. The reason could be that the time-dependent values of the tasks are not considered by PSP: Hence, it will allocate tasks whose values have considerably decreased — and will delay tasks whose values are still high. PSP improves

when the number of Servers is increased — but not as much as CDA which, in most cases, performs better. When communication delays are introduced, PSP's performance is similar to CDA's — which is different from the observations made in chapter 9.

Highest Bid Protocol (HBP)

In contrast to the experiments in chapter 9, HBP's performance is now comparable to that of the other protocols. In the PC Cluster, it's results are better than CDA's, especially for tight deadlines. For the PC Grid infrastructure, HBP performs better than CDA only for hard and tight deadlines, and only if there is background load in the system. A very strong degradation of HBP has been observed for loose deadlines: Again, as in chapter 9, the delay of the suspended tasks outweighs the gain of the high priority tasks. Unlike most other protocols, HBP degrades when the number of Servers in the system is increased. This is caused by the increased number of preemptions.

HBP with Threshold (HBP-T)

HPB-T is always among the best protocols. It outperforms both, CDA and HBP, but, in most situations, its results are not as good as PE-A's. In the PC Grid, however, HBP-T may even outperform PE-A, provided that deadlines are tight and load is moderate. Unlike the HBP protocol, HBP-T improves for a high number of Servers.

Preemptive Protocol (PE)

In situations with background load on the resources, the protocols PE-P and PE-A usually lead to the best results. In a PC Cluster, PE-A's performance is best by far, and PE-P is better than all the other protocols. The same is the case for a PC Grid — if the tasks have soft deadlines. For hard and loose deadlines, PE-A outperforms all other protocols, whereas for tight or moderate deadlines, results are mixed. If communication delays are introduced and task deadlines are tight, PE-P performs best, and PE-A's WCR is only slightly lower. However, in situations with moderate and loose deadlines, a strong degradation of PE-A can be observed.

Periodic Double Auction Protocol (PDA)

In most examined situations, PDA's results are worse than CDA's, and the differences are

largest if deadlines are tight. The reason for this poor performance is the delay between the transactions at the Electronic Marketplace (EMP). The only exception is a situation where deadlines are loose and the number of Servers is high.

First In First Out (FIFO)

As in the experiments in the previous chapter, FIFO's performance only matches that of CDA, if loads are low or moderate, i.e. when there is little competition among the tasks. With increased number of Servers, FIFO degrades, if deadlines are tight, but improves if deadlines are loose. In the latter case it will perform equally well as CDA.

Shortest Job First (SJF)

In most of the experiments, SJF's performance is below CDA's. Only for hard and tight deadlines, SJF's results are better. However, this observation has only been made for situations where there is background load on the resources (which operate in screensaver mode). For hard and tight deadlines, SJF's result also improves faster than CDA's, when the number of Servers in the system is increased. The reason, why SJF performs better for hard than for soft deadlines, is probably that the performance metric used in the latter case considers the task sizes in the weighting.

11 Experimental Grid Computing Framework

11.1 Introduction

So far, we have compared the performance of various market protocols via discrete-event simulation. Yet it remains to be shown that the marketplace and the protocols of the simulation model are implementable and able to operate in a real computational environment. Also, we need to determine whether the assumptions we have made about communication delays, processing delays, etc. are valid under realistic conditions. This chapter describes our basic Grid computing framework which we use for the validation of our simulation results. The framework has been developed as part of the AgentCities deployment grant *CoMAS* [49], and has also been used for the distributed computation of the PSIMAP application [Dafas *et al.*, 2003a]. It is based on the agent platform JADE [Bellifemine *et al.*, 1999] and is an almost exact implementation of the simulation model. JADE has been chosen, because it gave us more flexibility in implementing our architecture than dedicated Grid Computing frameworks such as the *Globus Toolkit* [Foster and Kesselman, 1997]. The implementation in JADE also required less effort than standard technologies like RMI and JINI, since it supports agent behaviours, asynchronous messaging, and multiple communication protocols.

11.2 Objectives

Overall, with our Grid computing framework we plan to achieve the following objectives:

Proof of concept: We want to demonstrate that the simulated system is realistic by implementing it on the Java-based agent platform JADE. Issues to be resolved include the performance and load measurement at the resources, the passing of data and code between machines in a geographically distributed environment, and the specification of task and resource constraints.

[49] Control and Management of Agents and their Services, iD: ACNET.02.30.

Deployment: We want to show that the framework can be deployed on a PC cluster in a local area network. Our experiments will be limited to a cluster scenario, i.e. large communication delays — as they occur in globally distributed networks — will not be studied. They have only been examined in the simulations.

Verification of simulation results: We aim to determine in how far the simulation results are valid. In particular, we want to find out, for which loads and numbers of resources the system behaves as in the simulations, and what the impacts of processing and communication delays are.

Applications: We want to show that real-world problems can be solved by this experimental framework. We will run a bioinformatics computation called PSIMAP. It is an example of a parameter sweep application which is submitted to the system as a burst of independent computations. Note that our Electronic Marketplace is a multi-user system where each user may submit such a computation.

11.3 General Description

The architecture and the interaction protocols of our framework are almost the same as in the simulation model which we described in chapter 6.

As shown in Figure 54, it consists of Clients, who want to execute tasks on resources, Servers, who provide these resources, and an Electronic Marketplace (EMP). The Electronic Marketplace allows Servers to advertise the resources and Clients to query them.

Whenever a Client sends a request to the EMP, it needs to specify the task's size, price bid, and constraints such as the task deadline, minimum resource speed, etc. At the EMP, the requests of the Clients are matched with the Servers' offers which contain the resource's speed, price, and current availability (in % of the total CPU capacity).

The resource's speed is given in MFLOPS, i.e. in millions of floating point operations per second. This figure is based on the execution of a sparse-matrix benchmark (see section D.3 in the appendix). Similarly, a task's computational size is specified in megaflops *times* milliseconds (MFLOPS*ms). Using these measures enables us to obtain a close estimate of

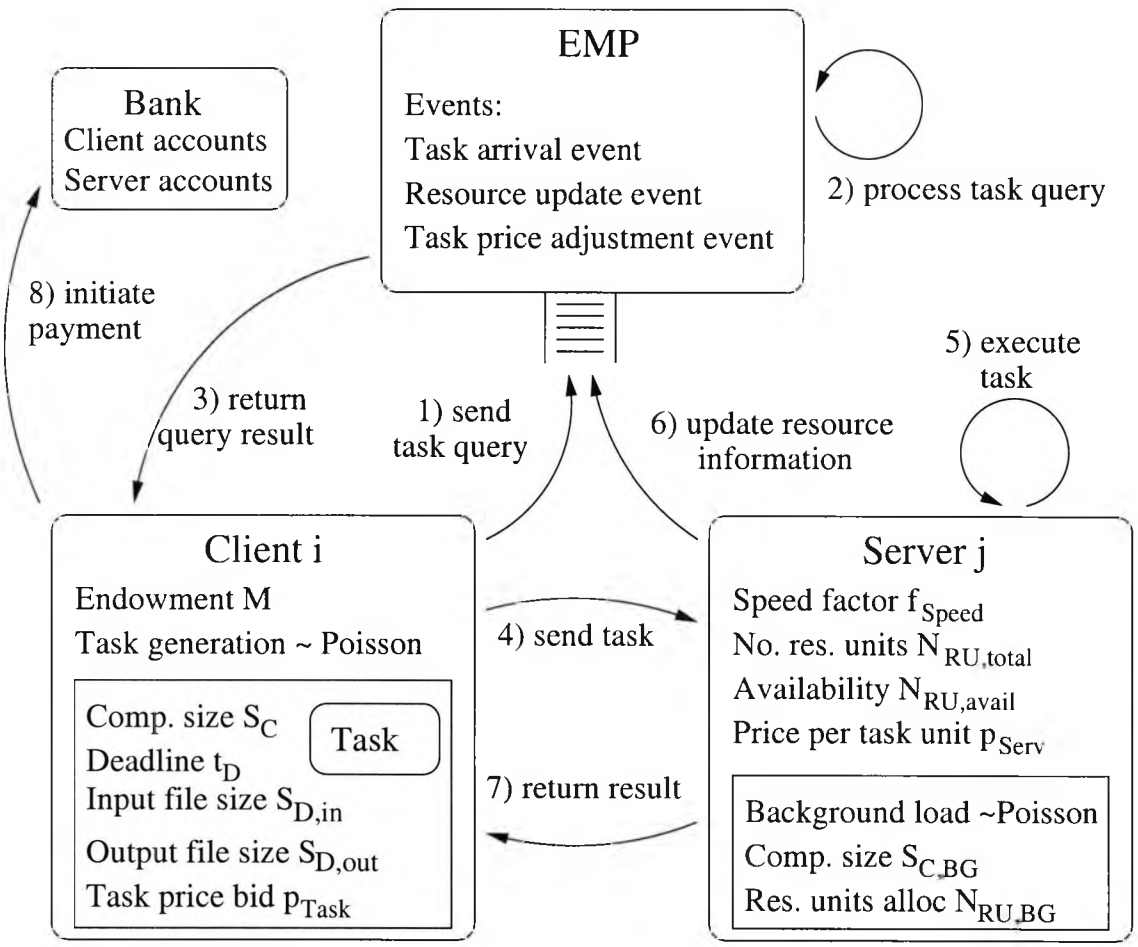

Figure 54: Model of the marketplace which has been implemented in our Grid computing framework.

how long a task will execute on a given resource.

Different protocols for the resource allocation have been implemented. These include the Continuous Double Auction Protocol (CDA), Continuous Double Auction with time-dependent price bids (CDA-TDB), Shortest Job First (SJF), First-In-First-Out (FIFO), and Prioritised First-In-First-Out (PRIO-FIFO).

11.4 Implementation

This section gives a brief overview of the most important features of our framework. More information about the implementation and the operation of the framework is given in chapter D of the appendix.

11.4.1 Communication

Within the system, we use RMI-style communication, i.e. the actors communicate via messages which contain a string of characters and a set of objects. The string represents a method which is to be invoked on the receiving agent, and the objects are the arguments to be passed to that method. The Java Reflection API is used for translating the string into the agent's method. Thus, a message handler is not needed.

This messaging facility is built on top of JADE and therefore allows to pass messages by using multiple communication protocols including RMI, Corba, HTTP, and JMS. This facilitates the communication between actors at geographically distributed locations.

11.4.2 Tasks

In our framework, a task is specified as a set of Java classes, which can be sent out from a Client for execution on a remote Server. The most important features of this framework, which concern the execution of tasks, are described below:

Interfaces for computational tasks. In order to deploy a computational task in our framework, the task needs to implement an interface which is described in section D.4. If a larger application is to be split up into smaller tasks which will be distributed and executed remotely, it also needs to implement an interface which is given in section D.5.

Passing input and output data. The input data is wrapped up in a serializable Java object so that it can be transmitted inside a FIPA message over any communication protocol. The same applies to the result data that is returned after the execution. To allow the transmission of code, Java classes (i.e. their bytecode) can be loaded over any protocol supported by JADE.

Specification of task and resource constraints. Our framework provides means for the Clients and Servers to specify constraints, which need to be met when matching tasks to resource offers. This feature is described in section D.6 of the appendix.

Class loading mechanism. A class loader has been implemented which can pass Java classes from a Client to a remote Server in order to execute them there. The Java classes are specified by their file names and paths on the Client's harddisk. Unless these classes are already available at the Server, their bytecode is read from these files, serialised, and sent to the Server. The bytecode is transmitted inside ordinary FIPA messages which JADE can send via RMI, Corba, HTTP, or JMS. At the Server, it is loaded, instantiated and executed.

11.4.3 Servers

The Servers in our system have been provided with the following facilities:

Measurement of the resource speed. When Servers register their resources at the EMP, they need to provide information about the resource's speed. To achieve this, our framework runs a benchmark at the Server resources which return their speed in MFLOPS. More information about this benchmark is given in section D.3 of the appendix.

Measurement of the resource utilisation. To obtain the current load at the Server resources, the Unix command *vmstat* is used which monitors the resource at regular time intervals. This method requires a Unix or GNU/Linux operating system — it is not available for Windows.

Resource scheduling policy. The Servers in our framework support the second of the scheduling policies which are described in subsection 6.2.3: It is assumed that the resource is time-shared, and that background load is given priority: the latter is achieved by executing our platform and our tasks with the lowest available priority, which can be expressed by using the `nice` command under Unix. The resource share, that is available to the framework, is allocated *exclusively* to one executing task, which cannot be suspended or preempted. Protocols which use proportional sharing, task suspension, or preemption are too hard to implement — unless modifications are made to the operating system (GNU/Linux) or the Java virtual machine [50].

[50] We used JDK 1.4 in our experiments. Unfortunately, Java up to version 1.5 does not have the capability to *safely* suspend and resume threads from outside. To our knowledge, this problem does not exist in the .NET framework [Platt and Ballinger, 2002]. Also, efforts are underway to provide resource control for Java [Czajkowski *et al.*, 2003].

11.5 Summary

In this chapter we have given an overview of our experimental Grid computing framework which serves as a proof-of-concept for our simulation model. We described our approach to resolve issues such as performance and load measurement at the resources, the passing of data and code between the machines, and the specification of the task and resource constraints. More details about the implementation and operation of this framework are given in chapter D of the appendix.

In the next chapter we will deploy our framework in a cluster of PCs in order to verify our simulation results. We will also show how it can be used for the execution of a parameter sweep application taken from bioinformatics.

12 Experiments

12.1 Objectives

In this chapter, we will show that the simulated system can operate in a real computational environment. To this end, we will deploy our experimental Grid computing framework, which is based on our simulation model, in a cluster of PCs in a local area network.

We will determine in how far the real system behaves as we observed in the simulations. In our experiments, we will study the impact of communication and processing delays on the performance of the Electronic Marketplace (EMP). We will also examine situations, in which the influence of the communication and processing delays is negligible. Comparing the results of these experiments to those obtained by our simulations should show whether our simulation model is correct. Finally, we will demonstrate the effectiveness of our framework for solving real-world problems by deploying a computationally intensive bioinformatics application.

12.2 Experimental Setup

In this section, we will first introduce the hardware and software infrastructure that is used in our experiments. Next, we will describe how performance measurements at the machines are carried out, and how load is generated by the Clients. Finally, we specify the parameters which are common to all our experiments.

12.2.1 Hardware and Software Infrastructure

The Client and the EMP are run on a dual-processor Pentium III (1GHz, 512 MB RAM) machine using GNU/Linux, Sun's JDK 1.4.2, and JADE version 2.6 [Bellifemine *et al.*, 1999]. The Servers are started up on separate 700 MHz Pentium III machines (512 MB RAM), which are distributed over several student labs of the university. Those machines use GNU/Linux, Blackdown JDK 1.4.1 and JADE version 2.6.

12.2.2 Performance Measurements and Load Generation

The speed of each machine is measured by running a benchmark which determines its performance in MFLOPS (see section D.3). This MFLOPS figure is based on a sparse-matrix multiplication taken from the SciMark2 benchmark [Pozo and Miller, accessed in 2003]. For the machines which were used in this experiment, an average performance of 64.9 MFLOPS has been determined. The standard deviation of these measurements has been between 0.1 and 0.3 MFLOPS. Since the hardware architecture of the machines is identical, it is probably caused by measurement inaccuracies or background load.

The load generated by the Clients consists of tasks which execute the same sparse matrix multiplication that has been used by the benchmark. The computation size of each task can be specified by passing its MFLOPS*ms value as parameter [51]. Hence, it is possible to determine how long a task will execute on a machine whose performance is known.

12.2.3 General Experimental Parameters

The total length of each experiment is set to 4400 seconds. During this time, tasks are generated by the Client. No measurements are made in the first 400 seconds: This is to ensure that the system reaches a steady state. After this initial period, the number of tasks which are statistically expected to be generated during an interval of 3600 seconds (= 1 hour) is considered in the result. To allow these tasks to complete, an additional final margin of 400 seconds is provided. For these experiments, it does not matter how many Clients there are in the system. Therefore, we use only one Client for the generation of tasks. During the experiment, each Server measures, at 10 second intervals, the current availability of its machine (in %). If it has significantly changed since the last measurement, this information is updated at the EMP [52].

To achieve the objectives stated in section 12.1, it is not necessary to examine all protocols and scenarios. We examine only the CDA protocol, and only for the scenario where

[51] The code of the tasks has been calibrated by executing it on a machine, whose MFLOPS figure has previously been measured by the benchmark.

[52] Note that this is not supposed to happen because our experiments are carried out at times when the machines are not used by the students. However, we found that, even if a machine is used, the background load is likely to be negligible.

tasks have different priorities. We have chosen CDA because it is the most studied protocol in our simulations. A detailed description of how the experiment is carried out is given in section D.8 of the appendix.

12.3 Results

In this section, the results of our experiments will be discussed. We will vary both the generated load and the number of resources in the system.

12.3.1 Variation of Load — 10 Servers

With the terminology introduced in chapter 7, our first experiment can be defined by the parameters {T2, SP1, Cx, (10 Servers), RD1, LV, BG0, TS1, BS1}. Note that the communication delays are beyond our control. Tasks have different weights, for which we use a uniform distribution $[0.0, 2.0]$ (T2), and their price bids are proportional to these weights. As performance metric, we use the *weighted completion time (WCT)*, which is defined as the mean of the completion times of the tasks multiplied by their weights. We use 10 identical Servers (RD1) without background load (BG0). The Server performance of 64.9 MFLOPS is represented by the resource size $N_{RU,total} = 100$ and speed factor $f_{Speed} = 0.649$ [53]. All tasks have the same computational size (TS1), and the task burst size is set to 1 (BS1). In the experiment, the total amount of load in the system is varied (LV).

The task size is a critical parameter in the experiment, as it determines how sensitive the results are to communication and processing delays. In order to be able to compare our results to those obtained in the simulations, we express the duration of the execution of the task in *time units*, whose size needs to be defined. In our experiments, the task size is chosen such that the task will need 1 time unit for its execution on a resource in the system. Hence, the task size is calculated by multiplying the size of a task unit by the Server performance (64.9 MFLOPS). Since the task size is proportional to the size of a time unit, the latter will have impact on the result. For this reason, we use different sizes for the time unit in our

[53] Note that the values for the resource sizes and speed factors are different from those used in the simulations. However, they are chosen in such proportions that they lead to the same results.

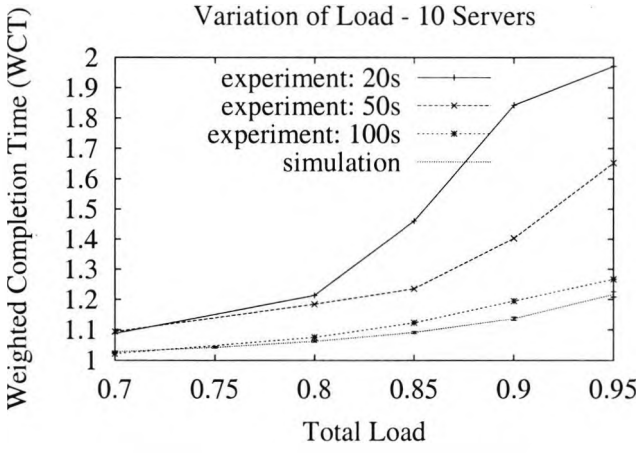

Figure 55: Variation of load for 10 Servers: Results of CDA for different sizes of a task unit.

experiment: 20s, 50s, and 100s.

The results of the first experiment for different values of a time unit are shown in Figure 55, where they are compared to the results of the simulation. For 70% load, the results of the experiment with 100s time units are about the same as those obtained in the simulations. For time units of 20s or 50s, the WCT is about 6 % higher. The gap between the experimental results and the simulation results becomes wider with increased load: At 95% load, time units of 100s lead to a 4% higher WCT than in the simulations. For 50s time units, it is 36% higher, and for 20s time units even 62% higher. We found that, with time units of 20s or 50s, not all tasks are able to complete when load is high. With 50s time units, this has been observed for 95% load, and with 20s time units, already for 90% load.

A possible reason for the differences between the simulation results and the experiments could be the fact that, in our experiments, we used smaller samples than in the simulations, and that no confidence interval has been given for the results. To rule this out, we ran simulations with exactly the same random seed and samples sizes as in the corresponding experiments. The results of the experiments and the simulations — both for different sizes of a task unit — are given in Figure 56. The figure shows that there are considerable differences between the simulations with different sizes for a task unit and the properly conducted simulations, for which a confidence interval is given (*'simulation: ideal'*). However, in all cases, the WCT in the experiments is much higher than in the corresponding simulations. Hence,

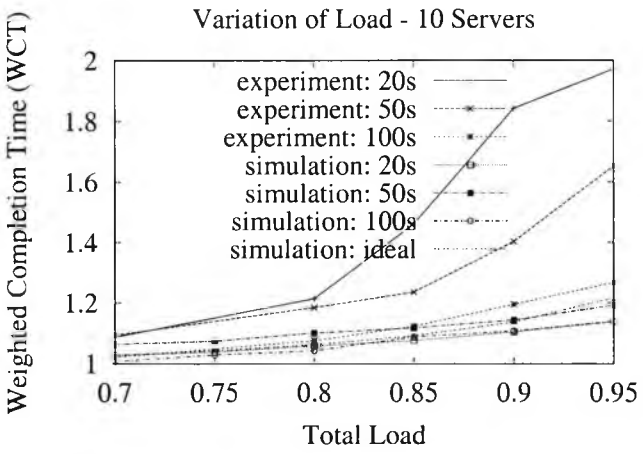

Figure 56: Variation of load for 10 Servers: Comparing the experimental results to simulation results with the same random seed and sample sizes.

the limited number of samples in our experiment is not responsible for the large differences between simulations and experiments.

Conclusion: For all amounts of load, the WCT in the experiments is higher than in the simulations, and the gap becomes wider with increased load. We ruled out that the limited number of samples in the experiment is responsible for the large difference between the experiments and the simulations. We found that, the smaller the task size, the larger is the WCT. It also appears that, for high loads and small task sizes, the system is not able to cope with the load — even though it is still below 100%. This scarcity of resources could be caused by delays introduced by the processing of tasks at the Electronic Marketplace.

12.3.2 Variation of Load — 32 Servers

In this experiment, we use the same parameters as before, except that, now, there are 32 Servers in the system. The results are shown in Figure 57. At 70% load, the results for all time unit sizes match the simulation results. If the load is increased to 85%, a considerable difference can be observed for the experiment with 20s time units, where the WCT is about 8.5% higher than in the simulations. For 50s time units, the WCT is only 0.5% higher, and for 20s even 0.5% lower than in the simulations. The latter could be caused by measurement inaccuracies or the limited number of samples in the experiments.

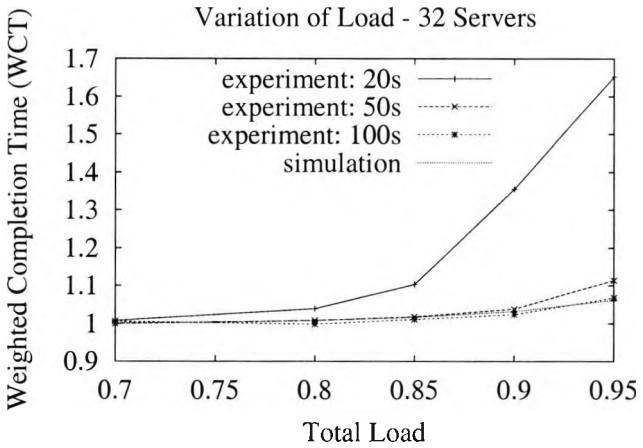

Figure 57: Variation of load for 32 Servers: Results of CDA for different sizes of a task unit.

At 95% load, the results for 50s time units are 5% higher and for 20s time units even 5% higher. However, for 100s time units, a difference of only 0.5% has been observed. As in the experiment with 10 Servers, not all tasks were able to complete for the time unit sizes 20s and 50s. This observation could be made at the same load levels as before.

Conclusion: For 32 Servers in the system, similar observations have been made as in the experiment with 10 Servers. However, this time the differences between simulation and experiments are smaller.

12.3.3 Variation of the Number of Servers — 80% Load

In the next experiment, we set the average load in the system to 80% and vary the number of Servers. This experiment can be represented by the parameter set {T2, SP1, Cx, SNV , RD1, (80% load), BG0, TS1, BS1}. The results are shown in Figure 58.

