Universiteit Antwerpen

Economic Grid Resource Management using Spot and Futures Markets

Proefschrift voorgelegd op 14 januari 2009 tot het behalen
van de graad van Doctor in de Wetenschappen,
richting Informatica, bij de faculteit wetenschappen,
aan de Universiteit Antwerpen.

Promotor:

prof. dr. Jan Broeckhove

Kurt Vanmechelen

CoMP Onderzoeksgroep Computationeel Modelleren en Programmeren

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

1.1 Motivation and Problem Description

The Internet can be considered as one of the milestones in the history of Information Technology (IT). Its development has revolutionized the way in which people communicate, share information, socialize and do business. The development of Internet standards has enabled the sharing of information on a global scale, turning information into a easily accessible good. Recently, the vision of extending the scope of what can be shared over computer networks to a more generalized concept of *services* which can offer a wide range of capabilities from access to distributed processing, network and storage resources to special purpose equipment, have underpinned the vision of a *grid* as a next generation Internet. The vision put forth by grid computing foresees an environment in which services are assembled, shared and used as a utility, over the boundaries of the organizations that host them. As such, a platform is envisioned where home users, scientists and businesses can appeal to a global service network in order to obtain access to their required IT capabilities. The specific set of grid computing services that have obtained a major interest in the scientific community, offer access to compute resources such as clusters, supercomputers and collections of desktop machines that are hosted at different sites all over the world. The integration of these resources can offer unprecedented computing performance that allows researchers to tackle problems deemed unsolvable beforehand. In the scientific community this model has already been put to use, as exemplified by the Large Hadron Collider (LHC) Computing grid that has gone live on September 10th 2008, in order to support the massive data processing requirements of CERN's LHC particle accelerator.

The implementation of the grid vision has been difficult due to a large number of technical challenges. These range from providing secure access to services to the problem of providing uniform access to a wide array of heterogeneous resources. To a great extent, these challenges have been resolved by major investments in international software development efforts which have in large part been publicly funded. These developments however, have given limited consideration to the fact that grids reinstate the model in which resources are *shared* on a large scale among users and providers that have a potentially loose prefatory relationship.

As a consequence, mechanisms are at present missing that deliver clear incentives to resource owners to share their resources in such an open environment. In addition, a flexible and efficient mechanism is required that determines who obtains access to resources and services on a grid, at what time and at what cost. This mechanism needs to incentivize users to use resources in a well-considered manner, impeding misuse and overuse of resources. In times of congestion, it must also be able to assign resources to users that value them the most, in order to get the most value out of the available infrastructure.

Currently, access privileges to grid resources are assigned to users and user groups based on explicit negotiations that result in static usage policies. In addition, resource management systems that are currently in use on production grids do not take a value-oriented approach when making resource allocation decisions, but use first-come-first-serve (FCFS) job scheduling and entirely system-centric optimization goals instead.

This has the following consequences:

- Providers do not openly share their resources as a utility as they have no incentive to do so. This limits the extent to which grids can realize the utility computing model, as well as the sustainability of current grid systems.

- There are no incentives for users to back off their loads during times of congestion, or to use resources in a well-considered manner. As a consequence, current systems do not offer provisions for stringent quality of service (QoS) requirements such as the formulation of a completion deadline for a set of jobs, as each user would simply require the highest possible level of QoS.

- The resource management system cannot make optimal scheduling and resource allocation decisions in terms of generating the maximal user satisfaction in the system, as it considers all requests for resources as being equal. As a result, the full potential of the infrastructure is not realized, especially in times of congestion.

1.2 Objective, Research Questions and Contributions

The objective of this thesis is to investigate how market mechanisms can be applied within the context of grid resource management to deal with the aforementioned issues. Markets have long been used to tackle the issue of efficiently sharing scarce resources in real economies. We investigate how these ideas can be applied to the problem of managing access to grid resources. We thereby consider the use of a market through which users and providers can interact in a self-interested manner to buy and sell access rights to resources.

A key property of markets is that the efficient division of resources at a global level emerges from the local interactions of the selfish user and provider agents in the market. This leads to a decentralized form of decision making and allows for the development of grid resource management systems that are driven by the generation of maximal value for users and providers, instead of pursuing a system-centric optimization goal. The introduction of cost levels that are dynamically determined based on the relationship between demand and supply, provides the necessary incentives to share resources and use them in a well-considered manner.

We will investigate both spot market organizations that are closely related to the current best-effort usage model in grids, as well as futures market organizations that support more stringent QoS requirements through the negotiation of access rights in future time. We thereby focus on QoS requirements in the form of hard deadlines that users place on the completion time of their computational workload. After considering the strengths and weaknesses of both approaches, we propose a hybrid market design in which both spot and futures market approaches are combined.

This leads to the following research questions and contributions.

- **How can different approaches to (market-based) grid resource management be efficiently evaluated and compared?**
  Due to the large-scale nature of grid systems and the limited availability of real-world platforms for experimentation, empirical evaluation of grid resource management systems is often performed using simulation.

  Our contribution is the design and development of a simulation framework that is tailored towards the evaluation of resource management systems for grids. Our framework goes beyond the state of the art as it directly supports the notion of experiments, provides means to speed up experiment execution through the use of clusters and grids, offers both discrete-time and discrete-event based simulation, and provides the possibility of directly comparing a new resource management system to a range of existing economic and non-economic resource management systems. In addition, we show how a scalable event-based simulation core can be developed through the use of continuations, and how Java annotations and Aspect-Oriented Programming can provide a clean programming model for developing simulations of distributed environments.

- **How can a market model in which a set of substitutable commodities are dynamically priced on a spot market be applied to the problem of grid resource management?**
  Our contribution is the application and evaluation of a commodity market model that dynamically establishes market-wide prices for substitutable commodities. We evaluate the proposed market-based approach in terms of price correctness and stability, fairness, scalability and infrastructural utilization through simulation. We thereby extend an existing method for finding equilibrium prices to cope with the specifics of pricing substitutable grid resources.

- **How does a market based on single-unit Vickrey auctions compare to the commodity market approach for organizing a spot market for computational resources?**
  Our contribution is the evaluation of a market model that uses single-unit Vickrey auctions and the comparison of the relevant metrics with the commodity market approach.

- **How can a market-based resource management system deliver application-level QoS by supporting the formulation of hard deadline requirements?**
  For many grid users, quality of service requirements are not formulated at the level of individual jobs but at the level of the entire application that is distributed on the grid. In order to support QoS requirements that relate to the deadline at which the entire application should finish, a market mechanism is required that relies on coallocation and resource reservation to meet these requirements.

  Our contribution is the development and evaluation of a futures market mechanism that is both scalable, realizes close to optimal value, and at the same time takes into account the discrete nature of computational jobs and the inherent limitations to the parallelizability of an application's workload.

- **Can a combination of a spot and futures market mechanism increase the total user value in face of high priority user requests for which the delay introduced by futures market mechanisms is unacceptable?**
  Mechanisms that plan in resource allocations in future time inherently introduce a delay in the allocation process. Therefore it is a challenge to support allocations and capacity planning in future time, while still offering means to quickly sell or buy non-storeable resources such as CPU cycles.

  Our contribution is the combination of a spot and futures market mechanism and the evaluation of the resulting approach in terms of realized user value. We thereby demonstrate how providers can react to price signals in both markets to divide their available supply of resources over both markets.

1.3 Structure of the Dissertation

The text is structured as follows. Chapter 2 introduces the key notions and motivations behind grid computing and resource management. In chapter 3, we introduce the motivation for and application of economic and market-based grid resource management. After these introductory chapters, chapter 4 presents an overview and performance evaluation of the Grid Economics Simulator (GES) that we developed for evaluating market-based approaches to grid resource management through simulation.

The first category of market organizations that we have considered follow the spot market paradigm. Chapter 5 presents and evaluates two spot market organizations namely commodity markets and single-unit Vickrey auctions. In order to tackle issues related to coallocation of resources over time in support of deadline requirements posed by users, we investigate the adoption of a futures market paradigm in chapter 6. Chapter 7 then combines the insights from chapters 5 and 6 by integrating a commodity market organization with a futures market, resulting in a hybrid approach. Finally, chapter 8 presents the main conclusions of this dissertation.

A number of chapters in this dissertation are partially based on the following contributions:

- Chapter 4:
  - *A Simulation Framework for Studying Economic Resource Management in Grids*
    K. Vanmechelen, W. Depoorter and J. Broeckhove
    Proceedings of the 8th International Conference on Computational Science (ICCS 2008), Lecture Notes in Computer Science, Vol. 5101, pp 226-235, Springer-Verlag, Heidelberg
  - *Scalability of Grid Simulators : An Evaluation*
    W. Depoorter, N. De Moor, K. Vanmechelen and J. Broeckhove
    Proceedings of Europar 2008, 26-29 August, Las Palmas de Gran Canaria, Spain, E. Luque, T. Margalef and D. Benitez eds., Lecture Notes in Computer Science, Vol. 5168, pp 544-553, Springer-Verlag, Heidelberg
  - *On the design and performance of a discrete-event core for the Grid Economics Simulator*
    K. Vanmechelen, W. Depoorter and J. Broeckhove
    Submitted to Software: Practice and Experience, Wiley
- Chapter 5:
  - *Pricing Grid Resources using Commodity Market Models*
    K. Vanmechelen, G. Stuer and J. Broeckhove
    Proceedings of the 3rd International Workshop on Grid Economics and

Business Models (GECON 2006), 16 May 2006, Singapore, H. Lee and S. Miller eds., pp. 103-112, World Scientific Publishing Co., Singapore

- *A Commodity Market Algorithm for Pricing Substitutable Grid Resources*
  G. Stuer, K. Vanmechelen and J. Broeckhove
  Fut. Gen. Comp. Sys., Vol. 23, no. 5, June 2007, pp. 688-701
- *A Comparative Analysis of Single-Unit Vickrey Auctions and Commodity Markets for Realizing Grid Economies with Dynamic Pricing*
  K. Vanmechelen and J. Broeckhove
  Proceedings of the 4th International Workshop on Grid Economics and Business Models (GECON 2007), August, 2007, Rennes, France, Lecture Notes in Computer Science, Vol. 4685, J. Altmann and D.J. Veit eds., pp. 98-111, Springer-Verlag, Heidelberg
- *Dynamic Pricing of Computational Resources in Grid Economies*
  K. Vanmechelen, J. Broeckhove, W. Depoorter and K. Abdelkader
  The Encyclopedia of Grid Computing Technologies and Applications, In Press

- Chapter 6:
  - *Economic Grid Resource Management for CPU Bound Applications with Hard Deadlines*
    K. Vanmechelen, W. Depoorter and J. Broeckhove
    Proceedings of the 8th IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2008), May, 2008, Lyon, France, pp 258-266, IEEE Computer Society
  - *Market-based grid resource co-allocation and reservation for applications with hard deadlines*
    K. Vanmechelen, W. Depoorter and J. Broeckhove
    Submitted to Concurrency and Computation: Practice and Experience, Wiley

- Chapter 7:
  - *Combining Futures and Spot Markets: A Hybrid Market Approach to Economic Grid Resource Management*
    K. Vanmechelen, W. Depoorter and J. Broeckhove
    Submitted to Future Generation Computer Systems, Elsevier

CHAPTER 2

Grids

2.1 Introduction

Over the last twenty years grid technologies and protocols have emerged to satisfy the need to use distributed resources and compose distributed services in a coordinated and secure manner. Through the definition of new concepts, abstractions, standards and protocols, grid technologies allow applications to access and share resources and services across wide-area networks [8]. As such, the deployment of grid technology is said to create a *grid* of interconnected resources and services. The term *grid*, first coined by Ian Foster and Carl Kesselman in the mid '90s, draws from the analogy to modern day power-grids in which electricity is transparently delivered to end-users. Indeed, the vision of organizing an equally transparent delivery of IT resources and services to companies and scientists has always been one of the key drivers in this field. The grid vision is often considered to be an extension of the Internet in the sense that it aims to provide seamless access to resources, services *and* information, just as the Internet provides seamless access to information through standardized protocols.

In the scientific community, the global and collaborative nature of scientific research has resulted in a pressing need for the development of grid technology. New modes of scientific inquiry based on large-scale numerical simulation, data analysis and the use of remotely accessible scientific apparatus, fuel this need [9]. A major example and a prime mover behind grid technology over the past years is the Large Hadron Collider Compute Project (LCG) [10]. This project is lead by CERN[1] and provides the compute infrastructure and software to support the analysis of the data

[1] CERN stands for Conseil Européen pour la Recherche Nucléaire or European Organization for Nuclear Research.

generated by the Large Hadron Collider (LHC)[2]. The LHC is a particle accelerator with a circumference of 26,7 km that can accelerate particles to the speed of light and then steer them into 600 million collisions per second. Physicists believe that data from these experiments can fill in the gaps in the *Standard Model* [11] of particle physics. Four *detectors* in the accelerator register collision data with high accuracy, generating a total output data stream of over 15 petabytes anually. Over 9000 scientists all over the world require timely access to this data for their research. In order to tackle this grand computing challenge, IT infrastructure that spans over 200 separate sites in more than 30 countries has been integrated in a compute grid. Presently it can process over 250,000 jobs a day and can tackle peaks of 500,000 jobs, with each job lasting several hours to several days on a high performance processor. Because workloads can be distributed on an infrastructure of such an enormous scale, new problems of unprecedented complexity, such as the LHC data analysis, can be tackled.

Aside from providing a solution to tackling these grand-challenge problems in science, grid technology is also a key enabler in the practical realization of a *utility computing* model. Such a model involves the transparent delivery of IT resources from third-party providers to users, allowing applications to scale up on-demand, without being constrained by the limits of the IT infrastructure of the user's own organization. Also, users with very irregular and infrequent needs for IT resources can make use of third-party infrastructure for all of their computing needs. A more common setup is that organizations can fall back on additional resources through the grid when their own infrastructure temporarily does not suffice to cover the compute demands. Because of the flexibility and transparency offered by the grid computing model, internal IT infrastructure no longer has to be dimensioned for peak loads. A typical phenomenon is that different IT resources that belong to the same organization are individually allocated to specific subentities within that organization. Since all of the individual resources are dimensioned for the peak loads of e.g. their department, a common situation is that many of the resources are idle. This underutilization of resources is both undesirable from a cost perspective as well as from a functional perspective. Costs related to floor space, cooling and electricity, software licenses, and personnel managing the infrastructure, all need to be payed irrespective of the utilization of resources [12]. In addition, economies of scale can reduce the relative weight of fixed costs related to hosting IT resources in relation to the variable costs. As such, this compute model has the potential of delivering significant cost savings for users in situations where the provisioning of IT infrastructure is fully or partially outsourced to providers on the grid, or when the internal resources of a company are shared through grid technology in order to improve utilization. From a functional perspective, underutilization in one part of the organization's infrastructure inherently implies a lost opportunity for delivering higher quality of service (QoS) to applications that hit the limits of what another part of the infrastructure can provide. Since, IT resources often directly support

[2]Although both LHC and LCG projects are lead by CERN, more than 50 countries collaborate in these projects.

business processes in organizations, increasing this level of QoS through resource sharing directly improves the quality of the organization's processes.

The practical realization of the grid computing model however, is a complex undertaking. The distributed, multi-organizational, and large-scale nature of grid systems leads to major challenges in system and process design and development. This chapter will provide an outlook on these challenges and present an overview on current approaches to tackle them.

2.2 Foundations

This section introduces a number of fundamental concepts related to grid infrastructures and the notion of *Virtual Organizations*.

2.2.1 Infrastructure

Grids integrate a wide range of different IT resources, ranging from computational, storage, and network resources, to special-purpose hardware such as telescopes or sensor networks, and to systems with dedicated software. Within these individual classes of resources, there is a significant level of heterogeneity in resource characteristics and attributes. A key challenge in the realization of grids is to provide the means to deal with this heterogeneity and with the need to coallocate different resources. This is partly realized by *virtualizing* resources, i.e. by making them accessible through a uniform interface and detaching the service delivered by a resource (e.g. offering storage space) from the individual hardware components that deliver this service (e.g. a Network Attached Storage (NAS) device linked to a cluster). In the following, we present a short overview of the resources that one currently finds in deployed grid systems.

Compute Resources

Compute resources offer the hardware for performing computations and for storing temporary results on scratch storage. The following classes of compute resources can be found in grid infrastructures:

- **Supercomputers** are machines with thousands of processors and terabytes of memory, providing extreme processing capabilities. They incorporate specialized hardware such as Non Uniform Memory Access (NUMA) subsystems and very low-latency interconnects. This allows them to undertake huge parallel computations efficiently, even when the individual processes within the computation have intricate data interdependencies. Since a supercomputer is often targeted at a specific (set of) problem(s), its design can include specialized processors, like the GRAPE processors used to calculate N-body problems in astrophysics [13]. Supercomputers are usually accessible to a specific group of scientists. They can be integrated in a grid infrastructure but typically are already accessible to the intended user group.

- A **cluster** is a collection of commercial-of-the-shelf computers that are connected through a dedicated high-performance interconnection network. They typically have a single point of access through a *head node*, which runs a local resource management system that manages the execution of jobs on the *compute nodes*. A cluster is usually connected to the backbone of an organization through a dedicated link. Depending on latency and bandwidth requirements, several interconnects can be used ranging from Gigabit Ethernet to Myrinet. Clusters are more general purpose than supercomputers and less tailored to requirements of specific parallel computations. Clusters are edging out supercomputers as high-performance computing (HPC) platforms, as evidenced by the evolution of the Top500 ranking [14]. This is in part because they offer better cost- effectiveness for applications that do not require the low-latency inter-process communication delivered by more advanced supercomputer architectures.

- **Desktops** are systems which are actively used by people, privately or within an organization, to perform their general computing tasks such as using a spreadsheat or email program. Desktop machines show considerable heterogeneity [15], even within a single organization. The incorporation of these machines in a grid infrastructure is more involved than for clusters or supercomputers. Since these desktop machines are actively used, the execution of jobs must only use spare CPU cycles and never hinder the user. In addition, the grid middleware that is to be installed on the desktop machine should be non-intrusive. Modern advances in virtualization software offer means to deal with these issues.

  Due to the often unpredictable and slow network connections of desktop systems, jobs are typically limited to having relatively small amounts of input and output data compared to their computational requirements. Another characteristic of desktop grids is that availability of individual machines is not guaranteed. Machines may be rebooted or shut down at any time. Therefore, the underlying system needs to be robust in dealing with machines leaving and re-appearing. In addition, desktops are in general considered to be non-trustworthy. As a result, computational results are inherently unreliable and issues concerning the protection of intellectual property rights and privacy issues are even more prominent, compared to a grid that consists of clusters or supercomputers. The extent to which these issues manifest themselves depends on the environment in which desktop grids are built (e.g. the global Internet, or within an enterprise).

  Despite these difficulties in managing a desktop grid, their potential is huge due to the sheer amount of desktop machines connected to the Internet or available within an organization, and is well proven by volunteer computing [16] projects such as SETI@home [17] and FOLDING@home[3] [18]. In addition,

[3]The FOLDING@home project which supports simulation of protein folding integrates resources from both desktop computers as well as Playstation 3 game consoles. It entered the Guinness Book of

modern advances in desktop hardware, with a notable shift towards multi-core architectures, have overdimensioned current desktop systems for daily computing tasks, leading to a high degree of underutilization of those systems, and a huge computational capacity that can potentially be harvested. Software for building desktop grids has historically been tightly linked with a specific application. Recently, toolkits such as BOINC [19] have been adopted for building application-neutral desktop grids[4].

Note that, although the issues concerning availability, trust and heterogeneity in network connections are typically more stringent in a desktop grid setting, the same issues exist in a grid consisting of clusters and supercomputers belonging to different organizations.

Storage Resources

Mass storage services for grids are often implemented using tape libraries. These storage devices contain one or more drives and a number of slots to hold tape cartridges. The cartridges are loaded and unloaded form the drives by means of a robot which leads to a considerable random access penalty. Moreover, tapes themselves are not very good at random access. As a result, tape libraries are primarily useful and cost effective for backup and archival of huge amounts of data. Tape libraries come in different formats, ranging from autoloaders with a single drive and several tape cartridges up to solutions spanning multiple racks with hundreds of drives, thousands of tape cartridges and multiple petabytes of storage capacity. Tape libraries are used at CERN to store data coming from the LHC experiments.

Hard disk storage solutions come in different forms and sizes, ranging from a single-disk Network Attached Storage (NAS) to an integrated multi-rack Storage Area Network (SAN) with thousands of hard disks. One of the benefits of these systems is the fast random access times they provide. Moreover the disks are usually running in a RAID configuration, delivering faster throughput and increased reliability. Since hard disk capacities enlarge constantly and since it is relatively easy to add racks, such storage solutions offer good scalability.

Hierarchical storage solutions combine the benefits of the systems previously mentioned. Typically, high-performance hard disk storage is combined with low-cost tape libraries. The disks hold currently active files while files that have not been accessed for some time are migrated to tape. The faster disks thus act as a cache for the tape storage.

Network Resources

The extent to which grids can realize the vision of efficient and transparent resource and service delivery depends on the latency and bandwidth capabilities of the network that connects the different resources. This relation between the grid vision

Records in November 2007 as the largest distributed computing network, being the first distributed computing system to offer petaflop performance (more than one thousand trillion floating point operations per second).

[4]BOINC is used as the middleware for the World Community grid [20], a large volunteer computing project sponsored by IBM.

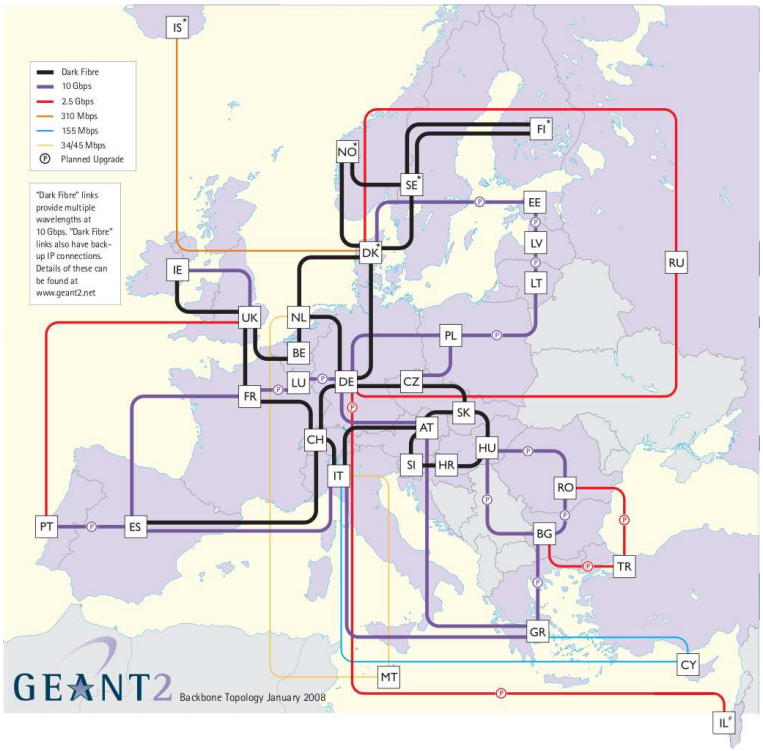

Figure 2.1: The GÉANT2 network [1]

and the capabilities offered by the network is closely linked to the ideas formulated by Gilder[5] in [22] :

> *When the network is as fast as the computer's internal links, the machine disintegrates across the net into a set of special purpose appliances* [6]

An example of such a high-speed network that is an enabler for grid technology in Europe is the GÉANT2 network shown in figure 2.1. This network connects 30 European research and education networks with over 30 million users in over 3500 institutions through multi-gigabit fiber optic links. This European backbone also connects to North America, Japan, Eastern Europe, the Mediterranean, the Middle East, Latin America and Asia-Pacific. It is a hybrid network, providing both routed and switched services over its extensive topology. GÉANT2 is the network upon which the European Enabling grids for e-Science (EGEE) grid has been built.

The actual evolution of the pan-European network serves as a good indication of the increasing capabilities of new wide-area network (WAN) technologies. The

[5] Gilder also formulated one of the three "technology laws", along with those of Moore and Metcalfe, which states that "the total bandwidth of communication systems triples every twelve months" [21].

[6] Originally this vision was articulated by Sun CTO Eric Schmidt in an email to Gilder stating: *"When the network becomes as fast as the processor, the computer hollows out and spreads across the network"*. This line of thought is also at the origin of Sun's credo *"The network is the computer"*.

first network, TEN-34 which was launched in 1997 used IP/ATM at 34 Mbps but the effective bandwith was around 2 Mbps to 10 Mbps. Its successor, TEN-155, was aimed at connecting the national research networks (NRENs) with IP/ATM at 155 Mbps. It also provided services on top of the network such as VPN and multicast. International connectivity was added in the next incarnation, GÉANT1, which used 10 Gbps fiber technology. The upgraded GÉANT2 network uses dark fiber and applies both Wave Division Multiplexing (WDM) and Time Division Multiplexing (TDM). Each wavelength offered by WDM delivers 10 Gbps and the network is engineered to support 40 wavelengths with 14 planned in use for 2008.

Note however that not all grid systems are built on top of high performance research networks. Many grid systems connect through the regular Internet, in which case more consideration must be given to the locality of data and processes in the network.

Other resources

Special-purpose devices, such as sensors [23, 24], accelerators, and telescopes [25] can also be connected to a grid in order to allow for data processing and storage, monitoring, and remote control [26].

2.2.2 Virtual Organization

A Virtual Organization (VO) is a well-defined group of people and/or institutions that share a common set of concerns and requirements. In order to meet these requirements, the participants typically share a set of hardware and software resources. This form of sharing goes beyond information sharing through e.g. document exchange, as found in *virtual enterprises*[7][28]. The sharing of these resources is conditional and temporary and each resource owner retains administrative control over their own resources. In addition, the sharing is highly controlled, with a clear definition of what is shared, who is allowed to share and under what condition the sharing occurs [3]. The relationship between members of a VO is based on equality, creating the need for mutual *authentication*. In addition, *authorization* mechanisms are required in order enforce that operations are consistent with the sharing relationship. The composition of a VO may also evolve over time, allowing for the admission of new members that have similar concerns and requirements.

In essence, VOs can be seen as virtual communities with their own set of rules that determine what computational and information resources their members can access. In EGEE for example, there are well over 100 different VOs spanning research domains such as astrophysics, biomedics and high-energy physics. Membership application for a grid is typically a formal process where the identity of the applicant is verified. If the applicant is allowed to access grid services, he is issued a personal grid certificate to do so. The VO membership application process may be similarly involved and requires proving your identity through the use of your personal grid certificate.

[7] The term virtual organization is also used in this context to denote the *an opportunistic alliance of core competencies distributed among a number of distinct operating entities within a single large company or among a group of independent companies [27].*

2.3 Definition

There is no universal accepted definition of "grid computing" or a "grid". This problem indisputably originates from that fact that the capabilities and properties of grid systems have always been evolving. The term grid computing was coined in the early '90s as a metaphor for making compute power as easy to access as electricity from the power grid. Unfortunately the analogy does not provide a clear definition of "grid computing". Consequently, many terms have been introduced for systems with grid-like properties without standardizing on what the "grid" actually stands for. It has lead to a proliferation of terms depending on the focus of the system. Examples are Compute Grids, Data Grids, Science Grids, Access Grids, Knowledge Grids, Bio Grids, Sensor Grids, Cluster Grids, Campus Grids, and Desktop Grids.

A further complication is the fact that the term grid has also been used in the context of locally shared resources that are under central control such as clusters[8]. A similar problem arose in the early days of the Internet in that vendors were claiming that private networks such as SNA and DECnet were part of the Internet, while others claimed that every local area network was a form of Internet. This ambiguous use of terminology was only clarified when the Internet Protocol (IP) became widely adopted for both wide area and local area networks.

There have been a considerable number of attempts to create a definition for grids [29]. Fortunately, there are some properties which almost all of these definitions have in common. Ian Foster, a pioneer in grid computing, combined them in a *three point checklist* [30]:

- A grid is a system with geographically distributed resources that are not under centralized control. Each organization retains full control over its resources.
- A grid uses standard, open protocols and interfaces for all essential activities performed by users (access, authentication and authorization, resource discovery, job submission,...). As such, the standardization and openess of protocols make grid systems available for a wide class of distributed applications and allow for the integration of heterogeneous resources. This clearly distinguishes grids from other types of distributed systems that are tailored to specific applications or systems.
- A grid should deliver non-trivial qualities of service. This means that a grid should handle resource failures and deal with the intrinsic heterogeneity of resources. Access to the grid has to be transparent with single sign-on functionality.

This checklist still leaves room for interpretation as to what *centralized control*, *open protocols* and *nontrivial quality of service* actually means. It is to be expected that through further developments the definition will become more discriminating.

[8] A resource management system (RMS) originally called GRD (Global Resource Director), was subsequently marketed by Sun Inc. under the name Sun Grid Engine (SGE), though it is actually a local RMS.

A more general and broadly used definition was also presented by Foster stating that *grid computing is coordinated resource sharing and problem solving in dynamic multi-institutional virtual organizations.*

2.4 History

Although the term "grid computing" emerged in the early nineties, and large-scale grid deployments were realized only recently, the idea of building large-scale distributed systems has existed for a long time. In the 1960s, J.C.R. Licklider formulated a vision that would lead to the construction of the ARPANET, a project that was at the nexus of the Internet as we know it today. Larry Roberts, one of the successors of Licklider, described his vision as follows [31]:

> *"Lick had this concept of the intergalactic network which he believed was everybody could use computers anywhere and get at data anywhere in the world. He didn't envision the number of computers we have today by any means, but he had the same concept - all of the stuff linked together throughout the world, so that you can use a remote computer, get data from a remote computer, or use lots of computers in your job. The vision was really Licks originally."*

In 1967, Len Kleinrock already expressed the key idea behind utility computing when he stated :

> *"We will perhaps see the spread of 'computer utilities', which, like present electric and telephone utilities, will service individual homes and offices across the country."*

During the mid-nineties, several projects in the field of *metacomputing* focused on providing a distributed platform for compute-intensive applications. Two representative projects from that time were I-WAY [32] and FAFNER [33].

The Factoring via Network-Enabled Recursion (FAFNER) project was started by Bellcore labs in 1995 as a response to the factoring challenge initiated by RSA Data Security Inc, a company that was named after the RSA public key cryptography algorithm. The goal of FAFNER was to build up a distributed metacomputing system to perform large number factorization in parallel. The distributed computing system was built using web servers and CGI (Common Gateway Interface) scripts for job distribution, monitoring and registration of contributing resources. The only requirement for a resource contributor was the ability to host a web server. The approach demonstrated by FAFNER would later on be adopted in other web-based distributed computing projects such as SETI@home [17].

The Information-Wide-Area-Year (I-WAY) project consisted of the construction of a high performance network linking 17 supercomputing centers and several virtual reality research sites. The goal was to integrate multiple internetworked supercomputers and advanced visualization systems in order to conduct very large scale

computations. Contrary to FAFNER, the I-WAY project was confronted with the problem of supporting a wider set of application groups. To standardize the computing environment, each contributing site installed an I-POP (Point of Presence) with a standardized software environment (I-Soft). A central resource broker (CRB) was deployed to distribute tasks to the I-POP servers, after which they were executed at a local site. I-WAY provided for a single-sign on feature. Using Kerberos authentication and encryption the CRB acted as a proxy, authenticating a user to third-party resources on the user's behalf. Contrary to FAFNER, I-WAY also supported coupled computations using the fast interconnects of the supercomputers through an adaptation of the MPICH library [34].

The middleware used by both I-WAY and FAFNER was not flexible and scalable enough to integrate resources on a global scale and make them available to a general class of applications. The I-WAY software base [35] was further developed and led to the first version of the Globus Toolkit [36], currently the base toolkit for many grid systems. The release of the Globus Toolkit offered grid middleware developers a modular toolkit with components that cover the basic services that are required for a grid system to operate. Such services include authentication and authorization, resource discovery and allocation, data access, and information services. As such, the toolkit was a first step towards standardized tools and protocols for building interoperable grid systems and for supporting the notion of dynamic virtual organizations. As standardization bodies such as the Global Grid Forum (GGF)[9] defined new standards for grid protocols, these were also taken up by the toolkit.

The Globus Toolkit has been used by a large number of national and international grid projects. Examples include the Open Science Grid, BEgrid, the European Data Grid (EDG) project and its successor, the Enabling Grids for e-Science (EGEE) project. The Globus Toolkit only provides basic services, and several additional components were developed or integrated by most of these projects in their middleware software stack. These components support e.g. global scheduling, file replication, automated file staging, and software package distribution and management.

In later incarnations of the toolkit (GT3 and GT4), the principles of Service Oriented Architectures were applied in order to further improve on the composability, interoperability and reusability of the components. The blueprint for this service-oriented architecture has been standardized in the Open Grid Services Architecture (OGSA). In practice, the implementation of the architecture by the Globus Toolkit heavily relies on Web service (WS) technology and open standards such as XML, WSDL, SOAP, WSRF and WS-Agreement. In parallel to this shift towards a service-oriented architecture, there is an increased focus on issues related to QoS guarantees and negotiation of QoS levels in recent developments in grid technology, middleware and standards. In addition, a more holistic view is adopted with an emphasis on distributed collaboration and dynamic formation of virtual organizations, as opposed to merely achieving the technical interoperability required to build large-scale distributed systems. In the scientific field, the application of this view

[9]The GGF merged with the Enterprise Grid Alliance in what is currently known as the Open Grid Forum (OGF).

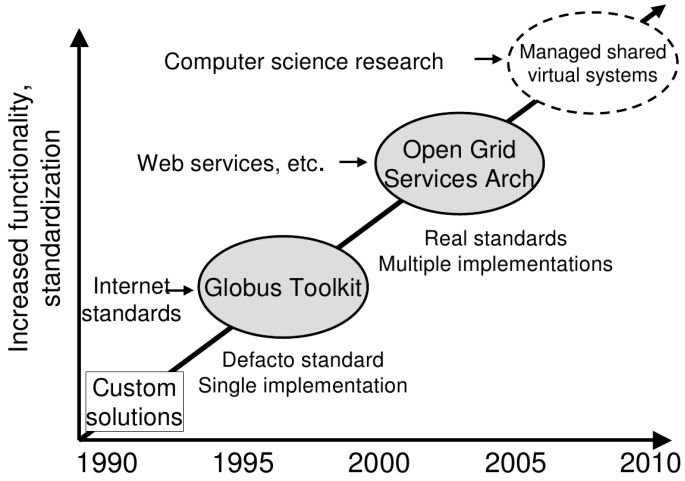

Figure 2.2: The evolution of grid technology according to Foster [2]

has lead to the notion of *e-Science*, a highly collaborative form of science for which grids are considered to be the key enabler. Finally, the large-scale deployment of grid technologies has shown the need for applying principles of autonomic systems such as self-management and self-healing (automatic recovery). Figure 2.2 presents a schematic overview of the history of grid technology according to Foster.

De Roure [37] has identified three generations of grid technology. He considers the first proprietary solutions such as those developed by the I-WAY and FAFNER projects as *first generation grids*. The middleware developments that mainly tackled the issues of scale and heterogeneity with a focus on delivering large scale computational power and data management facilities have lead to *second generation grids*. *Third generation grids* adopt a service-oriented model as prescribed by the OGSA standard.

Note that throughout these generations, other frameworks and systems such as Condor, Legion, and UNICORE have also emerged and some are still in use today. In fact, before the standardization of core grid service interfaces, the myriad of solutions resulted in several grid *islands* emerging, that were unable to cooperate and realize the vision of a global grid. Even today technical and practical issues, as well as the incomplete process of grid service and architectural standardization, still hinder full interoperability among different grid systems.

2.5 Architecture

2.5.1 Layered view

In Foster's seminal *The Anatomy of the Grid* paper [3], the architecture of a grid is divided into five high-level layers (see Figure 2.3).

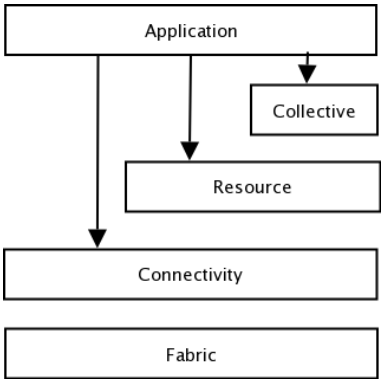

Figure 2.3: Layered grid architecture according to Foster [3]

- The **fabric layer** defines the resources that can be shared in a grid environment. These resources may be of physical nature, such as computational, storage and network resources, or their nature may be only logical, for example advanced network services and distributed file systems. These logical resources are implemented by means of their own protocol (e.g. NFS). Resources should provide two elementary services, namely an information service to allow for resource discovery and monitoring, and a management service to provide the resource owner with the necessary controls.

- The **connectivity layer** provides the core communication and authentication protocols used in the grid. Communication protocols such as those delivered by the TCP/IP protocol stack provide transport, routing and naming. The authentication protocols are built on top of them and have a broad set of complex requirements. One is *single sign-on*, or the ability, once authenticated, to access any available resources in the grid without further manual authentication. Another is *delegation* which allows the resources that are accessed by a user to access other resources on behalf of this user. The authentication protocols must also integrate with local authentication solutions such as the ones used in Windows and Linux. Finally, the protocols have to ensure data integrity and confidentiality as well.

- The **resource layer** uses the connectivity layer to securely monitor and control individual resources. This layer provides the actual information and management protocols. The information protocols enable the lookup of resource capabilities and usage monitoring. The management protocols provide functionality for access negotiation and the actual usage of the resources. They also are a natural accounting and policy enforcement point.

- The **collective layer** deals with all activities that are performed on collections of resources. Some fundamental examples are resource discovery, co-allocation, scheduling and brokering services, monitoring and diagnostic services and accounting and payment services. To deliver these services, the collective layer

accesses the underlying layers of the protocol stack.

- In the **applications layer** domain-specific applications built on top of the previous layers provide further abstractions for a specific set of users. An example is a grid portal focused on drug discovery which allows researchers to submit gene sequences of mutated virus strains, in order to estimate which drugs may be effective against these new variants.

2.5.2 OGSA : A standard service-oriented architecture

The standardization efforts within the Global Grid Forum (GGF) of the protocols and interfaces in Globus Toolkit 2, have lead to a standard for grid architectures called the Open Grid Services Architecture (OGSA) [38]. OGSA describes the base architecture for the components of the different layers presented in the previous section with the aim of increasing interoperability between different grid systems and increasing the composability and reusability of grid middleware components.

OGSA adopts the model of a *service-oriented architecture* (SOA) [39]. In a SOA, a *service* is an entity that provides some *capability* to its clients through a network-addressable interface. Clients appeal to a service's capabilities through a standardized *message exchange*. This allows for a loose coupling between clients and services and provides great flexibility in how services are implemented or where they are located. In effect, the core approach to service-oriented architectures draws largely upon interface-based design, one of the pillars of object-oriented programming which stresses the importance of separating interface from implementation. As such, it also yields similar advantages in terms of reusability, composability and loose coupling. The focus on language neutrality in a SOA, also enables services to be reused irrespective of the language in which they are implemented. In addition, a service-oriented architecture more explicitely deals with the (possible) distributed nature of services. A key concept in this regard is the notion of a *registry* in which service providers publish service descriptions and associated service contact points. This eliminates direct dependence of a client on the service's location in the network. Because of these properties, an SOA is appropriate for building grid systems that need to deal with the integration of heterogeneous distributed systems.

Like many other service-oriented architectures, OGSA relies on Web service concepts and technology to define the architecture. Web services (WS) are a platform- and language-independent stateless distributed computing technology. This platform- and language-independence is achieved through an adoption of eXtensible Markup Language (XML) for encoding the message data and describing the message exchange, and the use of standard Internet protocols for transmitting data across a network. OGSA standardizes on the use of the Web Service Description Language (WSDL) for *defining* service interfaces. The *invocation* of the interface's operations is not standardized although in practice the Simple Object Access Protocol (SOAP) is often used. SOAP usually relies on the HTTP protocol for the transmission of XML data, although other standard Web protocols such HTTPS or FTP can also be used. The notion of a registry is represented by a Universal Description, Discovery, and Integration (UDDI) registry, which is a web service in itself that allows clients

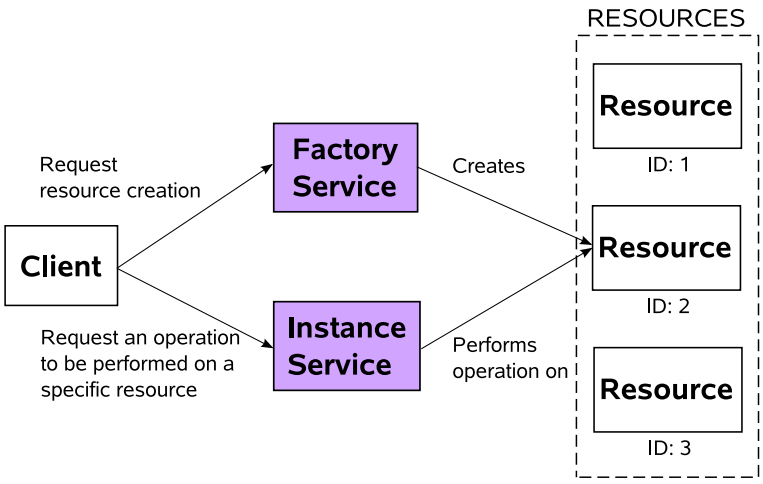

Figure 2.4: The WS-resource factory pattern [4]

to query service descriptions and obtain WSDL documents. More details on web services and the protocols involved can be found in [40].

During the integration process of grid and Web Services, it became clear that Web service technology was not fully able to deliver the necessary functionality for implementing grid services [41]. First, the stateless nature of a standard Web Service was not compliant with many grid services being stateful. In a joint effort between the Web and grid communities, the Web Service Resource Framework (WSRF) [42], an OASIS [43] standard, was introduced to support stateful Web services. In the WSRF model state is kept in an entity called a "resource", separately from the WS itself. Each resource has a unique End Point Reference (EPR). The client can then use this EPR to instruct a WS to refer to a particular state. Since a first-time client does not know the EPR in advance, it will bootstrap using a "Factory Service" which will create the resource and return the EPR. It then can interact with the "Instance Service" to perform actions on its assigned resource (see figure 2.4).

Another limitation of Web service technology also came to the fore with regard to the dynamic and potentially transient nature of grid services, which required provisions for lifecycle management and dynamic service creation. Finally, the typical client-server interaction pattern supported by Web services at that time did not cater for the delivery of asynchronous notifications from the Web service to the client.

Note that the OGSA does not prescribe the use of WSRF, although many implementations of the architecture use WSRF. An alternative software stack for implementing the OGSA is considered in [44] for example.

In OGSA, a *grid service* is defined as a Web service that conforms to a number of standard *interfaces* and *conventions* [2]. The interfaces relate to service discovery, creation and lifetime management, monitoring, notification. The conventions standardize with respect to naming and versioning.

Grid services are deployed in *hosting environments*. These environments typi-

cally carry the responsibility for dispatching invocations on the service's interface into implementation-specific actions (e.g. a call on a Java object implementing the service interface), and mapping grid-wide names (grid service handles) to implementation specific entities. In addition, they offer provisions for lifetime management and protocol processing that allows services to communicate across the network [45]. Implementations of such hosting environments are provided by component containers such as those found in the J2EE, Websphere or .NET frameworks.

2.6 Middleware

The software that organizes and integrates the resources in a grid is commonly referred to as grid *middleware*. Middleware automates the "machine to machine" interactions that create a single, seamless computational grid. It also offers implementations of grid services for authentication and authorization, and the management of data, information, resources and virtual organizations. Over the last years, many middlewares have been developed in support of several grid programming and usage models, and with varying degrees of functionality.

It is a vast effort to develop a middleware that covers the full range of functionalities required to realize the vision of secure and coordinated resource sharing across administrative and organizational boundaries. As a result, each middleware development project has focused on different aspects of these challenges, which has lead to software that is not general-purpose enough to serve the entire grid community, and at the same time is not interoperable with other grid middleware. This lack of interoperation and standardization has lead to wasted implementation efforts as a lot of middleware components were reimplemented in many projects instead of being reused.

2.6.1 The Globus Toolkit

Both the release of the Globus Toolkit and the definition of an architectural standard through OGSA have been important steps in dealing with the aforementioned problems. The use of the Globus toolkit frees middleware developers from building core services for common requirements such as authentication and authorization, data management, and job management from scratch. In addition, architectural standards such as OGSA have created better opportunities for integrating results of different projects focusing on specific requirements[10]. The latest two releases of the toolkit (GT3 and GT4) have been built according to the OGSA on top of WSRF. The relationship between the Globus Toolkit, Web services, WSRF and OGSA is represented in Figure 2.5.

An overview of the GT4 components is given in Figure 2.6. We briefly discuss these components here, an extensive overview can be found in [48].

[10] A prime example of a grid middleware project that pre-dated the standard, but recently migrated [46] to the OGSA is UNICORE [47].

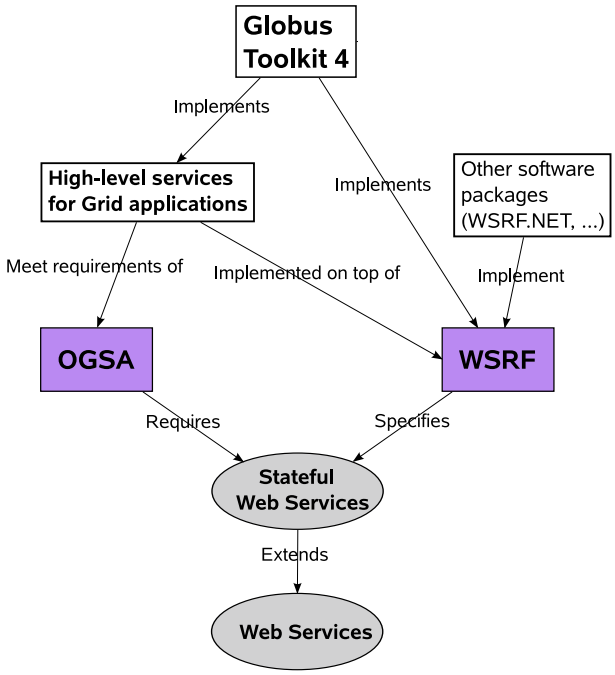

Figure 2.5: Relationship between OGSA, Globus Toolkit 4, WSRF and Web Services following [4]

The components are organized into the following five layers :

- **Security**
  The security layer includes components for authentication and authorization, credential delegation and single sign-on, as well as transport and message-level security. The layer is based on the Grid Security Infrastructure (GSI) which in turn relies on a Public Key Infrastructure (PKI) that is based on the X.509 standard. The authorization components in this layer provide both client-side and server-side authorization. The GSI also supports the delegation of credentials through *proxy certificates*. Through the delegation mechanism, a subset of the privileges of the user may be transferred to other entities. In a grid context, this is often required as a grid service might need to appeal to other services on behalf of the user in order to complete its task.

- **Data management**
  Remote data transfer is provided by the GridFTP and Reliable File Transfer (RFT) services. Whereas GridFTP requires the initiator to keep an open connection to perform a transfer, RFT acts much like a job scheduler in that users can submit a list of source and destination URLs to files, which are then moved on his behalf. In wide-area settings, data replication can significantly improve the turnaround time of data intensive applications and increase the scalability of the system [49, 50, 51]. Globus offers a Data Replication Service

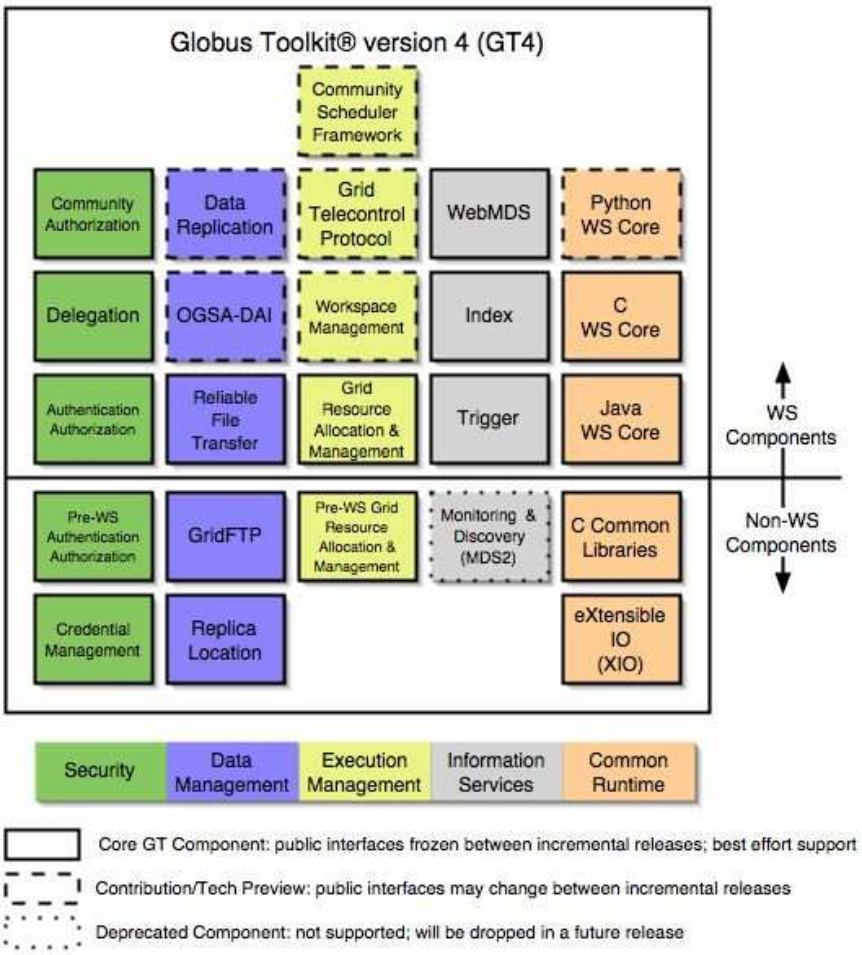

Figure 2.6: Globus Toolkit 4 components

(DRS) and a Replica Location Service (RLS) that manages the mapping between a *logical* file to potentially multiple *physical* files. The DRS internally uses both RFT and RLS for its operation. Note however, that these services only deliver the basic tools to replicate files, they do not include algorithms for automated and dynamic replication of data. In addition, OGSA Data Access Integration [52] (OGSA-DAI) components provide remote and standardized access to relational databases and XML data.

- **Information Services**
  The information services layer allows clients to monitor and discover services. This involves access to resource and service descriptions and attributes. This covers static information e.g. the operating system installed on a cluster, as well as dynamic information e.g. the expected waiting time for a job at a particular cluster. This information is disseminated to clients through the Monitoring and Discovery Service (MDS). MDS uses soft-state registration for dynamic information. It is also a framework for accessing system configuration and status information. The latest incarnation of MDS [53], is based on Web service standards such as WS-Properties, WS-Notification and WS-ServiceGroup to describe and disseminate information. Index services, which are typically organized in a hierarchy for scalability, aggregate data from multiple distributed information providers and cache information. The Globus Trigger service can be configured to trigger an action (such as sending an e-mail) when a condition on the monitoring data is fulfilled.

- **Execution Management**
  The execution management layer consists of components for initiation, monitoring, management, scheduling, and coordination of remote computations. The Grid Resource Allocation Manager (GRAM) implements the aforementioned requirements by interacting with different local resource management systems such as the Portable Batch System (PBS), Load Sharing Facility (LSF) and Sun Grid Engine (SGE). The capabilities of these systems are unified in a standardized interface.

- **Common Runtime**
  The common runtime layer includes various backends for hosting web services with implementations in various languages as well as a number of external libraries that are required for building grid services.

2.6.2 Requirements beyond the Globus Toolkit

Grid middlewares that are currently used for large-scale deployments such as the gLite middleware of the EGEE project, rely on the Globus Toolkit and combine components from the toolkit with a large number of software packages from other projects and with internal developments. The integration of such a large number of software packages is a complex and resource-intensive process. Indeed, the Globus Toolkit is merely a *toolkit* and lacks support for a wide range of requirements that need to be fulfilled in order to obtain a fully operational grid system.

Some critical aspects of grid systems that are not covered by components in the Globus Toolkit are :

- **Job management and scheduling**
  Although the Globus GRAM service can manage and deploy jobs on resources fronted by heterogeneous local resource management systems, it does not support complex workflows in which the jobs that are launched on the grid have interdependencies or data dependencies. The toolkit only provides the basic services to move and replicate data across the grid and launch individual jobs on a virtualized compute resource. In order to schedule complex workflows, global scheduling components are needed that prioritize job execution requests, coallocate resources and intelligently optimize data or job placement for the efficient execution of a grid application.

- **Package management**
  Grid middleware typically ships with a large number of software packages to provide for a standardized execution environment. Tools are needed to easily manage software packages integrated within the grid middleware and their dependencies. In addition tools are required to offer easy installation and configuration of the software packages that make up the middleware at a site that wishes to contribute resources.

- **Virtual Organization Management**
  The Globus Toolkit offers the technical components for enabling resource sharing across organizational boundaries. However, the actual definition and enforcement of the conditions under which this sharing occurs is not supported by the toolkit. In addition, the process of managing and issuing user certificates and VO membership through e.g. web-based registration portals is not supported by the toolkit.

- **Accounting**
  The Globus Toolkit does not provide any components for tracking resource usage by different users or VOs. This information is crucial however, in order to establish and maintain sharing relationships between stakeholders in the grid. Both the DGAS [54] and SGAS [55] projects however, provide accounting services for grids based on Globus.

2.6.3 Object-based middleware systems

The requirements on middleware of interoperability and platform independence have drive grid architectures towards WS-based architectures. However, a number of object-oriented approaches have also been developed. Some of these were built on top of distributed object frameworks such as CORBA [56], Java RMI [57] and Jini [58]. In such frameworks, objects are published and discovered in registries (similar to Web services) and they invoke one another's capabilities through remote procedure call mechanisms.

An advantage of the object-oriented approach is the reuse of the functionality for discovery, registration and communication of the framework [59]. Support for setting up transactions in distributed environments and notification mechanisms through remote events also contributed the attractiveness of this approach. Prior to the introduction of WSRF, Web services were unable to deliver these features.

Proponents of the object-oriented approach have also argued that it offers stronger programming abstractions and requires less development effort than does a web services approach.

Instances of object-based middlewares are JGrid [60], ICENI [61], CoABS [62], InteGrade [63], Legion [64] and Ibis [65] which is still an active project at present. Many of these projects aim for more dynamic and lightweight grids with respect to service composition and VO creation, as opposed to the more static and heavyweight approach of Globus-based middlewares.

2.7 Resource Management

According to [66], grid resource management, simply referred to as *resource management* in the following, is defined as *the process of identifying requirements, matching resources to applications, allocating those resources, and scheduling and monitoring grid resources over time in order to run applications as efficiently as possible.* This includes technical aspects (querying resource properties, sending application jobs to the batch queue of a local resource manager, reserving resources), as well as aspects related to optimization (what are the best resources to use?, when should a specific job be scheduled?). As such, an efficient resource management system is considered to be one of the key requirements for realizing a well-performing grid system. In resource management, a *job* is defined to be anything that needs a resource over a period of time. The process that involves decision making on what jobs are scheduled at what time and at what resource is referred to as *scheduling.*

The process of *scheduling* consists of (a) figuring out the possible alternatives of which resource is made available to which job at that what time and (b) figuring out which of the alternatives is optimal according to pre-set criteria.

According to Schopf [67] grid scheduling is comprised of the following three phases that take place after the job execution request by the user:

1. **Resource Discovery**
   In the resource discovery phase, the resource management system (RMS) queries the grid's information system to obtain a list of resources and selects those that the user has access to. In addition, minimal requirements on the execution environment are used to filter the candidate set of resources. These include for example, the operating system on which the job can run, the amount of RAM needed to run the job, and the set of libraries that should be available in the environment. The resource discovery process relies on the grid's information system to deliver this information. In addition, a standard language needs to be used in which resource providers can describe their

resources' characteristics and capabilities. Likewise, users should be able to express their requirements using agreed upon metrics and attributes. Common languages for achieving these goals are ClassAds [68], GLUE [69], and the Job Submission and Description Language (JSDL) [70].

2. **Resource Selection**
   From the candidate set of resources, a specific resource is selected for executing the job. This selection typically relies on dynamic information on the resource's attributes. Due to the fact that resources are shared and that dynamic information cannot be kept fully consistent in a wide-area grid setting because of scalability limitations, resource managers are often confronted with information that is out of date in this phase.

3. **Job Execution**
   After selecting the resource, the resource manager relies on standard interfaces and tools to submit the job to the resource's local resource manager. In addition, operations that prepare the remote execution environment for the execution of the job are performed. These include staging data such as input files and executables, and configuring environment variables. During the job's execution, the job is monitored and possibly automatically rescheduled to another resource if a resource failure is detected. The resource management system reports on the job's state throughout its lifecycle. Usually this is performed through a pull model, i.e. the user actively queries the RMS for the job's state. Upon the completion of the job the execution environment is sometimes *cleaned*, removing temporary files and settings. Some resource management systems also automatically stage output files to storage resources after job completion.

In a grid setting, schedulers usually form hierarchical structures with meta-schedulers (or *brokers*) at the top of the hierarchy and lower-level local schedulers (e.g. a cluster scheduler) providing specific capabilities at the bottom. These local resource management systems retain their autonomy and serve as a natural enforcement point for the organization's local sharing policies. Local policies are often enforced and expressed using a rule-based approach. Under the constraints of local policies, local resource managers then match the user's job requirements to physical resources that are allocated to the job and manage the job's execution. Different forms of scheduling policies are often supported such as First-Come First-Served (FCFS), Round Robin, priority-based schemes and fair-share scheduling. However, local resource managers have fairly limited provisions for advanced QoS and often provide a best-effort service. This complicates the scheduling task of the meta-scheduler significantly. Some local resource management systems support *reservations* of resources in order to improve QoS guarantees and move away from a best-effort model. A key issue then is to fulfill the conflicting requirements of providing guaranteed quality of service, while simultaneously optimizing for maximal resource utilization, fairness and overall job response time [71]. At present, local resource managers have limited capabilities for tackling this challenge without

significant administrator intervention. Examples of widely used local resource managers and schedulers are MAUI/TORQUE [72], Portable Batch System (PBS) [73], Load Sharing Facility (LSF) [74] and Sun Grid Engine (SGE) [75].