For a low Server number, the weighted completion times are higher than for a high Server number — in both the simulations and the experiments. However, large differences can be observed between the results with different sizes of a time unit: With 5 Servers and 20s time units, the WCT is about 50% higher than in the simulations, and for 50s time units still 10%. For task unit size 100s, however, the results are very close to those observed in the simulations, and this difference remains small for all examined Server numbers. With 100s

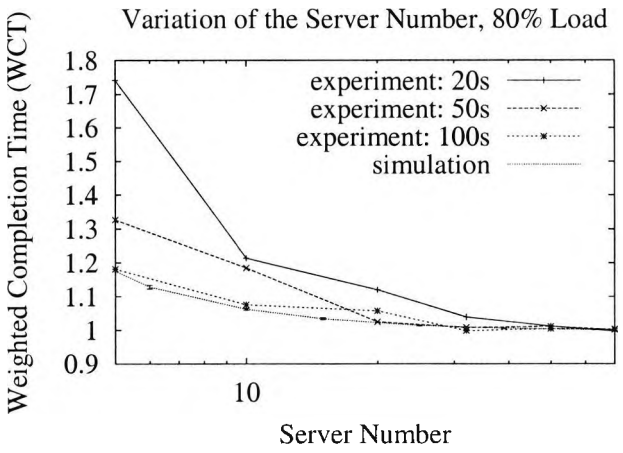

Figure 58: Variation of the Server number for 80% load in the system: Results of CDA for different sizes of a task unit.

time units, the largest difference has been observed for 20 Servers, where the WCT is only about 4% higher than in the simulations. The differences between all time unit sizes decrease as the Server number is increased, and for 70 Servers an almost exact match can be observed.

In this experiment all the measured tasks were able to complete their execution. The reason is that, at 80% load, a scarcity of resources was less likely to occur than for higher loads.

Conclusion:

With 80% load in the system and a low Server number, the differences between the results for the different task sizes and the simulation results are high, whereas for a high Server number, there is almost no difference at all. It appears that, for this 'moderate' load of 80%, our system scales well when the number of Servers is increased — even for small task sizes and hence a large number of queries to the EMP.

12.3.4 Variation of the Number of Servers — 90% Load

Next, we varied the number of Servers for the case that the average load in the system is set to 90% (see Figure 59): With 5 Servers, differences are now larger than in the previous experiment: For 20s time units, the WCT is 90% higher than in the simulations, and for 50s time units it is about 30% higher. For task unit size 100s, the WCT is still 5.3% higher.

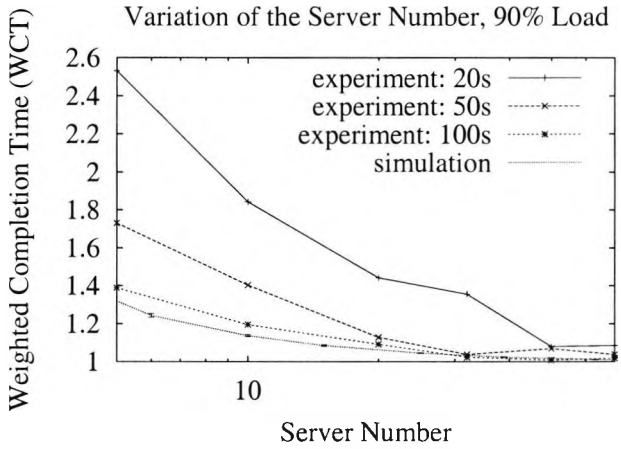

Figure 59: Variation of the Server number for 90% load in the system. Results of CDA for different sizes of a task unit.

As previously observed, the differences become smaller, when the number of Servers is increased to 70: With 20s time units, the WCT is about 7% higher than in the simulations, and for 50s time units it is still 3% higher. However, for 100s time units, there is less than a 1% difference to the simulations.

The load in the system has been high in this experiment. Therefore, in some measurements, not all tasks were able to complete. For a time unit size of 50s, this was the case when the Server number was set to 5. When the size of a task unit was set to 20s, all tasks completed only for the measurement with 50 Servers in the system.

Conclusion: For 90% load in the system, similar observations have been made as for 80% load, except that now there are larger differences between the results for different task sizes and the simulation results.

12.4 Deployment of a Bioinformatics Application

So far, we have shown that our simulated system is implementable and can operate in a real computational environment. We also investigated the validity of our simulation model and the scalability of our Electronic Marketplace (EMP). Next, we will demonstrate the effectiveness of our framework for solving real-world problems by deploying a computationally

intensive bioinformatics application called PSIMAP on our cluster of Linux PCs.

12.4.1 The PSIMAP Computation

PSIMAP [Park *et al.*, 2001; Dafas *et al.*, 2003a] is a bioinformatics application which is written in Java. Its aim is to determine the physical interactions between protein domains, which are fundamental to the workings of a cell. To study the large-scale patterns and evolution of the interactions, Park et al. view protein interactions in terms of whole protein families that interact with each other. In the computation of PSIMAP, a protein structure interaction map is derived from known structures which are obtained from a protein database. Running the full computation on one single machine would take several months. However, the application consists of repeated computations of the same algorithm with different parameter sets and is therefore a *parameter sweep application* (see section 2.1). It can easily be partitioned into independent subproblems that do not require any communication and can therefore be distributed over a loosely coupled network of computers. In [Dafas *et al.*, 2003a], several efficient algorithms are described which can, depending on the level of the required accuracy, reduce the computation time on a single machine to several weeks (days). By distributing it on a cluster of PCs, it can be further reduced to hours (minutes). The latter will be described in this section.

12.4.2 Distributing the Computation

To run the PSIMAP computation, we use one Client which submits all tasks of the computation to the Electronic Marketplace (EMP). At the EMP, we use the Continuous Double Auction Protocol (CDA) for the allocation of the resources to these tasks. Initially, the parameter space of the PSIMAP computation needs to be partitioned into separate parameter sets, each of which is wrapped up into a task which can be computed independently. Given a number c of machines, we split the parameter space into $n \geq c$ sets which are (almost) equal in terms of estimated execution times. c sets will execute in parallel at any time, while the remaining sets have to wait at the EMP until machines become available. If a small value is chosen for n, the load may not be evenly balanced, leading to poor performance. The reason

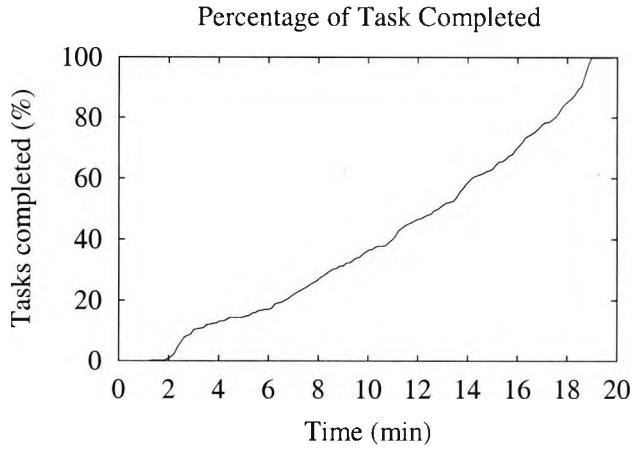

Figure 60: 'Short' PSIMAP computation: Percentage of completed tasks over time. The CDA protocol has been used for the resource allocations.

for this is that the estimates are not very accurate, so that the variation of execution times of the sets can be large. To this end, we choose $n >> c$, as it achieves a better load balancing, and thus a shorter overall computation time. Also, we give higher priorities to parameter sets with large estimated execution times: The weights of the tasks submitted to the EMP (and thus their price bids) are proportional to their sizes. This results in overlapping long computations with short ones, leading to a further reduction of the overall computation time.

12.4.3 Experimental Results

In the first experiment, we run the 'short' PSIMAP computation in which only the protein interactions at domain-to-domain level — rather than atom-to-atom level — are determined. The whole input data for the computation consists of 8800 parameter sets which are split up into 2000 tasks of about the same size. 76 machines are used for the experiment, in which we measure the percentage of completed tasks over time. Its results are given in Figure 60. The diagram shows that the computation is completed after 19 minutes. Compared to the computation on a single PC, which takes 4–5 hours, a speedup of 20 has been achieved. The reason for this low speedup is that the tasks concurrently read information from the database, which leads to delays and idleness of the machines.

Next, we repeat the experiment with the same parameters, except that we now deploy the

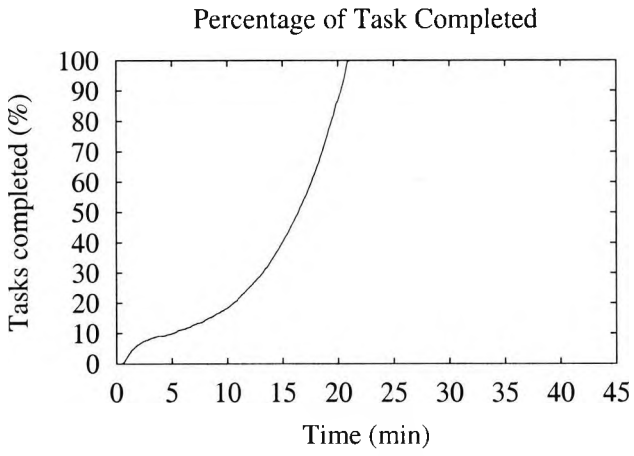

Figure 61: 'Long' PSIMAP computation: Percentage of completed tasks over time. The CDA protocol has been used for the resource allocations.

'long', more accurate PSIMAP computation where the interactions are determined at atom-to-atom level. 72 machines are used in the experiment. The results in Figure 61 show that the computation of tasks has finished after 20 minutes. The slope of the graph increases over time. This is caused by the resource allocation protocol which prioritises long-running tasks and allocates the shorter tasks later. The reason why the computation takes only marginally longer than in the previous experiment is the larger problem size, which leads to a better load-balancing and utilisation of the machines. In comparison to the computation on a single machine, which takes about 20 hours, a speedup of 60 has been achieved.

In the final experiment, we do not only deploy the long version of the PSIMAP application but also write the results into a database. The database access can lead to delays if several tasks perform the access concurrently — and will result in an idleness of the machines. To deal with this problem, each task executes multiple threads (3), and thus allows the interleaving of computation and communication. The result of the experiment, for which 68 machines have been used, is given in Figure 62. It shows that the whole computation completes after about 900 minutes (15 hours). The long duration of the computation is caused by a time-consuming operation on hash tables, which is performed when writing to the database.

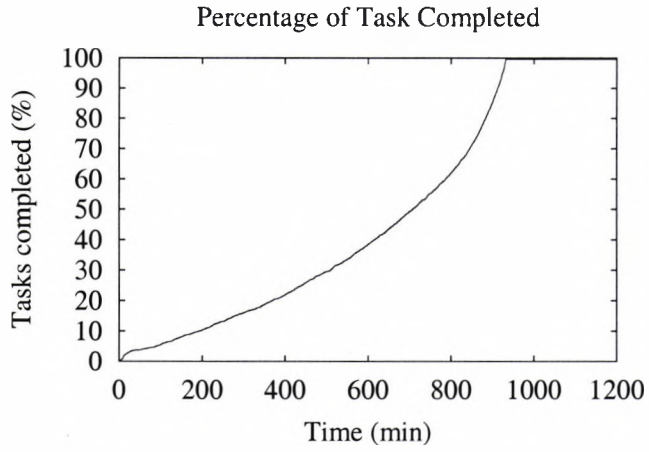

Figure 62: 'Long' PSIMAP computation where the results are written to the database: Percentage of completed tasks over time. The CDA protocol has been used for the resource allocations.

Conclusion:

The distributed computation of the PSIMAP application demonstrated the effectiveness of our framework for solving real-world problems. The speedup of the computation can be almost linear, provided that the application itself is scalable and not compromised by communication delays.

12.5 Discussion

To validate our simulation model, we carried out experiments with different tasks sizes, loads, and numbers of resources in the system. Although our experimental setup did not allow us to examine situations with heterogeneous resources, background load, or communication delays, we were still able to demonstrate that the system is implementable and will operate as predicted by the simulations. We found that, for large task sizes, the experimental results are very close to the simulation results. When smaller tasks are deployed, a higher relative delay of the tasks can be observed.

On the one hand, the differences between simulation and experiments become larger when the load in the system is increased. At a certain point, which depends on the size of the tasks, the system is no longer able to cope with the load. In spite of the amount of generated

load being less than 100%, there appears to be a scarcity of resources in the system. It could be caused by delays introduced by the processing of tasks which are waiting at the Electronic Marketplace.

On the other hand, the differences between simulation and experiments become smaller as the number of resources in the system is increased. The system scales — as long as there are enough resources available to serve the tasks. This is even the case for the experiment in which the task sizes are small, and therefore the number of queries to the Electronic Marketplace is high.

The differences between simulations and experiments are not caused by the *communication delays* in the system: In an experimental setup, we measured the message delay between two Jade platforms located at the machines of the EMP and of a Server. The average delay was about 12ms, i.e. 0.06% of the execution time of the smallest tasks in our experiments (20s). It can therefore be neglected. Hence, the differences between the simulations and the experiments are likely to be caused by the *processing delays* at the EMP.

We must note that, in our experiments, there are several factors which could lead to inaccuracies of the results. These include possible variations of the performance or load of the machines or network connections. Also, there could be errors in the measurements of Server speed or in setting the size of the tasks. As a result, the computations of the tasks could take longer or shorter than required and affect the outcome of the experiment. However, there is a close match between experiments and simulations in situations where we would intuitively expect it (i.e. for low load and a high number of resources). This shows that these errors cannot be very large.

By carrying out experiments with the PSIMAP application we also demonstrated the effectiveness of our framework for solving real-world problems. However, we must note that the PSIMAP application does not exploit the full potential of our Electronic Marketplace, because all its tasks belong to the same user and are all submitted at the beginning of the experiment. The allocation of computational resources is therefore relatively straightforward. This is usually not the case in *Computational Grids* which are open systems with multiple users, who may belong to different organisations. In Grid settings, like the ones examined in

our simulations, multiple users submit applications like PSIMAP and compete for resources, making resource allocation a more complex task.

An observation, which we made during our experiments is that, for an infrastructure like our university computer labs, the background load will be very small — even at times when it is frequently used by students. Therefore the 'no background load' case, which was examined in some of our simulations, is already a very close approximation of reality — unless the machines are operated in *screensaver mode*. However, this observation may not be representative for other environments.

12.6 Summary

In this chapter, we have shown that our simulated system is implementable and can operate in a real computational environment. This has been achieved by deploying our experimental Grid computing framework, which is based on our simulation model, in a local area network. We also investigated the validity of our simulation model and the scalability of our Electronic Marketplace (EMP). Overall, the results indicate that the system operates as predicted, as long as the tasks are large and the system is not overloaded. It also scales well with the number of resources. In addition to these experiments, we demonstrated the effectiveness of our framework for the distributed execution of real-world applications, such as PSIMAP.

13 Designer's Guidelines

13.1 Introduction

In the chapters 8 to 10, the performance of different market protocols and conventional protocols has been examined for various scenarios. The aim of this chapter is to give guidelines for the designer of an electronic marketplace. The question to be answered is: *Which protocol can be recommended in a given situation?*

13.2 Tasks with the Same Priority (T1-Scenario)

For the T1-Scenario we examined the sensitivity of three protocols [54] to the variation of different parameters. Since all tasks have the same priority in this scenario, little improvement can be expected from the other market protocols. The recommendations for the PC and Supercomputing infrastructures are shown in the two tables in Figure 63. Note that the first part of each table shows the results for cluster infrastructures, (i.e. resource diversity: none, as specified in the second column, RD1). The second part of each table shows the results for the Grid infrastructures (resource diversity: high, RD2).

For the entries in our tables we use the following terminology: The communication delay is considered 'low', if the mean latency is ≤ 0.01, 'moderate' if it is in the range 0.01–0.05, and 'high' if it is ≥ 0.05. The Server number is 'high', if ≥ 110. Regarding the task sizes, the term 'variable' means that the task sizes have a loguniform distribution (TS2). The load is considered 'low', if it is $\leq 50\%$, 'moderate' if it is in the range 50%–90% and 'high' if $\geq$ 90%.

For the background load share, i.e. the fraction of the total load being background load, we choose the following values: 'medium' for a 50% share (BG2) and 'high' for a 75% share (BG3). Regarding the granularity of the background load, the term 'fine-grained' represents the cases SP1 (for PC infrastructures) and SP3 (for Supercomputing infrastructures), i.e. 1 out of 10 resource units is allocated to each arriving background task. The term 'coarse-

[54] For the PC infrastructures, we examined CDA, RR, and PSP. For the Supercomputing infrastructures we examined CDA and RR.

PC Cluster & Grid infrastructures:

Ref. Nr.	Res. Divers.	Comm. Delays	Serv. No.	Task Size	Load	BG Load Share	BG Load Granul.	BG Task Size	Burst Size	Recom-mend.
1	none	none	32	ident.	any	none	–	–	low	CDA,RR
2	none	none	32	ident.	any	med./high	fine	medium	low	CDA
3	none	none	32	var.	any	medium	fine	low–high	low	CDA
4	none	none	32	var.	any	medium	coarse	low–high	low	CDA,RR
5	none	none	32/256	var.	high	medium	fine	medium	high	CDA,RR
6	none	none	32	var.	high	medium	coarse	medium	high	CDA,RR
7	high	none	32	var.	any	medium	fine	medium	low	CDA
8	high	none	32	var.	med.	medium	coarse	medium	low	PSP,CDA
9	high	none	32	var.	high	medium	coarse	medium	low	CDA
10	high	none	high	var.	high	medium	coarse	medium	low	CDA
11	high	high	32	var.	high	medium	coarse	medium	low	PSP

Supercomputing Cluster & Grid infrastructures:

Ref. Nr.	Res. Divers.	Comm. Delays	Serv. No.	Task Size	Load	BG Load Share	BG Load Granul.	BG Task Size	Burst Size	Recom-mend.
12	none	none	32	var.	any	med./high	fine	medium	low	CDA
13	none	none	32	var.	any	medium	coarse	medium	low	CDA,RR
14	high	none	32	var.	any	medium	fine	medium	low	CDA
15	high	none	high	var.	high	medium	fine	medium	low	CDA
16	high	high	32	var.	high	medium	fine	medium	low	CDA,RR
17	high	none	32	var.	high	medium	fine	medium	high	CDA,RR

Abbreviations:
Ref. Nr.: reference number of the parameter set. Res. Divers.: degree of resource diversity/heterogeneity. Comm. Delay: communication delay. Serv. No.: number of Servers in the system. Task Size: task size distribution (identical vs. variable). BG Load Share: fraction of the total load being background load. BG Load Granul.: granularity of the background load. BG Task Size: size of the background tasks. Burst Size: number of tasks submitted at a time. Recomm.: Recommendation for the system designer.

Figure 63: Guidelines for the T1-Scenario in which all tasks have the same priority. This scenario has been examined in chapter 8.

grained' represents the cases SP2 and SP4 in which the background tasks are allocated the whole resource. We assume the background task size to be 'medium' if it is equal to the average size of the tasks submitted by the Client. For the burst size, i.e. the number of tasks submitted at a time, we consider two cases: In the case 'low' the burst size is much smaller than the number of Servers in the system [55], and in the case 'high' it is comparable to the number of Servers.

[55] We used the value 1 in the simulations.

Recommendations

As shown in Figure 63, the CDA protocol performs best in most situations. The PSP protocol can be recommended for a situation in which the communication delays in the system are high (parameter set Nr. 11). It may also outperform CDA for a situation where the resource heterogeneity is high and the load is moderate (Nr. 8). RR performs equally well as CDA for the parameter sets {1, 4, 5, 6, 13, 16, 17}. In these situations there is no choice between resources of different quality. Therefore, RR should be given preference because it is computationally less expensive than CDA.

13.3 Tasks with Different Priorities (T2-Scenario)

The recommendations for the T2-Scenario are given in Figure 64. In all examined situations the background task size is set to 'medium'. The protocols CDA, PSP, HBP, PE-P, PE-A, FIFO, and RR have been examined for all parameter sets. For the other protocols the study has been limited to some parameter sets: SJF has been examined for {2, 3, 5, 6, 8}, PDA for {2, 3, 5, 6, 7, 8}, and CDA-RES for {2, 3, 5, 6, 7}. HBP-RES, HBP-T, and PE-T have been examined for parameter set Nr. 6.

Recommendations

In most situations, PE-A is far better than all other protocols, and it is followed by PE-P. However, we must note that the cost of suspending and resuming the preempted tasks has not been considered in our simulation model. Also, the cost of processing the tasks at the EMP — which will be higher for the preemptive protocols — has not been considered. Therefore, this recommendation is only valid for situations in which these costs are negligible.

If task migration is not possible (e.g. for implementation reasons) or not desirable (e.g. for security or performance reasons) CDA can be recommended. However, if there is a large number of Servers in the system and the load is high, PDA peforms better. The HBP-T protocol, which has only been examined for parameter set 6, can also outperform CDA. For high resource heterogeneity and moderate-to-high loads (Nr. 5 and 7), PSP may also perform better than CDA. If communication delays are high (Nr. 9), the preemptive protocols do not

PC Cluster & Grid infrastructures:

Ref. Nr.	Res. Divers.	Comm. Delays	Serv. No.	Task Size	Load	BG Load Share	BG Load Granul.	Burst Size	Recommendation
1	none	none	32	var.	any	none	–	low	1.PE-A/P 2.CDA 3.HBP
2	none	none	32	var.	any	medium	coarse	low	1.PE-A 2.PE-P 3.CDA
3	none	none	32	var.	any	medium	fine	low	1.PE-A 2.PE-P 3.CDA
4	none	none	32	var.	high	medium	coarse	high	1.PE-A 2.PE-P 3.CDA
5	high	none	32	var.	any	none	–	low	1.PE-A 2.PE-P 3.CDA, PSP
6	high	none	32	var.	any	medium	coarse	low	1.PE-A 2.PE-P 3.HBP-T
7	high	none	32	var.	any	medium	fine	low	1.PE-A 2.PE-P 3.CDA, PSP
8	high	none	high	var.	high	medium	coarse	low	1.PE-A 2.PE-P 3.PDA
9	high	high	32	var.	high	medium	coarse	low	1.PSP 2.HBP 3.CDA

Abbreviations:
Ref. Nr.: reference number of the parameter set. Res. Divers.: degree of resource diversity/heterogeneity. Comm. Delay: communication delay. Serv. No.: number of Servers in the system. Task Size: task size distribution (identical vs. variable). BG Load Share: fraction of the total load being background load. BG Load Granul.: granularity of the background load. Burst Size: number of tasks submitted at a time.

Figure 64: Guidelines for the T2-Scenario in which the tasks have different priorities. This scenario has been examined in chapter 9.

perform very well: now PSP performs best and is followed by CDA and HBP.

13.4 Tasks with Time-Dependent Priorities (T3-Scenario)

The recommendations for the T3-Scenario are shown in Figure 65 (PC Cluster) and 66 (PC Grid). As additional parameters we introduce the type of deadline that is used, i.e. 'hard' vs. 'soft' and 'tight' vs. 'moderate' vs. 'loose'. In all examined situations, the task sizes are variable (T2), the task burst size is set to 1, and the background task size is 'medium'. The protocols CDA, PSP, HBP, PE-P, PE-A, FIFO, and SJF have been examined in all situations. CDA-TDB has only been examined for situations where the tasks have soft deadlines ({4, 5, 13, 18}). PDA and HBP-T have been examined for the parameter sets {1, 2, 3, 6, 7, 8, 9, 10, 11, 12, 14, 15, 16, 17} and CDA-RES for the parameter sets {6, 7, 14}.

Recommendations

In the PC Cluster scenario, PE-A provides a far better performance than the other protocols. It is followed by PE-P. If task migration is not possible or desirable, HBP or HBP-T will be the best choice — depending on the type of deadline that is used. CDA can be recommended for the case 'soft & loose deadlines' (Nr. 5) [56].

PC Cluster infrastructure:

Ref. Nr.	Res. Divers.	Deadline Type	Comm. Delays	Serv. No.	Load	BG Load Share	BG Load Granul.	Recommendation
1	none	hard / tight	none	32	any	none	–	1.PE-A/P 2.HBP, HBP-T
2	none	hard / tight	none	32	any	medium	coarse	1.PE-A 2.PE-P 3.HBP
3	none	hard / loose	none	32	any	medium	coarse	1.PE-A 2.PE-P 3.HBP-T
4	none	soft / tight, soft / mod.	none	32	any	medium	coarse	1.PE-A 2.PE-P 3.HBP
5	none	soft / loose	none	32	any	medium	coarse	1.PE-A 2.PE-P 3.CDA

Abbreviations:
Ref. Nr.: reference number of the parameter set. Res. Divers.: degree of resource diversity/heterogeneity. Deadline Type: type of task deadline (hard vs. soft / tight vs. moderate vs. loose). Comm. Delay: communication delay. Serv. No.: number of Servers in the system. BG Load Share: fraction of the total load being background load. BG Load Granul.: granularity of the background load.