For a grid RMS, both the technical and optimizing aspects of resource management are more involved compared to local resource management systems such as those found in operating systems or batch schedulers for clusters. This can be attributed to :

- The more heterogeneous nature of grid resources.
- The lack of control a grid resource manager has over *shared* resources that are under *local control* of site administrators.
- The high degree of sharing which leads to a high of variance and unpredictability with respect to the availability of resources.
- The different objectives posed by users with respect to the optimization of the application's execution, and the conflicting preferences of the different stakeholders in a grid.
- The impact of the network latency and bandwidth limitations on the execution time of a job.
- The need for coallocating distributed resources of different types (e.g. network, computational, storage) over time.
- The larger scale at which grid resource managers need to operate, both in terms of the number of jobs and the number of resources that need to be managed.

2.8 Summary

This chapter has introduced the main concepts, ideas, challenges and driving factors behind grid computing. We have discussed how grid technologies and architectures have evolved from proprietary solutions to standards-based and service-oriented architectures, in order to foster reusability and interoperability. In addition we have presented the process of resource management in grids and discussed the main challenges in this field.

CHAPTER 3

Market-based Grid Resource Management

3.1 Introduction

The distributed and multi-organizational context in which grid systems operate presents significant challenges for grid middleware developers. A large part of these challenges are purely *technical*, in that they require the development of software components to fulfill technical requirements related to security, data management, monitoring, resource allocation and such. These requirements extend similar requirements for local resource management systems, to account for the large-scale, heterogeneous, multi-organizational and wide-area character of grids.

These technical challenges and requirements are well-understood and have lead to the development of standardized solutions and toolkits such as the Globus Toolkit [38]. In addition, large-scale deployments of current grid middleware solutions have demonstrated that the technical difficulties encountered when integrating distributed resources have, in large part, been solved. This is exemplified by projects such as EGEE which successfully integrate resources over hundreds of institutions, while supporting a wide spectrum of applications from various domains.

However, several non-technical issues related to grid resource management have not yet been resolved and impede the full realization of *the grid* as a global transparent infrastructure for on-demand resource provisioning. More specifically, current grid middlewares assume a static model of *cooperation* among the different organizations. As such, the *sharing* of resources that is inherent in the grid vision is assumed to naturally emerge and be sustainable. This assumption is highly debateable and unfortunately the current large-scale adoption of grid technology as in the EGEE project for example, does not offer empirical evidence for its justification. This is because the sharing proposition is embraced by the organizations that participate in the EGEE grid infrastructure, because it is directly *imposed* by the project's funding

agency[1]. The *sustainability* of the current sharing model that is used in scientific grids is questionable. In fact, within the context of the UK e-Science grid project there have already beens signs that a lack of enforcement of the shared infrastructure model, can lead to the reappearance of a situation in which resources are removed from the grid or are only shared on a limited scale (i.e. among a very limited number of parties)[2].

In addition, static bilateral sharing agreements are often hard to negotiate, and result in limited flexibility and *openness* with respect to the integration of new parties in the global grid infrastructure. As a result, several grid 'islands' or 'silos' emerge and the full potential of the infrastructure is not realized.

We argue that the source of this problem lies in the limited incentives resource providers currently have to openly share resources with other parties. This is not surprising, as current grid middleware does not offer any direct mechanism by which a provider is rewarded for offering resources. In fact, only recently have accounting schemes and systems such as DGAS been practically used within large-scale deployments such as EGEE. These systems can give an overview of the actual resource usage of different VOs, and can serve as a policy tool to steer further infrastructural developments (thereby possibly rewarding providers with a high degree of sharing for their efforts).

A second key issue is the use of a *best-effort* service level within grid resource management systems. Current systems offer very limited provisions for delivering quality of service (QoS). In EGEE for example, jobs are sent out to brokers which further direct them to compute resources at a specific *site*. These jobs are handled in a first-come-first serve manner and no prioritization among jobs is effected in order to offer more advanced QoS to users. In addition, resource usage is basically free from the viewpoint of an individual user[3].

This zero-cost, first-come-first-serve approach of the current resource management model takes away any possibility of introducing differentiated QoS levels because users are free to always request the highest QoS. Incentives are needed that push users to formulate their QoS requirements in a well-considered manner. Current resource management systems do not have the means to deliver such an incentive. This situation has been recognized by Schopf as one of the top ten questions in the development of grids for the scientific community [76].

A third problem is that resource management and scheduling approaches found in production-level grid middleware have a strong focus on the efficient scheduling of jobs from a *system-centric* point of view. The RMS schedules jobs in a way that maximizes the overall utilization of the infrastructure or that obtains the highest possible overall system throughput. These approaches to scheduling do not take into account the actual *value* grid users associate with their computations. As a result,

[1]For the EGEE project this is mainly the European Union through the Information Society Technologies (IST) programme.

[2]Personal communication with John Darlington, director of the London e-Science program at GECON 2007).

[3]Note that VO-wide sharing rules do restrict the share that an entire VO can get on the grid. These are defined based on the bilateral sharing agreements mentioned previously.

the broker may be performing optimally from a system-centric point of view, but suboptimally from a user-centric point of view [77]. The potential for value loss as a result of these suboptimal scheduling decisions increases with the load on the grid infrastructure and the heterogeneity of the valuations among users. As shown by Chun [78], although in the context of resource allocation on sensor networks, user valuations can vary by four orders of magnitude[4]. Traditional resource management systems lack the means for allowing users to express their preferences and objectives in a fine-grained manner. In addition, they lack the mechanisms by which resource allocations can be optimized in face of such preference elicitation. Indeed, given a set of heterogeneous preferences and disparate goals it is difficult for a RMS to decide on one common optimization goal to steer the allocation of resources.

In this chapter we will explain how market-based approaches to resource management[5] can provide a solution to these problems[6].

3.2 Market-based resource management

The introduction of an electronic market place for trading grid usage rights provides a promising approach to deal with aforementioned issues. Such a market place is open to all parties that wish to participate in the system as a provider or consumer. The incentives for (well-behaved) participation in a market stem from a common value model and accounting system that charges for the requested service or rewards for the service that is delivered. A number of benefits result from this approach.

3.2.1 Openness

Firstly, this improves on the negotiation model that is currently in place in which users first need to lobby for access rights which are subsequently enforced by long term, static policies. A market-based approach allows for greater flexibility in terms of accepting new parties in the infrastructure and in terms of determining usage rights for users, resulting in a more open and agile grid infrastructure. This is especially important for the potential user base that is formed by small research institutions or SMEs that have no prior relation to resource providers. Their relative cost for maintaining and procuring computing infrastructure is high due to limitations of scale and often strongly fluctuating requirements. As a consequence, these users are not able to engage in bartering during bilateral agreements as they do not own a significant amount of computing infrastructure. On the other hand, this user base has the greatest potential for benefiting from the added value a grid infrastructure can provide. The possibility to pay for usage rights in a piecewise

[4] To our knowledge, there have been no published records of such valuation data for grids.

[5] Also referred to as *economic models* for resource management [79].

[6] Although we will focus on the use of economic principles in resource management for grids, the *economics* of grid computing and associated business models are also a subject of study [12, 80, 81, 82, 83]. Both of these foci are part of the *Grid Economics* field which is at the confluence of grid systems and economics.

manner enables them to tap into this added value. In effect, this flexible negotiation model is a prerequisite for realizing the vision of on-demand provisioning that is brought forth by the grid computing paradigm. The recent uptake of cloud computing through offerings such as the Amazon Elastic Compute Cloud [84, 85], has clearly demonstrated the need for more openness and the role that charging mechanisms play in realizing the vision of utility computing.

3.2.2 Sustainability

Secondly, the use of market-based resource management introduces the possibility of charging a cost for sharing resources. This provides an incentive for resource owners to step into the grid infrastructure in a provider role. In this way, a well-organized market is an important step towards sustainable grid infrastructures where economic incentives stimulate organic growth when demand exceeds supply. Such an incentive is currently missing, a consequence of the funding structure of current grid projects and systems.

An important premise to this claim is the use of a unit of currency to which both consumers and providers attribute a value. Virtual currencies can be used in the system in order to decouple the monetary value of credits in the system from a specific monetary currency. In this case however, it is important for the correct operation of the market that the virtual currency is convertable to a good to which a provider attributes a certain value [86]. One possible approach is to provide an exchange service that offers the means to convert the virtual currency to a commonly accepted currency, e.g. the American dollar. This approach is currently applied in virtual worlds such as SecondLife [87] where the virtual Linden Dollar can be exchanged for American dollars on LindeX [88].

3.2.3 Value maximization

Thirdly, the market's operation and resulting prices fulfill the role of a communication bus over which market participants transmit their valuation information[7]. This information is typically transmitted in the form of *bids* which describe the service request, QoS requirements and the attributed value which is expressed in a standardized metric (e.g. American dollars). Depending on the market mechanism that is employed, this information can be used to prioritize conflicting servicing requirements in times of congestion in order to maximize the value of the grid infrastructure, as perceived by its users.

An important property of market-based approaches is that request prioritization and resource allocation decisions can be steered by an optimization strategy that only needs to consider the bids in order to select the set of service requests that will collectively generate the most value in the system. This is because the bids'

[7]The role of prices as an important medium for communicating complex preferences underlines the thoughts of Friedrich Hayek, one of the key founders of the Austrian School of economics, who states that the economic problem (of allocating resources to consumers), *is a problem of the utilization of knowledge which is not given to anyone in its totality [89].* As such, the key contribution of a price system is considered to be its efficiency in communicating information in a system in which the knowledge of the relevant facts is dispersed among many people [89].

valuation reports synthesize the complex preferences within the user population in a form that enables the RMS to aim for optimal value without the need for complex multi-criteria optimization. The valuation reports themselves stem from the goals, means and strategies used by individual agents that participate in the mechanism. As such, the optimal division of resources is said to *emerge* from the individual actions of these agents in the mechanism.

This results in a distributed form of decision making which has been used for thousands of years to control the complex division of scarce resources in real-world economies. This gives market-based approaches an advantage over current grid resource management approaches that use centralized control. Although approaches with centralized control can be optimized and fine-tuned to work efficiently, this is very time-consuming. In addition, the resulting system is more rigid and brittle compared to a market-based approach [79]. The large-scale and multi-organizational setting of grids further enlarge the problems tied to achieving efficient centralized control.

In order to obtain the maximal value in the system, a mechanism is required that encourages parties to report their valuations truthfully. This implies that users are endowed with limited budgets to back their valuations and that the market mechanism has the property of incentive compatibility[8] [90] or leads to a truthful Bayes-Nash equilibrium[9] [91]. In practice, incentive compatibility is hard to achieve, especially if mechanism properties such as budget balance and individual rationality[10] are also required [92]. However, mechanisms that are approximately incentive compatible can still obtain high efficiency as shown by Schnizler [93].

3.2.4 Well-considered resource usage

Fourthly, incentives are introduced that stimulate users to use the infrastructure conscientiously and to formulate their QoS constraints both in line with their own requirements and those of others using the system. This incentive arises from the fact that formulating stricter requirements than necessary, results in higher costs in times of congestion. The dynamic formation of prices thereby indirectly forces users to take into account the requirements of other users when optimizing their own utility, as they influence the cost level.

These incentives do not exist in systems that do not dynamically value resource usage based on the level of contention in the system. They enable an economic resource management system to obtain a higher overall satisfaction level across the user population.

[8] A mechanism is incentive compatible if participants in the mechanism fare best when bidding truthfully.

[9] A mechanism is Bayes-Nash incentive compatible if any agent's expected utility maximizing strategy in equilibrium with every other agent is to bid truthfully.

[10] A mechanism is individually rational if its outcome cannot induce a negative utility for the participating agents in equilibrium

3.3 Market mechanisms

In an *environment* with a set of *agents*, A market *mechanism* defines possible *actions* for each agent, and a mapping of the agents' actions into an *outcome* [94]. The outcome includes

- A component that determines the distribution of the goods that are sold through the mechanism to the agents. This is governed by a *choice rule*.
- A component that determines the payments that the agents should make in order to obtain the goods. The determination of these payments is governed by a *payment rule*.

The market mechanism thus defines how negotiations and trades are performed, i.e. according to which messaging protocol.

As an example, consider the English auction mechanism for a single good. The agents are the bidders in the auction. The sole action allowed by a bidder is to signal that it will accept the asking price broadcasted by the auctioneer. The choice rule of the English auction allocates the good to the highest bidder at the end of the auction. The payment rule of the auction makes the highest bidder pay the last broadcasted asking price.

3.3.1 Posted prices

In markets with posted prices, providers publish the prices they wish to receive for their goods. A consumer then considers these prices and acquires the goods required from the set of providers that deliver the greatest amount of utility. As such, there is no direct negotiation or price formation process. A posted prices approach to economic grid resource management has been presented by Buyya [95].

An advantage of this model is that it incurs minimal costs for establishing a trade. Indeed, in the real-world economy, stores for the general public typically adopt a posted pricing model as the organization and execution of a price negotiation process among a significant amount of buyers is too costly in that context. In addition, the operation of the market under a posted prices model is very simple and easy to understand for consumers and providers. A downside of this approach is that optimal efficiency is not realized because a posted pricing scheme cannot immediately steer the market towards a supply and demand equilibrium wherein goods are allocated to the agents that value them the most. Hence, the scheme is characterized by unrealized utility and revenue. In addition, goods might remain unallocated if the price levels are set too high.

These inefficiencies are depicted in Figure 3.1 for a market with a single buyer and seller, in which a second buyer enters the market later on. In the first time period, the provider overprices its resources. As a consequence, the provider's resources remain idle and the utility of the first buyer is not realized. In the second period, the demand curve for the first buyer rises above the posted price. This results in unrealized profit for the provider. In the third period, more than one buyer is interested in acquiring resources. However, the provider has no means to determine the buyer with the

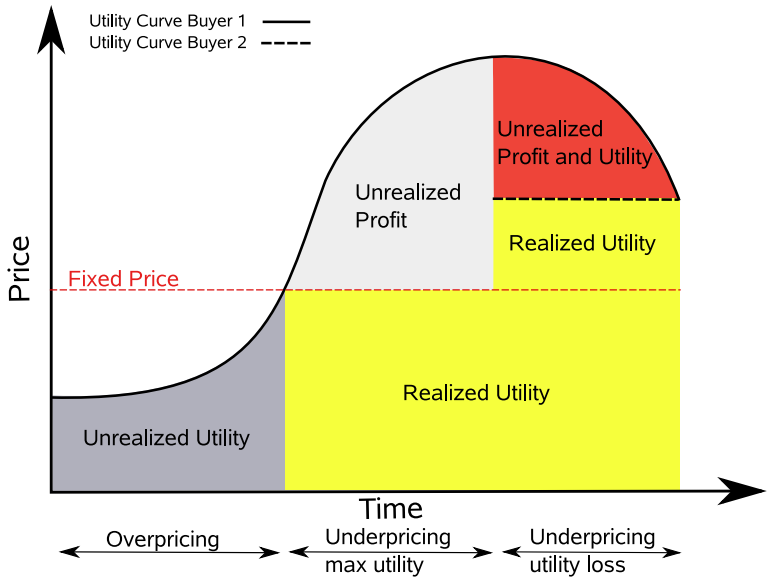

Figure 3.1: Inefficiencies arising from a fixed-pricing approach (following [5])

highest valuation for its resources[11]. As a result, the provider incorrectly decides to service buyer 2 who has entered the market with a lower valuation than buyer 1. This now leads to both unrealized utility as well as unrealized profit for the provider.

The magnitude of these inefficiencies depends on the heterogeneity of the valuations of the different consumers and how strongly these fluctuate over time. In addition, it depends on the providers' ability to advertise prices that approximate the equilibrium prices, and on the frequency with which the posted prices are updated.

In the setting of grid computing, wrongly configured prices can have a severe impact on the performance of the resource management system. Since most computational grid resources are typically non-storeable commodities (i.e. you cannot reclaim unused CPU cycles or network bandwidth), prices that are too high lead to lost opportunities for workloads to be scheduled on the system. Prices that are set too low result in the reappearance of a first-come-first-serve model and the associated disadvantages of this model discussed in the previous section.

3.3.2 Auctions

Auctions are one of the oldest[12] and most widely adopted market mechanisms, witness the use of auctions in online markets such as e-Bay, the New York Stock Exchange, or the UMTS/FCC spectrum auctions in Europe and the United States. Auction portals such as the Internet Auction List [97] presently host over 2500 auc-

[11] Note that if the provider would decide to query the buyers for their valuation, they would simply report the highest possible valuation as they know that the resources will be sold for the fixed price

[12] Historical records of the practice of auctioning in Babylon date back to 500 B.C. [96]. One of the most famous accounts of the auctioning process in ancient history involved the whole Roman empire being sold by an ascending auction in 193 A.D. by the Praetorian Guards.

tions for various goods, ranging from domain names to wines. Auctions are defined by McAfee [98] as

> *"A market institution with an explicit set of rules determining resource allocation and prices on the basis of bids from the market participants."*

In auctions buyers submit bids to the auctioneer. These bids describe their preferences e.g. price and quantity of goods. The auctioneer checks the *admissibility* of these bids according to the auction's bidding rules. The auctioneer then performs *winner determination* which involves a selection of the winning bids. In addition, the payments are determined for the winning bids. At that point the auction is said to *clear*. During the auctioning process, the auctioneer can reveal intermediate information on the status of the auction (e.g. the current highest bid).

According to Wurman [99], over 25 million types of auctions are possible. The taxonomy presented in [99] categorizes auctions based on aspects related to:

- **Bidding rules** that determine under which conditions bids can be considered admissible and who is allowed to bid. This also includes a categorization on the cardinality of buyers versus sellers in an auction.
- **Information Relevation** process that determines what information is revealed in which manner to the bidders during the auctioning process. This typically involves the sending of *price quotes* that represent a summary of the current bid state in the auction.
- **Clearing policy** that corresponds to aforementioned payment and choice rules. This policy determines when and how the auction clears or closes. This includes aspects related to closing conditions and timing, tie breaking and auctioneer fees.

The four basic auction types are [100] :

- The **English Auction** (or first-price ascending-bid auction) in which one seller offers a good for sale and buyers iteratively overbid each other until no one places a higher bid. Overbidding can be done by having the auctioneer announce prices at which bidders signal that they accept the new price, or by having bidders call bids themselves[13]. At that point, the good is allocated to the highest bidder at a price which corresponds to the latest bid. Common variants of the auction include a reserve price imposed by the seller, involve multiple units of the same good instead of a single good, or impose a minimal bid increase relative to the currently highest standing bid as an *activity rule*.
- The **Dutch Auction** (or first-price descending-bid auction) in which the auctioneer calls an initial (high) price which is subsequently lowered continuously until a bidder accepts the current price.

[13] As such this auction corresponds to the linguistic derivation of the word auction from the Latin verb *augere* which means *to increase*

- The **First-Price Sealed-Bid Auction** in which all bidders submit a sealed bid to the auctioneer. The good is sold to the highest bidder at the price given by its bid.
- The **Vickrey Auction** [101] (or second-price sealed-bid auction) which operates like the first-price sealed-bid auction, except for the fact that the highest bidder pays the price that corresponds to the second-highest bid.

Other types of auctions are :

- The **Double Auction** (or *exchange*), in which both buyers and sellers submit bids to the auctioneer. The seller bids are often referred to as *asks*. The auctioneer matches asks with bids and depending on the payment rule, determines a price for the trade. The seminal model of this type of auction is the k-double auction in which a single seller and a single buyer submit a bid b and ask a. If $b \geq a$, a trade is made at a price that equals $kb + (1-k)a$, where $0 \leq k \leq 1$[14]. Many variants of the double auction exist. One characteristic is the timing of when the auction is cleared. This can either be done continuously as new bids and asks arrive, or non-continuously in which case the auctioneer waits for all traders to place their bids. The first variant is referred to as the Continuous Double Auction (CDA), while the latter is called a Clearing-House (CH) or call market. Some markets combine both approaches, e.g. stock exchanges that open the trading day with a clearing-house auction and move on to continuous trading afterwards. Another variant of the double auction allows for multiple units of a homogeneous good to be traded instead of a single good.
- The **Walrasian Auction** which is a double-sided auction that directly applies the principles of general equilibrium theory [102] to determine equilibrium prices. An entity, called the Walrasian auctioneer[15] collects supply and demand levels for a limited set of standardized goods among the consumers and providers for a particular price point. Based on these reports, the price is adjusted to bring the market to equilibrium. This approach is adopted in the context of economic resource management by Wellman [103], Wolski [104] and Vanmechelen [105].
- The **Combinatorial Auction** which allows bidders to formulate bids that are conditional on the coallocation of a set of (heterogeneous) goods [94].

For a more extensive overview of auction theory and design we refer to [100].

3.3.3 Negotiations

Negotiations involve an iterative communication and decision making process between two or more agents [106]. In that sense, negotiations can be seen as a generalization of auction mechanisms [107].
In practice, the main differences between negotiations and auctions are:

[14] This is also referred to as a *k-pricing* scheme
[15] Named after Leon Walras (cfr. Chapter 5)

- Negotiations are less *open* than auctions and often involve a limited shortlist of sellers or buyers. The negotiation process of *bargaining* for example, is often bilateral.

- Negotiations can be used to discuss multiple facets of a trade. Not only can the price of trade be dynamically determined, but various additional conditions can be negotiated for (e.g. time of delivery or interest rates for buyer credit). Consequently, negotiations allow more complex trading contracts to be developed. In addition, negotiations can accommodate the complex relationships buyers and sellers might have. In settings such as business-to-business transactions, these can dominate considerations related to the price of a good [108].

- Negotiations do not necessarily involve a third-party entity that controls and standardizes the protocols and process of trading. Such negotiations are called unstructured or semi-structured [106].

- Negotiations involve agents on *both* sides of the trading process to adapt their offers iteratively to arrive at a mutual agreement.

- Negotiations rarely involve dissemination of information on the offers made, among the buyers.

- Negotiations often require more interaction back and forth and are more time-consuming compared to auctions [109].

The appearance of multi-attribute online auctions in which multiple possible heterogeneous goods are traded and in which more general utility models are used that are not solely a function of price, has blurred the distinctions between negotiations and auctions [106, 108]. According to Gimpel et al. [107] the key differences that remain relate to the structured versus unstructured nature of the protocol and negotiation process. In the context of automated electronic trading such as the one we will consider in our work however, structured protocols are a necessity in order to enable the encoding of the market mechanism in a computer program.

A number of theoretical studies on the differences between auctions and negotiations have indicated that, from the perspective of a seller obtaining maximal revenue, auctions are to be preferred over negotiations [110, 109, 111]. The bottom line is that the additional control and strategic options a seller has during the price formation process in a negotiation protocol, does not outweigh the potential of an auction attracting more bidders. As shown by Milgrom [109], as soon as auctions are expected to attract one more buyer compared to direct negotiations, they are to be preferred over negotiations. However, empirical studies have indicated that while auctions do result in markets with a higher degree of competition, they do not necessarily lead to significantly better prices for sellers compared to negotiations [112].

3.4 Application of markets to Grids

This section gives a high-level and generalized overview of how markets can be applied to grid resource management.

3.4.1 The Grid Value Chain

The idea of adopting a market-based model for resource allocation in grids can be employed at different levels (or tiers) of the grid value chain that is shown in Figure 3.2.

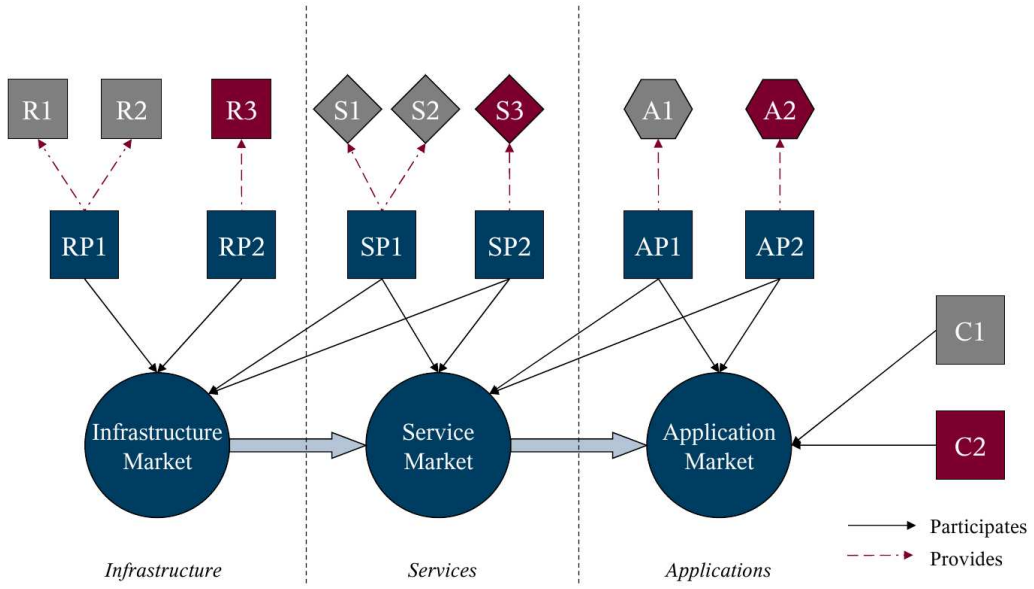

Figure 3.2: The grid value chain

Within the *application tier*, users can negotiate with application service providers (ASPs) through the market mechanism, in order to obtain access to a specific application under a given set of QoS constraints. The negotiated price for service delivery can include both the licensing costs for using the software, as well as the costs for obtaining the infrastructure and basic services required to run the application within the given QoS constraints. The ASP can then appeal to the market in the *services tier* to obtain access to core services (e.g. brokering services that offer uniform access to high-level virtualized computational services, data storage services with automatic replication support). Finally service providers negotiate in the *infrastructure tier* to buy access rights to raw resources (e.g. raw disk storage or processing power).

In the remainder of this work, we will focus on the market mechanisms that are used in the infrastructure tier although some of the ideas presented in chapters 5,6 and 7 can also be applied to the other tiers.

3.4.2 Entities and Services

The use of economic principles in grids leads to a number of new entities and services which we briefly discuss here.

Consumer

A *consumer* corresponds to an entity that participates in the market mechanism in order to acquire resources. We do not assume that this necessarily refers to an actual user actively formulating bids. Instead, a computational agent might act as a consumer in the mechanism on behalf of a user or a group of users to achieve a high-level goal. Consider for example a user that wants to finish a simulation study within the next 2 hours. The user is willing to spend up to 10 euros if the simulation is executed within this QoS constraint. Further assume that a market mechanism is used in which with each provider sells its resources through single-unit English auctions. The user would then communicate its high-level goal to an agent. This agent then :

1. Translates these requirements into a more concrete goal (e.g. reserve 50 CPUs on the grid for a duration of 1 hour for less than 10 euros)
2. Decides in which auctions to participate in order to acquire the resources
3. Instructs a set of bidding agents to participate in the English auctions on its behalf.

Provider

A *provider* corresponds to an entity that delivers resources to the grid infrastructure. Similar to the model used for consumers, a provider is represented by an agent in the system. It attempts to sell the provider's resources according to a high-level goal, e.g. obtain the maximal level of infrastructural utilization or generate the maximal amount of revenue.

When resources are traded in a market, a best-effort service model is no longer sufficient. More specifically, when payments are involved, consumers typically demand a more resilient form of QoS. This is because without QoS guarantees, it becomes very difficult for consumers to value resources. For instance, how would a consumer value a resource that has an indefinite waiting time? The delivery of QoS and market-based resource allocation thus go hand in hand. On the one hand, QoS is required for a correct valuation of resources as goods in markets, while on the other hand, markets are necessary for providing incentives for well-considered formulation of QoS constraints. Therefore, the provider typically hosts an additional service that allows consumers to formulate a Service Level Agreement (SLA) that governs the QoS constraints under which the provider needs to deliver the resources. Such a service can be integrated in a advanced local RMS, or can be deployed on top of a best-effort RMS.

Good

A key aspect of developing a market-based RMS is the definition of the *goods* that are traded in the market. As mentioned in section 3.4.1, we will focus on the raw

resource tier in this dissertation. Within this tier, important modeling choices still have to be made with respect to the types of resources that will be sold and the resource attributes that are exposed to the bidders. Accurate modeling of resource characteristics allows bidders to formulate their valuations in a fine-grained manner. However, it also increases the amount of information that needs to be delivered to bidding agents. In addition, it complicates the bidding agent's valuation logic. These modeling choices will be given further consideration in chapters 5,6 and 7.

RMS

Within the context of market-based resource management, the RMS is extended with a market mechanism that enables the grid stakeholders to formulate their preferences. The outcome of the mechanism is then used by the RMS as a value maximizing division of resources among consumers. Aside from this extension, the RMS maintains the properties of a traditional grid RMS in that it allows for the discovery of resources, takes care of the interaction with local resource management systems, and monitors the job's execution. Note that monitoring the execution of a job now also implies checking whether a provider fulfills its obligations to comply to the negotiated SLA(s).

Bank

A bank service is required in order to handle the transactions related to the payments that are linked to the trades between consumers and providers.

3.5 Summary

This chapter has highlighted a number of problems in traditional grid resource management systems and has motivated the use of market-based approaches in order to address these problems. After introducing some key terminology, we have given a high-level overview of a number of market mechanisms and have indicated how markets can be applied to grid resource management.

CHAPTER 4

The Grid Economics Simulator

4.1 Introduction

A key aspect of any research activity that engages in system and algorithm design is the *evaluation* of proposed systems and algorithms. Such an evaluation involves a scientific assessment of the properties and behavior of systems. In some cases, a theoretical analysis can prove that the system under study attains specific properties of interest. An example of such an analysis is a theoretical proof which establishes that the time complexity of an algorithm is quadratic in its input size. However, for complex systems an analytical approach is often infeasible or requires assumptions which are too stringent or unrealistic.