Figure 65: Guidelines for the T3-Scenario in which the tasks have different, time-dependent priorities. This scenario has been examined in chapter 10. In this table, the results for the PC Cluster infrastructure are given.

The recommendations for the PC Grid infrastructures depend on the amount of load in the system. For the situation 'hard & tight deadlines' without any background load in the system (Nr. 6) CDA-RES performs best for all amounts of load and is followed by HBP-T and PSP. When coarse-grained background load is introduced, CDA, FIFO, SJF, and PSP perform best when load is low (Nr. 7). PE-A is best if load is high and is followed by HBP-T and PE-P (Nr. 8). For moderate deadlines PE-A can be recommended if the load is low (Nr. 9) — unless task migration is not possible, in which case CDA, SJF, or HBP-T should be used instead. If the load is high (Nr. 10), the HBP-T protocol performs best and is followed by SJF and CDA. For the situation 'hard & loose deadlines' (Nr. 11 and 12), PE-A performs

[56] Note that HBP-T has not been tested for situations with soft deadlines.

best for all amounts of load. If task migration is not possible, HBP-T should be used instead, or — for low load — CDA or SJF which perform about equally well. With soft deadlines (Nr. 13) , PE-A performs best among all examined protocols. It is followed by PE-P and CDA-TDB.

PC Grid infrastructure:

Ref. Nr.	Res. Divers.	Deadline Type	Comm. Delays	Serv. No.	Load	BG Load Share	BG Load Granul.	Recommendation
6	high	hard / tight	none	32	any	none	–	1.CDA-RES 2.HBP-T 3.PSP
7	high	hard / tight	none	32	low	medium	coarse	CDA, FIFO, SJF, PSP
8	high	hard / tight	none	32	med. / high	medium	coarse	1.PE-A 2.PE-P, HBP-T
9	high	hard / mod.	none	32	low	medium	coarse	1.PE-A 2.CDA, SJF,HBP-T
10	high	hard / mod.	none	32	med. / high	medium	coarse	1.HBP-T 2.SJF 3.CDA
11	high	hard / loose	none	32	low	medium	coarse	1.PE-A 2.CDA, SJF,HBP-T,PE-P
12	high	hard / loose	none	32	med. / high	medium	coarse	1.PE-A 2.HBP-T, PE-P
13	high	soft	none	32	any	medium	coarse	1.PE-A 2.PE-P 3.CDA-TDB
14	high	hard / tight	none	high	high	none	–	1.CDA-RES 2.SJF 3.HBP-T
15	high	hard / tight	none	high	high	medium	coarse	1.SJF 2.HBP-T 3.PSP,CDA
16	high	hard / mod.	none	high	high	medium	coarse	1.HBP-T 2.CDA 3.SJF
17	high	hard / loose	none	high	high	medium	coarse	1.PE-A 2.HBP-T 3.PDA
18	high	soft	none	high	high	medium	coarse	1.PE-A 2.PE-P 3.CDA,SJF,FIFO
19	high	hard / tight	mod. / high	32	high	none / medium	coarse	1.PE-P 2.CDA, HBP
20	high	hard/mod., hard/loose	mod. / high	32	high	medium	coarse	1.CDA 2.PE-P 3.HBP

Abbreviations:

Ref. Nr.: reference number of the parameter set. Res. Divers.: degree of resource diversity/heterogeneity. Deadline Type: type of task deadline (hard vs. soft / tight vs. moderate vs. loose). Comm. Delay: communication delay. Serv. No.: number of Servers in the system. BG Load Share: fraction of the total load being background load. BG Load Granul.: granularity of the background load.

Figure 66: Guidelines for the T3-Scenario in which the tasks have different, time-dependent priorities. This scenario has been examined in chapter 10. In this table, the results for the PC Grid infrastructure are given.

Next, we consider situations in which the number of Servers and the load in the system are high. CDA-RES can be recommended for the situation 'hard & tight deadlines' if there is no background load in the system (Nr. 14). It is followed by SJF and HBP-T. If coarse-grained background load is introduced, SJF performs best for the case 'hard & tight deadlines' (Nr. 15) and HBP-T for the case 'hard & moderate deadlines' (Nr. 16). With 'hard & loose deadlines' PE-A performs best (Nr. 17). If task migration cannot be used, HBP-T and PDA can be recommended. For situations where tasks have soft deadlines (Nr. 18) PE-A performs best and is followed by PE-P. Among the non-preemptive protocols CDA, CDA-TDB, SJF, and FIFO can be recommended.

The recommendations for situations with high communication delays and high loads depend on the type of deadline that is used [57]. For hard and tight deadlines (Nr. 19) PE-P performs best. CDA and HBP come next and can therefore be recommended for the case that task migration is not possible. For hard and moderate/loose deadlines (Nr. 20) CDA provides the best results. It is followed by PE-P and HBP.

13.5 Comment

This chapter provided guidelines for the system designer regarding the choice of the best performing protocol for a given situation. We made the assumption of a managed system (see section 6.4) in which the pricing strategies of the Clients and Servers are enforced in order maximise the performance for the Clients. Hence, our recommendations would be different if the Clients and Servers were free to choose strategies which maximise their utility.

Also note that each recommendation is made for a situation in which the statistical properties of the system such as the load, background load, number of resources, etc. are constant. However, in reality these properties may vary over time. Therefore it will be necessary to implement the marketplace in a way that allows to change the protocol at runtime in order to provide the best possible performance to the Clients.

[57] Note that only few protocols have been examined here.

14 Summary and Future Work

In this chapter we discuss our simulation model and draw conclusions from our simulations and experiments. We briefly compare our observations to the results of the related work. Next, we describe the scenarios in which the marketplace could be deployed and discuss the implementation and scalability aspects that need to be addressed. Finally, we give directions for future work.

14.1 The Model

For our performance evaluations, we have chosen discrete-event simulation, as it allows us to arbitrarily set parameters determining message delays, processing delays, arrival times, etc. Concerning task arrivals and task size distributions, we decided to use synthetic workloads. This gave us the flexibility to explore situations with different amounts and granularity of load. To provide a realistic model, we used distributions that are based on workload logs collected from large-scale systems in production use. Similarly, for modelling communication delays, we used distributions which are based on empirical data. Interestingly, in both cases, we found that the choice of distribution [58] did not have much impact on the results of our simulations. What mattered was the average value chosen for the task sizes or communication delays. For this reason, we believe that using more realistic workload or communication models would not make much difference either.

Regarding the network infrastructure of the system, we used a simple model: All network links between the actors were *identical* and therefore led to the same *mean* communication delays [59]. Similarly, in an experiment, where the transmission of large data has been examined, we chose the same data sizes for all tasks. The reason for these choices is that we wanted to examine the impact of different communication delays: We assumed that the mean of *all* communication delays in the system is the main factor which determines the result.

What has not been considered in our simulation model, are processing delays at the central marketplace. They are likely to be negligible for small or medium-size systems in

[58] We experimented with different values for the standard deviation of the random variable.

[59] Note that network delays were only varied in some of the experiments and neglected otherwise.

which the marketplace is not queried frequently. In fact, our experiments in the lab show that these processing delays are very small in a moderately-loaded, medium-size system. These delays may increase if more queries are sent to the marketplace — which is likely to be the case for large systems. However, our experiments in the lab did not give us enough information for deriving such a model, as the results were biased by other factors. Also, such a model would depend on the implementation and the system hardware.

14.2 Simulations

For our simulations, we first gave an overview of the parameter space to be explored in the simulations. Then, we examined three different scenarios with different requirements of the users. We compared several market protocols and conventional resource allocation protocols. In this section we will describe our research process and present a summary of the results. Finally, we give a critique on the simulations.

Initial Scenario: Tasks with the Same Priorities

Initially, we investigated a simple scenario where all tasks have the *same* priority, and their average completion times have to be minimised. In such a scenario, the only useful feature of the market protocols is their *greedy* behaviour which results in the choice of the best performing resource at a given time. As market protocols, we examined the Continuous Double Auction Protocol (CDA) and the Proportional Sharing Protocol (PSP). For various scenarios, which are characteristic for computational clusters and Grids, we determined which of these two protocols is best and how great the benefit over a simple Round-Robin protocol can be. To determine how general our results are, we studied the sensitivity of the different protocols to various parameters. We found that in almost all situations CDA outperformed the two other protocols. PSP performed better only for moderate loads combined with high resource heterogeneity. It also degraded less than the two other protocols when communication delays were high. Our main focus in the simulations was on PC infrastructures. However, we also considered a scenario, where parallel applications were allocated to multiprocessor machines, and found that in many cases the results are similar.

Tasks with Different Priorities

Next, we examined a scenario that is more realistic for a computational Grid in which the clients and service providers belong to different organisations: Tasks have different priorities for their clients which are expressed by different price bids. To maximise performance from the clients' point of view, these priorities need to be considered for the resource allocation by the marketplace. For this reason, we used the *weighted* completion time of the tasks as performance metric: The higher the priority of a task, the more important is its early completion. Based on the experiences from the initial scenario, we could concentrate our experiments on scenarios and parameter sets which we considered relevant. We also introduced further resource allocation protocols, which can provide improvements in some of the examined situations. These protocols used features such as the suspension or migration of tasks, reserve prices, periodic auctions, time-dependent price bids, etc. Where some protocols failed, improvements were made (e.g. HBP-T).

Tasks with Time-Dependent Priorities

Finally, we considered a scenario which we believe is even more realistic for a Grid setting: The objective of the users is not just to have their tasks executed as fast as possible, but to meet certain deadlines which can be *hard* or *soft*. Similar scenarios have been investigated by [Chun and Culler, 2000], [Kim *et al.*, 2003], and [Nisan *et al.*, 1998]. We used performance metrics which reflect the value delivered to the users. These are the weighted completion rate (for hard deadlines) and the aggregate user utility (for soft deadlines). We considered those protocols whose results were promising in the previous scenario and also examined one protocol which has specifically been designed for situations with soft deadlines (CDA-TDB).

In the following, we will summarise our findings for the different protocols:

Exclusive Allocation of Resources without Preemption

The exclusive allocation of resources without preemption, as used by the Continuous Double Auction Protocol (CDA), will often lead to good results. Differences between CDA and the other protocols will occur at high loads in the system, i.e. when there is a strong competition for resources. In most cases, CDA is the better than the conventional scheduling protocols

RR, FIFO, and SJF, as it prioritises tasks according to their bids, and resources according to their performance. Usually, it is also better than the Proportional-Sharing Protocol (PSP).

Proportional Sharing of Resources

When the heterogeneity of the resources is high, proportional sharing may be preferable to the exclusive allocation of resources. In such situations, the Proportional Sharing Protocol (PSP) will outperform CDA — provided that the load in the system is not too high, and the weighted completion time (WCT) of the tasks is used as performance metric. PSP also copes better with high communication delays. Furthermore, it performs better than CDA in a situation where the tasks have tight deadlines, and the weighted completion rate (WCR) is used as performance metric.

Preemption: Suspension of Tasks

With the Highest Bid Protocol (HBP), we examined a protocol that allows to suspend low priority tasks in favour of high priority tasks in order to improve the overall performance. However, we observed that, in most cases, HBP leads to a poorer performance than CDA: It appears that the delay of suspended tasks outweighs the gain of the suspending tasks. We found that a solution to this problem is to use *thresholds* for the preemptions (as in the HBP-T protocol). With the right choice of threshold, HBP-T outperforms both CDA and HBP.

Preemption: Task Migration

We considered two protocols which enable the migration of tasks whose execution has already started. The first, PE-P (*passive*), allows the migration of a task only when preempted by another task. The second, PE-A (*active*) enables task migration whenever a better resource becomes available. With just a few exceptions, such as situations with high communication delays or tight deadlines, PE-A's results are far better than for the other protocols, and PE-P's results come next. One weakness of PE-P and PE-A, however, is that each migration may trigger another migration. This will often result in a chain reaction which — in reality — can be a huge burden on the central marketplace. A possible solution to this problem could be limiting the number of preemptions, as done by the PE-T protocol.

Use of Reserve Prices

The use of reserve prices (as in CDA-RES) can lead to better results than 'normal' CDA if there are differences in speed or background load among the resources. It can be a good solution in situations where the preemption of tasks is not possible or desirable. One risk of such a protocol is, however, that some low priority tasks may never be executed: This problem can be dealt with by gradually increasing their price bids or by enforcing price discounts at the resources. We must note that the comparison of CDA and CDA-RES only makes sense in a system, in which the Servers do not try to maximise their gain: In a free market, the service providers will set their reserve prices anyway.

Time-Dependent Task Priorities

The protocol CDA-TDB has specifically been designed for situations in which the tasks have *soft* deadlines: It bases the allocation decisions on the current price bids of the tasks, which are decreasing over time, as they reflect the actual values of the tasks. In our experiments, it did not lead to considerable improvements in comparison to 'normal' CDA with static price bids, and also, it was clearly outperformed by the preemptive protocols. Furthermore, a serious disadvantage of CDA-TDB is that it is computationally expensive. As we found in some limited trials with our experimental Grid computing framework, this can considerably delay the processing of tasks at the marketplace.

Periodic Auctions

While most of the auction protocols described in the related work in chapter 5 are periodic, all of our protocols — except PDA — are continuous, i.e. they carry out the transactions immediately. We opted for continuous auctions as we expected better performance. In fact, in most of our experiments the Periodic Double Auction Protocol (PDA) turned out to be less efficient than the Continuous Double Auction Protocol (CDA). Exceptions, however, can be found in situations with highly heterogeneous resources and moderate to high amounts of loads. We also observed improvements for situations where tasks had loose deadlines and the number of resources was high.

Conventional Scheduling Heuristics

In scenarios where tasks have different priorities it is obvious that market protocols lead to better results than those protocols which do not consider these priorities for the resource allocations. Yet, in order to determine in which situations the market protocols can make a difference, we examined several conventional scheduling heuristics, which include Round-Robin (RR), First-In-First-Out (FIFO), or Shortest-Job-First (SJF). Our results show that, with the market protocols, substantial improvements are achieved in situations with high loads and high heterogeneity of resources.

14.3 Simulations: Critique

The simulation results show that, in computational clusters and Grids, market protocols can provide an efficient allocation of resources which will benefit the users. The choice of the appropriate protocol, however, depends on various parameters, including load, resource heterogeneity, communication delays, etc. Guidelines for how to use these results have been given in chapter 13. It should be noted that we examined our protocols for systems, which are *stationary*, i.e. whose statistical properties, such as the task arrival rate, don't change over time. If these statistical properties change, e.g. for different times of the day or the year, our results will still be valid, as long as the duration of the tasks is relatively small. For situations where this is not the case, the protocols may have to be adapted, and predictive techniques may lead to improvements.

The Market as a Tool

In this thesis, we considered the market as a tool to maximise performance from the Clients' perspective. It could be argued that this will only be applicable in *managed systems* where the Servers do not have the objective to maximise their gain (e.g. where the resources are collectively owned by the users). However, we found that trying to maximise the performance for the Clients does not necessarily contradict the objectives of the Servers. The strategic behaviour of the Servers can even be exploited in a way that maximises performance.

Our results for the CDA-RES protocol show that allowing the Servers to set reserve prices can improve performance. The reason is that reserve prices can help to express the

differences in speed and load among the resources: The reserve prices exclude low priority tasks from using the well-performing resources so that they remain available to high priority tasks, which may arrive at a later time. These performance improvements by CDA-RES are achieved, even though the resources remain idle for considerable amounts of time. It appears that this 'waste' of processing power can be outweighed by the gain for the high priority tasks — and overall, the users can benefit from the opportunistic pricing of the Servers.

A problem with using reserve prices is that some low priority tasks may never be allocated to any resource. In a managed system this problem can be solved by enforcing price discounts at the Servers or by gradually increasing the task price bids ('task price adjustment'). However, in a free market, it will not be possible to use these techniques. Instead, the Clients will behave strategically and will therefore increase the bids of their tasks if they expect this to be beneficial to them — and hence provide a form of 'task price adjustment'.

In a free market in which the CDA protocol is used at the EMP, the Servers will need the ability to set reserve prices in order to maximise their benefit. However, if the a preemptive or proportional sharing protocol is used at the EMP, the Servers will not necessarily benefit from setting reserve prices, because the resource allocations can be changed whenever tasks with higher price bids arrive.

Type of Auction and Bidding Strategies

In this thesis, we assumed a managed system and therefore did not have to consider pricing/bidding strategies and resource accounting. Therefore, if we replaced the first-price auctions by second-price auctions, our simulation results would not be affected because the allocation decisions at the marketplace would remain unchanged (except for CDA-RES and PSP). However, for a system that is not managed, the choice of protocol (i.e. first-price vs. second-price) may have a considerable impact on the results because it determines the dominant strategies of the Clients and Servers, and hence the allocation decisions. The choice of the bidding strategy will become important in a scenario in which a Client has to allocate its limited funds to several tasks which it submits to the EMP (e.g. as part of parameter sweep applications, DAGs, etc.). This is subject of future work.

14.4 Experiments

With our experimental Grid computing framework we have shown that our Electronic Marketplace is implementable and can operate in a real computational environment. Although experimental and light-weight, the framework resolves many practical issues of distributed computing, which include the performance measurements at the resources, the measurement of load information, and the passing of data and code between Clients and Servers. Due to the JADE platform, which provides HTTP and various other communication protocols, a deployment of our framework in a geographically distributed system is possible: We enabled it to transfer Java code to remote machines via any of the protocols supported by JADE.

By running our framework in a local area network, we investigated the validity of our simulation model and the scalability of our Electronic Marketplace (EMP). Overall, the results indicate that the system operates as predicted, as long as the communication-to-computation ratio is small and the system is not overloaded. It also scales well with the number of resources. In addition to these experiments, we demonstrated the effectiveness of our framework for the distributed execution of real-world applications, such as PSIMAP. The main limitation of our experiments is that they have only been carried out in a local area network, which did not allow us to examine situations with heterogeneous resources and communication delays.

14.5 Comparison to Related Work

There have been only few other efforts to evaluate the performance of market protocols for computational clusters and Grids. These are described in chapter 5 in more detail. In most cases, the examined scenarios are slightly different and therefore the results are not directly comparable to ours.

Grid Settings

The authors of POPCORN [Nisan *et al.*, 1998; Regev and Nisan, 1998] examined the performance of several market protocols in a geographically distributed system. In contrast to our work, the authors studied economic properties of the market, such as price stability and

economic efficiency, rather than measuring performance from the user's perspective. Also, the evaluation was limited to periodic auctions which, according to our results, are less efficient than continuous auctions. Mechanisms such as proportional sharing, task preemption, or migration, which — as we found — can lead to considerable performance improvements, have not been examined either.

In the work reported in [Wolski *et al.*, 2001], different market protocols were examined for a Grid setting. Again, the performance metrics were different than in our work: They included the job throughput, utilisation of resources, and price stability. The experiments were also limited to protocols with periodic transactions.

Cluster Settings

In [Chun and Culler, 2002], market protocols were examined for a computational cluster which was modelled as a single, divisible resource consisting of identical processors. The authors studied a scenario where tasks had soft deadlines. They found that a first-price auction can outperform conventional scheduling protocols like SJF. They also observed that, for sequential workload, the use of auctions with time-dependent price bids does not lead to considerable improvements over auctions with static price bids. In spite of the differences in the setup, both observations are consistent with our findings for CDA-TDB, CDA, and SJF. However, the authors also found that the preemption of tasks 'does not add significant value': This observation could not be confirmed by our experiments, in which the preemptive protocols clearly outperformed CDA and CDA-TDB.

The authors of [Kim *et al.*, 2003] studied the unrelated machine case for a cluster setting and therefore required a different type of resource allocation protocols. Yet, in the experiments, the best results have been achieved with those protocols which considered task priorities when allocating resources. This is consistent with our findings.

The work by [Ferguson *et al.*, 1996] examined a cluster setting in which auctions were carried out at each processing node. Due to the differences in the setup, the authors' results cannot be compared to our results. However, their general observation, that market protocols can achieve better performance levels than non-economic protocols, has been confirmed by our simulations.

Resource Control for Mobile Agents

The authors of [Bredin *et al.*, 2001; Bredin, 2001] examined a proportional sharing protocol for the allocation of computational resources to mobile agents. They observed an 8% overhead for this market protocol in comparison to a locally optimal, non-economic protocol. This observation does not contradict our findings, as the authors did not consider the different priorities of the tasks in their performance metrics.

14.6 Applicability

In this section, we briefly describe the scenarios in which the marketplace could be deployed. We also discuss the implementation and scalability aspects that need to be addressed.

Scenarios

Our Electronic Marketplace could be deployed in various environments. Regarding the type of system infrastructure, we distinguish the following two cases: The first is a local cluster of identical resources which is small in size, and the second a *Grid* in which the resources are heterogeneous and geographically distributed. In both cases the resources can be PCs or multiprocessor machines. Regarding the ownership of resources which are offered at the marketplace, we envisage the following scenarios:

- The resources are owned by just one organisation, e.g. a research institution or company, which provides access only to its users, e.g. students or employees.
- The resources are *collectively* owned by different users or organisations to which the users belong.
- The resources are part of a web-farm owned by one single organisation, and the incoming service requests of *external* users are prioritised on the basis of the importance of the users, their past usage, etc. If the services are offered for free, resource accounting will not be needed.
- Both users and resource providers belong to different organisations. In such an environment, the issues of security and resource accounting are harder to resolve. Also,

the resource providers will be maximising their revenue — which is not necessarily the case in the other scenarios.

Implementation

From the implementation point of view, non-preemptive protocols like CDA are the simplest. Also, since the allocations of the tasks cannot be changed during their execution, they are the most reliable from the user's point of view. We found that the implementation of the preemptive and proportional sharing protocols is not straightforward, as it requires the ability to suspend and resume executing threads or other ways to control resource consumption. At the present time, this functionality is not supported by Java — which JADE needs to run — but efforts are underway to solve this problem [Czajkowski *et al.*, 2003]. Task migration, as used by PE-P and PE-A, is even harder to implement. Unless the program code is written in a way that facilitates migration, it will require *strong* mobility (see section 3.7): The complete execution state of a task has to be packaged up, in order to resume execution on the new Server. This facility is not provided by standard Java but has been implemented by the D'Agents mobile agent platform [Rus *et al.*, 1997], which uses a modified virtual machine.

Scalability

Bearing in mind that, in reality, computational tasks are at least a few minutes long, and the marketplace is able to process several requests per second, our central marketplace will scale for hundreds, if not thousands of resources. As we found in our experiments, this will already be the case with our experimental framework which is written in Java. With a more efficient implementation, an even better performance could be achieved. Beyond a certain point, however, it will be necessary to distribute the marketplace.

14.7 Future Work

The work presented in this thesis can be extended in several directions, which concern the simulation model, protocol design, and implementation. These will be described in this section.

Simulation Model

We could extend our simulation model by considering the processing delays at the central marketplace. This may be needed for the modelling of larger systems, in which the marketplace is queried frequently. Another extension — essential to the modelling of data-intensive applications — would be to consider memory requirements in the resource allocations. Our simulation model should allow the tasks to specify requirements for the memory or storage space needed, and model the resources accordingly. A further improvement, which could be relevant for the deadline-scenarios, concerns the task computation sizes: In reality they are often not known before their execution and only estimates can be given. The real task sizes should therefore be modelled as random variables.

To examine a real, geographically distributed system, more sophisticated models may be needed for the communication delays. These could be queuing-theoretic or based on data obtained from real network traces. Also, more realistic network infrastructures could be used for the simulations. Furthermore, our simulation model could be used to study a system that provides market-based resource control for mobile agents ([Bredin *et al.*, 2001; Bredin, 2001], see section 5.5). Another scenario, which we believe is relevant to Grid computing and which should be considered as future work, is the scheduling of applications with subtask dependencies (see subsection 2.3.1). Our model could also be extended to cover the *combinatorial case* in which each application requires a bundle of different resources for its execution (e.g. different machines, memory & storage space, network bandwidth, etc.).

Protocol Design

Regarding the design of the resource allocation protocols, it would be interesting to examine whether hybrid approaches can lead to any performance improvements. E.g. one could combine proportional sharing or preemption with reserve pricing, thresholds, or periodic auctions. In the scenario where tasks have deadlines (see chapter 10), protocols could be considered which not only consider the priority of tasks but also their urgency.

What should also be investigated is whether the use of performance prediction (see section 4.5) can improve the results. In some limited trials we examined a version of CDA which uses estimates of the load on the resources for its allocation decisions. However, we could

not find any considerable improvements. Yet, performance prediction may lead to better results in other scenarios. Another challenging task would be to enhance the pricing strategies of the Servers, so that they maximise their utility [60]. Similarly, the bidding strategies for the Clients could be improved.

Implementation

Concerning the implementation of the marketplace, the most important extension would be to actually deploy protocols which use proportional sharing or the preemption of tasks. This is currently not possible with Java. However, possible solutions would be to (i) use a modified Java virtual machine ([Suri, 2000], e.g.), (ii) use a framework that controls resource consumption through *bytecode rewriting* [Binder *et al.*, 2001], or (iii) schedule processes at the operating system level.

In this thesis we assumed that system properties such as the load, background load, number of resources, etc. remain constant. However, in reality these properties may vary over time. Therefore it will be necessary to extend the marketplace in a way that allows to change the protocol at runtime in order to provide the best possible performance to the Clients.

Furthermore, to make the marketplace more scalable, it might be necessary to distribute it over several machines. In [Wolski *et al.*, 2003], an architecture is proposed which consists of several branches which store replicated and synchronised information about the demand and supply. We find that a scalable marketplace does not necessarily have to be geographically distributed — rather the machines could be located at the same site. In fact, Napster [61] and eBay [62] are examples which show that centralised systems may even scale on a global level. In any case, the resource allocation protocols would have to be adapted to deal with the replicated data structures and the distributed processing of queries. Yet, by doing this, the operational behaviour of the protocols as described in this thesis would not necessarily have to change.

To finally deploy the marketplace, issues such as security, trust, and resource accounting have to be dealt with.

[60] In [Bredin, 2001], such strategies have been developed for resource control in mobile agent systems.
[61] http://www.napster.com
[62] http://www.ebay.com

A Resource Allocation Protocols

A.1 Resource scheduling policy: Proportional sharing

In this chapter, we describe resource scheduling policy 4 (see subsection 6.2.3), which uses proportional sharing, in more detail.

In this policy, several tasks can execute on a Server at a time. The amount of resources allocated to a task, $N_{RU,alloc}$, depends on its price bid, $p_{Task,i}$, in relation to the sum of price bids $\sum p_{Task,i}$ of all tasks executing on that Server, including the bid of the task itself. It is given by

$$N_{RU,alloc} = \frac{p_{Task,i}}{\sum_i p_{Task,i}} N_{RU,avail},$$

where $N_{RU,avail}$ is the total number of available resource units.

Every time a task or background task starts or completes execution on the Server, the other tasks need to be rescheduled. The tasks' resource shares change, and so does their execution speed. At such events, the effective execution speed $s_{Eff,i}$ of each executing task i is determined, and its completion event is rescheduled. First, the overall effective execution speed of the Server $s_{Eff,total}$ needs to be calculated. It is given by:

$$s_{Eff,total} = f_{Speed} \cdot N_{RU,avail}.$$

From this, the new effective execution speed of the task, $s_{Eff,i}$, can be obtained. It depends on the overall effective execution speed of the Server, on the price bid of task i, and on the sum of the price bids of all executing tasks. It is given by:

$$s_{Eff,i} = s_{Eff,total} \cdot \frac{p_{Task,i}}{\sum p_{Task,i}}.$$

The task's remaining size to be executed $S_{C,R,i}$ is calculated by

$$S_{C,R,i} = S_{C,R,i,prev} - (t - t_{prev}) \cdot s_{Eff,i,prev}.$$

In this equation, $S_{C,R,i,prev}$ is the remaining size of the task at the previous rescheduling event at time t_{prev}. t is the current time and $s_{Eff,i,prev}$ the effective execution speed of task i at the previous re-scheduling event. The remaining execution time $T_{Exec,R,i}$ of task i is given by:

$$T_{Exec,R,i} = \frac{S_{C,R,i}}{s_{Eff,i}}.$$

The completion event of task i is rescheduled for time $t + T_{Exec,R,i}$. The procedures for the main loop of the Server and for the rescheduling of executing tasks are given in Fig. 67 and Fig. 68, respectively.

```
SERVER CODE:
------------
while (experiment is running) {
   switch (event) {
      task start event: // when a task is received from a Client
         add task to the list of executing tasks;
         increase sum of price bids;
         RESCHEDULE EXECUTING TASKS;
         break;
      task completion event:
         remove task from the list of executing tasks;
         decrease sum of price bids;
         update sum of price bids information at the EMP;
         send task result to the Client;
         RESCHEDULE EXECUTING TASKS;
         break;
      background task start event:
         update resource information at the EMP;
         RESCHEDULE EXECUTING TASKS;
         break;
      background task completion event:
         process waiting background tasks (if any);
         update resource information at the EMP;
         RESCHEDULE EXECUTING TASKS;
         break;
   }
}
```

Figure 67: Proportional sharing: Main code

```
RESCHEDULE EXECUTING TASKS:
---------------------------
calculate the effective execution speed of the Server;
for all tasks i currently executing:
   calculate the effective execution speed for task i;
   calculate the remaining size of task i to be executed;
   calculate the remaining execution time for task i;
   (re-)schedule the completion event of task i;
```

Figure 68: Proportional sharing: Rescheduling of tasks

A.2 Resource allocation protocols

To give the reader more insight into the operation of the resource allocation protocols examined in this thesis, we provide the pseudo code of the most important procedures. Also, a more detailed description of the Round-Robin Protocol will be given.

A.2.1 Procedures common to all protocols

```
EMP MAIN LOOP:
--------------
while(experiment is running) {
   wait for a new event;
   switch (event) {
      task arrival event:
         PROCESS TASK QUERY (Task Query object);
         break;
      server resource update event:
         RESOURCE UPDATE EVENT (resource offer);
         break;
      task price adjustment event:
         TASK PRICE ADJUSTMENT EVENT;
         break;
   }
}
```

Figure 69: Main loop of the EMP

```
PROCESS TASK QUERY
------------------
if (task deadline has passed) {
   discard the Task Query object;
}
else {
   QUERY FOR SERVER (size, price, deadline);
   if (query successful) {
      send the query result to the task's Client;
   }
   else {
      add the Task Query object to the
       list 'taskQueryObjectList';
      indicate that 'taskQueryObjectList' is not sorted;
   }
}
```

Figure 70: Processing a task query at the EMP

A.2.2 Continuous Double Auction Protocol (CDA)

```
QUERY FOR SERVER (size, price, deadline):
-----------------------------------------
starting from the beginning of the list of resource offers:
   check all resource offers in order to find the 'best'
   (cheapest/fastest) offer which meets the task's
   constraints {size, price, deadline};

if (successful) {
   tag the best resource offer to indicate that it is reserved;
   return the result;
}
else {
   return null as result;
}
```

Figure 71: Continuous Double Auction Protocol: Query by a task

```
PROCESS RESOURCE UPDATE EVENT (resource offer):
-----------------------------------------------
if ('taskQueryObjectList' is not sorted) {
   sort 'taskQueryObjectList' according to the price bids
    in descending order;
   indicate that 'taskQueryObjectList' is sorted;
}

starting from the beginning of the 'taskQueryObjectList':
   - search 'taskQueryObjectList', until the first task is found for which
     the resource offer is acceptable w.r.t. {size, price, deadline};
   - while doing this: remove those Task Query objects from
     'taskQueryObjectList' for which the deadline has passed;

if (successful) {
   tag this resource offer to indicate that it is reserved;
   send the query result to the task's Client;
   remove Task Query object from 'taskQueryObjectList';
}
```

Figure 72: Continuous Double Auction Protocol: Resource update event

A.2.3 Proportional-Share Protocol (PSP)

```
QUERY FOR SERVER (size, price, deadline):
-----------------------------------------
starting from the beginning of the list of resource offers:
   check all resource offers in order to find the offer that is
   the fastest to execute the task and which meets the task's
   constraints {size, price, deadline}. (This takes into account
   the current background load and sum of price bids on that
   Server);

if (successful) {
   add the task's price bid to the sum of price bids of the
    Server offer which was chosen;
   return the result;
}
else {
   return null as result;
}
```

Figure 73: Proportional Share Protocol: Query by a task

```
PROCESS RESOURCE UPDATE EVENT (resource offer):
-----------------------------------------------
if ('taskQueryObjectList' is not sorted) {
   sort 'taskQueryObjectList' according to the price bids
    in descending order;
   indicate that 'taskQueryObjectList' is sorted;
}

starting from the beginning of the 'taskQueryObjectList':
   - search 'taskQueryObjectList' until the first task is found for which
     the resource offer is acceptable w.r.t. {size, price, deadline};
   - while doing this: remove those Task Query objects from
      'taskQueryObjectList' for which the deadline has passed;

if (successful) {
   send the query result to the task's Client;
   add the task's price bid to the sum of price bids of the
     Server offer which was chosen;
   remove Task Query object from 'taskQueryObjectList';
}
```

Figure 74: Proportional Share Protocol: Resource update event

A.2.4 Round-Robin Protocol (RR)

In the Round-Robin Protocol no pricing is used. The incoming task queries are matched with the *next* available resource offer which meets the task's constraints but which is usually not the *best*. For this purpose, an iterator (called 'serverListIterator') is used which cycles through the list of resource offers (see Fig. 75).

```
QUERY FOR SERVER (size, deadline):
----------------------------------
count = 0;
listSize = size of the list of resource offers;

do {
   get the resource offer at the current position of the iterator
   in the list of resource offers (called 'serverListIterator');

   if (resource offer is available
    && meets the task's constraints) {
      tag the resource offer to indicate that it is reserved;
      serverListIterator = (serverListIterator+1) MODULO listSize;
      return the result;
   }

   serverListIterator = (serverListIterator+1) MODULO listSize;
   count++;

} while(count<listSize);

return null as result;
```

Figure 75: Round-Robin Protocol: Query by a task

On arrival of a Task Query object at the EMP, the list of resource offers is searched until a resource is found which satisfies the task's constraints {size, deadline}. The search starts at the current position of the iterator. In case of a success, the resource offer is reserved. The iterator is incremented and the result returned. Otherwise, the iterator is also incremented and the next resource offer is considered. This step is repeated until all resource offers have been checked or a match has been found. If the query is successful, the result is sent to the task's Client. Otherwise, the Task Query object remains at the EMP until a suitable resource becomes available. It is stored in a list called 'taskQueryObjectList'.

Whenever a resource becomes available, the Task Query objects in this list are processed

in the order of their arrival, until the resource offer is taken or all elements have been checked and no match was found (see in Fig. 76). In case of a match, the iterator of the list of resource offers (called 'serverListIterator') is moved to the next position.

```
PROCESS RESOURCE UPDATE EVENT (resource offer):
-----------------------------------------------
listSize = size of the list of resource offers;
starting from the beginning of the 'taskQueryObjectList':
   - search 'taskQueryObjectList', until the first task is found for which
     the resource offer is acceptable w.r.t. {size, deadline};
   - while doing this: remove those Task Query objects from
     'taskQueryObjectList' for which the deadline has passed;

if (successful) {
   tag this resource offer to indicate that it is reserved;
   send the query result to the task's Client;
   remove Task Query object from the 'taskQueryObjectList';
   serverListIterator = (serverListIterator+1) MODULO listSize;
}
```

Figure 76: Round Robin Protocol: Resource update event

A.2.5 Task Price Adjustment Event

If enabled, the task price adjustment event (see pseudo code in Figure 77) occurs at regular time intervals — as long as the experiment is running. At first, the price bid p_{Task} of each TaskObject in 'taskObjectList' is adjusted according to:

$$p_{Task} = p_{Task,max} + \frac{t - t_{Creation} - T_{Neg}}{T_{Neg}} \cdot \left(p_{Task,max} - p_{Task,min}\right)$$

$$for\ t_{Creation} \leq t \leq t_{Creation} + T_{Neg}$$

In this equation $p_{Task,max}$ is the maximum price the task is willing to pay and $p_{Task,min}$ the minimum price that is initially requested. The price depends linearly on the time which is left to the end of the *negotiation time* T_{Neg}: The parameters used are the task deadline, t_D, the task creation time $t_{Creation}$, the (maximum) negotiation time T_{Neg}, and the current time, t. If the negotiation time has passed, the price is fixed at its maximum $p_{Task,max}$.

```
PROCESS TASK PRICE ADJUSTMENT EVENT:
-----------------------------------
if (experiment is running){
   schedule the next TASK PRICE ADJUSTMENT EVENT;
}
for (each task in 'taskObjectList') {
   adjust price;
}
sort 'taskObjectList' according to the price bids in descending order;
indicate that 'taskObjectList' is sorted;
for (each task in 'taskObjectList') {
   if (task deadline has passed) {
      remove TaskObject from 'taskObjectList';
   }
   else {
      QUERY FOR SERVER (size, price, deadline);
      if (query successful) {
         infinitesimal delay to avoid concurrent events;
         send the query result to the task's Client;
         remove TaskObjects from 'taskObjectList';
      }
      else {
         if (task price bid < lowest Server price) {
            exit method;
         }
      }
   }
}
```

Figure 77: Task price adjustment event

B Simulation Framework

B.1 Introduction

This chapter provides an overview of our simulation framework which has been used for the experiments described in this thesis. Also, we will describe the method that we used for dealing with randomness in the experiments.

B.2 Discrete-Event Simulation

Simulation can help to evaluate different resource allocation protocols before their implementation and without depending on a particular hardware infrastructure, network topology, or middleware. To simulate distributed systems, *discrete-event simulation* can be used. Basically, a discrete-event simulation operates with a list of events (in *virtual time*) and a central simulator object that executes these events in order. During the execution of the events, new events may be generated which will be added to the list.

The reason for choosing discrete-event simulation for our experiments is that it allows to arbitrarily set parameters determining message delays, processing delays, arrival times, etc. Hence, one does not depend on a particular system and can explore a strategy for different scenarios.

B.3 Choice of language and package

A discrete-event simulation can easily be implemented in Java or other object-oriented programming languages. As an alternative, a dedicated simulation language like SIMULA [Pooley, 1987] can be used.

We opted for Java because of prior experience with this language and because of the availability of various open-source simulation packages, which saved us a lot of programming effort. Also, writing the simulation in Java later enabled us to re-use large parts of our code when implementing our basic Grid Computing framework (see chapter 11). General-purpose simulation packages, which are based on Java, include Simjava [Howell and McNab, 1998],

Silk [Kilgore and Burke, 2000], and Desmo-J [Page *et al.*, 2000]. Other systems like Swarm [Minar *et al.*, 1996] or RePast [Collier, accessed in 2003] are designed for agent-based simulation. We have chosen Simjava (version 1.2), as it is open-source and relatively small, and therefore easy to control. It provides the core functions of a discrete-event simulator which we extended with additional Random number generators, statistical functions, and message passing capabilities. Furthermore, it uses continuous — rather than discrete — time, which enables us to choose arbitrary values for the delays, without any limitations introduced by discrete timesteps. In Simjava, all simulated actors are represented by `Sim_entity` objects which have their own threads and which may interact with each other. During the simulation, each thread may be started and stopped by the simulator according to the events which are stored in the event list. Also, it may create new events.

B.4 Simulation Framework: Base Package

Our simulation framework comprises several Java packages: a *Base* package, a statistical package, a random number generation package, and several packages which are specific to particular protocols. The *Base* package classes *ClientBase*, *EMPBase*, and *ServerBase* contain basic functionality for the Clients, the EMP, and the Servers. Most of this functionality is used by all resource allocation protocols. In the protocol-specific packages, these classes are extended by classes called *Client*, *EMP*, and *Server*. A brief overview of the classes in the *Base* package is given below:

- *Account*: the bank account for the Clients and Servers. It contains a method for transferring money to (or from) it.
- *Bank*: the bank which holds the accounts.
- *ClientBase*: contains the basic functionality of the Clients, like storing and sending tasks and collecting the results after their execution. It creates computational tasks with exponential inter-arrival rate and sends requests to the EMP.
- *CommEntity*: the base class for all communicating entities which have their own

thread, reside on a network node, and are able to communicate via remote method invocation — or passing of events. It extends the `Sim_entity` class.

- *CommInterface*: an interface for all communicating entities which reside on network nodes and are able to communicate via remote method invocation. Currently, it is only implemented by the *CommEntity* class. However, it could also be used for objects without active threads.

- *EMPBase*: the base class of the electronic marketplace. It defines all the basic functionality that the electronic marketplace requires. This class can be extended to add extra features to the EMP if necessary.

- *EMPQueryResult*: contains the result of a task's query to the EMP.

- *EMPResourceAd*: contains the parameters of a Server's resource offer ('*ad*') at the EMP.

- *Helper*: uses Java's reflection classes and can be used for a method invocation where the method name is passed as a string. It is needed for the remote method invocation or other invocations that are carried out after a certain delay.

- *Messenger*: used for the invocation of methods on objects which are accessible via a CommInterface.

- *MethodInvocation*: wraps up the information which is used for the invocation of a method on an object represented by a CommInterface. The method invoked is represented by a string.

- *Network*: contains a network topology represented by a list of network links. It is able to calculate the communication delay on a Network link specified by two network nodes.

- *NetworkLink*: contains the characteristics of a network link which are relevant to the communication delay.

- *Parameters*: stores the input parameters that are used for the simulation. It is able to print out its member variables (types and values) and to change these by reading from an input file. It uses Java reflection classes to manipulate the values. The class also has facilities to write results to a file, etc.
- *Resource*: represents a resource of a Server. It is characterised by a number of resource units which can be allocated to one or more tasks. It is associated with a Server.
- *ServerBase*: the Base class for the Servers. It contains the basic functionality that is required for a Server and can be extended.
- *Statistics*: stores the statistics which are recorded during an experiment.
- *StatisticsRecorder*: records certain statistics at periodic intervals and writes them to a file.
- *SuperTask*: this class can be used for representing a chain (pipeline) of computational tasks which may require different resources. (This scenario is not examined in this thesis.)
- *TaskData*: represents a computational task and contains all parameters that describe it. A TaskData object needs to be sent to a Server for execution.
- *TaskObject*: contains the information necessary for a tasks's resource query at the EMP. The actual task is represented by a TaskData object.

B.5 Simulation Framework: Protocol-specific Packages

The simulation framework contains several protocol-specific packages which cover the functionality necessary for one or more protocols. These packages include the classes *Client*, *EMP*, *Server*, and *Startup*. The latter is used for starting a simulation with the corresponding protocol. A list of these packages and of the protocols, which they implement, is given below:

- *CDA*: Continuous Double Auction Protocol, CDA-RES Protocol, PDA Protocol

- *SAP*: Seller-Adjusted Pricing Protocol, FIFO Protocol
- *RRP*: Round-Robin Protocol
- *PSP*: Proportional Share Protocol
- *HBP*: Highest Bid Protocol, HBP-T Protocol
- *PE*: Preemptive Protocol (PE-P, PE-A, and PE-T)
- *UCV*: CDA-TDB, PRIO-FIFO, and SJF

There are two further packages in our framework which contain protocols that have not been discussed in this thesis. These are:

- *RRQ*: Round-Robin with Server Queues
- *MCT*: Minimum Completion Time Protocol

B.6 Running a Simulation

The simulation is started by executing the *Startup* class of the protocol-specific package to be used. In the command line, the name of the file is passed from which the values for the input parameters will be read. These values are written into variables in the *Parameter* class. An example of a file containing input parameters is shown in chapter E.

The *Startup* class initialises an output trace which allows to log the events and values of the variables during the simulation. Each entry that is written to the trace consists of the name of the actor, which it is written by, a time stamp, and the output message.

Next, the *Startup* class initialises a *StatisticsRecorder* object, which records the values of important variables at periodic intervals and writes them to a file. These provide the user with snapshots of the system state at these recording events. The recorded variables include the total number of tasks created, tasks completed, tasks discarded, tasks currently in the system, tasks querying for a resource, and tasks executing. Furthermore, the average load and background load at the Servers, the average task completion time, and the average prices (ask, bid, and transaction) are reported.

Then, all actors in the system are created and the simulation is started. The actors extend the *CommEntity* class which is able to perform a *delayed* RMI-style invocation of methods on other *CommEntity* objects. This is implemented using Java's reflection classes and is easy to use, as the actors do not require a message handler. The communication delays are determined by the *Network* class.

During the simulation, certain variables are recorded by using *Observational* objects from our statistical package. Variables which are recorded include the mean waiting times, execution times, and completion times of the tasks, the amount of load and background load at the Servers, the prices of Clients and Servers, and many other variables. For each variable, several statistical figures, such as mean, standard deviation, minimum, and maximum, are recorded. After the experiment, a comprehensive summary of all statistics will be displayed, plus additional information, such as the task completion rate.

Our random number generator is based on Java's *util.Random* package. To avoid correlation between different random variables (e.g. task inter-arrival time or communication delays), we use a separate random stream for each random variable which is initialised with a separate random seed. Distributions which are supported by our random number generator include the uniform distribution, normal distribution, lognormal distribution, loguniform distribution, and bi-modal normal distribution. To ensure that the simulations are repeatable, the random seeds are *not* based on the system clock.

B.7 Experiments with Parameter Variation

Many of our experiments require to measure some specified variables, while one or more input parameters are varied. To implement this exploration of the parameter space, we wrapped up the above simulation framework by an external Java program.

There are two ways of running the simulator: either a single simulation is run and a comprehensive summary of all statistics is obtained, or a set of specified parameters is examined for a subset of the parameter space. In the latter case, the external program reads the default values for the input parameters from a file like the one shown in chapter E. These values are used as a basis for the experiments. The external program then uses *for*-loops to assign new

```
Base.Parameters  SAPricingTag                    none
Base.Parameters  totalLoadRatio                  none
Base.Parameters  BGLoadRatio                     none

Base.Statistics  recordedTasksCompletedOnTime    none
Base.Statistics  obsTimeInSystem          mean
Base.Statistics  obsWeightedCompletionTime       mean
```

Figure 78: Example file which specifies the parameters to be recorded

values to some chosen parameters. For each simulation run, the values of these parameters are overridden, and a new input file is created. The *Startup* class of the simulation framework is then executed with this newly created input file. In addition to this input file, the parameters to be recorded in all the simulation runs need to be specified in a separate file. An example of such a file is given in Figure 78. Each entry in this file represents a parameter to be recorded. It consists of three parts: the name of the class where the parameter can be found, the name of the object where the parameter is stored, and the name of the field within this object that will be recorded.

For instance, if the object is of type *Observational*, then the field can be the mean, standard deviation, number of recordings, etc. If the object is a primitive type like *double*, then '*none*' must be specified as field name. For each simulation run, the simulator will write a new line to an output file where the values of these variables are recorded.

B.8 Dealing with Randomness of the Results

As random numbers are used in the simulations, there is a certain degree of uncertainty when determining the *mean* of the variable which is measured (e.g. the completion time of tasks). In this section we will explain how we dealt with this problem in our experiments.

There are several commonly used methods of obtaining a reliable value for the mean. These include using long enough measurement intervals, repeating each simulation run many times with different random seeds, and ensuring that the measurements are made in a steady-state of the system. However, these methods are insufficient as it remains unclear how reliable the value obtained for the mean is. Therefore, in addition to the methods described above, we determined the *confidence interval* of the mean.

We chose a method, which is used for simulations by many researchers in the field, and which is described in [Jain, 1991]. The method consists of conducting m replications of the simulation run using different random seeds. In each replication, n measurements are made (in the system's steady state).

At first, a mean of the measured variables needs to be calculated for each replication i. It is given by

$$\bar{x}_i = \frac{1}{n}\sum_{j=1}^{n} x_{ij}, \quad i = 1,2,...,m.$$

Next, an overall mean for all replications is computed:

$$\bar{\bar{x}} = \frac{1}{m}\sum_{i=1}^{m} \bar{x}_i.$$

From these values, the variance of the means of the replications needs to be determined:

$$Var(\bar{x}) = \frac{1}{m-1}\sum_{i=1}^{m} (\bar{x}_i - \bar{\bar{x}})^2.$$

The confidence interval for the mean of the measured random variable is given by:

$$\left[\bar{\bar{x}} \mp z_{1-\alpha/2}\sqrt{\frac{Var(\bar{x})}{m}}\right].$$

The parameter $z_{1-\alpha/2}$ determines the width of the confidence interval. We decided to use the 95% confidence interval which is commonly used in science. For the 95% confidence interval, $z_{1-\alpha/2}$ has the value 1.96.