This is the main reason why many researchers in the field of distributed computing, and more specifically grid resource management, resort to *empirical* evaluation of systems. Empirical evaluation relies on results obtained from *experiments* in which the system under study is brought into operation. The system's behavior and properties are then evaluated based on the outputs measured during the experiment. A first possibility for empirical evaluation through experiments involves the deployment of the new system in a real-world setting. An example of this approach is the deployment of a new scheduling algorithm on a production-level grid infrastructure, while measuring the task throughput that is achieved.

The main advantage of this approach is that the experimental outcome is *realistic*, in the sense that it reports on *actual* behavior of the system in a real-world setting. As such, all the details that impact the performance and behavior of the system are guaranteed to be captured in the experiment. However, a great number of disadvantages inhibit the practical use of this approach in the context of grid resource management. First, there is a very limited *availability* of real-world grid platforms for experimentation. This is a consequence of the high costs inherent

in setting up and maintaining large scale grid infrastructures. Another aspect that makes real-world experimentation difficult, especially in the context of economic grid resource management, is that a market-based system operating in a realistic setting requires a potentially large number of individuals that interact with the mechanism through bidding. Secondly, the level of *reproducibility* of such experiments is severely limited. This is because real-world grid platforms are dynamic systems which are subjected to varying usage patterns and resource availabilities. Thirdly, for experimentation to result in a robust assessment of the system's properties, the system needs to be studied in a wide range of settings. For grid resource management systems, this requires experimentation under varying usage patterns and infrastructural arrangements. Real-world experimentation does not offer enough *flexibility* in this regard. Fourth, obtaining a high level of *efficiency* when running experiments on real testbeds is difficult. This is because on the one hand, the practicalities involved in setting up experiments can be very time consuming, while on the other hand the actual operation of the system under study can take a large amount of time. Finally, real-world experimentation makes it difficult to directly *compare* approaches with each other. This is caused by the limited reproducibility mentioned earlier, but also by the difficulty of packaging existing implementations and deployment procedures in a format that allows fellow researchers to reuse them efficiently for comparative analysis.

A second approach to empirical evaluation uses *simulation* in order to deal with these issues. Through simulation, one *imitates* the operation of a real-world process or system over time [113]. Such an imitation is achieved by the development of a simulation *model* which provides an approximation of the operation of a real-world system. Developing a simulation model requires that a number of assumptions are made with respect to the system's operation. As such, care must be taken that these assumptions do not invalidate the conclusions that result from experimentation through simulation. One way to deal with this problem is to *validate* the simulation model by comparing its outcome to results from real-world experimentation.

The field of numerical, computer-based simulation, in which the simulation model and interactions among entities in the simulated environment are expressed in program code, is especially well-suited to tackle the disadvantages that arise from using real-world experimentation:

- *Availability* : The only requirements for doing computer-based simulation experiments are compute power and simulation software. Therefore, simulated platforms for experimentation are readily available, even though in some cases the simulation software needs to be extended or developed from scratch for the purpose of the investigation.

- *Reproducibility* : The deterministic nature of computer-based simulation, the absolute control a researcher has over the simulated environment, and the availability of simulation software, results in experiments which are reproducible.

- *Flexibility* : As there is limited cost involved in setting up a wide range of experimental scenarios on virtually unlimited scales, a high level of flexibility

can be offered.

- *Efficiency* : Through computer-based simulation, time can be compressed as actions are imitated in silico. This allows for a significantly faster execution of an experiment compared to real-world experimentation. Advances in computer hardware and distributed computing further strengthen this efficiency advantage.
- *Comparability* : The use of simulation toolkits standardizes the experimental environment in which systems are evaluated and analyzed. This standardization contributes to the possibilities for direct comparative analysis of different systems based on standard metrics.

Even though there exists a number of software toolkits for the simulation of grids, they lack support for economic resource management systems (ERMS), or do not offer sufficient scalability and performance for studying such systems on a large scale. There is a need for such support however, as it allows easy *comparison* between different economic and non-economic approaches and enables researchers to *focus on the mechanism design and implementation*, while leveraging the strength of the existing general purpose framework in *setting up* the grid environment, *running* the simulation and *monitoring* the desired metrics.

This chapter will present the Grid Economics Simulator (GES) that we have developed in support of our research into economic forms of resource management. The key goal of the framework is to enable researchers to analyse and compare the properties and performance of a wide variety of economic and non-economic forms of resource management efficiently.

Because a grid is intrinsically a large scale system, a fundamental requirement for a grid simulator is scalability. An example of a large-scale production grid is the system built by the the European "Enabling Grids for E-sciencE" (EGEE) project. The EGEE infrastructure services over 10,000 users from 45 countries and offers a compute capacity of well over 40,000 CPUs, a figure which is expected to double over the course of the next year. In order to simulate such a vast infrastructure, a simulator has to be able to handle over 10,000 user entities and at least 100,000 computing nodes. In this chapter we will also evaluate the performance of SimGrid [114, 6], GridSim [115] and GES [116] in light of such large scale simulations.

4.2 Related Work

To provide some background on existing simulators and their capabilities, we review a selected number of them here. For a more elaborate overview one can consult [117].

4.2.1 Bricks

Bricks was designed as a performance evaluation system to analyse different scheduling approaches in high performance computing systems [118]. Two of the most

interesting features of Bricks are the use of a *scripting language* to describe the configuration and parameters of the simulation and its ability to *incorporate external components* such as the Network Weather Service [119] in simulations. Bricks has also been used to evaluate scheduling approaches [120] based on a fixed-cost charging model. The framework dictates a centralized approach for resource management however, limiting its general applicability. Development seems to have ceased and the framework is no longer available from the official project site.

4.2.2 MicroGrid

MicroGrid enables a researcher to create a *virtual Globus environment* of arbitrary composition [121]. It also allows for the execution of real applications within this environment. As such, it is actually an *emulator* rather than a simulator. This makes MicroGrid interesting for optimizing real grid applications with regards to the target configuration of the grid or conversely, allowing designers of grids to play with various parameters to optimize the grid architecture for a specific application.

Since MicroGrid is an emulator running real applications, it is very time consuming. Firstly, running the experiments is computationally intensive due to the system emulating the environment rather than simulating it. Secondly, real applications need to be developed. During this development, the researcher is confronted with all the technical difficulties of building grid applications, and cannot abstract away the details that are insignificant with respect to the goals of the empirical evaluation. It is also difficult to test a wide range of new resource management approaches as all of them have to be compatible with the Globus Toolkit, inhibiting experimentation in this area. Active development seems to have halted after 2004.

4.2.3 SimGrid

SimGrid [6] is an extensive toolkit that provides core functionalities for the simulation of distributed applications. The toolkit started out with a focus on centralized scheduling algorithms and was adapted later on to allow for decentralized scheduling [122]. The simulator takes into account the computational speed of nodes as well as latency and bandwidth of the network links connecting these nodes. SimGrid's analytical and flow-based network model allows for faster simulation times compared to approaches that use packet-level simulation of the network such as NS [123], SSFNet [124] and GTNetS [125].

The diagram in Figure 4.1 shows the different layers in the SimGrid software stack. At the top layer, four user interfaces modules are available. The GRAS module allows developers to implement distributed services and deploy them in a simulated setting or in a real world setting using a socket-based communication layer. This only involves linking the compiled user code against a different library. The SMPI module allows for the simulation of unmodified MPI code through the interception of MPI primitives. The SimDag module focuses on the simulation of scheduling heuristics for *task graphs*, which describe dependencies between the tasks of an application. Finally, the Meta-SimGrid (MSG) module allows a user to simulate distributed systems using the abstractions of *hosts*, *tasks*, *channels* and *processes*. A host represents a physical resource with computing capabilities that is

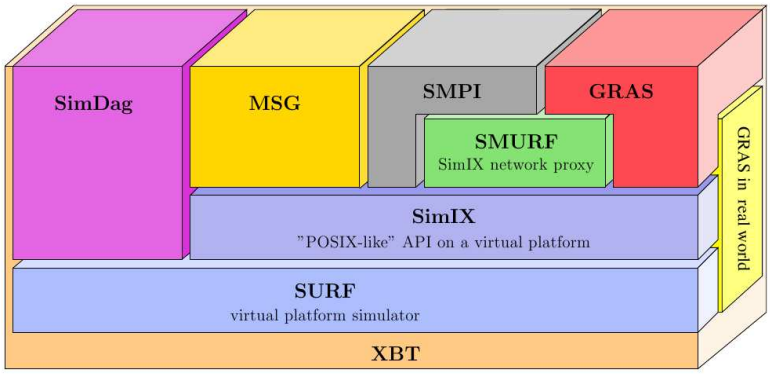

Figure 4.1: Overview of the SimGrid software stack [6]

able to execute tasks. Hosts can be linked to other hosts through channels. A host represents one entity in the simulator and corresponds to one thread. Currently, it is not possible to create multi-processor hosts. Therefore, each simulated CPU in a compute cluster has to be modeled as a host in order to accept tasks for execution[1]. A process is a piece of logic that runs on a specific host. Multiple processes can be deployed on the same host (e.g. a monitoring and a scheduling process).

Except for SimDag, all modules are built on top of SimIX which provides a POSIX-like API on top of the SURF simulation core. The core itself uses the XBT toolkit which provides data containers, exception handling and logging, and caters for portability. The SMURF module allows for the simulation to be distributed over multiple compute nodes. It is currently not fully implemented. The main goal of the distribution is to overcome scalability limitations of SimGrid that are mainly tied to memory requirements.

The SURF core models the simulated environment through a set of *actions* which require *capabilities* from *resources* in order for them to complete. The layers on top of the SURF core allow developers to express these actions and resources in a convenient manner, while also providing a programming model for concurrent processes that launch actions and wait for their completion. For each type of resource (e.g. network, CPU, disk), a *model* delivers information on the completion time of the next action on a resource. The SURF core iteratively advances time by querying the models for the first action to finish and then adjusting the internal clock accordingly, thereby updating the state of the actions and resources. In between iterations, user code is executed and can be notified on the set of actions that have finished.

Development of SimGrid is ongoing with more extensive support for MPI and modifications to the networking layer for improved scalability and accuracy.

SimGrid focuses heavily on task scheduling and network aspects of grids. To accommodate for economic resource management, substantial modifications would have to be made to make the simulated entities economic-aware and to model the

[1]This might be included as a future extension as communicated to us by the SimGrid developers

interaction patterns within a resource market.

SimGrid's codebase is entirely written in a procedural programming paradigm using the C language. Java bindings that call into the C core using JNI have recently been added for the MSG layer in version 3.3 of the framework. However, as we will see in section 4.7, the bindings do not offer sufficient scalability for simulating large-scale grids.

4.2.4 GridSim

The GridSim toolkit allows researchers to model heterogeneous resources, users, applications, resource brokers and schedulers [115]. GridSim uses a model wherein each individual user has its own application-level scheduler called a resource broker. Likewise, schedulers take on the task of managing the resources of a single provider. As such, GridSim focuses on decentralized application-level scheduling, with limited support for schedulers that perform global optimization.

GridSim is written in Java on top of the SimJava 2.0 basic discrete-event infrastructure. The simulator includes QoS-aware packet-level simulation of the network, supports advance reservations and also provides an output statistics framework. In addition, it supports both space and time-shared resource allocation policies. Contrary to SimGrid, it is possible to model clusters as a single entity. GridSim can also model a Grid's information system with *GridInformationService*s (GIS) which are responsible for the registration, indexing and discovery of resource providers. It is possible to create a single GIS entity in the simulation, but also to organize multiple GIS entities in a hierarchy comparable to a DNS tree. Additionally, GridSim includes components oriented towards data grids, the most important one being the *ReplicaCatalogue* (RC). Like the information services, replica catalog services can be organized hierarchically.

GridSim has been used to simulate the operation of an application-level scheduling algorithm that takes deadline and cost considerations into account in a market with fixed prices [95]. The toolkit has also been used in the context of evaluating the communication demands of different auction protocols [126]. However, the auction mechanisms were not fully integrated with the resource management and scheduling components of GridSim. This limits their use in evaluating the impact of the auction market approach on the actual outcome of the scheduling and resource management process.

Every simulated entity in GridSim extends the `GridSimCore` class which includes both an `Input` and `Output` object to send and receive events. All three of these classes extend the `Sim_entity` class of SimJava. Since every `Sim_entity` is actually a Java thread, this means that every user or resource provider entity requires at least three threads during simulation. As we will show in section 4.7 this also limits scalability. GridSim entities use the `sim_pause()`, `sim_get_next()` and `sim_schedule()` primitives of SimJava to pause, receive the next `Sim_event` object or schedule such an event.

4.2.5 jCase

jCase [93] is a tool for evaluating combinatorial auctions through simulation. The Multi-Attribute Combinatorial Exchange (MACE) bidding language supported by jCase has been applied in the context of grid resource management. The tool supports multiple algorithms for computing the transfer prices in a combinatorial auction. It also supports both the commercial CPLEX [127] solver, as well as the open source lp_solve [128] solver for determining the optimal set of winners in a combinatorial auction through a linear program. jCase however, is not a general purpose grid simulation framework and specifically targets combinatorial auctions. Currently it also lacks support for simulations of dynamic systems over time.

4.3 GES Overview

The Grid Economics Simulator (GES) has been developed for the evaluation of various economic approaches to resource management in their ability to efficiently allocate and schedule tasks in a grid. This section presents an overview of the simulator's architecture, operation and features. The GES codebase currently includes about 60,000 lines of Java code in approximately 600 classes.

4.3.1 Key Abstractions and Model entities

This section reviews the high-level abstractions and entities in our simulation model. Depending on the resource management system that is simulated, these basic entities are extended with additional model attributes that are specific to the RMS.

A *consumer* represents a grid user that wants to execute computational *jobs*. Every job has a normalized running time, i.e. the time it takes to finish the job on a reference CPU. Each consumer has a queue of jobs that need to be executed and for which resources must be acquired from *providers* through participation in the market. A consumer is provided with a budgetary endowment that may be replenished periodically throughout the simulation.

Jobs are allocated by the RMS to *grid resources* which are supplied to the grid through providers. Every provider hosts a number of CPU resources that are sold through the computational market. Providers interact with consumers through a market mechanism in order to agree upon a price for the execution of a job. When a trade agreement is reached, the provider will bill the consumer. The dispatch of a job to the CPU can be immediate or alternatively happens when the starting time of the job's reservation on a resource is reached. After the execution of a job has been initiated, a job remains allocated to its resource and cannot migrate. The actual runtime of a job depends on the processing power of the resource on which it is deployed, and the share of this power that is received by the job. We therefore focus on CPU-bound tasks and do not model the effects of disk or network I/O during the job's runtime. However, the transmission of input and output data before and after the execution of the job can be modelled.

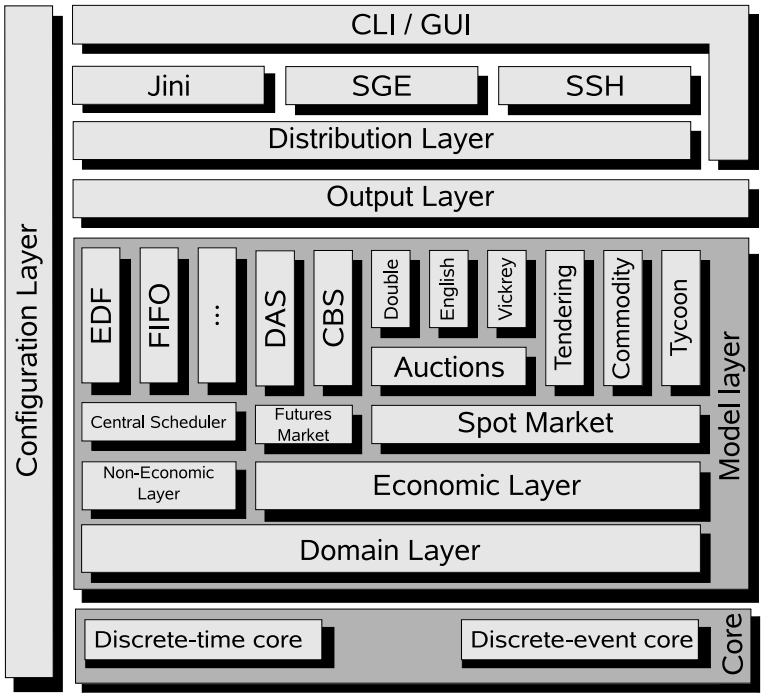

Figure 4.2: Overview of the architecture of GES

The *environment* brings together consumers and providers. It also dictates the interaction pattern used for obtaining resource allocations. An environment can be non-economic or economic, in which case it is called a *market*. A market has a *bank* facility that hosts accounts for each consumer and provider. The bank also handles all transactions necessary for paying the bills associated with resource usage. A market follows either a spot market or futures market allocation paradigm. The former is characterized by immediate dispatching of a job to a resource while the latter supports trading for resources in a future timeframe. An overview of the different market paradigms and scheduling approaches supported by the simulator is given in section 4.3.3.

Note that since the focus of GES is on economic grid resource management, we have described the key abstractions and entities from an economic point of view. In the context of the simulation of non-economic resource management systems, aspects related to billing and pricing are of no concern.

4.3.2 Architecture

An overview of the GES architecture is given in the layer diagram of Figure 4.2.

One of the key design goals of the architecture are extensibility and reusability. This "extend-and-refine" philosophy can be found throughout the whole `model` layer and its components. The `domain` layer contains base classes for all domain entities such as `Consumer`, `Provider`, `Job`, `GridResource` and `GridEnvironment`. The `Bank` entity is situated in the `economic` layer. Support for traditional forms of resource

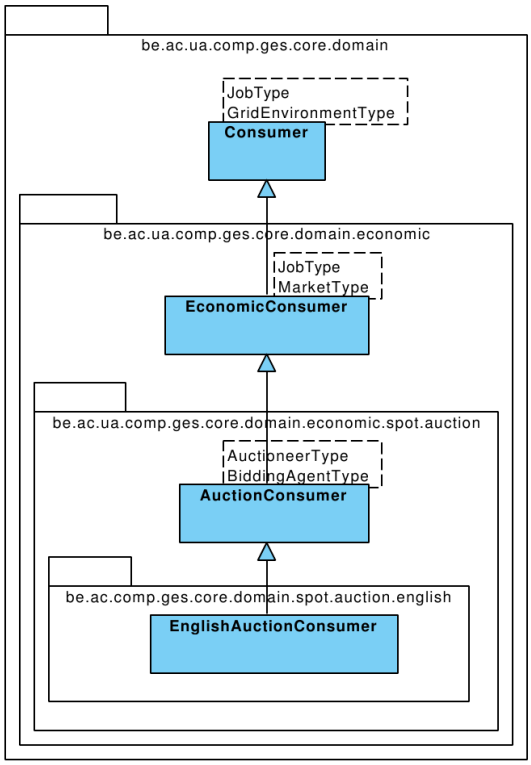

Figure 4.3: Domain model extension in GES

management is provided through the `non-economic` layer. Class extension is heavily used from the domain layer up to the specific RMS implementation as shown in Figure 4.3. The `Consumer` base class of the `Domain` layer for example, hosts all logic related to job creation, management and monitoring. The `EconomicConsumer` class in the `economic` layer extends this base class to add methods, data members, and output metrics related budgetary management. This class is then further extended by the `AuctionConsumer` class to add base logic related to participation in auction protocols. This base logic is extended and specialized further in the `EnglishAuctionConsumer` class to include bidding logic that is specifically tailored towards an English auction.

Existing components can be easily reused, extended and added to the framework when new resource management systems are explored. An overview of the different RMS systems that are currently supported by GES is given in section 4.3.3.

Further examples of reusability within GES can be found in the `economic` layer which provides components for accounting, billing and transactions, in the `futures market` layer which includes advance reservation mechanisms for preemptible and non-preemptible workloads, in the `auctions` layer that supports pluggable protocols for auctioning, and in the `tendering` layer where new negotiation strategies can be plugged in.

The model components rely on the `core` layer to simulate actions. GES provides

both a discrete-time core and a discrete-event core. In the discrete-time core, time advances in fixed steps and all actions within one step are considered to take place within the timeframe governed by that step. In the discrete-event core, actions lead to events which are associated with a specific firing time. The discrete-event core advances the simulation time by firing individual events one after the other. Both cores will be discussed further in sections 4.4 and 4.5 respectively.

Simulations can be distributed over multiple processing nodes through the `distribution` layer. This layer interfaces with compute resources that host a Jini-enabled compute service or clusters fronted by a Sun Grid Engine head node or a SSH setup. Currently, distribution is supported at the granularity of a simulated scenario. There are two key reasons why the distribution of simulations over multiple nodes proves to be useful. The first one is that in every simulation, we have to choose a seed for the random number generator that is used to initialize stochastic variables. To obtain statistically meaningful results, it is necessary to perform the same simulation a number of times with different seeds. The second reason is the fundamental requirement to assess RMS implementations in a wide variety of settings, leading to a large number of simulations that need to be performed. Distribution enables these to be executed concurrently. Possibilities for distributed execution of the individual entities in the simulation are planned for future releases. Such finer level of distribution is useful to remove scalability limitations that result from the limited amount of memory that is available on a single computer.

The `gui` layer allows the user to create and run scenarios, and to monitor them dynamically, i.e. while the simulation is in progress. A screenshot of the user interface is given in Figure 4.4. A persistency framework allows for storing both scenario configurations and configurations of the UI layout. In addition, a user can take a snapshot of the simulated environment and store this snapshot on disk. At a later point in time, the snapshot can be reloaded to memory in order to continue the simulation from a restore point. The snapshot mechanism serializes state to disk through the use of Java object serialization [129]. As such, snapshots are considered to be short-lived in the sense that they remain valid up to the point that a change in a class file of the domain model, breaks the serialization format.

Aggregated metrics over simulation runs and over a selection of simulated entities (e.g. a collection of consumers) are supported in the form of means, variances, standard deviations and box plots. After data collection and analysis, data can be directly exported from the simulator's UI to standard data formats such as `csv` or graphical formats such as `eps` and `png`.

4.3.3 RMS Frameworks

GES comes with built-in support for a number of resource management systems, both non-economic and economic. All non-economic RMS implementations use an offline central scheduler that can be configured to use different scheduling policies, some of which are very familiar in the scheduling literature [130]:

- An `Earliest Deadline First` (EDF) policy that schedules in the jobs of the consumer with the earliest deadline first.

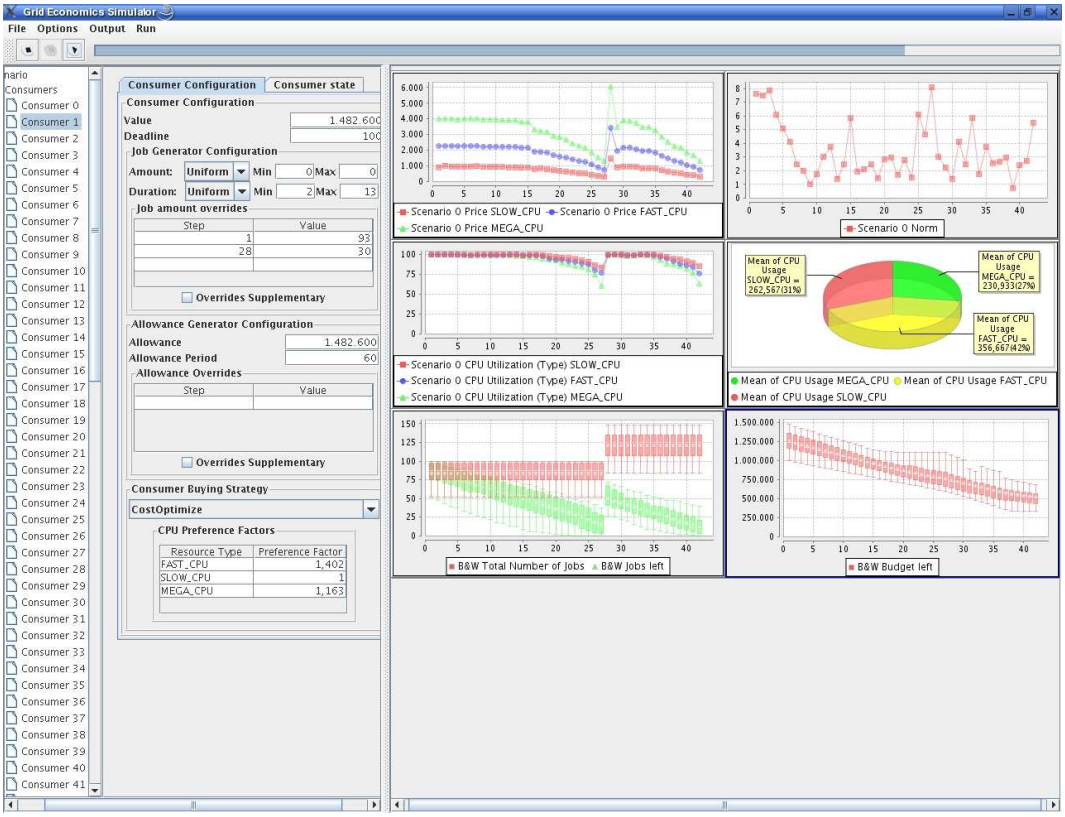

Figure 4.4: Screenshot of the GES graphical user interface

- A `Priority` policy where jobs are processed in order of the priority level that has been assigned to a consumer.
- A `Round Robin` policy that alternates between jobs from different consumers.
- A `FIFO` policy that schedules jobs in first-in-first-out manner as they arrive.
- A `DONE` (Deadline-Oriented Non-Economic) policy that aims to maximize the number of consumers that meet their deadline. It follows a greedy approach, scheduling in consumer requests in order of increasing workload.

The economic RMS's implemented in GES are divided into two separate branches. The first one encompasses the spot markets while the second one incorporates the futures markets. Spot markets are characterized by very dynamic price setting and quick reaction to changing market conditions but also suffer from the exposure problem [131]. Futures markets with support for advance reservation and co-allocation solve this problem at the expense of increased complexity and reaction time to changing market conditions.

The spot markets that are implemented in GES are the following:

- A `Selective Tendering` market with congestion control [132], where consumers request quotes from a group of selected providers. A consumer that is unable to obtain an allocation after requesting a certain number of quotes, backs off and tries again at a later point in time.
- An `Auction` market which supports double auctions as well as English, Dutch, First-Price Sealed-Bid and Vickrey auctions [133].
- A `Commodity` market that uses a Walrasian Auctioneer [134] for pricing. Multiple price adjustment schemes can be used ranging from a routine based on Smale's method [135] of global optimization to various optimization routines delivered by the Matlab Optimization Toolbox.
- An implementation of the market mechanisms used in `Tycoon` [136].

The futures markets supported by GES are:

- The `CBS` [137], a centralized brokering system where consumers have to direct their application processing requests to a central broker entity that will negotiate with the providers. The broker aims to maximize the total value generated by fulfilling the consumers' requests through coallocation of resources at multiple providers. Consumers are able to formulate a hard deadline requirement for the completion of their application's execution.
- The `DAS` market [137], a decentralized auctioning system where each provider holds auctions for selling its resources over a future time frame. The market allows consumers to place sealed bids for their jobs at the different auctions. Providers then determine the winners of the auctions using a greedy heuristic. After winner notification, a consumer has the chance to withdraw its bid, if it was unable to coallocate a sufficient amount of resources in order to run its application within the deadline requirements.

```
1  public interface DiscreteTimeControllable {
2    /**
3    * Hook method for notifying the entity that a new simulation step
4    * is about to be entered
5    */
6    public void startStep();
7
8    /**
9    * Hook method for simulation logic that should be invoked when the
10   * simulation advances one step
11   */
12   public void doStep();
13
14   /**
15   * Hook method for notifying an entity that a simulation step has
16   * ended
17   */
18   public void endStep();
19 }
```

Listing 4.1: DiscreteTimeControllable interface

4.4 Discrete-Time Core

The first core provided by GES adopts a discrete-time approach. In discrete-time simulation, time progresses in steps of equal length. A developer hooks into the core's control loop by implementing the `DiscreteTimeControllable` interface shown in Listing 4.1. Objects that implement this interface can be registered with the simulation core to be notified upon advancement of the simulation to a new step. Upon receiving such notification, they trigger the necessary actions in the environment.

In practice, a single object of the `GridEnvironment` base class is typically registered with the core. This object represents the simulated environment and takes care of notifying other simulated entities within the environment such as consumers and providers, when the simulation advances one step. The base classes of these different entity objects already provide logic that needs to be triggered upon each step. This includes updating the budget of consumers, collecting revenue from running jobs, registering output metrics for each simulation step and so forth. A developer extends the base classes with new methods tailored to the resource management system under study.