For the above formula to be valid, the number of replications, m, has to be chosen high enough (≥ 30). We used $m = 40$ in our experiments, which resulted in narrow confidence intervals for most cases.

C Additional Simulation Results

In this part of the appendix, additional simulation results will be given.

C.1 Tasks with the Same Priority: Supercomputing Cluster

In this and the following section we will explore the *Supercomputing Cluster* and the *Supercomputing Grid* infrastructures. We only examine RR and CDA because PSP requires time-shared resources. The question to be answered is in how far these results differ from those observed for the *PC Cluster* and *PC Grid*.

C.1.1 Different Amounts of Load

The first experiment is defined by the parameters {T1, SP3, C1, SN1, RD1, LV, BG2, TS2, BS1}. This means that we have a system with N_{Serv}= 32 identical space-shared resources. The Servers have the resource size $N_{RU,total}$ = 10 and speed factor f_{Speed} = 1.0. The task computation size S_C has a loguniform distribution, and the burst size is 1. Communication delays are neglected. The average total amount of load in the system is varied between 0 and 100% of the system capacity, and half of this load is background load. Background tasks have the computation size $S_{C,BG}$ = 1.0 and are allocated $N_{RU,BG}$ = 1 resource units at a time.

As shown in Figure 79 (left), CDA provides the best results for the whole range of loads. RR performs worse because resources are allocated arbitrarily, whereas CDA selects the fastest available resource. Unlike for the PC infrastructures, the gap between CDA and RR does not close for higher loads. Overall, the completion time of both protocols is higher than in the PC Cluster infrastructure (Figure 13 (left)). The reason is that the resource share allocated to a task remains constant even when a background task completes execution. The free resource units either remain idle or are allocated to other tasks.

However, in contrast to the PC Cluster infrastructure, the performance now does not degrade much when the background load in the system is increased (BG3). As shown in Figure 79 (right), the mean completion times for CDA and RR are now only marginally higher than before. The reason is that background load is no longer prioritised. Hence, for

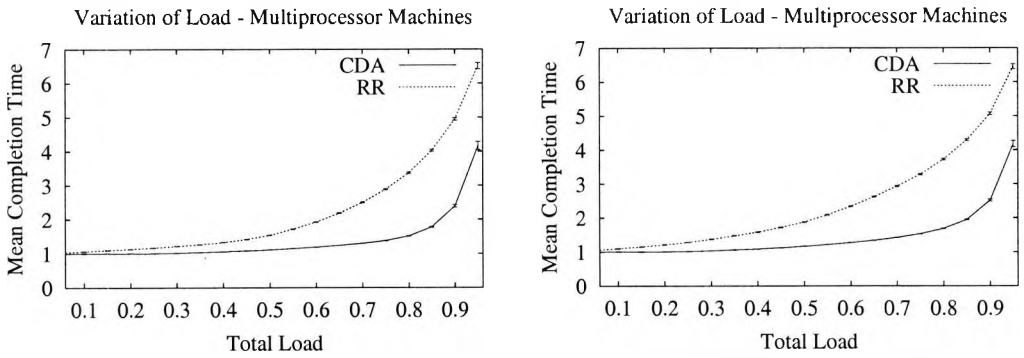

Figure 79: Supercomputing Cluster. Left: Variation of load in the system. Right: Variation of load when background load is higher.

a task it does not make much difference if it competes with other tasks or with background load.

Conclusion: CDA performs better than RR. The gap is wider than for the PC infrastructures where time-shared resources have been used. This is due to the fragmentation of resources, which is caused by allocating constant resource shares to the tasks. We also found that increasing the share of background load within the total load does not make much difference to the results as background load is not given priority.

C.1.2 Granularity of Background Load

As in the PC scenario, the completion time of the tasks is also affected by the granularity of the background tasks. In the experiment in Figure 80 (left) we use the same parameters as in Figure 79 (left), except that now each background task takes up all resource units of the Server, i.e. $N_{RU,BG}$=10 (SP4). As for the PC infrastructure, RR now performs equally well as CDA because all resources that are available execute at the same speed. Note that the two protocols perform much better than in the PC Cluster infrastructure (in Figure 13(right)), because background load does not have priority, and because resources are not fragmented as in Figure 79 (left).

Conclusion: Like for the PC infrastructures, the granularity of the background load has a large impact on the performance of the protocols. The difference is that the protocols'

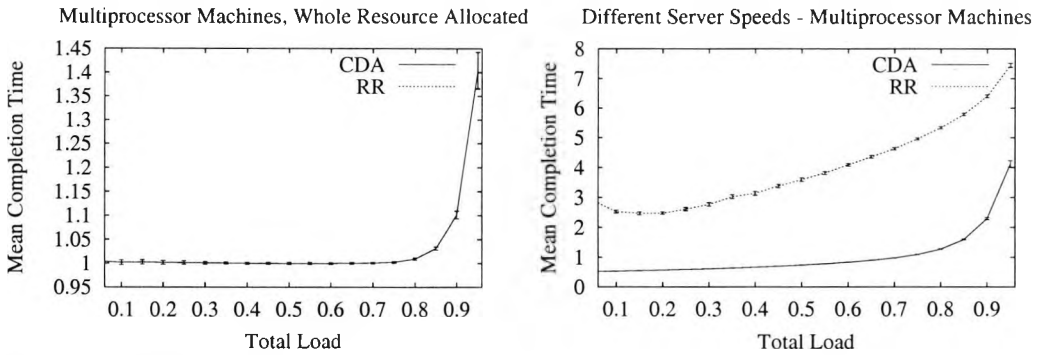

Figure 80: Left: Supercomputing Cluster. Variation of load in the system when background load takes up the whole resource. Right: Supercomputing Grid. Variation of load in the system when resources are heterogeneous.

performance improves in a situation where the resources are either completely available or unavailable (SP4).

C.2 Tasks with the Same Priority: Supercomputing Grid

Finally, we examine a *Supercomputing Grid* infrastructure which is characterised by higher resource heterogeneity, higher number of resources, and higher communication delays than computational clusters. Compared to the PC Grid infrastructures, the only difference is the scheduling policy.

C.2.1 Resource Heterogeneity

First, we examine different amounts of load in the system (LV) when resource heterogeneity is high (RD2, i.e. $f_{Speed,min} = 0.0015625$). The parameters of the experiment are given by {T1, SP3, C1, SN1, RD2, LV, BG2, TS2, BS1}. The results are shown in Figure 80 (right). For the whole range of loads, CDA performs much better than RR.

Conclusion: With heterogeneous resources, RR performs worse than CDA, now even for lower loads. The differences are much larger than for the PC infrastructures.

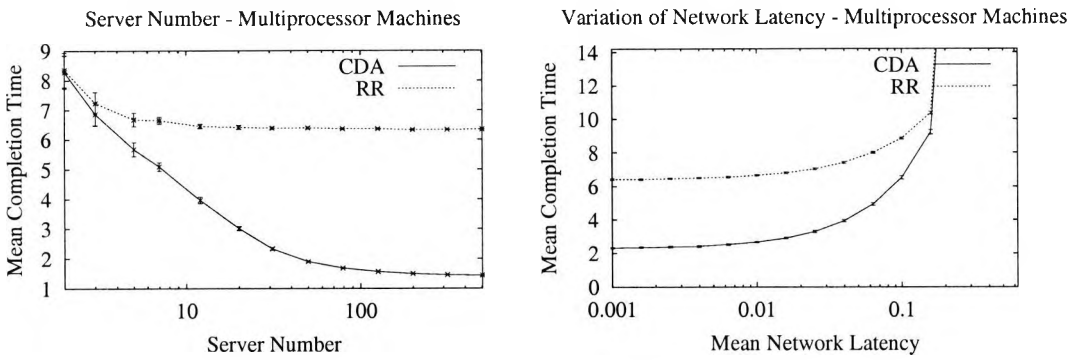

Figure 81: Supercomputing Grid. Left: Variation of the Server number. Right: Variation of the network latency.

C.2.2 Different Server Numbers

Next, we vary the number of Servers, N_{Serv}. As parameters we choose {T1, SP3, C1, SNV, RD2, L1, BG2, TS2, BS1}, i.e. the total load is fixed at 90% (L1) and resources are heterogeneous (RD2). The results are shown in Figure 81 (left). As already observed in Figure 17 (right), the mean completion time of the protocols improves as N_{Serv} is increased. However, now the mean completion time of Round-Robin remains at a very high level (> 6). This can only be explained by the fragmentation of the space-shared resources. Since in Round-Robin resources are allocated arbitrarily, it is very likely that tasks are sent to resources where already tasks are being executed. Their resource share will remain constant until their completion, leading to longer execution times.

Conclusion: Like in the *PC Grid*, the performance of CDA improves considerably with increased number of Servers. The difference is that now, RR's mean completion time remains at a high value, and the gap between the two protocols becomes wider.

C.2.3 Communication Delays

As for the PC infrastructures, we also examine the impact of network latency. The simulation parameters of the experiment shown in Figure 81 (right) are given by {T1, SP3, C2V, SN1, RD2, L1, BG2, TS2, BS1}. Except for the scheduling policy they are the same as in Figure 18 (left).

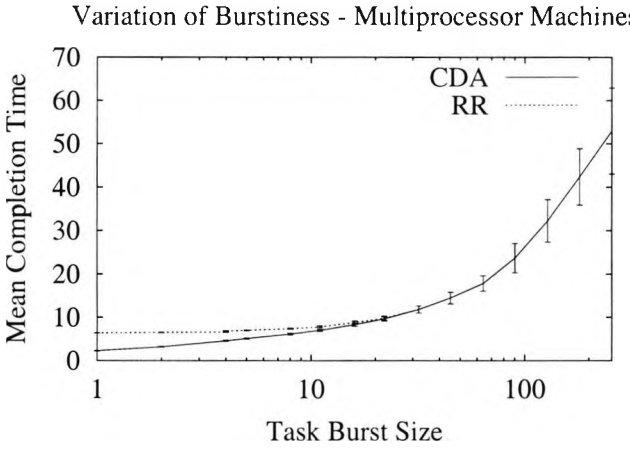

Figure 82: Supercomputing Grid: Heterogeneous resources. Variation of the task burst size.

A sharp rise of the mean completion time can be observed for RR and for CDA, when the latency is increased over 0.1. Note that this rise occurs for a higher latency than in Figure 18 (left). The reason could be that due to the fragmentation of resources, the Servers are less likely to remain completely idle, resulting in fewer resources being wasted.

Conclusion: Regarding variations of the communication delays, similar observations could be made as for the PC infrastructures.

C.2.4 Task Burstiness

Finally, we examine the effect of changing the task burst size, *BS*. As parameters of the experiment we use {T1, SP3, C1, SN1, RD2, L1, BG2, TS2, BSV}. These are the same as in Figure 80 (right), except that now the total load is fixed at 90%, while the burst size is varied. The results in Figure 82 show that with increased *BS* the mean completion time increases. This observation is similar to that in Figure 14 (left). Note that the gap between RR and CDA decreases when *BS* is increased.

Conclusion: Varying the task burst size leads to similar results as in the PC infrastructures.

C.3 Tasks with Different Priorities: PC Cluster

C.3.1 With Background Load

We examined the PDA protocol for the experiment defined by the parameters {T2, SP2, C1, SN1, RD1, LV, BG2, TS2, BS1} which is described in subsection 9.1.2. In Figure 83 (left), the results of PDA with different time intervals δt between the transactions are compared to CDA's results. In none of the cases any improvement could be observed. As expected, a very small $\delta t = 0.01$ leads to almost the same result as CDA. The same observations have been made with fine-grained background load, for which the results are shown in Figure 83 (right).

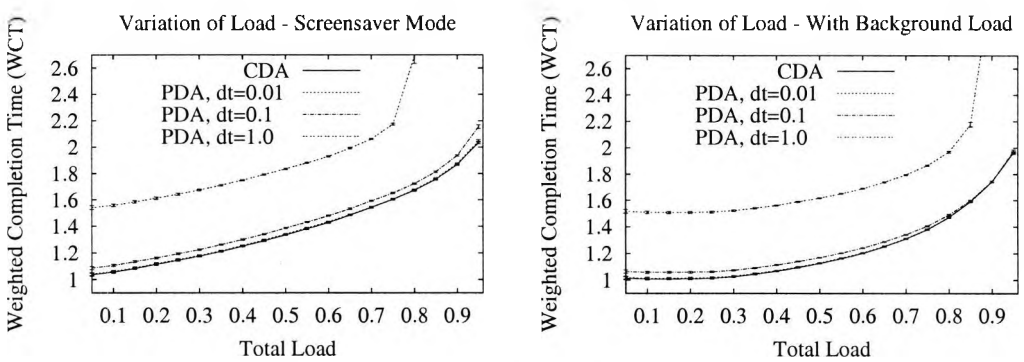

Figure 83: Tasks with different priorities: PDA with different time intervals between the transactions. Left: Coarse-grained background load (screensaver mode, SP2). Right: Fine-grained background load (SP1).

C.3.2 More Background Load: Fine-Grained Background Load

The parameters chosen for the experiments in Figure 84 (left and right) are almost the same as in Figure 24. The only difference is that now the average background load amounts to 75% of the average total load in the system (BG3). The difference between HBP and the other protocols is now smaller. For low and moderate loads HBP outperforms Round-Robin. Also, PSP's performance is now closer to that of CDA.

Conclusion: As the share of background load in the system is increased, HBP and PSP perform relatively better (than before) when compared to the other protocols.

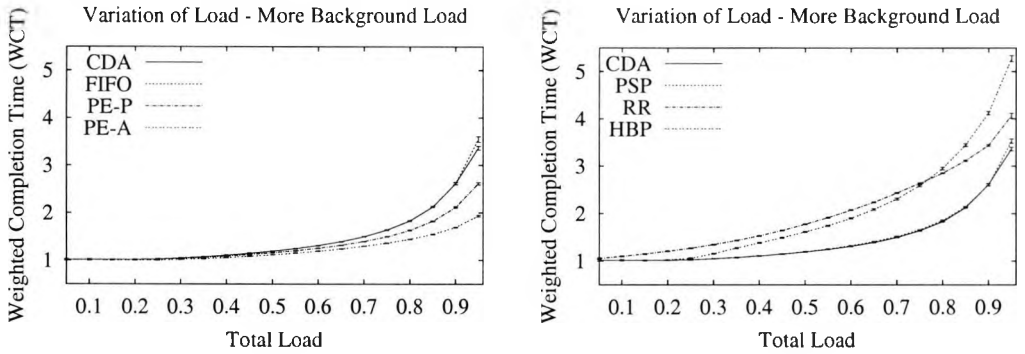

Figure 84: Tasks with different priorities: Variation of load, more (fine-grained) background load.

C.4 Tasks with Different Priorities: PC Grid

C.4.1 Resource Heterogeneity: Screensaver Mode

In Figure 85, some additional results are given for the experiment which has been described in subsection 9.2.1.

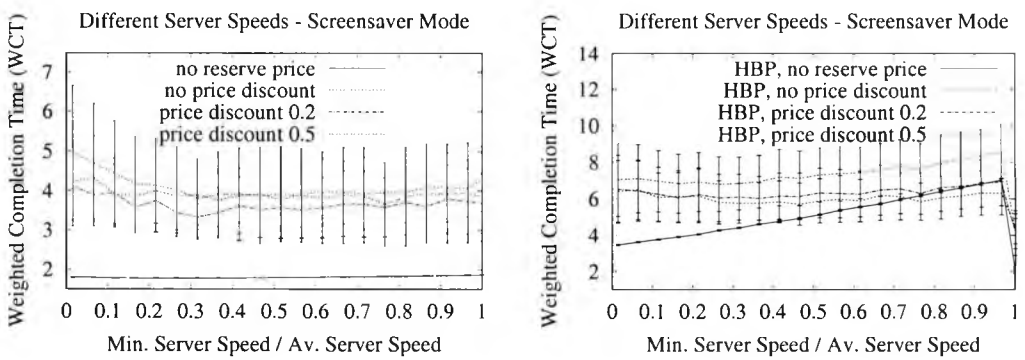

Figure 85: PC Grid: Variation of resource heterogeneity, screensaver mode. Left: CDA-RES with different price discounts. Right: HBP-RES with different price discounts.

C.4.2 Resource Heterogeneity: Fine-Grained Background Load

In Figure 86, we provide the results of HBP, FIFO, and the preemptive protocols for the experiment described in subsection 9.2.2.

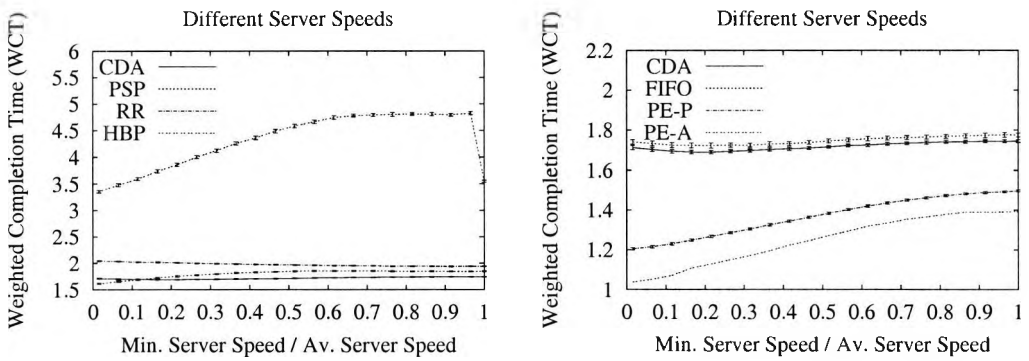

Figure 86: PC Grid: Variation of resource heterogeneity for fine-grained background load. Left: Results for HBP. Right: Results for CDA, FIFO, PE-P, and PE-A.

C.4.3 Variation of Load: No Background Load

In this experiment, we use the same parameters as in subsection 9.2.4, except that there is no background load in the system. The parameters of this experiment are given by {T2, SP1, C1, SN1, RD2, LV1, BG0, TS2, BS1}. The results are shown in the Figures 87 and 88.

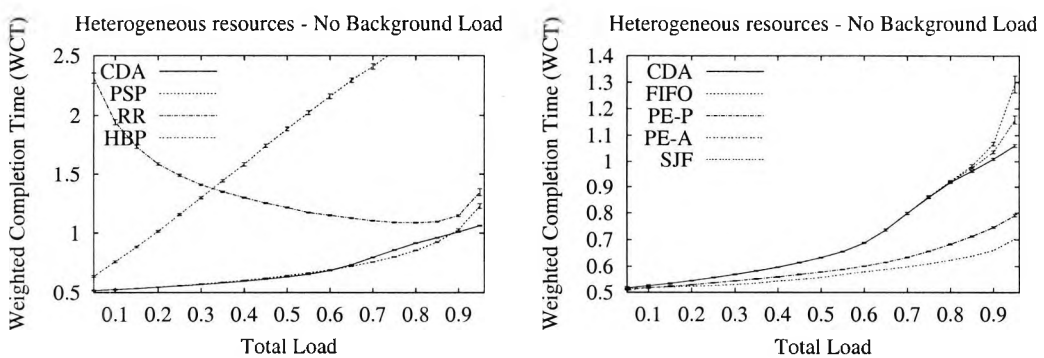

Figure 87: Variation of load for heterogeneous resources, no background load.

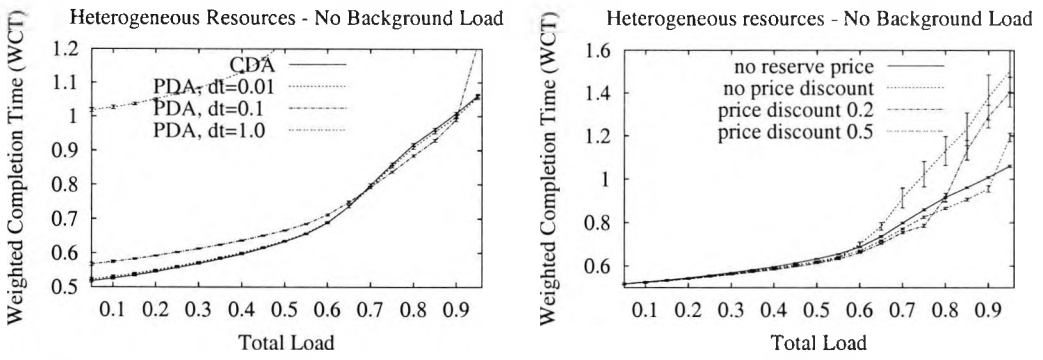

Figure 88: Variation of load for heterogeneous resources, no background load. Left: Results for PDA with different intervals between the transactions. Right: CDA-RES with different price discounts.

C.4.4 Variation of Load: Screensaver Mode

Figures 89 and 90 show additional results to the experiments which have been described in subsection 9.2.3.

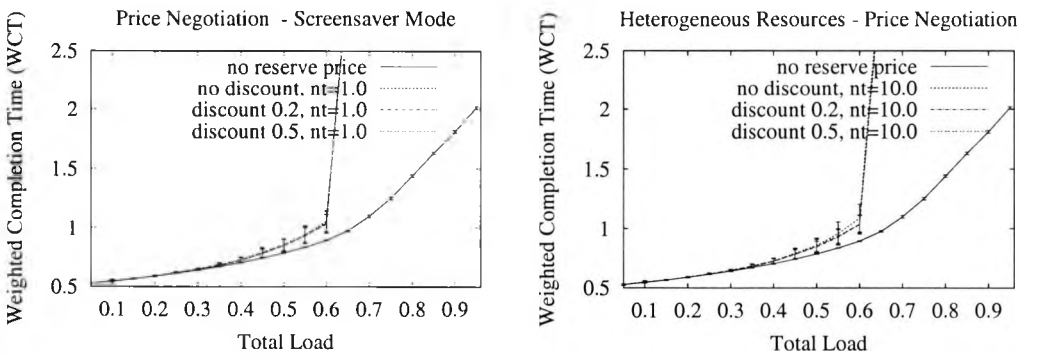

Figure 89: Screensaver mode and heterogeneous resources: CDA-RES with different price discounts. Left: Task negotiation time 1.0. Right: Task negotiation time 10.0.

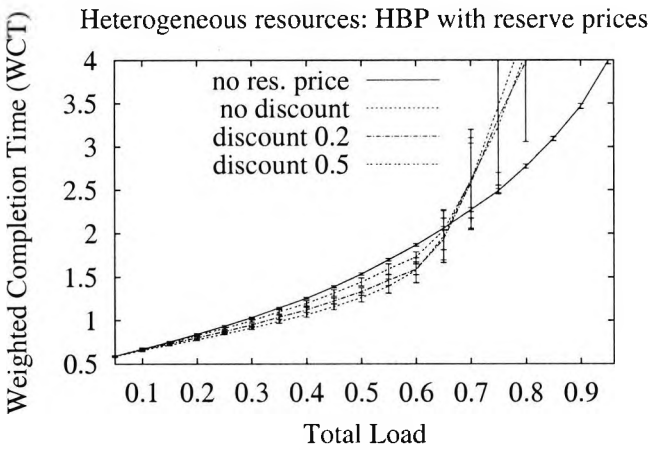

Figure 90: Variation of load for heterogeneous resources: HBP-RES with different price discounts.

C.4.5 Variation of Load: Fine-Grained Background Load

In Figure 91, we give additional results to the experiments which have been described in subsection 9.2.4.

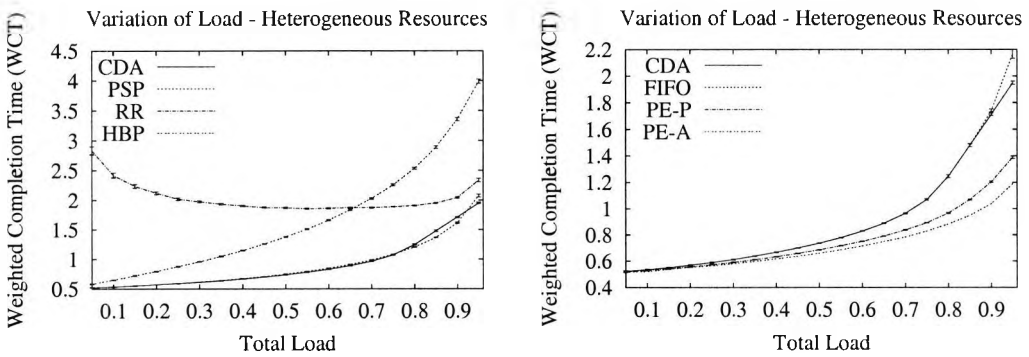

Figure 91: Fine-grained background load: Variation of load for heterogeneous resources.

C.5 Tasks with Time-Dependent Priorities: PC Cluster

C.5.1 PC Cluster

Figures 92 to 94 show additional results for the situations which have been examined in section 10.2.

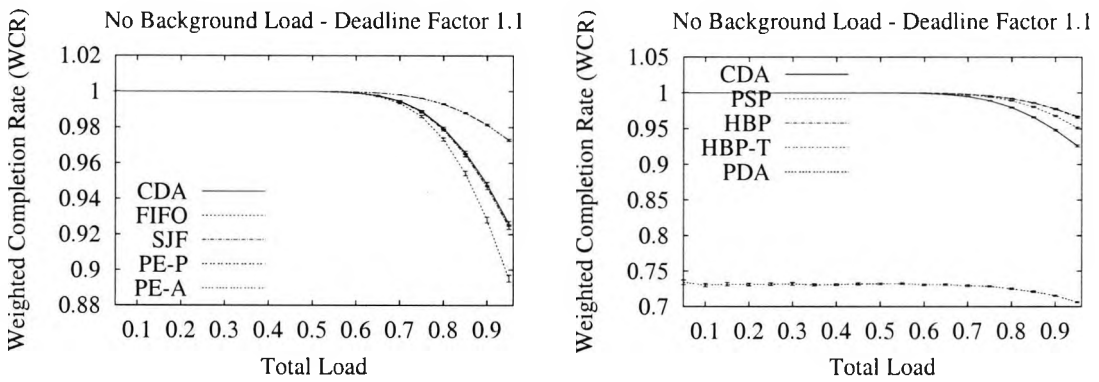

Figure 92: Variation of load, no background load. Tight deadlines (deadline factor 1.1).

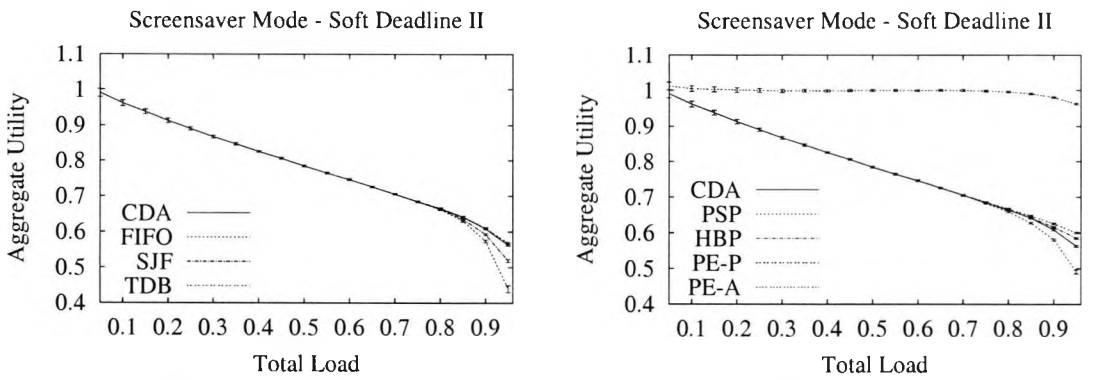

Figure 93: Variation of load, screensaver mode: Soft deadlines II (moderate).

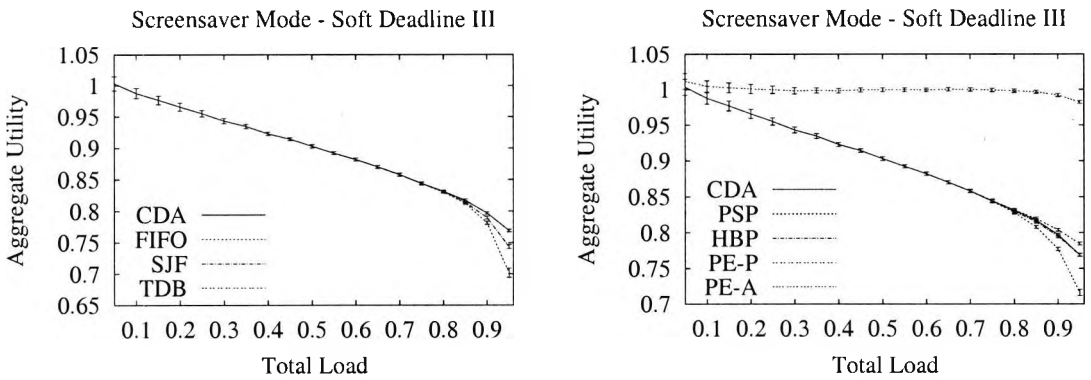

Figure 94: Variation of load, screensaver mode: Soft deadlines III (loose).

C.5.2 PC Grid: Variation of Load

In this subsection, we show some additional results for the experiments which are described in subsection 10.3.2.

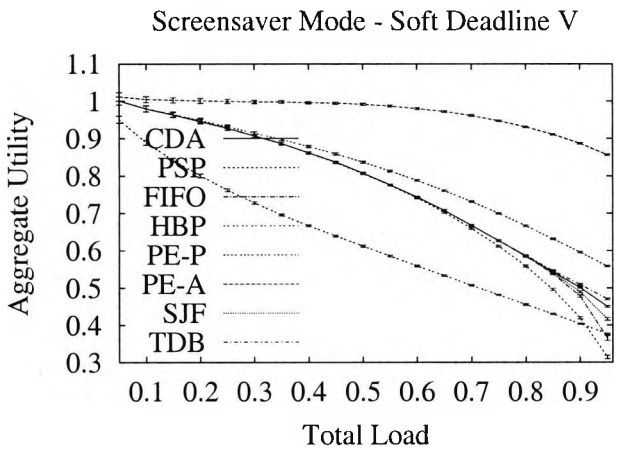

Figure 95: Variation of load, screensaver mode: Soft deadlines V (moderate).

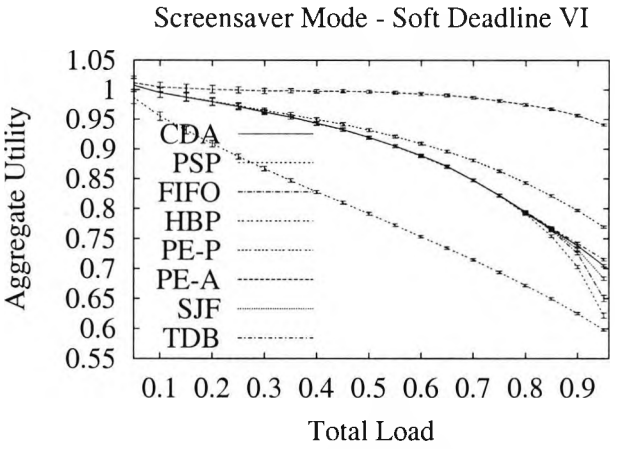

Figure 96: Variation of load, screensaver mode: Soft deadlines VI (loose).

C.5.3 PC Grid: Different Server Numbers

Here, we show some additional results for the experiments where the number of Servers in the system is varied (see subsections 10.3.3 and 10.3.4).

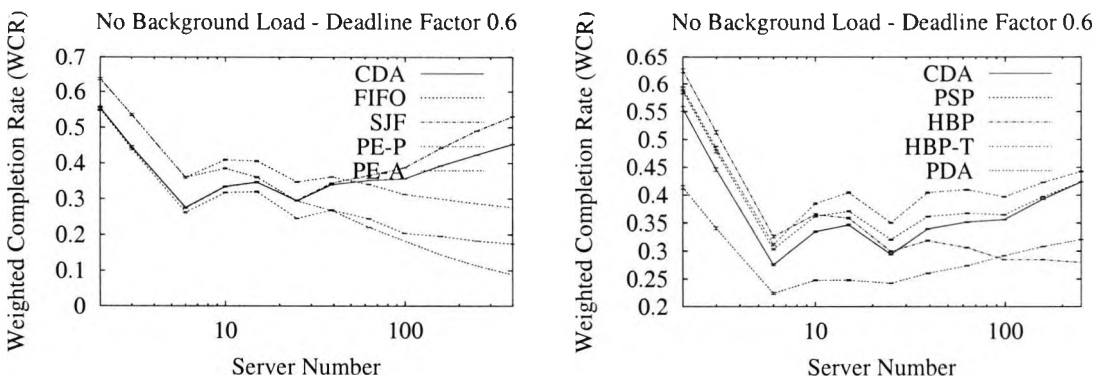

Figure 97: Variation of the Server number, no background load: Tight deadlines (deadline factor 0.6).

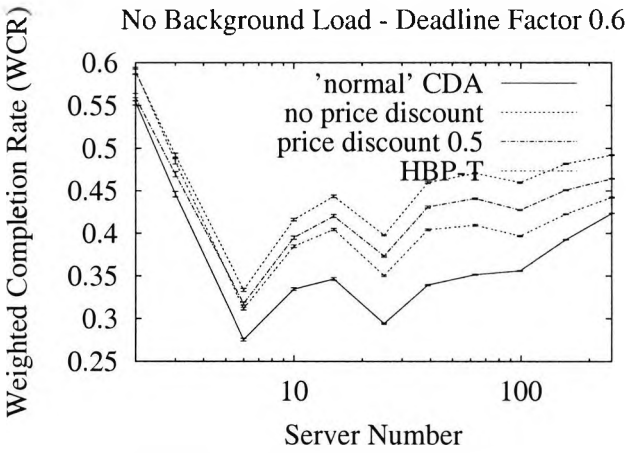

Figure 98: Variation of the Server number, no background load: Results of CDA-RES with different price discounts when tight deadlines are used (deadline factor 0.6).

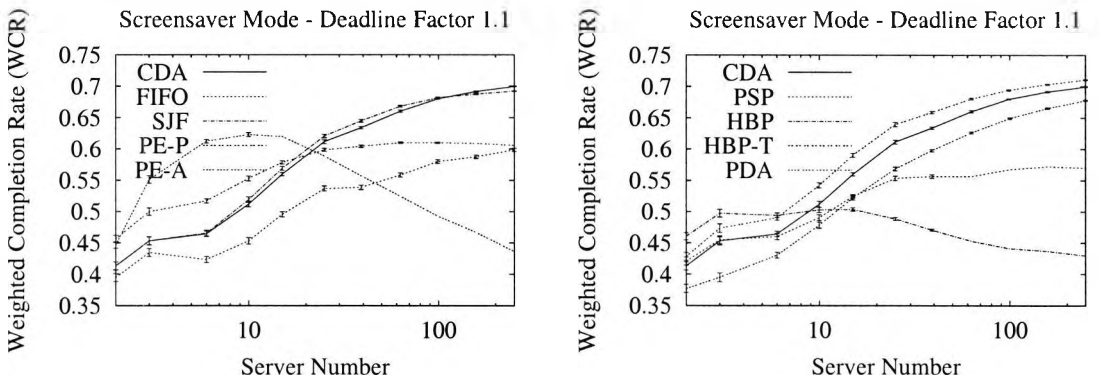

Figure 99: Variation of the Server number, screensaver mode: Moderate deadlines (deadline factor 1.1).

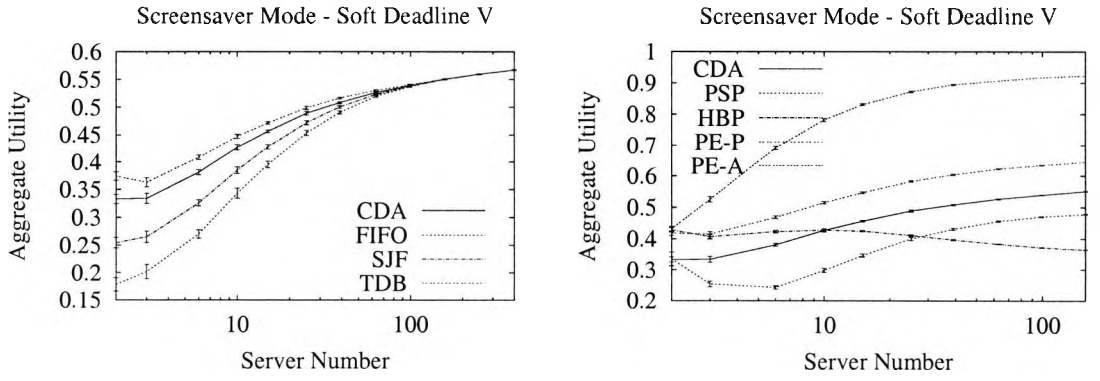

Figure 100: Variation of the Server number, screensaver mode: Soft deadlines V (moderate).

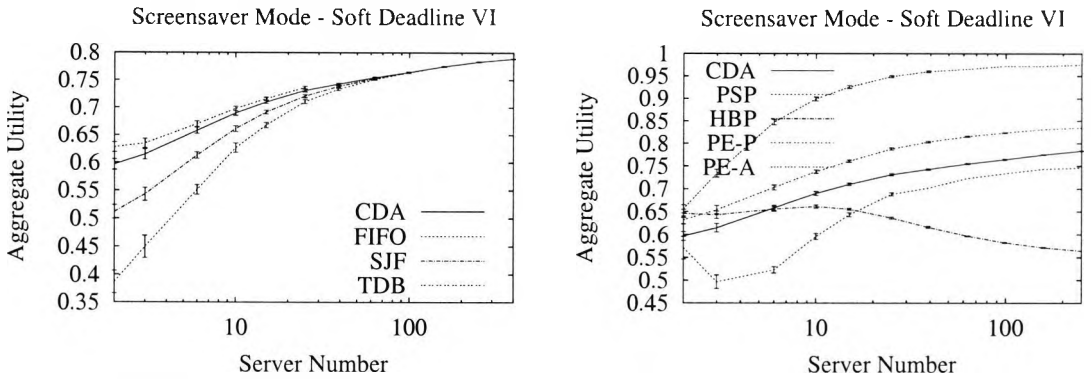

Figure 101: Variation of the Server number, screensaver mode: Soft deadlines VI (loose).

C.5.4 PC Grid: Communication Delays

Next, we provide some more results for the experiments where communication delays are varied (see subsection 10.3.5).

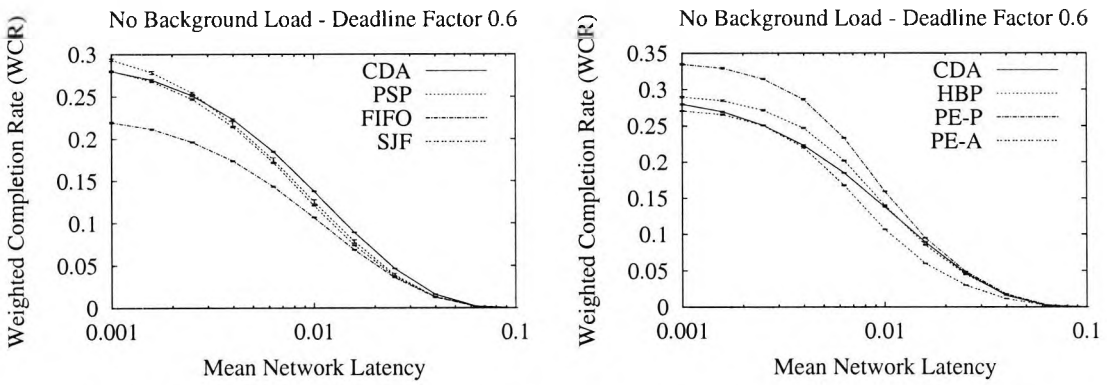

Figure 102: Variation of the communication delay, no background load: Tight deadlines (deadline factor 0.6).

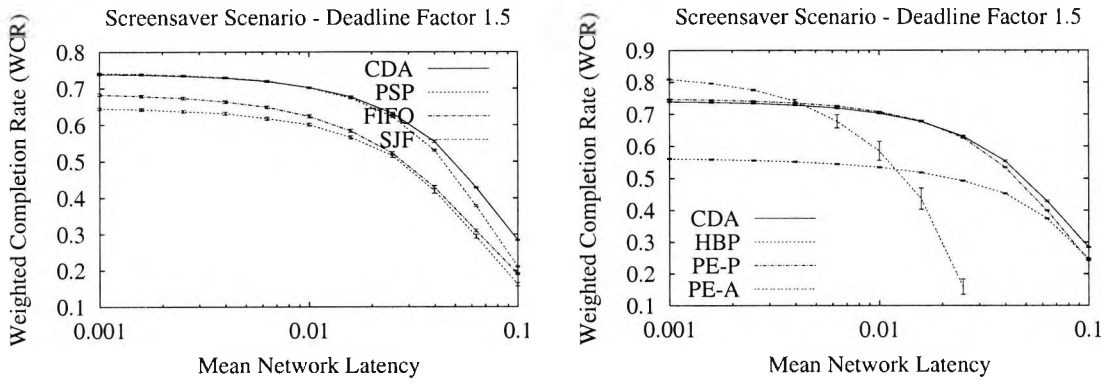

Figure 103: Variation of the communication delay, screensaver mode: Loose deadlines (deadline factor 1.5).

D Experimental Framework: Additional Information

This part of the appendix provides additional information about the implementation and operation of our experimental Grid computing framework which has been used in thesis.

Like the simulation framework, our experimental framework comprises several Java packages: a *Base* package, a statistical package, a random number generation package, and several packages which are specific to particular protocols. The statistical package and random number generation package are almost identical to the corresponding packages in the simulation framework which are described in chapter B.

D.1 Base package

Most of the classes in the *Base* package have exactly the same purpose as the corresponding classes in the simulation framework. These include: *Account*, *Bank*, *ClientBase*, *EMPBase*, *EMPQueryResult*, *EMPResourceAd*, *Helper*, *MethodInvocation*, *Parameters*, *Resource*, *ServerBase*, *Statistics*, *StatisticsRecorder*, *TaskData*, and *TaskObject*.

These classes are described in section B.4. They had to be adapted because the actors in our Grid framework have to communicate via facilities provided by the JADE platform rather than putting messages into the event queue of a discrete-event simulation package. In addition to the above classes, the *Base* package includes the following other classes:

- *ClassData*: stores the bytecode of classes. It is *Serializable* and therefore can be transported in a message.
- *FutureInvocationBehaviour*: a *CommEntity*'s behaviour which provides the means for executing a method on that *CommEntity* after a specified delay.
- *LANRunBase*: base class used for wrapping up a Jade platform which is used for starting up an experiment. It needs to be extended by a protocol-specific class.
- *MyClassLoader*: a class loader which has been designed for passing bytecode inside a Jade message for execution on a remote Server.

- *ProcessRequestBehaviour*: a *CommEntity*'s behaviour which will process messages in the background, which have been received by Jade's *ReceiverBehaviour*. Whenever a *MessageInvocation* object is received, it is executed immediately. This behaviour never terminates.

- *ReaderThread*: a thread which is used for monitoring the CPU utilisation of a Server's resource and which is running in the background. The monitoring requires a Unix or Linux system which provides the *vmstat* command.

- *RemoteStartup*: used for creating agents at remote machines and receiving parameters that are needed for the experiment. It is also used for synchronising the measured simulation time at the beginning of an experiment.

- *StartupAgent*: base class for a Jade agent which starts up an experiment. It needs to be extended by a protocol-specific class.

- *TaskResult*: contains the result of the task after its execution on a remote Server. It also contains the address of its Client and the task's ID.

D.2 Protocol-specific Packages

The functionality, which is specific to a particular protocol, is contained in a separate package. In our experiments, we have been using the *GCDA* package [63]. It contains the following protocols: Continuous Double Auctions (CDA), Continuous Double Auctions with time-dependent price bids (CDA-TDB), PRIO-FIFO, and Shortest Job First (SJF). The package includes the classes *Client*, *EMP*, *Server*, and other classes, which are used for starting up the experiment. Among the latter, the most important ones are:

- *LANRun1:* telnets to remote machines and spawns new processes. On each of these machines, it executes *LANRunRemote*. *LANRun1* requires the *expect* package [Libes, 1994] to be installed in order to run the login scripts.

[63] GCDA stands for Generalised Continuous Double Auction Protocol.

- *LANRunRemote:* needs to be run at the remote machines which are used in the experiment. It starts a Jade platform on which a *RemoteStartup* agent is created.

- *LANStartupAgent:* starts the experiment at the local machine. This class is started up by *LANRun1*.

D.3 Benchmark package

The framework also provides a *benchmark* package which contains the classes used for the performance measurement at the Servers. It also includes the computational task which is used in our experiments. The *Benchmark* class is used for measuring the speed of the Server resource (in millions of floating point operations per second, MFLOPS). It executes a sparse-matrix multiplication that is provided by additional classes which are part of the SciMark2 benchmark [Pozo and Miller, accessed in 2003]. The ModifiedBenchmark class is an adaptation of the Benchmark class which is designed to execute a sparse matrix multiplication whose computation size can be specified in MFLOPS*milliseconds. Hence, given the size of the computation, it is possible to tell how long the computation will run on a Server resource whose MFLOPS figure is known. The *ModifiedBenchmark* class is needed for the experiments which are designed to verify our simulation results.

D.4 Interface for a Computational Task

In order to use a computational task in our framework, it needs to implement the interface shown in Figure 104. When the task is executed on a remote machine, the method *executeTask* will be called. The input parameters of the task are wrapped up in a *Serializable* object which needs to be passed as argument to this method. After the task's execution, the method *getTaskResult* will be called by the Server. It will return the result object, which will then be sent to the *Client*.

Note that there are other parameters, which are needed for a task's execution, and which have to be specified when creating a *TaskData* object at the *Client*. These are the name of the main class of the task (which implements this interface) and a list of all other classes which

```
public interface ComputationalTask {
   /** Execute a task
    *  @param parameters   the task's input parameters */
   void executeTask(Serializable parameters);

   /** Get the task's result after its execution
    *  @return Serializable   the task's result */
   Serializable getTaskResult();
}
```

Figure 104: Interface for the computational task to be implemented

are needed for its execution. This list is only required if the classes need to be loaded to the remote machine.

D.5 Interface for a Parameter Sweep Application

This interface is designed for parameter sweep applications. Such applications need to execute the same code many times with different parameters. Therefore, they can easily be split up into many smaller tasks which can be distributed and executed independently. To run such an application — which we refer to as '*super task*' — it needs to implement the *ComputationalSuperTask* interface whose code is shown in Figure 105.

To split up the super task, the method *divideSuperTask* needs to be called in which the parameter *n* specifies how many tasks are created. The method returns an array of *Serializable* objects, each of which wraps up an individual task's input parameters. To execute a task, both the code and its specific input parameters need to be sent to the remote Server. When the Client receives the result of a task's execution, it needs to be passed to the method *collectTaskResults*.

D.6 Specifying Task and Resource Constraints

Our framework allows to specify constraints and goal functions when matching tasks to resource offers. A task's query is wrapped up by a *TaskObject*, and a resource offer at the EMP is represented by an *EMPResourceAd* object. These classes can be extended: By

```
public interface ComputationalSuperTask {
   /** Return an array of n serializable objects which
    *  wrap up the input parameters of the tasks.
    *  @param n    number of tasks
    *  @return Serializable[]    input parameters of all tasks */
   Serializable[] divideSuperTask(int n);

   /** Collect the result of a task
    * @param result    result of the task
    * @param i         ID of the task
    * @return int    number of tasks completed so far */
   int collectTaskResults(Serializable result, int i);

   /** OPTIONAL: Get the computation size of a Task
    *  @param n          number of tasks in the super task
    *  @return double    task computation size in MFLOPS*ms */
   double getTaskComputationSize(int n);
}
```

Figure 105: Interface for a parameter sweep application

overriding the methods described below, a task's or resource's constraints can be customised.

In the following, we will describe a mechanism of our framework which deals with task and resource constraints:

Checking if a query is still active

For each *TaskObject*, which is processed by the EMP, it needs to be determined whether the query is still active or whether it has expired (e.g. due to the task's deadline which may have passed). This is done by calling its method *taskConstraintsDiscard* which will return 1, if the task should be removed from the EMP, and 0 otherwise.

Checking if a resource satisfies the task's constraints

Each *EMPResourceAd*, which is considered for execution of a task, needs to be passed as parameter to the method *taskConstraints* of the *TaskObject*. This method determines whether the resource offer satisfies the task's constraints. These constraints may express the requirement of a particular machine architecture, operating system, or software package — or the resource's capability to meet the task's deadline. As result, the method will return two arguments: The first one indicates whether the constraints are satisfied (1:yes, 0:no). The other

argument may be used for passing results which can be used as a shortcut in the further processing by the methods described below. It may help to avoid repeated computations of the same parameter.