In addition, an implementation needs to be provided for the `doStep` method in the new environment class that specifies the sequence of high-level actions that need to be performed within the environment at each step. Listing 4.2 shows such an implementation for the `AuctionMarket` environment. The environment first builds a list of active auctions at line 6 and then instructs consumers to join a number of these auctions at line 15. After the joining phase, providers are instructed to run their auctions at line 20 which triggers the bidding process and unfolds the

```
1  public void doStep() {
2    //Build up a list of all providers hosting an auction for a free
3    //CPU resource
4    ArrayList<AuctionType> auctions = new ArrayList<AuctionType>();
5    for(AuctionProvider<AuctionType> provider : this.providers) {
6      auctions.addAll(provider.getAuctions());
7    }
8
9    //Shuffle the list of consumers to get a random join sequence
10   Collections.shuffle(this.consumers, this.getRandomNrGen());
11
12   //Instruct all consumers to join a set of the open auctions
13   for(AuctionConsumer<AuctionType,BiddingAgentType> consumer
14     : this.consumers) {
15     consumer.joinAuctions(auctions);
16   }
17
18   //Instruct all providers to run their auctions
19   for(AuctionProvider provider : this.providers) {
20     provider.runAuctions();
21   }
22
23   //Instruct all consumers to schedule their jobs
24   for(AuctionConsumer consumer : this.consumers) {
25     consumer.schedule();
26   }
27
28   this.trade();
29 }
```

Listing 4.2: `doStep` method of the AuctionMarket class

auction protocol. Thereafter, consumers are instructed to use the acquired resources for scheduling their jobs at line 25. Finally, a call to the `trade` method of the `EconomicGridEnvironment` base class on line 28 will trigger the payments from consumers to providers.

For the spot markets (see 4.3.3), the main phases in each simulation step are shown in Figure 4.5. First, the controller updates the joblist and budget of the consumers. Then, depending on the market mechanism used, the consumers, providers or both are instructed to start negotiations. In order to execute jobs, the consumer accepts a bill and sends it to the bank in phase 4. In phase 5, all monetary transactions take place. Finally, providers are instructed to start the execution of the relevant jobs and update the remaining execution time of running jobs. Consumers are notified of the jobs that have finished in phase 7. In practice, a subclass of the `GridEnvironment` class takes on the controller role.

The discrete-time approach offers an attractive programming model in that it allows a developer to centralize the simulation logic in the hook methods of the `DiscreteTimeControllable` interface. As such, one has a clear overview of the sequence of actions that need to be executed at each simulation step, and it becomes

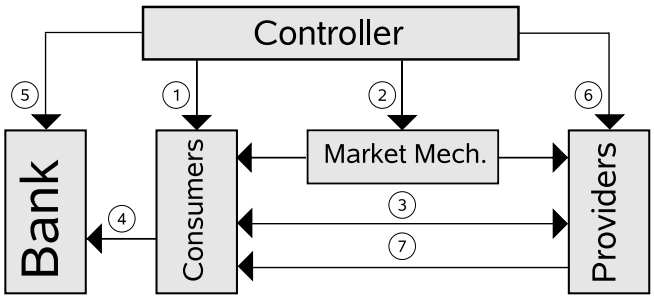

Figure 4.5: Overview of a simulation step in GES

straightforward to translate them into explicit code. In addition, this explicit encoding of the order in which actions are triggered within a single time step, relieves the developer from the complexity associated with modeling concurrent processes. This differs from event-based simulation as used in GridSim, SimGrid or the GES discrete-event core.

In event-based simulation, control logic is distributed over multiple entities. Because entities are modelled as concurrent processes, more effort is required to deal with different types of events arriving at an entity at different times. On the other hand, discrete-time simulation does not lend itself well to fine-grained simulation of real time, and cannot accurately model concurrent processes. Therefore, discrete-time simulation is well suited to simulations that focus on the high-level outcome of resource management approaches without considering the effects of the underlying interaction protocols in detail.

Consider for example the evaluation of an English auction protocol for trading resource access rights. We assume that a provider iteratively organizes auction rounds for selling resources that are currently available at its site. In each round, users submit their bids to the auction which include the price they are willing to pay for one time unit of resource usage, as well as the amount of resources they are willing to buy. The highest bidders get to use the provider's resources for the price they bid. For the evaluation of the protocol, we are interested in how the bidder's strategies influence the price dynamics in the market over time and what the distribution of revenues among the different providers is.

In a discrete-time simulation we can abstract away from the intricate timing issues related to the auction protocol. Examples of such issues are the time we consider the auction to remain open, or the time it takes for a bidder to send a bid to the auction. In the discrete-time model, we can assume that in each simulated time step auctions are held and that their execution unfolds within that timestep. This allows us to focus on the core logic of the auction mechanism and bidding logic without having to deal with intricate timing issues. As such, the resulting code is

much simpler in the discrete-time model versus the discrete-event model[2].

In a discrete-event simulation, time can be simulated in a more fine-grained manner. Therefore, network delays in the bidding process as well as timing and synchronization issues related to the auctioning protocol can be simulated and their effect on performance metrics such as system utilization can be evaluated. As such, a discrete-event model lends itself to a more realistic and detailed simulation of market mechanisms that can focus on aspects related to the actual *implementation* of a mechanism. This realism comes at the cost of higher implementation complexity as all the details of the protocol implementation need to be present in the simulation code.

4.5 Discrete-Event Core

This section describes the discrete-event core of GES, and highlights the software development techniques that were used in its development. More specifically, we show how a combination of *Aspect Oriented Programming*, *annotations* and the use of *continuations*, has contributed to the development of an object-oriented discrete-event simulation core that offers the necessary performance and scalability properties for simulating large-scale grids.

4.5.1 Basics

The Discrete-Event Engine (DEE) of the GES Discrete-Event System (DES) follows the process interaction approach [138]. As such, it supports the scheduling and execution of both *events* and *processes*. Events can be conditional or unconditional. Conditional events are fired when a specific condition is met and unconditional events when their fire time is reached. A process is equivalent to a thread; it is a logical stream of execution over a simulated timeframe. During the execution of events or processes, other events or processes can be created and scheduled.

The control loop of the DEE is shown in Figure 4.6. First, the next event (NE) is taken from the Future Event List (FEL). The NE is defined as the event with the earliest fire time in the FEL. If the fire time of the NE is the current time, it is executed immediately. If the fire time of the NE is higher than the current time, each conditional event (NCE) in the Conditional Event List (CEL) is checked to verify whether its firing condition is met. If it is, the NCE is executed. The scanning of the CEL repeats until no conditional event can fire anymore. After this, time is advanced by adjusting the event core's internal clock to the firing time of the NE, and the NE is executed.

On top of the DEE, a number of extensions have been developed. The *Entity extension* enables the creation of entities. Entities are elements of the system under study and are defined as having a (network) location. Entities can be composed of other entities which basically means that they co-exist at the same (network)

[2]The simulation results that will be presented in the following chapters have all been obtained with the GES discrete-time core.

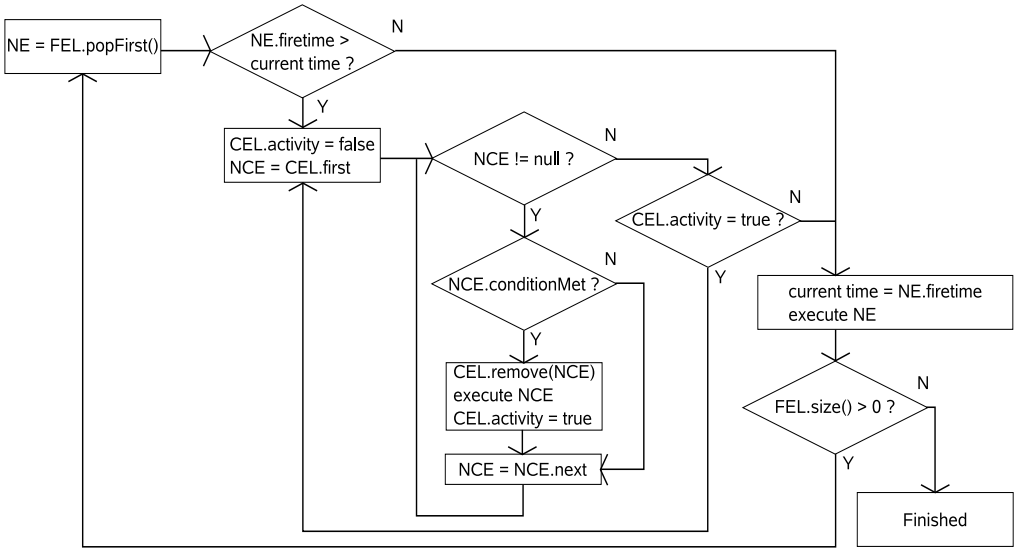

Figure 4.6: Discrete-Event Engine control loop

location. The composition happens automatically when an entity is created in the control flow of another entity. There are three ways to create entities; the first one is to annotate a class with the `@Entity` annotation, the second is to implement the `EntityInterface` and the third one is to create a `Process`. The first two methods create passive entities which do not undertake any action on their own. The last one creates an active entity which can interact with other entities on its own accord.

The *Process Method extension* allows for the transformation of methods annotated with `@ProcessMethod` into processes. This means that when such a method is called, a process is created which executes the method body. The *Asynchronous/Synchronous Method extension* ensures the conversion of calls to methods tagged with `@AsynchronousMethod` or `@SynchronousMethod` to (a)synchronous calls with certain delays. An asynchronous method call will not block the caller but the execution of the call may be performed with a predetermined delay. Such calls can only be made on `void` methods. A synchronous method call will block the caller while both the execution and the return of the call may be subject to a predetermined delay. The *Asynchronous/Synchronous Network Method extension* indicates that calls to methods tagged with `@AsynchronousNetworkMethod` or `@SynchronousNetworkMethod` will be alterred into (a)synchronous network calls with delays depending on the location of the entities in the network and network conditions.

The different states of `Event` and `Process` objects are listed in Figure 4.7. When `Events` are created their state is `instantiated`. After scheduling, it changes to `scheduled`. As long as events are not executed they can be unscheduled, becoming `instantiated` once more. During the execution the event's state is set to `running`, followed by `finished` afterwards. A `Process` is actually an `Event` of which the execution can suspend. Therefore it requires additional state information when it

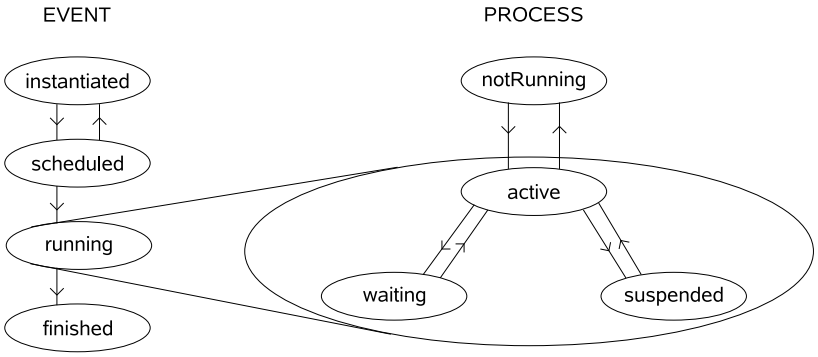

Figure 4.7: Event and Process states

is `running`. A process that is actively executing instructions is `active`. Otherwise it can either be `waiting` or `suspended`. A `waiting` process can be resumed by any `ResumeEvent` while a `suspended` one can only be resumed by a `ResumeEvent` which has been registered with the process beforehand. Other events that arrive to resume a `suspended` process will throw an exception.

4.5.2 Aspect Oriented Programming and Annotations

Aspect Oriented Programming (AOP) [139, 140] allows a software developer to provide code for cross-cutting concerns through the definition of so-called *aspects*. Aspects extend a language with three new features. A *pointcut* describes join points, which are well defined points in the code or flow of execution where *advice* can be injected or *weaved* in. The *advice* may be injected before, after or around a specific join point. By means of *inter-type declarations* it is possible to augment classes with new methods, data members or interfaces. The flexibility offered by tools such as AspectJ in the definition of such pointcuts, allows for easy integration of a cross-cutting concern in an existing codebase.

In this section, we will show how the combination of AOP and Java annotations [141] has contributed to the clean programming model offered by GES. We will focus on one of the cross-cutting aspects in our simulator, namely the introduction of a network delay that needs to be introduced each time an entity invokes a Remote Procedure Call (RPC) on another entity. We will use the capabilities offered by the AspectJ compiler for the application of the AOP paradigm. Annotations have been added as a standard feature to the Java platform since version 1.5.0.

For a synchronous RPC in which the calling entity's thread of control blocks until the result is returned, we use the aspect given in Listing 4.3. This aspect rewrites all calls to non-`void` methods that are tagged with the `@SynchronousMethod` annotation (line 3) to simulate both remote call and return delay. The call delay is simulated by the `callEvent` created on line 8 and sent to the callee on line 23. The `send` call implementation which is delivered by the `EntityAspect` base class will appeal to a singleton representing the network in order to compute the delay for the

```
privileged aspect SynchronousNetworkMethodAspect extends EntityAspect {
  Object around(final EntityInterface src, final EntityInterface dst) :
    call(@SynchronousNetworkMethod !void *.*(..)) && this(src)
      && target(dst) {

    IProcess proc = src.getCurrentProcess();

    PayloadEvent<PayloadEvent<Object>> callEvent =
          new PayloadEvent<PayloadEvent<Object>>() {
      public void run() {
        ResumeEvent<IProcess, Object> returnEvent =
            new ResumeEvent<IProcess, Object>(proc,ResumeEvent.DELAY) {
          public void run() {
            this.getProcessToResume().resumeNow(this);
          }
        };

        returnEvent.setPayload(proceed(src,dst));
        this.setPayload(returnEvent);
        send(returnEvent,dst,src);
      }
    };
    send(callEvent, src, dst);
    proc.suspendNow(ProcessState.suspended);

    return callEvent.getPayload().getPayload();
  }
```

Listing 4.3: Synchronous Method Aspect

transmission and schedule the delay in the core's event queue. After the `callEvent` is scheduled, the original `Process` is suspended on line 24. When the execution of the `callEvent` is triggered by the event core, it invokes the target method by calling AspectJ's `proceed()` method on line 18, and the return value is set as payload of the newly created `returnEvent`. The `returnEvent` is then sent back to the caller with the appropriate delay. This `returnEvent` is responsible for resuming the original `Process`, which will continue on line 25 returning the value of the called method on line 26. AspectJ thus allows us to substantially alter method calls to the point of encapsulating the call itself in a method of an anonymous inner class. This feature is used in order to *capture* the call on a network entity and transfer its point of execution to the run method of a GES event.

```
1 @Entity
2 public class Provider {
3   @SynchronousNetworkMethod
4   public boolean scheduleJob(Job j) {
5     return this.queueJob(j);
6   }
7 }
```

Listing 4.4: Provider class exposing a synchronous network method

```
1 @Entity
2 public class User {
3   private boolean scheduleJob(Job j) {
4     Provider provRef = this.getProviderForJob(j);
5     boolean success = provRef.scheduleJob(j);
6     return success;
7   }
8 }
```

Listing 4.5: User class invoking the synchronous network method

The combination of annotations and AOP offers a clean solution to the integration of the delays that result from a RPC. As Listing 4.4 shows, the developer simply annotates the provider's method for accepting jobs in order to turn it into a network method. Listing 4.5 shows the caller which invokes the `scheduleJob` method on a `Provider` reference. A key advantage of our approach is that a developer can hold on to an object-oriented paradigm with strong type checking. Without the use of AOP and annotations this is impossible to achieve, as the execution of the target method needs be extended to incorporate the necessary logic for e.g. blocking the calling thread, computing the message delay and inserting the necessary events that are tied to the message passing process in the event queue. Without the techniques presented in this section, such logic can only be appealed to through a standardized interface which is, by design, independent from the actual remote method being called.

This forces the user to fall back on a procedural style of coding. In GridSim

for example, which is mainly designed according to the principles of object-oriented programming, all interactions between distributed entities need to pass through a method with the following signature:
`void send(Sim_port destPort,double delay,int gridSimTag,Object data)`
where `destPort` determines the entity the message is sent to, `delay` the required message delay, `gridSimTag` a unique ID that informs the callee on the type of message sent, and `data` the payload of the message. The `send` call will block the current active thread until the return value is received by the caller. The adoption of this message passing model now forces the developer into a lower-level procedural style of coding with no compile-time type checking. More specifically, a developer needs to associate a unique integer value to every remotely calleable method. The callee then needs a switch structure which dispatches the processing of a received event to a local method. In addition, all payloads to the RPC call need to be packaged in a typeless data structure. The receiving end is required to perform the necessary casts and type checks after receiving the payload. In effect, the developer needs to tackle many of the issues that come up when implementing RPC-like communication protocols. As a result, more development effort is required, code maintainability is reduced and the chances for coding errors are increased.

As can be observed from Listings 4.4 and 4.5, the use of annotations and AOP shields the developer from this complexity and the resulting disadvantages. A key advantage of annotations is that they allow the client developer to explicitely state where an aspect from our library should be applied without relying on naming conventions for pointcuts and without the user having to edit the aspect itself to change the advice.

4.5.3 Continuations

The simulation of grids in general, and market mechanisms for grids in particular, involves a large number of individual entities that are concurrently performing actions that change the state of the environment. Examples of such actions are a user transmitting a job to a provider for execution, a resource broker querying different resource providers on the possibility of allocating resources in a specific timeframe, or a provider placing an offer in an auction for putting up its resources for sale. This pervasive concurrency naturally leads to a design based on threads. Indeed, many grid simulation tools such as SimGrid and GridSim use a massive multithreading[3] approach. However, we argue that although threads are a good way to develop systems with *actual* concurrent behavior, it is not necessary to use real threads to *simulate* this concurrency. Indeed, the use of a massively multithreaded model in many grid simulators is not motivated by the need for concurrent execution of simulator code, but by the need for a programming concept that allows one to *model* concurrency. More specifically, the developer needs a mechanism to suspend the execution of a method when it needs to wait for an external event. Such an external event originates from a concurrent flow of control in the simulation.

[3]In a massive multithreaded model, each simulated entity is tied to one (or multiple) threads of control

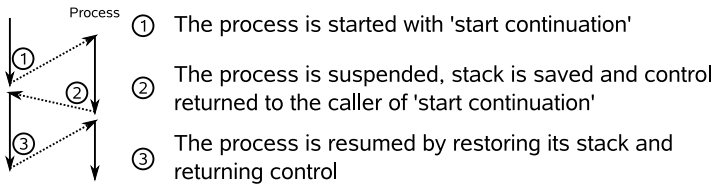

Figure 4.8: Use of continuations

The adoption of massive multithreading to achieve this goal has severe consequences. Firstly, if native threads are used, it results in limited scalability with respect to the number of entities that can be modelled. On a machine with a standard Linux kernel, the number of native threads is limited to just over 32,000. Considering for example that each entity in GridSim [115], requires one thread for its *body* and two threads for managing the incoming and outgoing communication flows, less than 10,700 entities can be modelled in GridSim. On 32-bit Linux machines, the existence of a 3GB limit to the amount of virtual memory that can be allocated to a user process also limits scalability as each thread requires at least 64kB of virtual memory with the default being 320kB on most systems.

Secondly, as we have discussed previously, (unconditional) events in the event queue of a discrete-event simulator are strictly ordered on their firing time. As such, a discrete-event core has enough information to determine the next piece of logic that must be executed, with zero overhead. In a threaded model, the transfer of control to a specific thread is managed by a thread scheduling component in the kernel that cannot use this information. Therefore, the scheduler has a chance of activating the *wrong* thread resulting in additional overhead and loss of performance.

Thirdly, the use of a threaded model in a simulation core complicates the programming model as it involves intricate logic to switch control between processes through the use of e.g. mutexes and semaphores. This is a consequence of the fact that control is not automatically returned to the discrete-event core when an entity's thread of control is blocked. In addition, the complexity inherent to developing parallel programs in terms of process synchronization and race conditions is introduced in the code. This complexity is caused purely by the use of a threaded model, and is therefore not inherent in the problem of simulating concurrent processes.

The last two points result from threads being a general tool for parallel programming. In particular, a threaded model fits well to the problem of programming systems in which actions occur at undetermined times and concurrently running processes need to be synchronized, or excluded from concurrent access to memory. For a discrete-event simulation core to function properly however, code never needs to be executed in parallel. The only requirement is the possibility for an executing method body to be preempted and resumed later on.

Continuations provide exactly this capability, without inducing the overhead and complexity that comes with a more general purpose threading model. A continuation represents the rest of a computation at a certain point in its execution.

Basically it is a control structure containing the stack and a program counter. All continuation libraries work in a similar way and support three basic instructions as shown in Figure 4.8. They allow a computation to be started with continuation support (point 1). This computation can then suspend (point 2), after which execution resumes right after point 1. At a later time, the computation may be resumed (point 3). Projects that offer continuation libraries for Java are Javaflow [142], Rife [143], and Kilim [144]. All three libraries have to perform bytecode transformations in order to store and restore the stack's contents and the program counter. Although our present implementation of the event core uses Javaflow, our design allows other continuation libraries to be plugged in easily. The integration of Kilim seems especially promising as it has reported superior performance compared to other libraries [144]. Unfortunately, the combination of Java generics with AOP used in our codebase, is presently not compatible with Kilim's bytecode rewriting phase. We are working together with Kilim's author to resolve this issue. In this respect, we would also like to mention that AspectJ's support for Java generics was not fully stable at the time of our developments. We submitted four bug reports, two of which have been resolved in the latest 1.6.2 release of AspectJ.

4.5.4 API

The most important methods and annotations of the GES discrete-event core API that are oriented towards users, are listed in Table 4.1.

Name	**Description**
createEventSystem(..)	Creates the event system with a specific bootstrap `Event`.
schedule(..)	Schedules an `Event` or `Process`. `Events`/`Process`es can be scheduled without delay, with a specific delay, with a specific execution time or with a specific fire condition.
unSchedule()	Removes the `Event` or `Process` from the event engine's FEL and CEL.
suspend(..)	Can be called from within the control flow of a `Process`'s run method. It suspends the execution of a process, puts it in the `waiting` state, and returns control to the event engine. Processes can be suspended with or without a timeout. The suspend method also returns the event responsible for resuming the process.

Name	Description
resume(..)	Can be called by other `Process`es or `Event`s to resume a process (until it is finished or suspends again). It can be called with or without a reason (encoded as an integer) and payload. A `ResumeEvent` will be created encapsulating the reason and payload.
delay(..)	Can be called from within the control flow of a `Process`'s run method. It delays the execution of a process for or until a specific time and puts it in the `suspended` state.
startJoining()	Starts monitoring for processes that are being scheduled and instructs them to alert a join manager when they are finished.
doJoining()	Stops the monitoring and puts the process in the `suspended` state until all processes scheduled between the calls to `startJoining` and `doJoining` have finished executing.
@Entity	Objects of this class are to be regarded as entities.
@ProcessMethod	This method should be transformed to a `Process`.
@AsynchronousMethodAspect	Calls to this method should be asynchronous with a specific call delay.
@SynchronousMethodAspect	Calls to this method should be synchronous with a specific call and return delay.
@AsynchronousNetworkMethodAspect	Calls to this method should be asynchronous with a call delay depending on the network location of caller and callee as well as network conditions.
@SynchronousNetworkMethodAspect	Calls to this method should be synchronous with a call and return delay depending on the network location of caller and callee as well as network conditions.

Table 4.1: API features

```
1  public class AuctionConsumer extends EconomicConsumer {
2
3    ...
4
5    @ProcessMethod
6    public void startNegotiations(ArrayList<AuctionType> auctions) {
7      this.discoverAuctions(auctions);
8      this.startBidding();
9    }
10
11   private void startBidding() {
12     Process.startJoiningCurrent();
13
14     for(BiddingAgentType agent : this.biddingAgents) {
15       //Activate the bidding agent...
16       agent.schedule();
17     }
18     // Wait until all our agents have finished bidding...
19     Process.doJoiningCurrent();
20   }
21 }
```

Listing 4.6: Auction Consumer

Example

To illustrate key elements and usage of the DES core API, we will outline an example that models the bidding process in an auction market. The bidding process is as follows:

1. At a certain time, a number of auctions are opened.
2. Consumers that are looking for resources are instructed to start negotiating by the controller which calls the `startNegotiations()` method in listing 4.6. This call is transformed into a Process by means of the `@ProcessMethod` annotation.
3. Consumers then discover the open auctions and start bidding with potentially multiple bidding agents. The formulation of their bid depends on the auction paradigm (and bidding agent) used. The consumer waits until all of its agents have stopped negotiating by means of the *joining* methods used in `startBidding()`.
4. After an auction has closed, the winner and losers are notified.

The actual bidding is done through bidding agents (listing 4.7). These bidding agents will first join the auctions and then start bidding. After a bid has been placed, the bidding agent process suspends on line 12, awaiting an external event such as an update of the asking price, or the closing of the auction. As can be seen in the listing on line 23, the method to update the asking price is annotated, converting a call to this method into an asynchrounous RPC. Upon the arrival of a price update,

```
1  public class BiddingAgent extends Process {
2
3    ...
4
5    public abstract Bid<BiddingAgentType> getBid(double minimumBidLevel);
6
7    public void run() {
8      this.joinAuction();
9
10     while(!this.isFinishedBidding()) {
11       this.placeBid();
12       this.suspend();
13     }
14   }
15
16   private void placeBid() {
17     Bid<BiddingAgentType> bid = getBid(this.currentAskPrice);
18     if (bid != null) {
19       this.auction.placeBid(bid);
20     }
21   }
22
23   @AsynchronousNetworkMethod
24   public void askingPriceUpdate(double askingPrice) {
25     this.currentAskPrice = askingPrice;
26
27     if(auction.getHighestBidder() != this &&
28        (auction.getHighestBid()) > budget) {
29       this.withdraw();
30     }
31     this.resume();
32   }
33 }
```

Listing 4.7: Auction Bidding Agent

the bidding agent checks the new state of the auction to determine whether it wants to withdraw. The suspended bidding process is then resumed on line 31.

Resources are sold through auctions created by the resource providers. The auctioning process shown in Listing 4.8 follows an *event-based* paradigm. At its core, the `processEvent` method processes events based on their type. This paradigm is supported by returning the `ResumeEvent` from the suspend method, which allows the `processEvent` method to identify the reason why the auctioning process has been resumed.

While the auction is running, a number of rounds are held. During a round, every bidding agent has the chance to bring out its bid using the `placeBid` method. The annotation on this method indicates that calls to it need to be converted into synchronous network calls. Upon receiving a bid, the auction process is resumed and the auction's `bidPlaced` method is called which registers the bid. In addition, the method will determine whether the auction will move to a new round, or whether

```
1  public class Auction extends Process {
2
3    ...
4
5    /**
6     * The time to wait for bids to arrive in a bidding round.
7     */
8    private int biddingRoundTimeout = 500;
9
10   public void run() {
11     // Wait for the desired amount of bidder competition...
12     this.waitForCompetition(nrAgents, joiningDelay);
13
14     // Run the auction
15     while(this.isAuctionRunning() && this.getBidders().size() > 0) {
16       IResumeEvent event = suspend(biddingRoundTimeout, ROUND_TIMEOUT);
17       this.processEvent(event);
18     }
19   }
20
21   private void processEvent(IResumeEvent event) {
22     if (!this.isAuctionIdle()) {
23       switch (event.getResumeReason()) {
24       case BIDDER_JOINED :
25         this.joined((BiddingAgentType)event.getPayload());
26         break;
27
28         ...
29
30       case BID_PLACED :
31         this.bidPlaced((Bid)event.getPayload());
32         break;
33       case ROUND_TIMEOUT :
34         if (this.isAuctionRunning()) {
35           this.endOfBiddingRound();
36         }
37         break;
38
39         ...
40       }
41     }
42   }
43
44   @SynchronousNetworkMethod
45   public void placeBid(Bid<BiddingAgentType> bid) {
46     this.resume(BID_PLACED, bid);
47   }
48 }
```

Listing 4.8: Auctioning process

the auction is to be closed. A round can also be ended after a period without any (bidding) activity. To this end, the auction process supplies a delay value to the suspend method on line 16. If the process, is not resumed by any event within the timeframe determined by the delay, the core will notify the process by sending it an event with the given `ROUND_TIMEOUT` reason.

4.6 Integration of third-party software

This section comments on how we have integrated a number of existing software components in GES. These third-party components relate mainly to the simulation of networks, synthetic generation of clusters and networks, and optimization routines.

4.6.1 Network simulation

In order to support market mechanisms that take network-related aspects into account, we have integrated existing components of the SimGrid project in GES. Many dedicated network simulators have been developed in the network community. The most well-known network simulator is NS-2 [123], which supports a wide range of network protocols and queueing models. It is currently followed up by NS-3 [145], an open source project to develop an eventual NS-2 replacement. Other examples are SSFNet [124], OPNET Modeler [146], GTNetS [125] and OMNeT++ [147]. All of these tools are able to perform accurate packet-level simulation of the network. This is because their target user base is focused on the evaluation of new algorithms that operate on a level of abstraction which requires them to consider individual network packets and network protocol stacks. Some grid simulators perform packet-level simulation but do so with limited accuracy. GridSim for example considers a fragmentation of network payloads into packets but does not accurately model the TCP flow protocol in the transmission of these packets over the network.

In a grid context however, network simulation can involve hundreds of data transfers of multiple gigabytes of data. In such settings, it is difficult for packet-level simulation to offer the performance and scalability to efficiently simulate grid resource management systems. This is why more coarse-grained approaches have been developed in the grid community that employ analytical models which only consider TCP *flows* instead of individual packets. Both OptorSim and SimGrid take this approach. The models used in OptorSim however, have been found to be highly inaccurate in the presence of heterogeneous link characteristics [6].

The models developed in SimGrid have been validated with respect to the GTNeTS packet-level simulator. The results of that study show that SimGrid is able to model end-to-end transmission times for network data flows with high accuracy in cases of moderate congestion on the network [148]. In settings with high network contention, or when flow transmission times are bounded by the latency of a link, accuracy is significantly lower. This is partly because SimGrid models do not take the effects of TCP slow start into account. When high accuracy is required in such settings, SimGrid also supports the use of GTNeTS as a backend for simulating the

network. As reported by Fujiwara et al. [148], the SimGrid network models can deliver significant gains in runtime performance compared to GTNeTS. Depending on the amount of flows and data sizes, these can amount to many orders of magnitude.

```
1  public interface NetworkModel {
2  /**
3  * Sends the <code>event</code> from the <code>src</code> to
4  * the <code>dst</code> Process.
5  * @param event The event to send.
6  * @param src The source of the event.
7  * @param dst The destination of the event.
8  * @return whether the event was sent successfully
9  */
10 boolean send(Event event, EntityInterface src, EntityInterface dst);
11 }
```

Listing 4.9: NetworkModel interface

The main challenges in integrating SimGrid's networking layer into GES are the following:

- SimGrid's Java bindings to the MSG layer only support native threading which limits scalability as explained in section 4.5.3.
- An integration will result in the state of the network being managed by SimGrid components, while the state of the other resources is managed within GES. As such, both the GES and SimGrid simulation cores will need to coexist.
- SimGrid's analytical approach to simulation differs from GES in the sense that there is no event queue. The core only holds on to a list of *actions* which use *resources*. The simulation core iteratively queries the resource models for determining the next point in time at which an action will finish.

We have tackled the first challenge by integrating the networking code found in the lower-level SURF layer instead of the MSG layer (cfr. Figure 4.1 for an overview of the SimGrid architectural layers). As the SURF layer is not tied in with any concurrency or threading model, we are not hindered by the limitations of the massive multithreading approach used in SimGrid. The second and third challenges have been addressed by integrating both simulation cores as depicted in the sequence diagram of Figure 4.9.

Upon the call to `send` (1), the time in the SURF core is first synchronized with the current time in the GES core (3), before the actual communication action is initiated in the SURF layer (7). The Java GES code calls into the C SURF code through the use of the Java Native Interface (JNI) specification. The two cores are linked through a mapping in the `SurfNetworkModel` class which associates a unique GES event ID with each event that passes through the networking layer (5). This ID is passed on to the SURF core by means of metadata that is tied to the communication

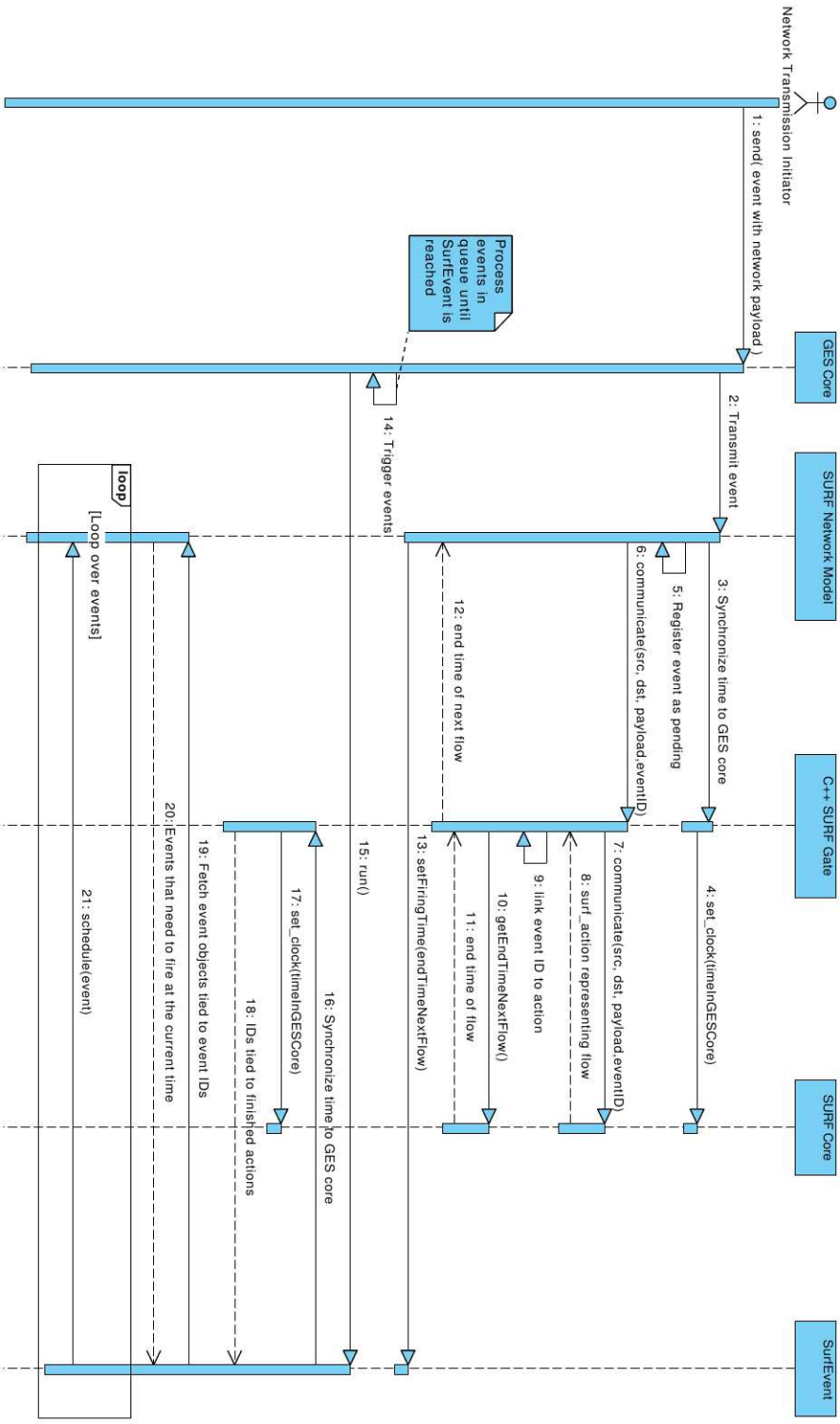

Figure 4.9: Integration of SimGrid's SURF layer within GES

action which models the data transmission (6). The `SurfNetworkModel` provides an implementation of the `NetworkModel` interface shown in Listing 4.9.

In the event queue of the GES core, a `SURFEvent` is installed which is configured with a firing time F_{SURF} that corresponds to the next action finish time in the SURF core (13). The `run` method of this event will call into the SURF core using a JNI interface and force the SURF core to update its time to F_{SURF} (16). Such an update returns the GES event IDs associated to all the communication flows that have finished at that time. The `SurfEvent` then queries the `SurfNetworkModel` for all the associated GES events tied to these flows (19), and schedules these events for immediate execution (21).

The firing time of the `SURFEvent` object in the GES core is updated to the new firing time :

- After the `SURFEvent` has been triggered by the GES core.
- Before firing any event in the GES event queue, if the firing time of the `SURFEvent` is marked as dirty. We mark the time as being dirty after the initiation of a communication flow in SURF. A `SURFEventCoreListener` is installed in the GES core to plug in this logic in the event core. Note that we do not recompute the firing time after the initiation of each new communication flow as this is a relatively costly operation. In the event that multiple flows are initiated without being interleaved by events from the GES core, only one recomputation of the firing time is done.

We note that the sequence diagram in Figure 4.9 shows a simplified view on the integration. For example, in the event that two events are transmitted from a source to the same destination, we ensure that events do not arrive out of order. This can occur if the transmission of the first event incurs a longer transmission delay compared to the second event. SimGrid does not impose such orderings on the finishing time of flows between two entities.

Our integration only required one change to the SimGrid codebase. Because syncing the two simulation models requires the GES core to set the time in the SURF core, a `set_clock` method was added to `surf.c`.

4.6.2 Infrastructure generation

As mentioned in the introduction of this chapter, experimentation using simulation regularly involves a study of a system under varying infrastructural arrangements. In order to support the automatic generation of infrastructure configurations, we have integrated the GridG [149] and GiST [150] tools within GES.

The GridG tool is designed to support the synthetic generation of network topologies, as well as the annotation of nodes and links within a topology with information relevant in the context of computational grids. Network topology generation is supported by a wide-range of tools such as Tiers [151], Waxman [152] and Inet [153]. A key feature of GridG is that it generates topologies that follow *both* the structural hierarchical laws for internet topologies as well as the so-called power-laws [154]. GridG uses the Tiers generator to obtain a topology that has a three-level

hierarchical structure (LAN, MAN and WAN). The Tiers generator does not add any redundant links to the topology. GridG then adds such redundant links and transforms the topology so that it follows the *outdegree power law*. This law is one of the following three power laws identified by Faloutsos et al. [154] for the topology of the internet:

- The *rank exponent law* which says that the outdegree d_v of a node v is proportional to the rank of the node raised to a constant.

$$d_v = \beta r_v^R \tag{4.1}$$

- The *outdegree exponent law* says that the frequency f_d of an outdegree d is proportional to the outdegree raised to the power of a constant O.

$$f_d = \alpha d^O \tag{4.2}$$

- The *Eigen exponent power law* says that the eigenvalues λ_i of the topology graph's adjacency matrix, are proportional to the order, i, raised to the power of a constant ϵ.

A topology generator that follows these power laws is also called a *degree-based* generator.

Like many other degree-based generators, GridG only enforces the outdegree exponent law, which often results in the generated topology also following the other laws. Lu et al. explain this observation by showing the interrelations between the different laws [149]. GridG relies on the constants defined in [154] to apply the outdegree law with $\beta = exp(4.395)$ and $R = -0.49$.

After topology generation, GridG annotates network links with bandwidth and latency attributes. GridG reuses the values generated by the Tiers tool for this. As an alternative, a user can supply latency and bandwidth distributions for the WAN, MAN and LAN levels. The distributions for the switching bandwidth also need to be supplied by the user. In addition, GridG annotates the hosts in the topology with information on their OS, CPU speed, the amount of memory and CPU cores, and the size of disks. The correlations between the different attributes (e.g. a machine with a fast CPU often has more memory than a slow CPU), are encoded using a rule-based framework. This framework organizes the rules in a decision tree, with the architecture of the node forming the root. A disadvantage of the GridG approach is that it mainly focuses on individual machines without explicit support for clusters. Additionally, the results of the rule-based approach have not been validated by comparing them to real systems.

Therefore, we have decided to use the GiST [150] tool to generate the clusters on the network. We thereby associate one cluster with each host endpoint in the network topology generated by GridG. GiST generates cluster configurations based on a statistical analysis of over 10,000 CPUs that are hosted by 112 production clusters. An additional advantage over GridG annotation is that it also supports extrapolation of resource attribute values in order to generate future resource configurations based on observed trends. GiST annotates a cluster with the following

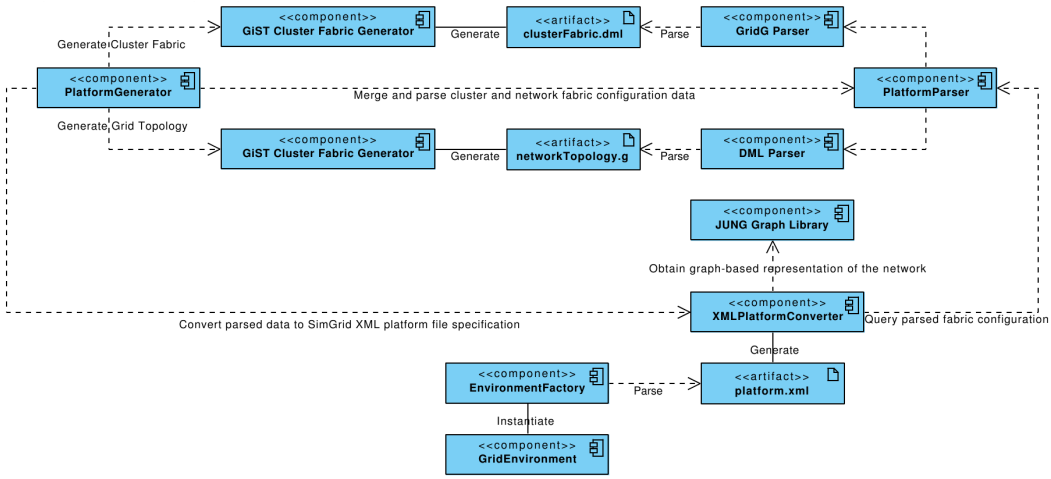

Figure 4.10: Integration of GridG and GiST tools

attributes; processor architecture, clock speed, level 2 cache size, memory and disk size per node, number of processors per node, number of nodes in a cluster, and network technology (e.g. 10-Gbit Ethernet, or Myrinet). The distributions that the tool uses for the attribute values are derived from the 112-cluster dataset. The statistical models were validated with a dataset that included 191 Rocks [155] clusters totaling 10,073 CPUs [150]. However, no recent validation was done on the models that perform extrapolation of the attribute values over time. Although GiST allows for new training data to be fed to the modeler in order to improve the distributions over time, we have not found any updated training data that is readily available[4].

A schematic overview of the integration is given in Figure 4.10. First, the `PlatformGenerator` uses the command-line GridG and GiST tools to generate a GridG file describing the network topology, and a Domain Modeling Language (DML) file describing the set of clusters and their attributes. A parser is then invoked which uses a `DML Parser` and a `GridG Parser` component to obtain an in-memory representation of the configuration data and to associate the set of clusters with nodes in the network. This in-memory representation is then used by a converter which generates an XML file according to the XML schema developed by the SimGrid project for describing platform configurations. Since SimGrid uses a static routing approach, all the routes between endpoints in the network need to be specified in the platform file. To generate these routes, we use the Java Universal Network/Graph Framework (JUNG) [156] library to obtain an in-memory, graph-based representation of the network. We then use Dijkstra's shortest path routing algorithm [157] on this graph to obtain the routes between all endpoints in the network and insert these routes in the XML file.

The resulting platform file can then be fed as a configuration file to the `EnvironmentFactory` which builds up the simulated environment through instantiation of the network model and all domain entity objects. For network models that rely

[4]The training data that comes with the GiST distribution dates from 2004

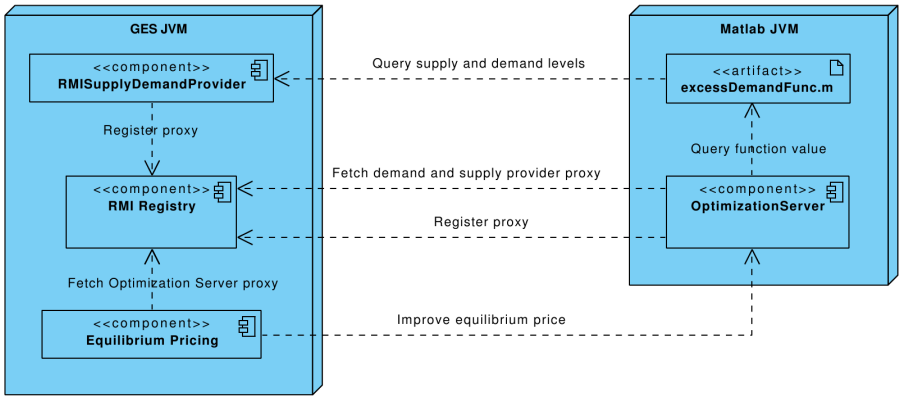

Figure 4.11: Integration of Matlab optimization routines

on the components delivered by SimGrid's SURF layer, the usage of a SimGrid platform file is mandatory. For other models, the `EnvironmentFactory` also accepts the in-memory representation of the network delivered by the `PlatformParser`.

4.6.3 Integration of Matlab Optimization Toolbox

As mentioned in section 4.3.3, GES supports a spot market in which a central process, the Walrasian auctioneer, adjusts the global price levels in the market for a set of commodity goods. This adjustment is based on reported demand and supply in the market for a particular price point. Through price adjustments, the difference between demand and supply is to be minimized in order to obtain the price level for which the market attains an equilibrium. In order to support this minimization, we have integrated routines from the Optimization Toolbox and the Genetic Algorithms And Direct Search Toolbox of Matlab. The optimization routines from these toolboxes accept a Matlab function to be minimized, along with a set of options.

The integration is schematically depicted in Figure 4.11. An RMI server that exposes the Matlab optimization routines is installed in the Matlab JVM and registered with the RMI registry in the GES JVM. An RMI proxy to this server object is then obtained by the pricing component that represents the Walrasian auctioneer through a lookup operation on the RMI registry. When the simulation requires prices to be adjusted to equilibrium, this proxy is used to obtain a new price vector. Upon the invocation of the remote optimization call, the server object will use the Matlab JMI library to invoke the requested optimization routine with the given options from Java code. The Matlab `excessDemandFunc` function that reports the difference between supply and demand is called by this routine throughout the minimization process. Since supply and demand levels are determined by bidding agents that reside in the GES JVM, this Matlab function needs to call the `RMISupplyDemandProvider` that reports on the demand and supply levels for a given price point.

4.6.4 Other libraries

The following open source libraries are also used by GES:

- JFreeChart : Charting library used for building the analysis framework in the GES UI.
- Colt : Library for high performance scientific and technical computing in Java.
- JUNG : Java Universal Network/Graph Framework used for creation and manipulation of graphs.
- Xerces : Java XML parser.
- Log4j : Java logging library.
- JUnit : Unit testing framework.
- Jini : Framework for developing object oriented distributed systems. We use Jini to provide for a computational. backend for distributed execution of simulation experiments on desktops and clusters.
- Java CoG Kit : Library used for distributed execution of simulation experiments on Globus resources.

4.7 Performance Study

As mentioned in section 4.2, many frameworks have been developed for simulating grid systems. We have chosen to compare the performance of GES to SimGrid and GridSim because of their maturity, extensive user base and active development status. The main differences between the three simulators relate to the chosen simulation paradigm and threading model. While GES offers both single-threaded discrete-time and discrete-event based simulation, both SimGrid and GridSim use massively multithreaded discrete-event based simulation. In this regard, SimGrid offers the choice between the use of native `pthreads` and `ucontexts` as user space threads to overcome the limitations related to the amount of native threads that can be supported by the OS. The use of `ucontexts` however, is only available for developers coding in C or C++. Similarly to GridSim, the Java bindings to SimGrid only support native threads, a consequence of the threading model used in current JVMs.

4.7.1 Discrete-time core performance

We will test the general scalability of the different simulators and determine whether they are capable of simulating a system on the scale of the EGEE grid. In addition, we will investigate how the three simulators scale in terms of the number of jobs in the system.

All tests use variations of a base scenario with the following parameters:

- Number of consumers $N_{\mathrm{c}} = 1,000$
- Number of providers $N_{\mathrm{p}} = 100$
- Number of jobs per consumer $N_{\mathrm{j_c}} = 10$
- Job length in units of time $L_{\mathrm{j}} \in [7, 13]$
- Number of cpus per provider $N_{\mathrm{cpu_p}} = 10$

Since we want to focus on the performance properties of the simulators' cores, we split up the consumers in groups of 10 and associated each group with a single provider which schedules incoming jobs in a round robin fashion. Every job is atomic, CPU bound, and is considered to be part of a trivially parallel workload (i.e. there are no interdependencies between jobs).

For GES and GridSim, we did not use a simulated network. For SimGrid, we were obliged to use a trivial network setup because the network provides the only interaction channel for entities. Since it is currently impossible in SimGrid to aggregate multiple CPUs in one entity, we have modelled a provider node by combining one forwarding host with $N_{\mathrm{cpu_p}}$ CPU hosts. The forwarding host is connected to its CPUs with one link. Consumers are also connected to their provider with one link. Every job is routed over these links from the consumer to the forwarder and then to a CPU node. All links were configured with maximal bandwidth and minimal latency properties. Jobs were attributed zero bytes of network payload. Because of their higher potential scalability, we used `ucontexts` with a 64kB stack size instead of `pthreads`. We used SimGrid's MSG layer to simulate this scenario.

This simple setup uses the most basic set of features supported by the simulators for simulating job execution on grids. As such, the results are representative for the performance of the simulation cores as they are not influenced by scheduling logic, complex network configurations or advanced logging and output management.

All tests were performed on the CalcUA cluster at the university of Antwerp which hosts 256 Opteron 250 nodes running a 64-bit Linux distribution. All nodes used in the tests were equipped with 8 GB of RAM. During testing we measured the simulation time in milliseconds and the maximum memory usage, both real and virtual, by polling the simulation process every second. We used version 1.6.0 of Sun's JVM for the execution of all Java code.

Test I: General Scaling

This scenario is designed to evaluate the general scalability of each simulator. We scale N_{c} from 10 to 10,000 while changing N_{p} such that $N_{\mathrm{c}}/N_{\mathrm{p}} = 10$. The other parameters are maintained at their default values. In effect, this test scales up the entire base scenario.

Figure 4.12 shows the time it took to perform the simulation on a logarithmic scale as a function of the number of simulated consumers. As shown in the graph, GES scales up better than both GridSim and SimGrid. The difference in simulation

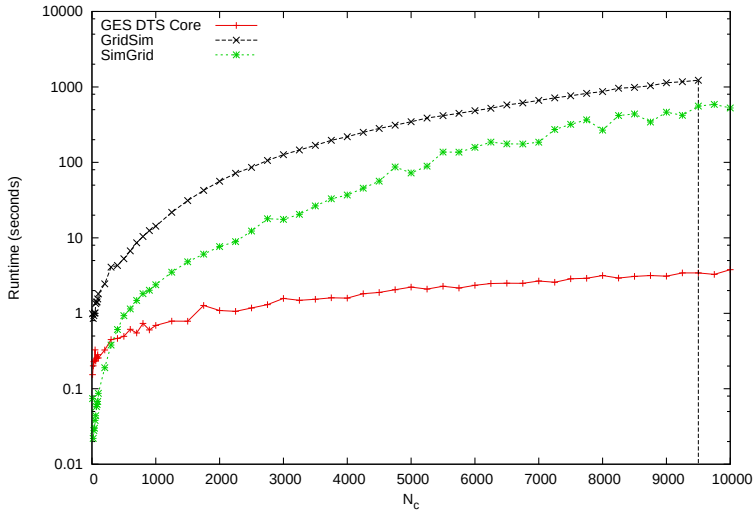

Figure 4.12: Simulation time as a function of the number of consumers

time is well over two orders of magnitude when simulating a grid with over 5,000 consumers. GridSim was unable to simulate an environment with 10,000 consumers because it would need to create well over 32,000 threads, which is more than a normally configured Linux kernel can handle. When scaling up beyond $N_c = 10,000$, we were able to determine that SimGrid could handle a maximum of 12,000 consumers while GES had no problem simulating 100,000 consumers.

Figure 4.13 shows the actual memory usage as a function of the number of consumers. SimGrid does not scale linearly in this regard. The use of threads can only account for a linear increase in memory usage. At a scale of 10,000 consumers, we simulate 21,000 entities[5]. The required stack size for these entities only covers 1.3 GB of the memory usage. We have confirmed that the majority of the remaining memory usage can be ascribed to the static routing table used by SimGrid. More specifically, SimGrid only supports a global static routing table which is initialized at the start of the simulation. This table contains the routes from each individual CPU to each other individual CPU. As such, the routing table scales quadratically in the number of CPUs and limits SimGrid's scalability. We are currently extending SimGrid with a dynamic routing protocol in the form of OSPF in order to overcome these limitations. While GridSim does scale linearly, its actual memory usage is still substantially larger compared to GES' memory usage.

Test II: Job Scaling

This scenario evaluates the influence of the number of jobs on the simulation time. We keep the total workload per consumer at 100, and scale $N_{\mathrm{j_c}}$ from 1 to 100. The simulation time is plotted in Figure 4.14. We can see that SimGrid handles the increasing number of jobs better than GridSim and that GES is virtually unaffected by the amount of jobs. It is clear that when using the discrete-event paradigm, the

[5] 10,000 consumers and 1,000 providers with 10 processors each

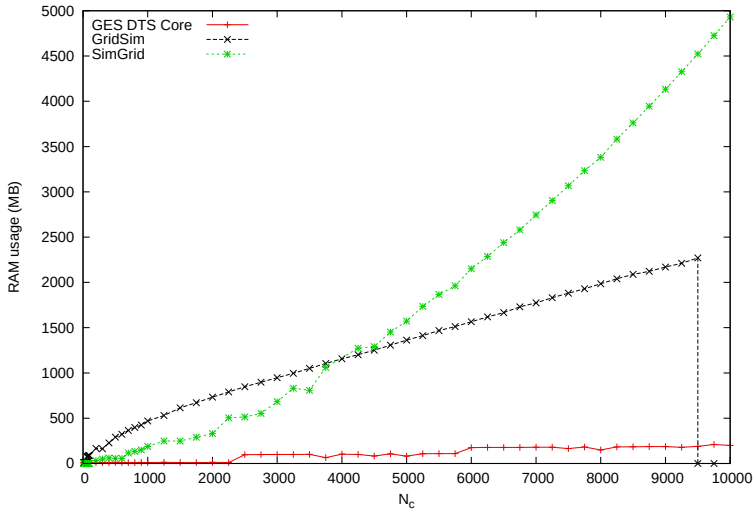

Figure 4.13: Real memory requirements as a function of the number of consumers

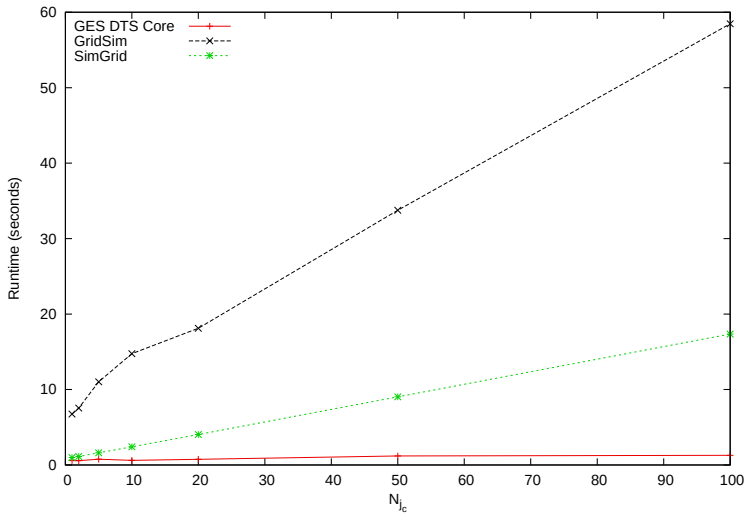

Figure 4.14: Simulation time as a function of the number of jobs

simulation time scales linearly with the number of jobs while this is not the case for the discrete time paradigm. A discrete time paradigm in contrast scales linearly with the size of a job while this has no effect on a discrete-event system. The suitability of either paradigm is determined by the resolution at which we wish to model time. While it is interesting to model time in high resolution for the analysis of interaction protocols, it is not necessary to do so for the simulation of job execution. In section 4.7.3 we will directly compare the performance of the Discrete-Time System (DTS) core versus the Discrete-Event System (DES) core.

4.7.2 Threading and Virtual Memory

Both GridSim and SimGrid use threads for each simulated entity. GridSim uses the Java threading model while SimGrid offers a choice between `pthreads` and `ucontexts` on Linux. Both Java threads and `pthreads` are native while `ucontexts` are user space threads. On a Linux machine with a normally configured kernel, the number of simultaneous native threads is limited to just over 32,000. Since user space threads are not visible to the kernel, they can overcome this limitation. Moreover `ucontexts` offer more configurable stack sizes than `pthreads`.

Each thread or context will also allocate a stack in virtual memory. This stack will be created in real memory only when it is needed on a per page basis. Therefore, threading does not necessarily have a direct impact on *physical* memory usage. However there are limits on the amount of *virtual* memory a process can allocate. This limit is 4 GB on any 32-bit architecture, of which in general only 3 GB can be used by the process itself on a Linux machine. On 64-bit Linux machines, the practical upper limit of virtual memory available is the sum of the physical memory and the swap size unless oversubscribing is allowed. To mitigate these limitations, it is possible to tweak the size of the thread stack. The typical thread stack size for Java threads is 320 kB on a 32-bit and 1024 kB on a 64-bit Linux machine. It is possible to reduce this size with a JVM configuration parameter to a minimum of 64 kB. For `pthreads`, the minimum stack size is 56 kB while `ucontext` stacks can be even smaller. Another advantage of `pthreads` and `ucontexts` is their ability to adjust the stack size per thread while Java does not offer this flexibility.

The limitations on the scalability of a threaded approach are depicted in Figures 4.15 and 4.16. The graph in Figure 4.15 includes both the maximum native thread limit as well as the 32-bit memory wall. It also demonstrates the effect of tweaking the stack size on the virtual memory usage. The graph in Figure 4.16 shows the maximal virtual memory allocation of the three simulators as a function of the number of consumers in the general scaling scenario. It shows that the JVM by itself allocates 1 GB of virtual memory. This is used as a code cache which contains the interpreter and code generated by the compilers. From the graph we can also observe that GridSim is capable of claiming close to 35 GB of virtual memory. The magnitude of this allocation is due to the standard thread stack size of 1 MB. SimGrid allocates a significantly smaller amount of virtual memory. When we contrast this with the actual memory usage in Figure 4.13 we can observe that SimGrid uses almost all of its virtual memory while GridSim uses less than a tenth of its allocation.