Determine the task's goal function for a resource

If there exist several *EMPResourceAds* which satisfy a task's constraints, it needs to be determined which one is *best*. This is done by evaluating it with respect to a task's *goal function* (i.e. preference). The TaskObject's method *taskGoalFunction* is called, passing the *EMPResourceAd* as parameter, together with additional parameters that have been returned by the method *taskConstraints*. The method *taskGoalFunction* will return the value of the goal function of this task for the case that it is executed on that resource. It may depend on price, speed, or any other parameters.

Determine a Server's goal function for a task

When several tasks are competing for a resource which satisfies their constraints, it needs to be determined which one is *best* from the Server's perspective (e.g. which one pays the highest price, is the largest, etc.). This is achieved by calling the method *serverGoalFunction* of the *EMPResourceAd* object for each task that is checked. The *TaskObject* and further parameters, which may have been returned by the method *taskConstraints*, are passed to *serverGoalFunction* as arguments. The result is the Server's goal function which needs to be maximised.

D.7 Random Numbers, Statistics, and Parameter Variation

The generation of random numbers and measurement of statistics is almost the same as in the simulation framework described in chapter B. One difference is that the Statistics, which are recorded on remote machines, need to be passed back after the experiment.

As in the simulation framework, it is possible to carry out experiments with parameter variations. In this case the experiment is repeatedly run while one or more of its input variables are varied (see section B.7). Several specified variables can be measured.

D.8 Running an Experiment

This section provides a brief description of how the experiments with our Grid framework are carried out.

Reading the experimental parameters

An experiment is started by executing the *LANRun1* class of the GCDA package. In the command line, the name of the file needs to be specified, from which the values for the input parameters will be read. The values read from that file are written into variables in the *Parameter* class and will be used in the experiment. The input file looks similar to the one used in our simulation framework (see chapter E).

Creating platforms on remote machines

Next, the names of the remote machines, on which the Servers will be run, are read from another file. A script written with the *expect* package [Libes, 1994] is used to log onto each of these remote machines. A Jade platform is started there, and a *RemoteStartup* agent is created.

Initialising the local platform

LANRun1 creates a *LANStartupAgent* on the local machine. It has a similar function as the *Startup* object in our simulation framework (see chapter B): It initialises an output trace which allows to log the events and values of the variables during the experiment. This *LANStartupAgent* initialises the *StatisticsRecorder* object which will record the values of important variables at periodic intervals, writing them to a file. Then, all actors that will run on the local machine are created: the *EMP*, the *Bank*, and the *Client*.

Initialising the remote platforms

The parameters of the experiment are also sent to all *RemoteStartup* agents which reside on the remote machines. Like the *LANStartupAgent*, each *RemoteStartup* agent initialises an output trace and a *StatisticsRecorder* on its machine. After a small delay, all *RemoteStartup* agents are requested to create the *Server* agents. Each *Server* agent sends a message to the

Bank to open its account. It also starts monitoring its CPU utilisation (in %) which is done by using the Unix command *vmstat*. Next, the Server uses the *Benchmark* class to measure its speed in MFLOPS*ms. Finally, it registers its resource offer (speed, availability, and price) at the EMP.

Starting the experiment

The *LANStartupAgent* waits for a long enough time in order to allow the speed measurements at the Server resources to finish. After this delay, all agents are started. Those running on remote machines are activated by notifying their *RemoteStartup* agent. At the start of the experiment, the local time of each agent is synchronised by recording the current value of the system clock at its machine. This value will be used as an *offset* for all future time measurements (i.e. for all measurements, this offset will be deducted from the value of the system clock). Since this *reset* of the timer is carried out at approximately the same moment for all agents in the system, their time measurements will be comparable throughout the experiment. Also, the termination time of the experiment is scheduled. On the local machine, this is done by the *LANStartupAgent*, and on the remote machines by the *RemoteStartup* agents. Furthermore, the Poisson arrival process for the task generation by the *Client* is activated.

Completing the experiment

At the termination event, the *RemoteStartup* agents send the recorded statistics to the local machine. All statistics, that have been collected locally and at the remote machines, are made available to the user by printing them to the screen (or an output file).

E Example of an Input File

Finally, we give an example of an input file which specifies the parameters used by our simulation framework which is described in chapter B. A similar input file is used by our experimental Grid Computing framework (see chapter 11).

```
//INPUT/OUTPUT FILES:
//-------------------

topologyFileName topology.txt       // topology data file name
loadTraceFileName loadtrace.txt     // load trace data file name
traceFileName  outtrace.txt         // trace of the experiment
recorderFileName  outrecorder.txt   // record of time-dependent variables
variablesFileName invar.txt         // to be used for series of experiments
variablesOutputFileName outvar.txt  // to be used for series of experiments
autotraceTag 0                      // enable/disable Simjava's (internal)
                                    // trace

//-----------------------------------------------------------------------

//SIMULATION PARAMETERS:
//----------------------

initDelay 0.000000000001            // there must be a delay between the
                                    // startup of the entities to avoid
                                    // randomness of the scheduler

minimalDelay 0.000000000001         // something similar; used for the
                                    // taskPriceAdjustmentEvent()

initialSeed 1020775884309           // one simulation run should also be
                                    // carried out with other seeds

statisticsRecorderInterval 7.7777   // interval for recording
statistics


runLength 1000.0                    // duration of the experiment

initialRecordingMargin 100.0        // statistics should only be recorded
                                    // after an initial margin - before the
                                    // system gets to its steady state

finalRecordingMargin 50.0           // the creation/completion of task
                                    // executions should only be recorded
                                    // in the statistics if the tasks are
```

```
                                    // started well before the end of the
                                    // experiment
                                    // (t < runLengh-finalRecordingMargin )

fixedTaskNumberTag 0                // tag indicating whether as many
                                    // tasks are recorded as would
                                    // (statistically) be expected in the
                                    // recording interval
                                    // (0: disabled, 1: enabled)

//--------------------------------------------------------------------------

//SYSTEM PARAMETERS:
//------------------

scenarioTag 0                       // tag indicating the type of scenario
                                    // 0: independent tasks, 1 resource type
                                    // 1: tasks with pipeline structure
                                    //    and different resource types

numberResourceTypes 8               // only for scenarioTag=1:
                                    // number of resource types

clientNumbers 1                     // since all tasks (or supertasks) are
                                    // treated independently, it does not
                                    // matter by how many clients they are
                                    // created

serverSpeedRatioMin 0.1             // minimum speed ratio of the Servers
serverSpeedRatioMax 0.1             // maximum speed ratio of the Servers

serverNumbers 10                    // number of servers

serversResourceMin 10               // minimum number of resource units
                                    // per Server

serversResourceMax 10               // MUST be the same as the min value

serverRandomTag 0                   // indicates whether the resource size
                                    // and/or speed ratio distributions are
                                    // random (1) or deterministic (0)

resourceSchedulingTag 3             // scheduling policy
                                    // 2: space-shared
                                    // 3: time-shared
                                    // 4: proportional sharing
                                    // 5: suspension of tasks enabled
                                    // 6: preemption of tasks enabled
```

```
networkDelayTag 0                    // tag indicating whether the topology
                                     // file is used (1) or whether the
                                     // network links between all actors
                                     // are identical (0)

latencyScalingFactorB 1.0            // scaling factor for the mean of the
                                     // network latency
                                     // Means: the values for 'meanB' will
                                     // be multiplied by this value

latencyScalingFactorD 1.0            // scaling factor for the stdev of the
                                     // network latency  ('stdevD')

throughputScalingFactorA 1.0         // scaling factor for 'meanA'
                                     // (inverse of the throughput)

throughputScalingFactorC 1.0         // scaling factor for 'stdevC'
                                     // (inverse of the throughput)

minValueScalingFactor 1.0            // scaling factor for the min value of
                                     // the communication delay

networkDistTag 0                     // tag indicating whether network
                                     // delays are deterministic or random
                                     // (0: deterministic. 1: random)

//-------------------------------------------------------------------------

//LOAD RELATED PARAMETERS:
//------------------------

taskSizeMin 1.0                      // min. computational size of a task
taskSizeMax 1.0                      // max. computational size of a task

taskDataSize 0                       // in this scenario
resultDataSize 0                     // no large data transfers

taskDeadlineTag 0                    // tag indicating whether deadlines are
                                     // used (0: no, 1: yes)

taskDeadlineFactorMin 100.0          // minimum deadline factor
taskDeadlineFactorMax 100.0          // maximum deadline factor

totalLoadRatio 0.8                   // total average load in the system in
                                     // relation to the system's capacity
                                     // (both Client task load and
                                     //  Server background load)
```

E EXAMPLE OF AN INPUT FILE

```
taskWeightTag 0                    // tag indicating whether task weights
                                   // are used
                                   // 0: all task weights are set to 1.0
                                   // 1: uniform dist between taskWeightMin
                                   //    and taskWeightMax
                                   // 2: bimodal normal distribution,
                                   //    Parameters: taskWeight...
                                   //    MeanLow, StDevLow, MeanHigh,
                                   //    StDevHigh, FractionLow

taskWeightMin 0.0                  // if tag=1: minimum task weight
taskWeightMax 2.0                  //           maximum task weight

taskWeightMeanLow 1.0              // if tag=2: mean value of the lower mode
taskWeightStDevLow 0.5             //           stdev of the lower mode
taskWeightMeanHigh 100.0           //           mean value of the higher mode
taskWeightStDevHigh 50.0           //           stdev of the higher mode
taskWeightFractionLow 0.9          //           fraction of samples belonging
                                   //           to the lower mode

taskBurstSize 1                    // number of tasks in a burst

BGLoadTag 1                        // background load enabled/disabled

BGTaskSize 1.0                     // computational size of a background task

BGAllocatedUnits 1                 // number of resource units
allocated to
                                   // a background task (on which it will
                                   // execute in parallel)

BGLoadRatio 0.45                   // total average background load on the
                                   // Servers in relation to the system's
                                   // capacity

BGLoadFactorMin 1.0                // if Servers have different mean BG load,
                                   // (and the means are uniformly
                                   // distributed): The load ratio of the
                                   // server with the minimal load is set to
                                   // BGLoadFactorMin*BGLoadRatio. In order
                                   // to maintain the average load ratio
                                   // BGLoadRatio, the load factor of the
                                   // server with the highest load is set to:
                                   // BGLoadFactorMax = 2-BGLoadFactorMin.
                                   // If all Servers should have the same
                                   // mean background load, simply set
                                   // BGLoadFactorMin to 1.0
```

```
randomiseExecutionTag 0              // tag indicating whether the actual
                                     // execution time of the task at the
                                     // Server is random (with normal
                                     // distribution):
                                     // 0 (deterministic):
                                     //    actual exec. time = task runtime
                                     // 1 (random): positively truncated
                                     //    normal distribution with
                                     //    mean = task runtime
                                     //    stdev = sqrt(taskRunTimeStDev)
                                     //             * task runtime

taskRunTimeStDev 1.732050808         // if the actual execution time is
                                     // random, this is its standard deviation
                                     // ( this value is normalised to the mean
                                     //   i.e. the real stdev will be
                                     //   mean * taskRunTimeStDev )

loadTraceTag 0                       // tag deciding whether the average
                                     // background load of the Servers
                                     // is set according to the load
                                     // trace read from a file (0:n, 1:y)

loadTraceSize 24                     // if loadTraceTag==1:
                                     // number of samples in the load trace

loadTraceTimeStep 20.0               // if loadTraceTag==1:
                                     // time step between two samples of the
                                     // load trace

// --- FOR scenarioTag==1 (supertask-scenario) -----------------------

meanTaskNumber 10                    // for scenarioTag==1:
                                     // mean number of tasks per supertask
                                     // (exponential distribution)

minTaskNumber 1                      // for scenarioTag==1:
                                     // min. number of tasks per supertask
                                     // (exponential distribution)

maxTaskNumber 100                    // for scenarioTag==1:
                                     // max. number of tasks per supertask
                                     // (exponential distribution)

//--------------------------------------------------------------------------

//PROTOCOL-SPECIFIC:
```

E EXAMPLE OF AN INPUT FILE

```
//------------------

clientMoney 100000000000.0          // Endowment of the Client. this amount
                                    // does not matter much at the moment
                                    // (all tasks are autonomous)

deadlineSafetyMargin  0.0           // when using task deadline:
                                    // the safety margin to be used by the
                                    // EMP when deciding whether a resource
                                    // will complete it on time. This is to
                                    // consider communication delays.

SAPricingTag 1                      // how Servers adjust their prices
                                    // 0: no price adjustment
                                    // 1: adjustment according to current load
                                    // 2: adjustment according to past revenue

                                    // for SAPricingTag==1:
serverUnitPriceMax 0.0              // the maximum Server price
serverUnitPriceMin 0.0              // These values are used to obtain the
serverReservationPriceMax 0.0       // distribution of minimum prices
                                    // (reservation prices) of the Servers.

serverPriceUpdateTimeStep 1.0       // for SAPricingTag==2:
                                    // interval between the price updates of
                                    // a Server

serverPriceDiscount 0.0             // for SAPricingTag==2:
                                    // amount to be deducted when calculating
                                    // the reservation price from the past
                                    // revenue

taskPriceAdjustmentTag 0            // 0: no taskPriceAdjustmentEvents
                                    //    (i.e. tasks do not adjust their
                                    //     price bids)
                                    // 1: taskPriceAdjustmentEvents
                                    //    IN ADDITION TO immediate processing
                                    //    of tasks or resource updates
                                    // 2: FOR CDA: taskPriceAdjustmentEvents
                                    //    WITHOUT immediate processing of task
                                    //    or resource updates
                                    // 3: FOR CDA: no immediate processing,
                                    //    i.e. offers and bids are matched at
                                    //    periodic intervals ONLY, HOWEVER,
                                    //    WITHOUT price adjustment (NOTE: FOR
                                    //    ANY OTHER PROTOCOL THIS TAG WILL
                                    //    DISABLE TASK PRICE ADJUSTMENTS!!!)
```

E EXAMPLE OF AN INPUT FILE

```
identicalPricesTag 0                    // if set to 1: all task price bids are
                                        // identical allowing a shortcut in the
                                        // simulation

taskPriceAdjustmentInterval 0.1         // interval between price
adjustments
                                        // of tasks

resourceSelectionTag 1                  // as a task looking for resources:
                                        // 0: select cheapest resource
                                        // 1: select fastest resource

taskSelectionTag 0                      // as a Server looking for tasks:
                                        // 0: maximise unit price
                                        // 1: maximise overall payment
                                        // 2: maximise task size

predictionTag 0                         // tag indicating that load prediction
                                        // is used (0: no, 1-3: yes).
                                        // 1: window, 2: mean over all values
                                        // 3: mean over all values in each
                                        // interval of a prediction period
                                        // This is relevant to scheduling
                                        // policies 3-5.

predictionUpdateInterval 10.0           // time interval for the updates of
                                        // the load predictions

predictionWeightingFactor 0.5           // if load prediction is used: this is
                                        // the relative weight of the historical
                                        // information for calculating the
                                        // estimate (the remaining weight is
                                        // assigned to the current value of
                                        // the load)

bidPredictionWeightingFactor 0.5        // if load prediction is used: similar
                                        // as above, but instead of the load this
                                        // factor is used for the estimation of
                                        // the sum of price bids (in PSP) or the
                                        // highest price bid (HBP)

loadWindowSize  10                      // for predictionTag==1: the number of
                                        // load samples stored
                                        // for predictionTag==3: the number of
                                        // intervals in a period used for
                                        // prediction

priceBidWindowSize  10                  // for predictionTag==1: the number of
```

```
                                // price bid samples stored
                                // for predictionTag==3: it is set to
                                // the same value as loadWindowSize

predictionIntervalLength 20.0   // for predictionTag==3: length of an
                                // interval within a prediction period
                                // (it makes sense  to set it to the
                                // same value as the loadTraceTimeStep)

minExecutionSpeed 0.0           // RELEVANT TO THE PSP PROTOCOL:
                                // minimum effective execution speed
                                // requested by the tasks at the EMP

endowmentDistributionTag 0      // determines how the endowment of a task
                                // depends on its weight
                                // 0: polynomial: price ~ weight ^ n
                                // 1: exponential: price ~ n ^ weight
                                // (n is the endowmentDistributionOrder)

endowmentDistributionOrder 1.0  // the 'n' (order) for the distribution of
                                // the endowment: e.g. for 'polynomial'
                                // with n=1: price ~ weight (linear)

recalculateSumTag 0             // relevant to the PSP protocol:
                                // indicates whether, on completion of
                                // a task, the Server should recalculate
                                // the sum of price bids (1), or just
                                // deduct the difference from the last
                                // figure (0)

clientUnitPriceMin 1.0          //  base price of the Clients' tasks
clientUnitPriceMax 1.0          //  the actual price bid of a task is
upperPriceLimitMin 1.0          //  calculated by multiplying the base
                                //  price by the task's weight
                                //  (these parameters must be identical)

UCVProtocolTag 0                // for the UCV protocol package:
                                // tag indicating which protocol
                                // is used: (0:CDA, 1:PRIO_FIFO, 2:SJF)

slowA 0.0                       // only for the UCV protocol: if task
                                // slowdown<=slowA, the price bid is zero
slowB 0.0                       // only for the UCV protocol: if
                                // slowA<slowdown<slowB, the price bid
                                // rises from zero to the maximum

slow1 0.1                       // if slowB<slowdown<=slow1, the max
```

```
                                    // value is delivered
slow2 0.2                           // if slowdown>=slow2, nothing is paid

slow2Max -999.9                     // if a uniform distribution is used for
                                    // variable 'slow2' of the tasks:
                                    // slow2 is min value, slow2Max is max
                                    // value (-999 = disabled)

DFactor 0.0                         // only for UCV-CDA protocol:
                                    // factor for the differential component
                                    // (i.e. the speed of decline in value)
                                    // in determining the task's priority

priceQueueA 0.0                     // only for the PRIO-FIFO protocol:
priceQueueB 0.0                     // prices of the three FIFO queues
priceQueueC 0.0                     // (priceQueueA>priceQueueB>priceQueueC)

negotiationTime 0.1                 // if task price adjustment == 1 OR 2:
                                    // the duration until a task reaches
                                    // its maximum price bid

activeMigrationTag 1                // only for PE (resourceSchedulingTag==6):
                                    // allow active migration

speedImprovementFactor 1.0          // only for PE: increase of effective
                                    // speed needed for preemption
bidImprovementFactor 1.0            // only for PE: increase of the bid
                                    // needed for preemption

                                    // if (scenarioTag == 1) AND
                                    // (taskPriceAdjustmentTag == 1 OR 2):
subTaskMinimumFactor 0.75           // factor used for the initial bid of
                                    // subtasks
subTaskMaximumFactor 1.25           // factor used for the final bid of
                                    // subtasks
```

References

[Abraham *et al.*, 2000] A. Abraham, R. Buyya, and B. Nath. Nature's Heuristics for Scheduling Jobs on Computational Grids. In *Proc. of the 8th Intl. Conference on Advanced Computing and Communications (ADCOM 2000)*, Cochin, India, December 2000.

[Abramson *et al.*, 2002] D. Abramson, R. Buyya, and J. Giddy. A computational economy for Grid computing and its implementation in the Nimrod-G resource broker. *Future Generation Computer Systems (FGCS) Journal*, 18(8):1061–1074, October 2002.

[Aida *et al.*, 2000] K. Aida, A. Takefusa, H. Nakada, S. Matsuoka, S. Sekiguchi, and U. Nagashima. Performance evaluation model for scheduling in global computing systems. *Intl. Journal of High-Performance Computing Applications*, 14(3):268–279, Fall 2000.

[Ali *et al.*, 2002] S. Ali, J.-K. Kim, H. J. Siegel, A. A. Maciejewski, Y. Yu, S. B. Gundala, S. Gertphol, and V. K. Prasanna. Greedy Heuristics for Resource Allocation in Dynamic Distributed Real-Time Heterogeneous Computing Systems. In *Proc. of the Intl. Conference on Parallel and Distributed Processing Techniques and Applications, PDPTA '02*, Las Vegas, Nevada, June 2002.

[Amdahl, 1967] G. Amdahl. Validity of the single-processor approach to achieve large scale computing capabilities. In *Proc. of the AFIPS Conference*, pages 483–485. AFIPS Press, 1967.

[Artsy and Finkel, 1989] Y. Artsy and R. Finkel. Designing a process migration facility: the Charlotte experience. *Computer*, 22(9):47–56, 1989.

[Azar, 1998] Y. Azar. On-line load balancing. In *Online Algorithms - The State of the Art*, pages 178–195. Springer, 1998.

[Backschat *et al.*, 1996] M. Backschat, A. Pfaffinger, and C. Zenger. Economic-based dynamic load distribution in large workstation network. In *Proc. of the 2nd Intl. Euro-Par Conference*, volume 2, pages 631–634, Lyon, France, 1996. Springer.

[Baeumer *et al.*, accessed in 2003] C. Baeumer, M. Breugst, S. Choy, and T. Magedanz. Grasshopper - a universal agent platform based on OMG MASIF and FIPA standards. http://www.grasshopper.de/download/doc/GrasshopperAndStandards.pdf, accessed in 2003.

[Baker *et al.*, 2001] M. Baker, R. Buyya, and D. Laforenza. The Grid: A Survey on Global Efforts in Grid Computing. Technical Report 2001/92, Monash University, Melbourne, Australia, May 2001.

[Bellifemine *et al.*, 1999] F. Bellifemine, A. Poggi, and G. Rimassa. JADE - A FIPA-compliant agent framework. In *Proc. of the 4th Intl. Conference and Exhibition on the Practical Application of Intelligent Agents and Multi-Agents (PAAM'99)*, pages 97–108, London, UK, 1999.

[Bernerns-Lee, 1999] T. Bernerns-Lee. *Weaving the Web: The Past, Present, and Future of the World Wide Web by its Inventor.* Orion Publishing Group, 1999.

[Binder *et al.*, 2001] W. Binder, J. Hulaas, A. Villazon, and R. Vidal. Portable Resource Control in Java: The J-SEAL2 Approach. In *Proc. of the Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA'01)*, Tampa Bay, Florida, October 2001. ACM Press.

[Braun *et al.*, 1998] T. D. Braun, H. J. Siegel, N. Beck, L. Bölöni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, and B. Yao. A taxonomy for describing matching and scheduling heuristics for mixed-machine heterogeneous computing systems. In *Proc. of the 17th IEEE Symposium on Reliable Distributed Systems*, pages 330–335, West Lafayette, Indiana, October 1998. IEEE.

[Braun *et al.*, 1999] T. D. Braun, H. J. Siegel, N. Beck, L. Bölöni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, B. Yao, D. Hensgen, and R. F. Freund. A Comparison Study of Static Mapping Heuristics for a Class of Meta-tasks on Heterogeneous Computing Systems. In *Proc. of the 8th Heterogeneous Computing Workshop (HCW'99)*, April 1999.

[Bredin *et al.*, 1998] J. Bredin, D. Kotz, and D. Rus. Market-based Resource Control for Mobile Agents. In *Proc. of the 2nd Intl. Conference on Autonomous Agents*, Mineapolis, USA, 1998. ACM Press.

[Bredin *et al.*, 1999] J. Bredin, D. Kotz, and D. Rus. Economic Markets as a Means of Open Mobile-Agent Systems. In *Proc. of the Workshop on Mobile Agents in the Context of Competition and Cooparation (MAC3) at the Third Intl. Conference on Autonomous Agents AA99*, Mineapolis, USA, May 1999. ACM Press.

[Bredin *et al.*, 2001] J. Bredin, D. Kotz, D. Rus, R. T. Maheswaran, C. Imer, and T. Basar. A Market-Based Model for Resource Allocation in Agent Systems. In A. Omicini, F. Zambonelli, M. Klusch, and R. Tolksdorf, editors, *Coordination of Internet Agents — Models, Technologies, and Applications*. Springer, 2001.

[Bredin, 2001] J. Bredin. *Market-based control of mobile-agent systems.* PhD thesis, Dartmouth College, New Hampshire, USA, 2001.

[Buyya and Murshed, 2002] R. Buyya and M. Murshed. GridSim: A toolkit for the modeling and simulation of distributed resource management and scheduling for Grid computing. *Concurrency and Computation: Practice and Experience (CCPE)*, May 2002.

[Buyya and Vazhkudai, 2001] R. Buyya and S. Vazhkudai. Compute Power Market: Towards a market-oriented Grid. In *Proc. of the 1st Intl. Conference on Cluster Computing and the Grid, CCGrid 2001*, Brisbane, Australia, 2001. IEEE.