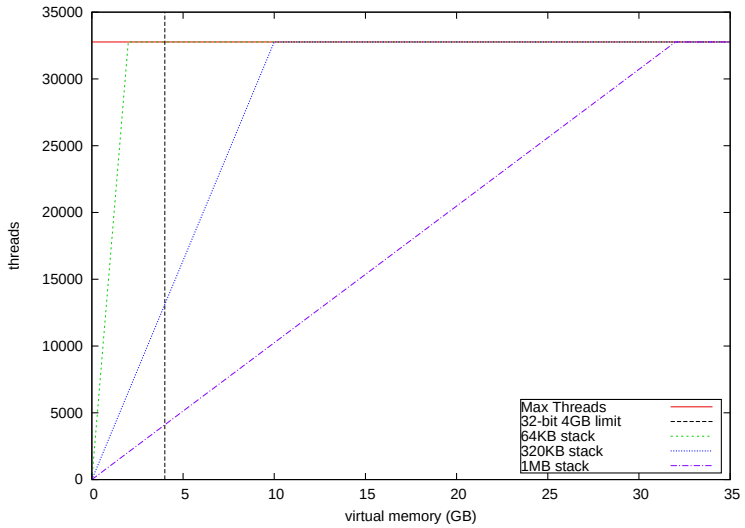

Figure 4.15: Number of native threads as a function of available virtual memory

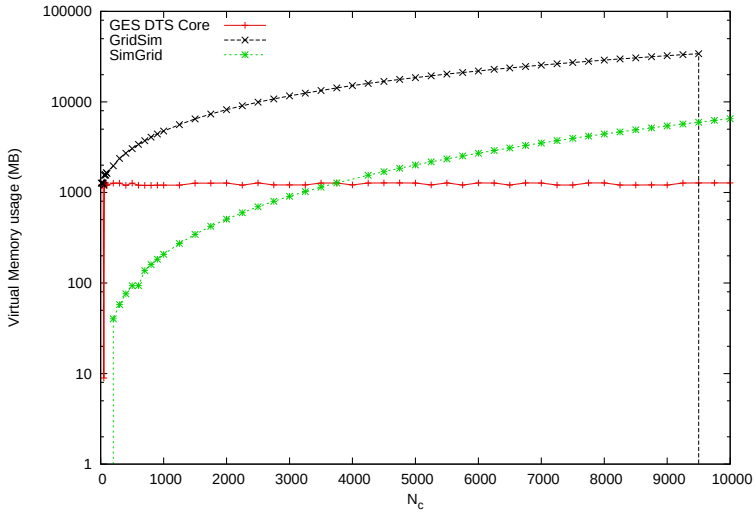

Figure 4.16: Virtual memory requirements of the simulators under general scaling scenario

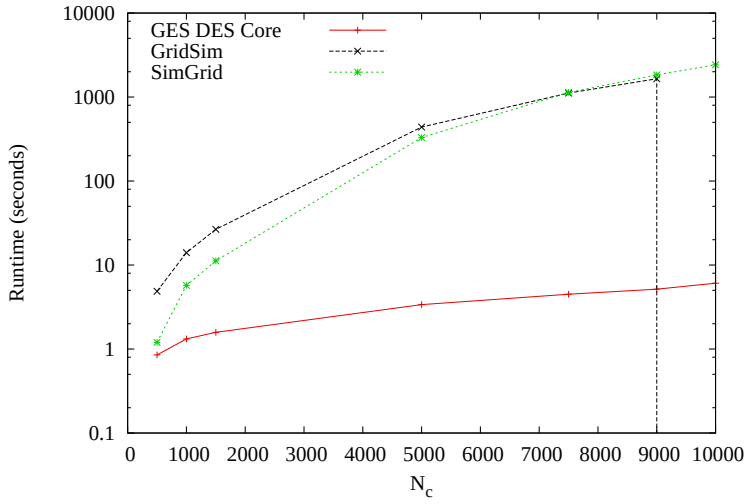

Figure 4.17: Simulation time for SimGrid, GridSim and GES using the discrete-event core, as a function of the number of consumers

4.7.3 Discrete-event core performance

In order to evaluate the performance of our discrete-event simulation core, we use a similar setup to the one described in section 4.7.1 in which we associate a 5 minute timeframe with each discrete time step. We will simulate N_c consumers that will each send 10 atomic jobs out to one of the N_p providers in the grid infrastructure. Each job has a running time between 35 minutes and 65 minutes (in increments of 5 minutes), on a reference architecture that delivers 2GFlops/s. Each provider hosts 10 CPUs that deliver 2GFlops/s each. At the start of the simulation, consumers start submitting their jobs to providers after a uniformally distributed delay of one to ten simulated seconds. The simulation completes when all jobs have finished their execution. Because we focus on the performance of the simulators' cores, no planning logic is present in the consumer to select a specific provider to send a job to. We simply split up the consumer population in groups of 10, and associate with each group a provider that will schedule incoming jobs in a round robin fashion.

All tests were run on a Intel Quad Core desktop with 8GB of RAM and 2.83 GHz cores. We used a 64-bit Linux distribution along with version 1.6.0 of Sun's JVM for the execution of all Java code.

Figure 4.17 shows the time it took to perform the simulation on a logarithmic scale as a function of the number of simulated consumers. We scaled N_c from 10 to 10,000 while changing N_p such that $N_c/N_p = 10$. As shown in the graph, the GES DES core delivers significantly higher performance compared to GridSim and SimGrid. Simulations took longer for GridSim and especially SimGrid compared to the results in Figure 4.12. We have verified that this is not due to the difference in the hardware platform on which the tests were executed. The main cause lies in the fact that consumers now submit their jobs at the start of the simulation after

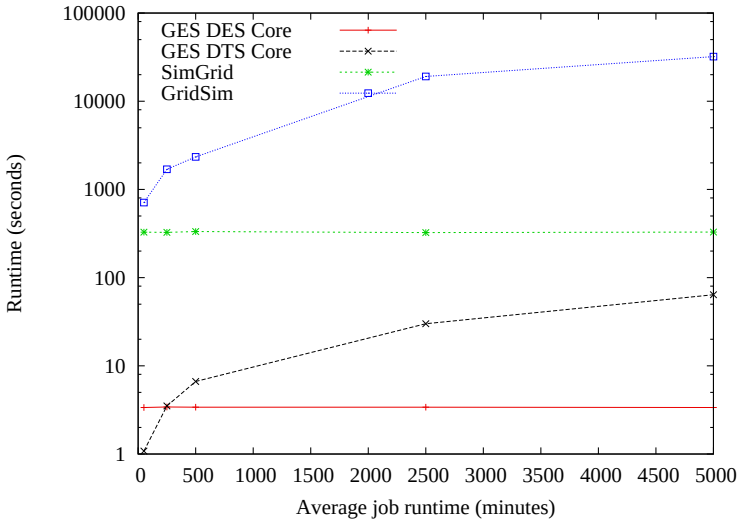

Figure 4.18: Simulation time for SimGrid, GridSim and GES using the discrete-time and discrete-event cores, as a function of the average job runtime

a random delay. In the discrete-time core tests such a delay was not introduced and all submissions were synchronized. Because of the limited spread in the length of job runtimes, a relatively large number of jobs then finish at the same time. This significantly reduces the processing power that the SimGrid and GridSim cores need to dedicate to job management. SimGrid for example, solves the resource sharing model for each action ending time that occurs in the simulation. If the ending times of jobs are not synchronized, an ending time needs to be determined for each individual job. This requires a significantly higher number of model solve calls compared to the simulations for which the results where shown in Figure 4.12.

The results in Figure 4.18 indicate that under increasing job lengths (or alternatively a decreasing amount of real time associated to one time step in the DTS core), the performance of the DTS core diminishes almost linearly. This is to be expected, as in a discrete time simulator the runtime depends directly on the amount of discrete time steps simulated. For the results in Figure 4.18 we equated one simulated time step in the DTS core with five minutes of real time.

4.8 Summary and Conclusions

Due to its flexibility, performance and low cost, simulation has been the tool of choice for researchers to explore new resource management systems for grids. In this chapter, we have introduced the Grid Economics Simulator which specifically supports research into economic resource management systems. We have presented the simulator's architecture and features and have given an overview on the integration of third-party software components within GES. These components introduce

functionality related to efficient simulation of networks, synthetic generation of clusters and networks, and optimization routines.

A key challenge in the design of GES was the development of an efficient simulation core and a clean programming model. We have shown how the combination of Aspect Oriented Programming and Java annotations contribute to this goal. Many simulation cores adopt a massive multithreaded approach to simulate concurrency which impacts scalability, performance and complexity of the resulting code. We have presented an alternative in the use of continuations which provide a minimally sufficient implementation for programming a discrete-event core without incurring the overhead costs and complexity related to a more general thread-based model.

In light of grids being large-scale systems, we have compared the scalability and performance of GridSim, SimGrid and GES. As we have observed from the results of our tests, both SimGrid and GridSim are unable to scale to the level that would allow them to simulate very large scale grid infrastructures such as EGEE. For GridSim the problem is rather fundamental in that it requires an amount of threads during simulation that reaches the upper limit of threads manageable by a normally configured Linux kernel. While SimGrid can sidestep this issue by using `ucontexts`, it still reaches an upper limit of 23,000 simulated entities on a machine with 8GB of RAM due to the extensive memory use that results from its network model.

Depending on the resolution at which time is to be simulated, it is better to choose either the discrete-event or discrete-time simulation paradigm. Whereas a detailed analysis of the impact of communication delays requires a high resolution, a lower resolution is adequate for simulating the execution and scheduling of jobs in a grid system. The programming models used in discrete-time and discrete-event cores also show substantial differences. In general, a discrete-time core offers a more simple programming model, as it abstracts away from the concurrency details that are present when using discrete-event simulation.

Irrespective of the choice for a discrete-time or discrete-event model, one can decide to use a threaded or non-threaded simulation core. We have quantified the impact of this choice through a comparative analysis of GES, SimGrid and GridSim and have shown its effect on simulation performance and scalability. As our performance evaluation shows, the proposed single-threaded discrete-event and discrete-time cores can significantly outperform the massive multithreaded cores of two widely used grid simulation tools.

CHAPTER 5

Spot Markets

5.1 Introduction

Current production-level grid resource management and scheduling systems adopt a usage model in which users send out individual jobs to a broker for immediate execution. In practice, jobs are handled by the grid's global and local scheduling systems in a first-come-first-serve manner, possibly delaying the actual start of their execution. From the user's point of view however, no restrictions on this potential delay are formulated. The formulation of job execution requests by users is thus adhoc, and does not include any restrictions on when the job should finish in future time.

In this chapter, we will stay close to this usage model through the use of *spot markets* for our economics-based resource management model. In spot markets, trades imply an immediate, or almost immediate, delivery of goods. As such, they follow the usage model in which demand for computational resources is expressed adhoc, and the fulfillment of the demand is expected to be realized within a limited timeframe. A spot market is typically characterized by a relatively fast trading process which is an advantage in the context of trading non-storeable commodities or commodities with significant storage costs, such as energy.

In the vision put forth by the grid computing paradigm, users treat raw computational, network, and storage resources as interchangeable, and not as specific machines, networks and disk systems. This corresponds closely to the notion of a *commodity market*. In such a market, a limited number of standardized *commodities* are defined which are traded on a *commodities exchange*. This exchange constitutes a double-sided market mechanism that operates under a given set of rules for trading and pricing of goods. Commodities exchanges typically support both *spot trading*, as well as *forward trading* contracts which allow for the future delivery of a given

quantity of goods at a given price. An example of a real-world commodity spot market is the Title Transfer Facility (TTF) in the Netherlands. This spot market for natural gas defines four commodities (H, L, G and G+), based on the quality of the natural gas which is measured by its calorific value. It allows producers to quickly sell off surplus gas to consumers that require additional capacity. In this chapter, we will focus on spot trading in accordance with the current usage model on grids. The next chapter will take considerations with respect to forward trading into account in order to allow users to formulate more advanced QoS constraints on the execution of their jobs.

In order to determine the *spot price*, i.e. the price at which commodities are currently traded in the exchange, a wide range of processes for price formation can be applied. We will apply the process first described by Walras [158]. In this approach, a centralized process, called the *Walrasian auctioneer* [1], iteratively adjusts the price of the different commodities based on the demand and supply reports of consumers and providers with respect to this price. The price adjustment process continues until a market *equilibrium* is achieved in which the demand for the different commodities equals the supply. As a consequence of the *first fundamental theorem of welfare economics* [160], a Pareto optimal (or Pareto efficient) allocation is achieved when the market is brought to equilibrium[2]. This means that a change in price cannot increase the utility of one participant in the market without worsening the utility of at least one of the other market participants. In this case, the market is also said to attain *economic efficiency*.

An attractive feature of this commodity market model is that the market price is optimized through a simple interaction between the market and its participants. Participants only have to signal their demand and supply levels for the different commodities to the Walrasian auctioneer, given the commodities' prices. In non-commodity market organizations wherein specific resources are sold in separate markets (e.g. single unit auctions that sell access rights on a specific CPU of a provider), participants need to deal with the extra complexity of deciding in which market to participate. In addition, consumers that decide to participate in multiple markets at the same time, need to determine how they will divide their available budget across these markets. If these decisions are taken in a non-optimal way, this can lead to inefficiencies. Although market organizations such as combinatorial auctions, where bids are formulated for a combination of items, offer a means to deal with some of these issues, the determination of an optimal winner combination given a set of bids in such auctions is NP-complete [161] and practical applications of this model in the context of grid resource management have shown the intractability of this approach [162].

A second advantage of the commodity spot market model is that participants are encouraged to report their true valuations (in terms of demand and supply levels), because they have no information on the demand levels of other participants and do

[1] This term was first coined by Leijonhufvud [159] to depict the auctioneer described by Walras

[2] Note that this follows only for a *complete market* under the assumptions of *perfect competition* and *non-satiation*. The extent to which these assumptions can be justified in the context of studying real economies is one of the key aspects that distinguish the different *schools* in economics.

not now how close the suggested price is to equilibrium. In addition, the centralized approach taken in the commodity market model, results in a large number of agents participating in the mechanism. This leads to a *competitive market* in which the *power* of an individual agent to influence the market-wide prices is very limited. In this setting, consumers are said to exhibit *price-taking* behavior. As a result, the potential payoff that an individual agent can obtain by acting strategically by an adjustment of its own behavior based on the (expected) behavior of other agents in the mechanism is close te zero [163]. In the context of practical market-based optimization, these are important considerations as such strategic behavior often increases the computational and communicative complexity of the the logic that governs the bidding behavior of agents in the market [164].

There are also some disadvantages linked to the commodity market approach in the context of grid resource management. First, the notion of locality is not taken into consideration. In a commodity market, all resources in the same commodity category are deemed equal and the added cost for network usage is not incorporated in the price. The creation of multiple local markets that trade resources within a *region* of the network is a possibility to introduce locality considerations at a high level. This decentralization does increase bidding complexity however, as consumers must now face the same decisions with respect to participation in multiple markets that we outlined for the single-unit auction market, albeit in a less extensive form. Care must also be taken that splitting the resource trading process over multiple markets does not fragment the market too much in which case the assumptions with respect to price-taking behavior would no longer hold.

Secondly, as soon as one has to start paying for resource usage, one can no longer deem all resources equal. Besides performance, aspects such as quality of service, available (cache) memory, and network bandwidth contribute to the value of a compute node. The commodity model limits the specificity with which consumers can value resources. Techniques such as *fuzzy clustering* [165] can help in the formation of well-chosen commodities by grouping all the compute resources in n categories based on a, fuzzy, combination of their relevant properties. As a result, one can come up with n clusters of compute resources: e.g. low-end, average and high-end systems. The approach of partitioning resources in a number of *service classes* has also been deemed advantageous for increasing resource and scheduling efficiency by Volckaert et al. [166]. In [166], two resource and service partitioning algorithms are described, one based on genetic algorithms and the other on the optimization of an Integer Linear Program (ILP).

Once a partitioning is obtained, prices can be determined using commodity market principles. This time however, one has three *substitutable* commodities, which makes resource allocation decisions more challenging. Is it preferable to buy A resources of type X, or B resources of type Y, or maybe some combination? This indicates clearly that pricing of substitutable goods using commodity market principles, is an important step in the construction of grid markets.

5.2 Related Work and Classification

Early prototypes of economic resource management systems that use a spot market model are Spawn [167] and Popcorn [168]. Spawn supports applications that can increase or decrease their parallelization degree through the use of hierarchical funding structures in a peer-to-peer environment. Each peer runs a Vickrey auction to sell idle processing power. In the Popcorn system, goods called JOP (Java operations per second) are traded in a resource market using three alternative pricing mechanisms; Vickrey auctions, a simple double auction and a clearinghouse double auction. Analysis of price dynamics, social efficiency and price stability in Popcorn showed promising results for an environment with non-strategic buyers and sellers.

More recently, HP Labs has developed an economic resource manager [136] for grids using a proportional sharing mechanism. The mechanism assigns the user an allocation share on a resource that corresponds to the relative weight of its bid compared to all the other active bids on the resource. A *best-response* algorithm is used by each individual bidder to compute the optimal division of credits among the different resources, given the current price levels on the spot market and the consumer's budgetary limits. Because resource shares are immediately adjusted as new bids arrive at a provider, bidders have no guarantees on the size of the allocations they will receive after placing a bid. Recently, extensions to this work have been proposed in order to address this issue [169, 170].

An evaluation of different auctioning protocols in the context of a spot resource market is given in [171]. The authors consider the effect the different protocols have on the resulting payments when considering a mix of risk-averse and risk-neutral buyers. Their empirical study confirms the intuition that Vickrey auctions favor buyers, first-price sealed-bid auctions favor sellers and double auctions favor neither. A similar study was done by Pourebrahimi et al. [172] for different levels of resource congestion. The bidding strategies used in [172] have been shown to outperform a zero-intelligence bidding strategy and a bidding strategy based on a simple learning approach [173], in terms of job throughput and a better adaptability of price levels to the congestion levels in the network.

Gomoluch et al. [174] compare the aforementioned proportional-sharing approach with a continuous double auction and a round robin scheduling policy for a setting with independent tasks and under homogeneous network link characteristics. Their results confirm the expectation that market-based approaches can deliver significant added value under periods of resource contention or if resource heterogeneity is high. Under low load and a homogeneous resource infrastructure, a simple round robin approach can obtain similar performance in terms of average completion time without the communicative and computational overheads caused by market mechanisms. The proportional sharing approach was shown to outperform the continuous double auction in situations with high resource heterogeneity. It was also found to be better in settings with communication delays that are large compared to the runtimes of jobs. Under very high congestion however, the CDA delivered significantly better results. The concurrent execution of multiple tasks on one resource in the

proportional sharing approach, was shown to significantly increase the individual completion times of jobs in this setting.

According to the classification in [174], our work uses a state-based, non-preemptive approach. State-based means that allocations are based on the current snapshot of the system state as opposed to model-based where the allocations are based on a predictive model of job runtimes and resource availabilities. Non-preemptive means that tasks are assigned to hosts once and stay there as opposed to preemptive strategies where tasks are allowed to be interrupted and possibly migrate between hosts. When one classifies based on the underlying economical model [175], our work belongs to the category of commodity markets. Some other possible models are bargaining [176], posted price models [177], contract based models [178, 179], bid-based proportional resource sharing models [180, 181], bartering [182] and various forms of auctions [183, 184].

Cost-based resource management and market models have also proven to be useful for resource management in the context of cluster computing [185, 180, 186, 187]. Furthermore, market models for resource sharing have been applied to agent systems [188, 189], telecommunication networks [190, 191, 192, 193, 194], databases [195], and data mining [196]. An overview of their application within the domain of grid computing can be found in [197], [198] and [199].

The use of a commodity market model for obtaining global equilibrium prices for resources in a grid context has also been proposed in [7, 104]. We take this approach further and extend it to allow for trading and pricing of substitutable goods. This more closely models grid markets where multiple resources are available to process the jobs, but with varying performance characteristics. We also introduce significant improvements to optimization algorithms used to compute the equilibrium prices. Finally, we extend the evaluation of the commodity market approach, thereby considering aspects such as computational and communicative requirements, fairness of allocations, the impact of introducing a large amount of commodity goods, and a statistical analysis of the performance of the price adjustment process in terms of obtaining market equilibrium.

5.3 Commodity Market model

Our market model consists of three parties: consumers, providers and the market itself. Consumers formulate demand for the different commodities in the market while providers formulate supply for these commodities. Given these demand and supply reports, the market establishes an equilibrium price, matching demand and supply, using a pricing algorithm. Consumer and provider models, which are loosely based on [104], are described in sections 5.3.2 and 5.3.3 respectively. The pricing algorithm is detailed in section 5.4.

5.3.1 Resource and Job Model

A complete and accurate grid resource model should include a large set of different resource types, each with their own specific attributes. Examples include CPU time, temporary and permanent storage, network bandwidth, main memory, and more specialized resource types for specific hardware components. Evidently, the scope of simulated resources is an important design decision, but so is the extent to and manner in which these resource types are introduced as tradeable goods into the market. Fully exposing all resource types and attributes allows for a very accurate valuation of resources by grid users. A downside to this approach is the increase in the complexity of the market's pricing mechanism, the interactions between the market and its participants, and the participants' valuation logic. In practice, a balance needs to be found between allowing for accurate valuations and computational complexity. The approach taken by Schnizler [93] for example, allows for fully accurate valuations (except for aspects related to resource locality), but is computationally intractable in the sense that the outcome of the market mechanism cannot be computed within a reasonable timeframe for realistic grid settings. We believe that in the end, practical computability of the mechanism's outcome is key if market-based resource allocation is to be used in practice. In that respect, we believe that the benefits of market-based resource allocation can be realized to a large extent, without requiring fully accurate valuations. This is further motivated by the fact that *valuing* resources by itself is not a trivial process for users. Complex resource models only make this problem worse. As such, we envision an operational model with two tiers as schematically depicted in Figure 5.1:

- The market serves the role of a high-level admission controller which accepts requests for job execution based on high-level and aggregated supply and demand information.
- Components for system-centric optimization take on the task of planning the accepted workloads on specific grid resources taking into account more fine-grained resource attributes

Given these considerations, we take the approach of restricting our resource model to CPU resources and to consider these the single type of resources that are tradeable in the market. In the following we will therefore consider a commodity market which we will subsequently refer to as *the market*, of two substitutable *commodities*: low-end and high-end CPUs (`fast` CPUs and `slow` CPUs)[3]. These CPUs are *logical* entities, i.e. they do not necessarily correspond to a physical CPU. Through virtualization, physical CPUs can be partitioned into a number of logical CPUs. As all results presented in this chapter are based on simulations and not on a real world grid setup, the problem of clustering CPU resources into a finite set of resource categories is not an issue. However, once the proposed market organization will be integrated in real grids, clustering of available resources will become an important aspect for practical deployment of the market mechanism.

[3]In section 5.10 we will investigate the impact of further scaling up the number of commodity goods.

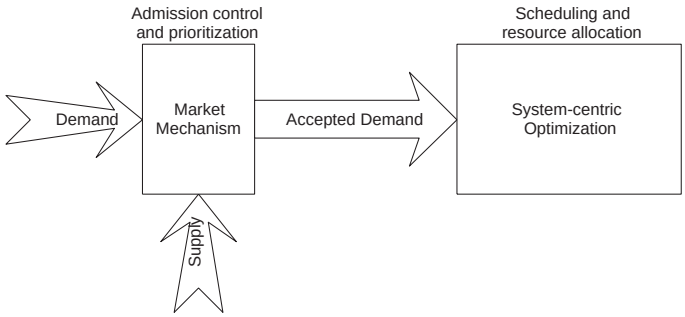

Figure 5.1: Two-tier operational model for market-based resource allocation

In a setting with substitutable goods, one needs a measure by which goods in the substitutable categories can be compared. For compute nodes, we express this as a speedup factor s_i, which denotes the speed of a CPU from category i relative to the speed of a reference machine M_{ref}. Without loss of generality, one could also express a resource's speed in a common measure for compute performance such as MIPS or GFlops/s.

As a consequence, consumers will be faced with the problem of deciding which of the two categories to buy for a particular job. This differs from the work presented in [104, 7] wherein a commodity market model is set up with two complementary goods in the form of CPUs and disk storage. We argue that the presented commodity model with substitutable goods is more useful in practice, as we believe that trading for disk storage certainly needs to include locality considerations if it is to lead to satisfactory results in a grid context. In addition, the introduction of multiple CPU categories allows for more differentiated QoS.

In accordance with the limitation of our resource model to CPUs, we model jobs as CPU-bound computational tasks. Every job has a normalized runtime T, determined by the time it takes for the job to finish on a reference machine M_{ref}. However, in the following we will not assume that providers or consumers know this runtime. When jobs are allocated to a CPU in category i, their actual runtime is determined by $\frac{T}{s_i}$. Jobs are also taken to be atomic, in the sense that they are always allocated to a single CPU.

5.3.2 Consumer model

Consumers formulate demand in the market by expressing their willingness to buy CPUs from providers in order to run their jobs. In accordance with the spot trading used in the market, providers only bring available CPU resources to the market. As such, jobs start running immediately when allocated to a provider's CPU resource.

Each consumer is endowed with a limited budget which is periodically replenished according to a budget refresh period, termed the *AllowancePeriod*. At the start of such a period, leftover budget from the previous period is discarded. We do not assume a particular funding source for consumers. In practice, funding rates could

be determined by system administrators, could be set by consumers themselves through monetary payment, or could result from a feedback loop that redistributes the credits earned by providers of a particular (virtual) organization to the users of that organization.

To prevent the scenario where every consumer tries to schedule all of his jobs at the start of its allowance period, which would result in an extreme increase in price, expenditures are spread out evenly across the allowance period. Furthermore, we do not assume that consumers know the runtime of their jobs. Therefore, we need to prevent that they accept prices that are not sustainable for them over the entire allowance period.

We define the market price vector $\mathbf{p}$ as $\mathbf{p} = (p_1, ..., p_n), p_i \geq 0$, where p_i represents the price per resource unit, per time unit of the ith commodity. Given such a price vector, consumers will have to report their demand for each resource type to the Walrasian auctioneer.

In order to determine the budgetary constraints under which demand can be formulated, a consumer calculates its average rate of spending ($rate_{avg}$). This denotes the rate at which it would have spent credits for the jobs it has run so far if it had been charged the currently suggested price vector $\mathbf{p}$. It then computes the sustainable spending rate $rate_{sustainable}$, i.e. the total rate of spending it can sustain until the next replenishment of its budget. If $rate_{avg} < rate_{sustainable}$, a consumer will express demand. The available spending rate at the time of demand formulation is then given by $rate_{available} = rate_{sustainable} - rate_{processing}$ with $rate_{processing}$ the amount of credits needed for currently processing jobs.

After determining the budgetary limits, a consumer needs to decide which commodities to buy. Depending on the consumer's preferences with respect to performance of CPU resources, certain resource categories will be preferred over others. This is expressed through a $ValuationFactor_i$ term. These considerations lead to an adjusted price for each category, given by

$$AdjustedPrice_i = \frac{p_i}{PerformanceRatio_i * ValuationFactor_i} \tag{5.1}$$

The right hand side of equation 5.1 reflects the price normalized to unit performance and factors in the valuation.

The consumer then expresses demand, limited by the current available rate of expenditure given by $rate_{available}$, in the category with the lowest adjusted price. In cases where $AdjustedPrice_i$ equals $AdjustedPrice_j$, a random ordering is assigned to both CPU categories. If there is budget left after expressing demand in the category with the lowest adjusted price, the consumer considers the next best category to spend its remaining budget on. This continues until the available spending rate has been fully used, or demand is formulated in the category with the highest adjusted price.

The valuation factor factor is a simple abstraction for the complex logic a consumer might follow to prefer one CPU category over another besides pure cost considerations. A consumer might follow such a logic wherein he is willing to pay more than double the price for a CPU of category 1, which is only twice as fast as

one of category 2, if his job needs to finish within a deadline for example.

5.3.3 Provider model

Providers host a configurable number of resources of different types. In order to determine the amount of resources of a given type that a provider will offer for sale, it first calculates the average revenue per time step rev_i accrued by selling resources of this type in the past. If the suggested price p_i is greater than rev_i, all available resources of this type are offered for sale. If the price is lower, only the fraction $\frac{p_i}{rev_i}$ of the available resources is offered. Providers are thus prepared to limit their supply of free CPUs to the market in order to keep prices high, thereby taking on the strategy of maximizing revenue instead of utilization.

However, the fewer resources are sold, the lower rev_i becomes, which in turn leads to an increase in the number of resources offered at price level p_i. The speed of these downward price adjustments is controlled by an elasticity attribute E, which determines the length of the time period that is taken into consideration for obtaining rev_i. If E equals zero, the provider does not take past revenues into account when determining supply, and always makes every CPU that is not currently executing jobs, available to the market. If E equals infinity, the provider will consider all of the revenue that was generated in the past. This way, E determines the provider's reluctance for adjusting its supply when confronted with downward price adjustments.

Consumers will be charged the market price at the time of resource acquisition as a fixed rate for the entire duration of the job. An alternative to a fixed rate is to allow a variability in the charged rate based on the market price evolution. Another option is to allow variability on the performance a consumer receives for a given rate over the job's execution period, an approach adopted in [136]. These alternatives allow for faster reallocation of resources according to the dynamics in the market. However, the resulting uncertainty makes budgetary planning and resource usage planning much more difficult for consumers.

5.4 Pricing Algorithm

As mentioned in chapter 3, resource cost should be dynamic, depending on demand, supply, value and general wealth. If a dynamic pricing model is introduced, prices can synthesize and communicate complex consumer and provider preferences with respect to the different resource types. As such, they allow the commodity market to exhibit emergent behavior in that the individual reactions of consumers and providers to price changes will lead to a division of resources that maximizes allocative efficiency, and result in an equilibrium between supply and demand.