[Buyya *et al.*, 2000] R. Buyya, J. Giddy, and D. Abramson. An Evaluation of Economy-based Resource Trading and Scheduling on Computational Power Grids for Parameter Sweep Applications. In *Proc. of the 2nd Workshop on Active Middleware Services*, Pittsburgh, USA, 2000. Kluwer.

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

[Cabrera, 1986] L. F. Cabrera. The influence of workload on load balancing strategies. In *USENIX Summer Conference*, pages 446–58, 1986.

[Casanova *et al.*, 2000] H. Casanova, A. Legrand, D. Zagorodnov, and F. Berman. Heuristics for Scheduling Parameter Sweep Applications in Grid Environments. In *Proc. of the 9th Heterogeneous Computing Workshop (HCW'00)*, pages 349–363, May 2000.

[Casanova, 2001] H. Casanova. Simgrid: a Toolkit for the Simulation of Application Scheduling. In *Proc. of the 1st ACM/IEEE Intl. Symposium on Cluster Computing on the Grid (CCGrid 2001)*, Brisbane, Australia, May 2001.

[Casavant and Kuhl, 1988] T. L. Casavant and J. G. Kuhl. A Taxonomy of Scheduling in General-Purpose Distributed Computing Systems. *IEEE Transactions on Software Engineering*, 14(2):141–154, 1988.

[Chavez *et al.*, 1997] A. Chavez, A. Moukas, and P. Maes. Challenger: A Multi-agent System for Distributed Resource Allocation. In *Proc. of the 1st Intl. Conference on Autonomous Agents*, Marina del Ray, CA, USA, 1997. ACM Press.

[Chien and Konstantinidou, 1994] A. A. Chien and M. Konstantinidou. Workloads and Performance Metrics for Evaluating Parallel Interconnects (Position Paper). *IEEE TCCA Newsletter*, Fall 1994.

[Christoffel, 2001] M. Christoffel. Cooperation and Federation of Traders in an Information Market. In *Proc. of the AISB Symposium on Information Agents for E-Commerce (AISB'01 Convention)*, University of York, UK, March 2001.

[Chun and Culler, 2000] B. N. Chun and D. E. Culler. Market-based Proportional Resource Sharing for Clusters. Technical Report CSD-1092, University of California at Berkeley, Computer Science Division, January 2000.

[Chun and Culler, 2002] B. N. Chun and D. E. Culler. User-centric Performance Analysis of Market-based Cluster Batch Schedulers. In *Proc. of the 2nd IEEE/ACM Symposium on Cluster Computing and the Grid (CCGrid 2002)*, Berlin, May 2002. IEEE/ACM.

[Cirne and Berman, 2001a] W. Cirne and F. Berman. A Comprehensive Model of the Supercomputer Workload. In *Proc. of the 4th Workshop on Workload Characterization*, Austin, Texas, 2001. IEEE.

[Cirne and Berman, 2001b] W. Cirne and F. Berman. A Model for Moldable Supercomputer Jobs. In *Proc of the 15th Intl. Parallel & Distributed Processing Symposium (IPDPS)*, San Francisco, CA, 2001. IEEE.

[Cogan *et al.*, 2001] P. Cogan, J. Gomoluch, and M. Schroeder. A Quantitative and Qualitative Comparison of Distributed Information Processing using Mobile Agents realised in RMI and Voyager. *Intl. Journal of Software Engineering and Knowledge Engineering (IJSEKE)*, 11(5):583–606, October 2001.

[Collier, accessed in 2003] N. Collier. RePast: An Extensible Framework for Agent Simulation. http://repast.sourceforge.net/docs/repast_intro_final.doc, accessed in 2003.

[Czajkowski *et al.*, 2003] G. Czajkowski, S. Hahn, G. Skinner, P. Soper, and C. Bryce. A Resource Management Interface for the Java Platform. Technical Report TR-2003-124, Sun Microsystems, May 2003.

[Dafas *et al.*, 2003a] P. Dafas, D. Bolser, J. Gomoluch, J. Park, and M. Schroeder. Fast and Efficient Computation of Domain-Domain Interactions from known Protein Structures in the PDB. In *Proc. of the German Conference on Bioinformatics*, Munich, Germany, June 2003.

[Dafas *et al.*, 2003b] P. Dafas, J. Gomoluch, A. Kozlenkov, and M. Schroeder. Structural Protein Interactions: From Months to Minutes. In *Proc. of Parallel Computing 2003 (ParCo2003)*, Dresden, Germany, September 2003.

[Douglas and Ousterhout, 1991] F. Douglas and J. Ousterhout. Transparent process migration: design alternatives and the Sprite implementation. *Software Practice and Experience*, 21(8):757–85, 1991.

[Downey, 1997] A. B. Downey. A Parallel Workload Model and Its Implications for Processor Allocation. In *Proc. of the 6th Intl. Symposium on High Performance Distributed Computing*, Portland, Oregon, August 1997. IEEE Computer Society Press.

[Fatima, 2000] S. Shaheen Fatima. TRACE - an adaptive organizational policy for MAS. In *Working Notes of UKMAS 2000 - The Third UK Workshop on Multi-Agent Systems*, St Catherine's College, Oxford, UK, December 2000.

[Feitelson and Rudolph, 1998] D. G. Feitelson and L. Rudolph. Metrics and Benchmarking for Parallel Job Scheduling. In D. Feitelson and L. Rudolph, editors, *Job Scheduling Strategies for Parallel Processing, IPPS/SPDP'98 Workshop (Lecture Notes in Computer Science)*, volume 1459, pages 1–24. Springer, Orlando, Florida, 1998.

[Feitelson *et al.*, 1997] D. G. Feitelson, L. Rudolph, U. Schwiegelshohn, K. C. Sevcik, and P. Wong. Theory and Practice in Parallel Job Scheduling. In D. Feitelson and L. Rudolph, editors, *Job Scheduling for Parallel Processing: IPPS'97 Workshop (Lecture Notes in Computer Science)*, volume 1291. Springer, Geneva, Switzerland, 1997.

[Ferguson *et al.*, 1996] D. F. Ferguson, C. Nikolaou, J. Sairamesh, and Y. Yemini. Economic Models for Allocating Resources in Computer Systems. In *Market-Based Control: A Paradigm for Distributed Resource Allocation*. World Scientific, 1996.

[FIPA, accessed in 2003] FIPA. FIPA Specifications. http://www.fipa.org/specifications/index.html, accessed in 2003.

[Floyd and Paxson, 2001] S. Floyd and V. Paxson. Difficulties in simulating the Internet. *IEEE/ACM Transactions on Networking*, 9(4):392–403, 2001.

[Foster and Kesselman, 1997] I. Foster and C. Kesselman. Globus — A Metacomputing Infrastructure Toolkit. *Intl. Journal of Supercomputer Applications*, 11(2):115–128, 1997.

[Foster and Kesselman, 1998] Ian Foster and Carl Kesselman, editors. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kauffman, 1998.

[Foster *et al.*, accessed in 2004] I. Foster, C. Kesselman, J. M. Nick, and S. Tuecke. The Physiology of the Grid. http://www.globus.org/research/papers/ogsa.pdf, accessed in 2004.

[Fox *et al.*, 1994] G. C. Fox, R. D. Williams, and P. C. Messina. *Parallel Computing Works*. Morgan Kaufmann Publishers, Inc., 1994.

[Fox, 1992] G. C. Fox. Lessons from Massively Parallel Applications on Message Passing Computers. *IEEE Computer*, pages 103–114, 1992.

[Garey and Johnson, 1995] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman and Co., 1995.

[Geist *et al.*, 1994] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam. *PVM: Parallel Virtual Machine*. MIT Press, 1994.

[Ghanea-Hercock *et al.*, 1999] R. Ghanea-Hercock, J. C. Collis, and D. T. Ndumu. Co-operating Mobile Agents for Distributed Parallel Processing. In *Proc. of the Third Intl. Conference on Autonomous Agents (AA99)*, Mineapolis, USA, May 1999. ACM Press.

[Gomoluch and Schroeder, 2001a] J. Gomoluch and M. Schroeder. Flexible Load Balancing in Distributed Information Agent Systems. In *Proc. of the ACAI 2001 and EASSS 2001 Student Sessions*, Prag, Czech Republik, July 2001. Czech Technical University in Prague.

[Gomoluch and Schroeder, 2001b] J. Gomoluch and M. Schroeder. Information agents on the move. *Software Focus*, 2(2):31–36, Summer 2001.

[Gomoluch and Schroeder, 2002] J. Gomoluch and M. Schroeder. Flexible Load Balancing in Distributed Information Agent Systems. In V. Marik, O. Stepankova, H. Krautwurmova, and M. Luck, editors, *Multi-Agent Systems and Application II. Selected Revised Papers: 9th ECCAI-ACAI/EASSS 2001, AEMAS 2001, HoloMAS 2001, LNAI 2322*. Springer, 2002.

[Gomoluch and Schroeder, 2003] J. Gomoluch and M. Schroeder. Market-based Resource Allocation for Grid Computing: A Model and Simulation. In *Proc. of the First Intl. Workshop on Middleware for Grid Computing (MGC2003)*, Rio de Janeiro, Brazil, June 2003.

[Gomoluch and Schroeder, 2004] J. Gomoluch and M. Schroeder. Performance Evaluation of Market-based Resource Allocation for Grid Computing. *To appear in Concurrency and Computation: Practice and Experience*, 2004.

[Harchol-Balter and Downey, 1997] M. Harchol-Balter and A. B. Downey. Exploiting process lifetime distributions for dynamic load-balancing. *ACM Transactions on Computer Systems*, 15(3):253–285, 1997.

[Hewlett Packard, accessed in 2003] Hewlett Packard. HBTC - Hewlett-Packard and grid computing. http://www.hp.com/techservers/grid/, accessed in 2003.

[Hohl, accessed in 2003] F. Hohl. The Mobile Agent List, a repository of mobile agent systems. http://mole.informatik.uni-stuttgart.de/mal/mal.html, accessed in 2003.

[Howell and McNab, 1998] F. Howell and R. McNab. Simjava: A Discrete Event Simulation Package for Java with Applications in Computer Systems Modelling. In *Proc. of the First Intl. Conference on Web-Based Modeling and Simulation*, San Diego, CA, USA, January 1998.

[IBM, accessed in 2003] IBM. IBM Grid Computing home page. http://www-1.ibm.com-/grid/, accessed in 2003.

[IEEE, 1990] IEEE. *IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries*. IEEE, New York, 1990.

[Jain, 1991] R. Jain. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*, chapter 25, pages 430–431. Wiley-Interscience, 1991.

[Jul *et al.*, 1988] E. Jul, H. Levy, N. Hutchinson, and A. Black. Fine-grained mobility in the Emerald system. *ACM Transactions on Computer Systems*, 6(1):109–33, 1988.

[Kagel, 1995] J. H. Kagel. Auctions: A Survey of Experimental Research. In J. H. Kagel and A. E. Roth, editors, *The Handbook of Experimental Economics*, chapter 7, pages 501–580. Princeton University Press, 1995.

[Kephart *et al.*, 2001] J. O. Kephart, J. E. Hanson, and A. R. Greenwald. Dynamic Pricing by Software Agents. *Computer Networks*, 32(6):731–752, March 2001.

[Keren and Barak, 1998] A. Keren and A. Barak. Adaptive Placement of Parallel Java Agents in a Scalable Computer Cluster. In *Proc. of the Workshop on Java for High-Performance Network Computing*, Stanford University, Palo Alto, CA, USA, February 1998. ACM Press.

[Kilgore and Burke, 2000] R. A. Kilgore and E. Burke. Object-Oriented Simulation of Distributed Systems using Java and Silk. In *Proc. of the Winter Simulation Conferences (WSC'00)*, Orlando, Florida, USA, December 2000.

[Kim *et al.*, 2003] 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. Dynamic Mapping in a Heterogeneous Environment with Tasks Having Priorities and Multiple Deadlines. In *Proc. of the 17th Intl. Parallel and Distributed Processing Symposium (IPDPS 2003)*, Nice, France, April 2003. IEEE.

[Lange and Oshima, 1998a] D. B. Lange and M. Oshima. Mobile Agents with Java: The Aglet API. *World Wide Web Journal*, 1998.

[Lange and Oshima, 1998b] D. B. Lange and M. Oshima. *Programming and Deploying Java Mobile Agents with Aglets*. Addison-Wesley, 1998.

[Lee *et al.*, 1998] L. C. Lee, H. S. Nwana, D. T. Ndumu, and P. De Wilde. The stability, scalability and performance of multi-agent systems. *BT Technology Journal*, 16(3):94–103, July 1998.

[Levy *et al.*, 2001] L. Levy, L. Blumrosen, and N. Nisan. OnLine Markets for Distributed Object Services: the MAJIC system. In *Proc. of the 3rd USENIX Symposium on Internet Technologies and Systems*, San Francisco, CA, USA, 2001.

[Liang and Bracha, 1998] S. Liang and G. Bracha. Dynamic class loading in the Java Virtual Machine. In *Proc. of the Conference on Object Oriented Programming Systems Languages and Applications (OOPSLA'99)*, pages 36–44, Denver, CO, USA, 1998. ACM Press.

[Libes, 1994] D. Libes. *Exploring Expect.* O'Reilly & Associates, 1994.

[Maheswaran *et al.*, 1999a] M. Maheswaran, S. Ali, H. J. Siegel, D. A. Hensgen, and R. F. Freund. Dynamic mapping of a class of independent tasks onto heterogeneous computing systems. *Journal of Parallel and Distributed Computing*, 59(2):107–131, 1999.

[Maheswaran *et al.*, 1999b] M. Maheswaran, T. D. Braun, and H. J. Siegel. Heterogeneous Distributed Computing. In J. G. Webster, editor, *Encyclopedia of Electrical and Electronics Engineering*, pages 679–690. John Wiley & Sons, New York, NY, 1999.

[Messer and Wilkinson, 1996] A. Messer and T. Wilkinson. Power to the process. In *Workshop on Parallel, Emergent, and Distributed Computing*, Reading,UK, May 1996. MIT Press.

[Metcalfe and Boggs, 1976] R. Metcalfe and D. Boggs. Ethernet: Distributed Packet Switching for Local Computer Networks. In *Proc. of the National Computer Conference*, volume 19, 1976.

[Milojicic *et al.*, 1998] D. Milojicic, M. Breugst, I. Busse, J. Campbell, S. Covaci, B. Friedman, K. Kosaka, D. Lange, K. Ono, M. Oshima, C. Tham, S. Virdhagriswaran, and J. White. MASIF: The OMG Mobile Agent System Interoperability Facility. *Personal Technologies*, 2:117–129, 1998.

[Minar *et al.*, 1996] N. Minar, R. Burkhart, C. Langton, and M. Askenazi. The Swarm Simulation System, A Toolkit for Building Multi-Agent Simulations. http://www.swarm.org, June 1996.

[Möller *et al.*, 1999] S. Möller, U. Leser, W. Fleischmann, and R. Apweiler. EDITtoTrEMBL: A distributed approach to high-quality automated protein sequence annotation. *Journal of Bioinformatics*, 15(3):219–227, 1999.

[Moss, 1999] S. Moss. The cost of rational agency. Technical Report 99-51, Manchester Metropolitan University, Centre for Policy Modelling, 1999.

[Natrajan *et al.*, 2001] A. Natrajan, M. Humphrey, and A. S. Grimshaw. Capacity and Capability Computing in Legion. In *Proc. of the 2001 Intl. Conference on Computational Science*, San Francisco, CA, May 2001.

[Nguyen *et al.*, 1996] T. D. Nguyen, R. Vaswani, and J. Zahorjan. Parallel application characteristics for multiprocessor scheduling policy design. *Lecture Notes in Computer Science*, 1162:175–199, 1996.

[Nisan *et al.*, 1998] N. Nisan, S. London, O. Regev, and N. Camiel. Globally distributed computation over the Internet - the POPCORN project. In *Proc. of the 18th Intl. Conference on Distributed Computing Systems*, Amsterdam, Netherlands, 1998. IEEE.

[Obelöer *et al.*, 1998] W. Obelöer, C. Grewe, and H. Pals. Load management with mobile agents. In *Proc. of the 24th EUROMICRO Conference*, pages 1005–1012, 1998.

[Object Management Group, 1992] Object Management Group. *The Common Object Request Broker: Architecture and Specification*. Wiley, 1992.

[Pacheco, 1997] Peter S. Pacheco. *Parallel Computing with MPI*. Morgan Kaufmann, 1997.

[Padala *et al.*, 2003] P. Padala, C. Harrison, N. Pelfort, E. Jansen, M. P. Frank, and C. Chokkareddy. OCEAN: The Open Computation Exchange and Arbitration Network, A Market Approach to Meta computing. In *Proc. of the Intl. Symposium on Parallel and Distributed Computing (ISPDC'03)*, Ljubljana, Slovenia, 2003.

[Page *et al.*, 2000] B. Page, T. Lechler, and S. Claassen. *Objektorientierte Simulation in Java mit dem Framework DESMO-J*. Libri Books on Demand, 2000.

[Pallas, accessed in 2003] Pallas. UNICORE. http://www.unicore.de, accessed in 2003.

[Park *et al.*, 2001] J. Park, M. Lappe, and S. A. Teichmann. Mapping Protein Family Interactions: Intramolecular and Intermolecular Protein Family Interaction Repertoires in the PDB and Yeast. *Journal of Molecular Biology*, 307:929–938, 2001.

[Pinedo, 1995] M. Pinedo. *Scheduling. Theory, Algorithms, and Systems*. Prentice Hall, 1995.

[Platform Computing, accessed in 2003] Platform Computing. Platform LSF 5. http://www.platform.com, accessed in 2003.

[Platt and Ballinger, 2002] D. S. Platt and K. Ballinger. *Introducing Microsoft .NET*. Microsoft Press, 2002.

[Pooley, 1987] R. J. Pooley. *An Introduction to Programming in SIMULA*. Blackwell Scientific Publications, 1987.

[Pozo and Miller, accessed in 2003] R. Pozo and B. Miller. SciMark 2.0. http://math.nist-.gov/scimark2/index.html, accessed in 2003.

[Preist *et al.*, 2001] C. Preist, A. Byde, C. Bartolini, and G. Piccinelli. Towards agent-based service composition through negotiation in multiple auctions. In *Proc. of the AISB Symposium on Information Agents for E-Commerce (AISB'01 Convention)*, University of York, UK, March 2001.

[Regev and Nisan, 1998] O. Regev and N. Nisan. The POPCORN Market — an Online Market for Computational Resources. In *Proc. of the First Intl. Conference on Information and Computation Economies*, Charleston, South Carolina, USA, 1998. ACM Press.

[Rus *et al.*, 1997] D. Rus, R. Gray, and D. Kotz. Transportable information agents. *Journal of Intelligent Information Systems*, 9, 1997.

[Sahni and Thanvantri, 1995] S. Sahni and V. Thanvantri. Parallel computing: Performance metrics and models. Technical report, Computer Science Department, University of Florida, May 1995.

[Sandholm, 2000] T. Sandholm. Distributed Rational Decision Making. In G. Weiss, editor, *Multi-agent systems*. MIT Press, 2000.

[Schroeder and Boro, 2001] M. Schroeder and L. Boro. Does the restart method work? Preliminary results on efficiency improvements for interactions of web-agents. In *Proc. of the Workshop on Infrastructure for Agents, MAS, and Scalable MAS (at Agents'01)*, Montreal, Canada, 2001.

[Sherwani *et al.*, 2002] J. Sherwani, N. Ali, N. Lotia, Z. Hayat, and R. Buyya. Libra: An Economy driven Job Scheduling System for Clusters. In *Proc. of the 6th Intl. Conference on High Performance Computing in Asia-Pacific Region (HPC Asia 2002)*, Bangalore, India, December 2002.

[Shoch and Hupp, 1982] F. Shoch and J. Hupp. The Worm programs - Early experience with a distributed computation. *Communications of the ACM*, 25(3):172–80, 1982.

[Strasser *et al.*, 1996] M. Strasser, J. Baumann, and F. Hohl. Mole — a Java based mobile agent system. In *Proc. of ECOOP'96, Workshop on Mobile Object Systems*, Linz, Austria, 1996. Springer.

[Streit, 2001] A. Streit. On job scheduling for HPC-clusters and the dynp scheduler. In *Proc. of the 8th Intl. Conference on High Performance Computing (HiPC)*, Hyderabad, India, December 2001. Springer.

[Strohmaier *et al.*, accessed in 2003] E. Strohmaier, J. Dongarra, H. Meurer, and H. Simon. 22nd TOP500 List. Supercomputer Conference (SC2003), accessed in 2003. http://www.top500.org/lists/2003/11/.

[Sun Microsystems, accessed in 2003a] Sun Microsystems. Java Message Service API - Java Message Service Tutorial. http://java.sun.com/products/jms/tutorial/index.html, accessed in 2003.

[Sun Microsystems, accessed in 2003b] Sun Microsystems. Java Remote Method Invocation - Distributed Computing for Java. http://java.sun.com/marketing/collateral/javarmi-.html, accessed in 2003.

[Sun Microsystems, accessed in 2003c] Sun Microsystems. Jini architectural overview. http://wwws.sun.com/software/jini/whitepapers/architecture.pdf, accessed in 2003.

[Sun Microsystems, accessed in 2003d] Sun Microsystems. Project JXTA Technology: Creating Connected Communities. http://www.jxta.org/project/www/docs/JXTA-Exec-Brief-032803.pdf, accessed in 2003.

[Sun Microsystems, accessed in 2003e] Sun Microsystems. Sun Cluster Grid Architecture. http://wwws.sun.com/software/gridware/SunClusterGridArchitecture.pdf, accessed in 2003.

[Suri, 2000] N. Suri. An Overview of the NOMADS Mobile Agent System. In *Proc. of ECOOP'2000*, Nice, France, 2000.

[Takefusa, 2001] A. Takefusa. Bricks: A Performance Evaluation System for Scheduling Algorithms on the Grid. In *Proc. of the JSPS Workshop on Applied Technology for Science (JWAITS 2001)*, San Diego Supercomputer Center, University of California, San Diego, USA, January 2001.

[van Nieuwpoort *et al.*, 2001] R. van Nieuwpoort, T. Kielmann, and H. E. Bal. Efficient Load Balancing for Wide-Area Divide-and-Conquer Applications. In *Proc. of the Symposium on Principles and Practice of Parallel Programming (PPOPP'01)*, Snowbird, Utah, USA, 2001. ACM Press.

[Waldo, 2001] J. Waldo. Mobile code, distributed computing, and agents. *IEEE Intelligent Systems*, 16(2), March/April 2001.

[Waldspurger *et al.*, 1992] C. A. Waldspurger, T. Hogg, B. A. Huberman, J. O. Kephart, and W. S. Stornetta. Spawn: A Distributed Computational Economy. *IEEE Transactions on Software Engineering*, 18(2):103–117, 1992.

[Waldspurger, 1995] C. A. Waldspurger. *Lottery and Stride Scheduling: Flexible Proportional-Share Resource Management.* PhD thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, MA, USA, 1995.

[Webopedia.com, accessed in 2004] Webopedia.com. Web Services. http://www.webopedia.com/TERM/W/Web_services.html, accessed in 2004.

[Wellman *et al.*, 2001] M. P. Wellman, W. E. Walsh, P. R. Wurman, and J. K. MacKie-Mason. Auction protocols for decentralized scheduling. *Games and Economic Behavior*, 35:271–303, 2001.

[Wims and Xu, 1999] B. Wims and C. Xu. Traveler: A Mobile Agent Based Infrastructure for Global Parallel Computing. In *Proc. of the First Intl. Symposium on Agent Systems and Applications (ASA'99) / Third Intl. Symposium on Mobile Agents (MA'99)*, Palm Springs, California, October 1999. IEEE.

[Wolski *et al.*, 2001] R. Wolski, J. S. Plank, J. Brevik, and T. Bryan. Analyzing market-based resource allocation strategies for the computational Grid. *Intl. Journal of High Performance Computing Applications*, 15(3):258–281, Fall 2001.

[Wolski *et al.*, 2003] R. Wolski, J. Plank, and J. Brevik. G-Commerce – Building Computational Marketplaces for the Computational Grid. Technical Report CS-00-439, University of Tennessee, 2003.

[Wolski, 1997] R. Wolski. Implementing a Performance Forecasting System for Metacomputing: The Network Weather Service. In *Proc. of the ACM/IEEE SC97 Conference*, 1997.

[Xu and Wims, 2000] C.-Z. Xu and B. Wims. A Mobile Agent Based Push Methodology for Global Parallel Computing. *Concurrency: Practice and Experience*, 14(8):705–726, 2000.