Consider a market of n substitutable commodities and corresponding prices each measured by non-negative real numbers. A price vector $\mathbf{p} = (p_0, p_1, ..., p_{n-1}), p_i \geq 0$, where p_i represents the price of a unit of the ith commodity. Let R^n denote an n-dimensional real Cartesian space, and let $R^n_+ = \{x \in R^n \mid x_i \geq 0\}$. We consider

an economic setting with aggregate supply and demand functions $S, D : R_+^n \rightarrow R_+^n$, with S, D functions of prices in R_+^n. The condition for economic equilibrium is that supply equals demand, or $D(\mathbf{p}) = S(\mathbf{p})$ as a condition on the price vector $\mathbf{p}$. The derived excess demand is the function $\xi : R_+^n \rightarrow R^n$ given by $\xi(\mathbf{p}) = D(\mathbf{p}) - S(\mathbf{p})$. Thus, $\xi_i(\mathbf{p})$ is the excess demand for the ith good at price vector $\mathbf{p}$. As defined, it can be positive or negative; negative excess demand can be interpreted simply as excess supply.

Note that one of the commodities is typically chosen as a *numeraire*, so that the prices of the other goods are expressed in terms of this numeraire good. Typically, the numeraire represents the value of one unit of currency and it is fixed at one. As such, we have

$$\forall \mathbf{p} \in R_+^n : p_0 = 1 \tag{5.2}$$

In general equilibrium theory, common hypotheses on ξ are [200, 201]:

1. $\xi(\alpha\mathbf{p}) = \xi(\mathbf{p})$ for $\alpha > 0$ (Homogeneity of degree zero)
2. $\mathbf{p}.\xi(\mathbf{p}) = 0$ (Walras' Law)
3. if $p_i = 0$, then $\xi_i(\mathbf{p}) > 0$ (Desirability)
4. $\xi(p)$ is a single-valued and continuous function
5. $\xi(p)$ is continuously differentiable

The homogeneity law states that a scaling of the numeraire together with the price levels for all non-currency goods with a factor α, will not lead to a change in excess demand. That is, excess demand is only impacted by the value of the price vector components relative to each other, and not by their absolute values.

The second hypothesis corresponds to Walras' Law which is based on Walras' identity

$$\forall \mathbf{p} \in R_+^n : \sum_{i=0}^{n} p_i D_i = \sum_{i=0}^{n} p_i S_i \tag{5.3}$$

The identity follows from the observation that the total monetary value of what a trader plans to buy should equal the total monetary value of what the trader plans to sell during a particular period. As an example, consider a trader which plans to buy two units of bread and one unit of butter under a current market price vector $\mathbf{p}$, with p_0 the numeraire being one euro, p_1 the price of bread at two euro and p_3 the price of butter at three euro. The trader will therefore need to offer $p_1 * 2 + p_3 * 1 = 7 * p_0 = 7$ units of supply in the currency commodity to cover its demand. A similar reasoning holds for the trader supplying bread and butter but demanding currency in return. Because this property holds for each individual trader, equation 5.3 follows at the level of aggregate supply and demand.

Walras' Law then states that the sum of excess demands over all commodities in the economy should equal zero, irrespective of the fact that $\mathbf{p}$ constitutes an equilibrium price vector:

$$\forall \mathbf{p} \in R_+^n : \sum_{i=0}^{n} p_i D_i - p_i S_i = 0 \tag{5.4}$$

which is equivalent to the second hypothesis.

The desirability property states that zero prices always lead to positive excess demand. This property is linked to the notion of non-satiability, i.e. a consumer always gains additional utility by obtaining more units of a good at zero prices.

Finally, the excess demand function is assumed to be single-valued, continuous and continuously differentiable.

It has to be pointed out that the excess demands for the different goods are interrelated. Some reasons for this are that (1) goods are interchangeable, so the more one buys from good i, the less one needs from the other goods, and (2) consumers are considered to have a limited budget, so the more they buy from good i, the less credits they have left to buy other goods.

In order to obtain economic efficiency, the market needs to be brought to equilibrium. This market equilibrium is characterized by a market-wide price vector $\mathbf{p}*$ under which aggregate supply and demand equal one another. A consequence of Walras' Law is that

$$\forall \mathbf{p} \in R_+^n : \forall i \neq j \ \xi_i(\mathbf{p}) = 0 \Rightarrow \xi_j(\mathbf{p}) = 0 \tag{5.5}$$

Therefore, there is no need to explicitely consider the demand and supply in p_0 when searching for an equilibrium price vector, provided that budgetary constraints with respect to monetary transactions for trades are enforced. In the following, we will thus refrain from explicitely modeling the currency commodity.

In practice, finding an equilibrium price vector for a market with n commodities (excluding the numeraire), is equivalent to an n-dimensional minimization problem for the surface defined by the norm of the excess demand function $|\xi(\mathbf{p})|$. In the following we will also refer to norm of the excess demand $|\xi(\mathbf{p})|$ as the ED_Norm.

An example search space for a market model with two substitutable goods in the form of CPU resources of different speed is shown in Figure 5.2. Many fundamentally different techniques for solving such minimization problems exist. In the next section we will outline a well-know technique from quantitative economics, namely the Global Newton method of Smale [135].

5.4.1 Smale's method

Smale proves [135] that, given a market consisting of n commodities with price vector $\mathbf{p}$ and associated excess demand vector $\xi(\mathbf{p})$ an equilibrium point $\mathbf{p}^*$ exists such that $\xi(\mathbf{p}^*) = 0$ under the hypothesis on ξ presented previously.

Further assuming that ξ has continuous first and second order derivatives, $\mathbf{p}^*$ is found by solving the global Newton differential equation

$$D\xi(\mathbf{p})\frac{d\mathbf{p}}{dt} = -\lambda\xi(\mathbf{p}) \tag{5.6}$$

where the sign of λ is $-sgn(DetD\xi(\mathbf{p}))$. Here $D\xi(\mathbf{p})$ is the linear transformation with matrix representation

$$D\xi(\mathbf{p}) = \left(\frac{\partial \xi_i}{\partial p_j}\right) \approx \left(\frac{\Delta \xi_i}{\Delta p_j}\right) \tag{5.7}$$

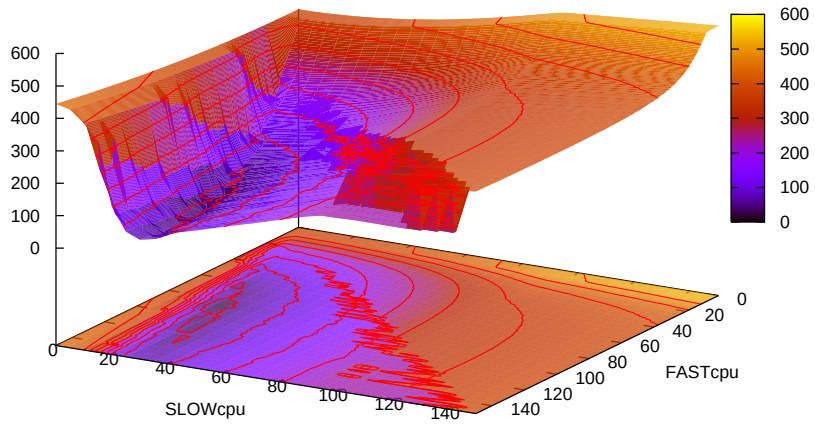

Figure 5.2: Example of a search space for the equilibrium price vector, in which the Z-axis denotes $|\xi(\mathbf{p})|$

To ensure that a solution of 5.6 exists, the following boundary condition is imposed on the price domain M and the boundary of this domain δM :

Boundary condition 1 $\forall \mathbf{p} \in \delta M, DetD\xi(\mathbf{p}) \neq 0$, *and there is a choice,*
(a) $sgn(\lambda(\mathbf{p})) = sgn(DetD\xi(\mathbf{p})), \forall \mathbf{p} \in \delta M$, *or*
(b) $sgn(\lambda(\mathbf{p})) = -sgn(DetD\xi(\mathbf{p})), \forall \mathbf{p} \in \delta M$,
which makes $-\lambda(\mathbf{p})D\xi(\mathbf{p})^{-1}\xi(\mathbf{p})$ *point into* M *at each* $\mathbf{p} \in \delta M$.

Intuitively, the boundary condition states that at the boundary of the price domain, an appropriate sign can be chosen for λ such that the first step in the search procedure points into the price domain. A more elegant and weaker boundary condition is given by Herings [202] by imposing that for every $\mathbf{p} \in M, \xi(\mathbf{p}) - \xi'(\mathbf{p})i$ is not radially outward pointing, where $\xi'(\mathbf{p}) = \sum_{i=1}^{n} \xi_i(\mathbf{p})/n$ is the mean excess demand at $\mathbf{p}$, and i is a vector of ones of dimension n.

In practical applications of Smale's method for settings without an analytical description of the excess demand functions, the partial derivatives can be approximated by discretization as indicated in equation 5.7. A limitation of Δp to 1 as in [104] however, leads to problems as we will describe later on.

5.4.2 Extension of Smale's method

The use of Smale's Global Newton method (cfr. equation 5.6) to find $\mathbf{p}^*$ relies on a number of conditions which are no longer guaranteed for a straightforward application to the proposed commodity market model. The problematic assumptions are:

1. If $p_i = 0$, then $\xi_i(\mathbf{p}) > 0$. If at a certain time step no consumer needs to schedule a job, this condition is violated and cannot be corrected.

2. The search space is continuous in every point. This condition is violated whenever resources are traded in units.

3. $D\xi(\mathbf{p})$ is nonsingular, i.e. the slope of the excess demand function in any of the commodities should never be 0. This condition is violated in situations where providers/consumers want to sell/buy an equal amount of goods for prices p and $p + \epsilon$.

4. Negative prices are not allowed.

Except for the first, these problems can be addressed by introducing a series of modifications to the search space and some extensions to the search algorithm. We will refer to the resulting algorithm as ESGN (Extended Smale Global Newton).

Search Space Modifications

The first modification relates to the search algorithm stranding on a local minimum which is characterized by negative prices in some of the price vector components. For negative prices, providers will report zero supply while consumers will report their maximal demand (which is bounded by the number of jobs in the consumer queue). The search then might not be able to determine a non-zero first order derivative in order to guide the algorithm towards an equilibrium price. We artificially heighten the excess demand value for negative prices in order to deal with this issue. The more negative the price, the higher the excess demand. Since we are dealing with a minimization algorithm, this modification will guide the algorithm away from negative prices.

Secondly, an adjustment of the price vector can result in the algorithm stranding on a plateau of the excess demand surface which is characterized by *structural oversupply*. In this setting, the price vector is located in a region of the excess demand surface in which all providers offer all of their resources to the market, while consumers cannot formulate any demand because prices are too high. The determination of a non-zero first order derivative also poses a problem in this situation. To guide the algorithm back to lower prices, we have introduced a positive slope on this part of the excess demand surface. This adjustment is important in the context of grid settings as the total potential demand in the market (i.e. the total number of jobs in the consumer queues), will typically exceed the total potential supply (i.e. the total number of CPUs hosted at the different providers). As such, the adjustment of a low price level, which results in a very high excess demand, to a high price level with full oversupply is often a (significant) improvement.

Thirdly, due to the discrete nature of the excess demand function, the search space is not continuous. We have therefore allowed providers and consumers to express their demand and supply in fractional units during the price formation phase. As a result, the global excess demand function will be smoothed and the partial derivatives guiding the search will deliver more accurate information. During the scheduling phase, a provider will round its fractional supply to the nearest

integral value. For consumers, fractional demands are introduced through an even distribution of the budget that remains after formulating integral demands over the different commodities. This leftover budget is then used to express an additional fractional demand for each commodity. The reported demand will be rounded down to the nearest integral value during the job scheduling phase.

Search Algorithm Extensions

The modifications to the search space presented in the previous section, do not guarantee that the original algorithm finds an equilibrium price. Starting from Smale's algorithm, we have incorporated three extensions in order to further improve its performance for search spaces generated by the commodity market.

The first of these involves an additional rule to deal with negative prices. We have found that the practical implementation of Smale's method as presented in [203], can return negative prices even with the search space modifications mentioned previously. Whenever a price vector is suggested as an update to $\mathbf{p}$ that contains a negative price component, this price component is rejected and the previous one is re-used.

A second scenario involves the algorithm stranding on a ridge, meaning that a change in one of the price vector's components does not trigger a change in excess demand. In this situation, we remove all 0-columns and corresponding rows from $D\xi(\mathbf{p})$ and as a consequence, the price for these goods will not change.

Finally, we have observed that if the step size Δp is fixed to 1, as is the case in [104], the algorithm can get stuck in a local minimum. In order to alleviate this situation, we have introduced a variable step size. If the algorithm detects the presence of a local minimum it will increase the step size in order to step out of the local minimum and proceed towards the global minimum.

5.4.3 Implementation of the Algorithm

The activity diagram for the implementation of the ESGN algorithm is shown in Figure 5.3. We use the familiar Euclidian vector norm for $|\xi(\mathbf{p})|$:

$$|\xi(\mathbf{p})| = \sqrt{\sum_{i=1}^{n} \xi(p_i)^2} = \sqrt{\sum_{i=1}^{n} (D_i - S_i)^2} \tag{5.8}$$

with n the number of different commodities, and for a given price, S_i the supply of resource i and D_i the demand for resource i. A norm greater than 0 points to an inefficiency in the sense that the suggested market price $\mathbf{p}$, does not lead to a market equilibrium wherein goods are allocated to consumers that value them the most.

The algorithm starts by assigning the previous price vector $\mathbf{p_0}$ to the current price vector $\mathbf{p}$. The recursion depth RD, the price step PS and the step size SS are all initialized to 0. The base excess demand norm n_0 is calculated using the base price vector $\mathbf{p_0}$. A new price vector $\mathbf{p'}$ will only be accepted if its norm is better than n_0. Finally, Δp is initialized to 2^{SS}, which comes down to 1.

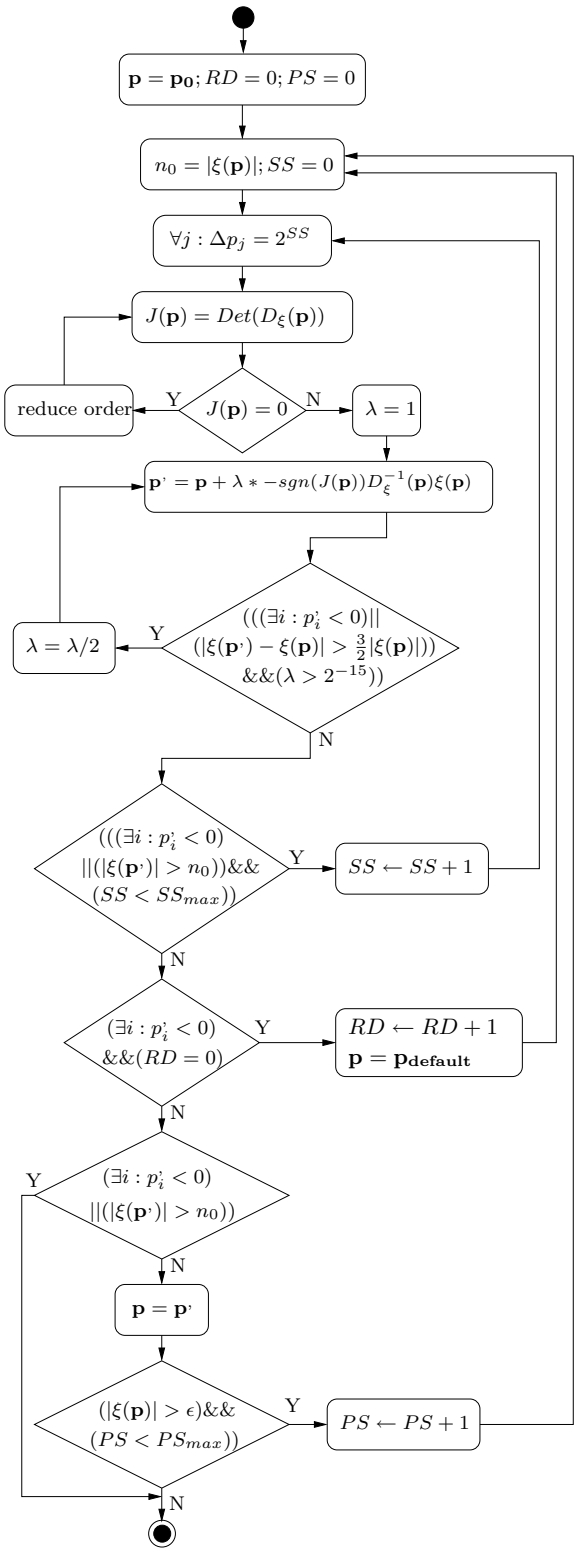

Figure 5.3: The Pricing Algorithm

Next, $J(\mathbf{p})$, the determinant of $D\xi(\mathbf{p})$ is calculated. If it equals 0, $\mathbf{p}$ is located on a ridge and an order reduction is necessary. Otherwise, λ, a factor which influences the extent of the price step, is set to 1 and the new price vector $\mathbf{p}'$ is calculated using the algorithm described in [203]. If $\mathbf{p}'$ contains a negative price or the change in the norm of the excess demand is too big, the value of λ is divided by two and the price setting part of the algorithm is restarted. This iterates until all prices are positive and the change in $|\xi(\mathbf{p})|$ is within range or until λ is so small that the magnitude of the price changes would fall below machine precision.

If there are still negative prices or the new value for $|\xi(\mathbf{p})|$ is greater than n_0, and SS did not reach its maximum value yet, SS is increased by one and the algorithm is iterated. Currently, SS_{max} is limited to 10, which results in a maximal price step of 1024. If no better excess demand norm can be found within this range, the algorithm assumes it has attained the global minimum.

If at this point, there are still negative prices, the algorithm is restarted from its initial price vector $p_{default}$, which lies at the boundary of the price domain. This recursion is allowed only once to prevent an infinite loop. We have observed this phenomenon in situations where (1) prices are very close to 0 because there was a long period of over supply, (2) there is a sudden and strong spike in demand due to a job peak, and (3) the demand for one good is much higher than for the others.

If the price vector still contains negative prices or it leads to a value for $|\xi(\mathbf{p})|$ that is greater than n_0, the algorithm ends by reusing the old price vector. On the other hand, if the new price vector results in a better excess demand norm, it will become the current price vector. Now, one step in finding the minimum is finished. The algorithm starts over using this price vector as starting point. This continues until $|\xi(\mathbf{p})|$ is less than ϵ or the algorithm has iterated more than PS_{max} times.

5.5 Operational overview

The diagram in Figure 5.4 shows a schematic overview of the operation of the commodity market organization. The spot market first instructs its equilibrium pricing component to establish a new price vector for the market (step 1). The pricing component queries the excess demand provider to deliver the excess demand for the current price level (step 2). This involves queries on both consumer and provider bidding agents. Depending on the actual implementation of the mechanism, these queries could involve a network message or a local method call, as we will see in section 5.10.4. It then iteratively adjusts the price and queries the resulting excess demand level with the excess demand provider, to find the equilibrium price. Note that the excess demand level is based on the unfulfilled demand and available supply as there is no renegotiation for resources that are already allocated, i.e. jobs that are already allocated to resources do not contribute to the demand and only available CPU resources are accounted for in the supply. Likewise, providers only supply available resources to the market. As soon as the new equilibrium price vector has been determined, a ticketing service in the market queries the excess de-

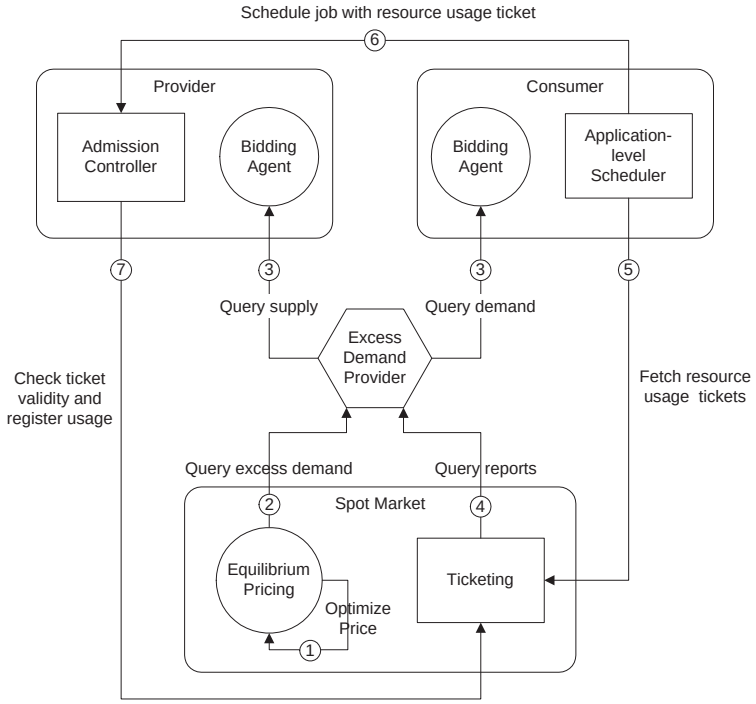

Figure 5.4: Operational overview of the commodity market

mand provider on the reported demand and supply levels of the individual bidding agents (step 4) to obtain the necessary information for usage ticket accounting. The consumer's application-level scheduler which takes care of scheduling the individual jobs of the consumer's application with providers on the grid, subsequently fetches resource usage tickets from the ticketing service (step 5).

The scheduler presents these tickets to the provider upon the transmission of an execution request for a job (step 6). An admission controller then checks the validity of the tickets with the market's ticketing service in order to make sure that the tickets are still valid for use (step 7). If they are still valid, the tickets are marked as being used in the ticketing service.

We note that our organization separates the price determination for tickets with the actual application-level planning and scheduling. In a commodity market model such as the one presented here, the commoditization of resources inherently implies the loss of detailed information on the goods that are traded such as available memory and network locality. This however, does not imply that the resulting resource allocations cannot be optimized with respect to these resource attributes. The application-level schedulers can perform the necessary checks to ensure that jobs are deployed on resources with sufficient memory and disk space, and that optimizations with respect to data locality are performed. To this end, they can

obtain the necessary information from the grid's information and data management systems.

In terms of the frequency of price adjustment, a number of strategies can be adopted. Although a fixed period could be used (e.g. every 5 minutes), a reactive approach can improve the responsiveness of the market and save on price determination overhead. In this setting, the market keeps track of the number consumers and providers that are awaiting the initiation of a new pricing round so they can buy or sell additional tickets. A well-chosen threshold value on this metric can then trigger the price formation process at appropriate timings. Note that if providers are required to wait for a significant amount of time before a new price formation process is started, this can induce an inefficiency as CPU resources are non-storeable commodities. Therefore, the selection of the threshold value involves a trade-off between having sufficient competition in the market which is required to obtain meaningful prices, and limiting the adverse effects of delayed resource pricing and trading on system utilization. In settings wherein providers host a significant amount of resources, local resource management systems can use queues and job runtime estimation techniques to counter such inefficiencies. In order to deliver the promised amount of performance, virtualization techniques can also be used that increase the amount of processing power dedicated to a job so that the waiting time in the queue does not impact the overall quality of service level that is received by the user.

5.6 Simulation Process

We have implemented the market model discussed in the previous sections in GES. This enables us to efficiently study the dynamics of commodity markets and the state of the grid environment in a controlled manner and under differing factors such as market size, resource categorizations, budget allocations and participant strategies. Each simulation step in the commodity market consists of the following actions:

1. The budget of each consumer is updated. If this is the first step in the simulation or if a consumer's allowance period $AllowancePeriod$ is reached, the budget is reset to the allowance a. Both parameters are bounded random numbers.

2. Each consumer's job queue is updated. As we are interested in the effects of demand shocks in the market, we associate a job peak period job_{peak} with a consumer. For all consumers that have reached the start of such a period, a configurable number of jobs is added to the consumer's job queue. For all other consumers, a job is added to the queue according to the consumer's job creation probability. The job peak period, the number of jobs added and the job creation probability are bounded random numbers.

3. If there are non-occupied resources in the market, equilibrium price levels are established for these resources.

4. Actual trading occurs: in random order, every consumer allocates resources on the market at the price level determined in the previous step.

5. Consumers are charged for all of their allocated jobs and associated revenue is transferred to the provider accounts.

6. The remaining runtime of all active jobs is adjusted and all finished jobs are removed from the system. The availability state of the provider's resources is updated.

When an equilibrium price is not found by our search algorithm in an overdemand market situation, allocation is performed on a first-come-first-serve basis. Therefore, it is important that we randomly iterate over the consumer population in step 4, so no consumer is favored when resource allocation takes place in such a situation.

5.7 Algorithm Evaluation

The evaluation of the pricing algorithm can be broken down into two parts. The stability and efficiency of the algorithm itself, discussed in this section, and the fairness and correctness of the market mechanism, discussed in section 5.8.

To test the stability and efficiency of the algorithm we have constructed a simulation scenario with elements that require significant price adjustments by the algorithm through the introduction of demand shocks. The simulation parameters are presented in Table 5.1. There are 100 consumers, each submitting new jobs in accordance with their job creation probability. Additionally, they submit a random number of jobs at the start of their job_{peak}, which corresponds to every 100 time steps. We have opted to make each consumer's job peak period coincide, so that the market instability is maximal at these steps. Each consumer receives a random allowance every 500 steps. The allowance period is a multiple of the peak period in order to show the interplay between them. Jobs are characterized by a random normalized runtime. There are 50 providers in the simulation, each hosting a random number of `fast` CPUs and `slow` CPUs. The simulation is run for 2000 steps. The price evolution in this simulation is shown in Figure 5.5. The dotted vertical lines represent the peak steps. One can see that prices rise sharply whenever a demand peak has occurred. Furthermore, the price of a `slow` CPU follows the same trend as the price of a `fast` CPU, but is always about 65% lower. This is due to the fact that, on average, consumers prefer to buy `fast` CPUs as long as they aren't three times more expensive than a `slow` CPU. This behavior follows from `fast` CPUs having a performance ratio that is twice as large as a `slow` CPU, while the average valuation factor for a `fast` CPU over the consumer population is 1.5.

As the demand diminishes, prices drop. They do not drop suddenly because of the large value of the elasticity attribute E (cfr. section 5.3.3) and the gradually diminishing consumer job queues. One can also observe that the consumer behavior is a very defensive one. The further it is away from the next allowance period, the

Table 5.1: Parameters common to all performed simulations

Parameter	**Value**
Simulation steps	2000
Number of consumers	100
Number of providers	50
Amount of `fast` CPUs per provider	{1,2,...,8}
Amount of `slow` CPUs per provider	{2,3,...,15}
E attribute for all providers	∞
$ValuationFactor_{\texttt{fast}}$	[1.0,2.0]
$ValuationFactor_{\texttt{slow}}$	1.0
`fast` CPU speed s_{fast}	2.0
`slow` CPU speed s_{slow}	1.0
Job length	{2,...,10}
Allowance period	500
Base Allowance	{500.000,...,1.250.000}
job_{peak}	100
Jobs submitted at job_{peak}	{1,2,...,200}
New job probability per step	10%

less money it is willing to spend. This is caused by the fact that the consumer's spending strategy does not explicitly take into account expected periods of low prices in between the job peaks. After such a period of lower demand, the consumers can therefore sustain a higher spending rate as the expected rate of expenditure, was not obtained. This effect increases as we approach the end of an allowance period, as the consumer divides the remaining budget by the number of steps until the end of the allowance period, to determine its spending rate. We would like to note that in periods of excess demand, more than 600 resources are traded per simulated time step.

The stability and efficiency of the pricing algorithm can be analyzed by looking at the absolute excess demand norm $|\xi(\mathbf{p})|$, found through minimization. Figure 5.6 shows this norm for each simulation step and its relative frequency distribution over the whole simulation. Table 5.2 lists some key statistical figures. From this we conclude that in general the value of $|\xi(\mathbf{p})|$ is low; in more than 50% of all steps a price vector is found which leads to an excess demand norm lower than 1, and in 95% of the simulation steps, $|\xi(\mathbf{p})|$ was less than 5.13. The spikes in the norm of the excess demand occur at the steps where the price drop ends and a new (approximate) market equilibrium is found. After a couple of steps, the market stabilizes and $|\xi(\mathbf{p})|$ falls back to a near 0 level.

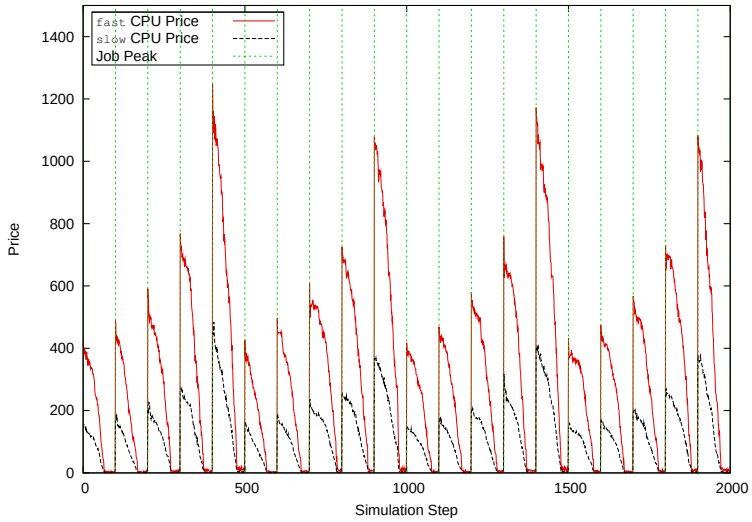

Figure 5.5: Price evolution

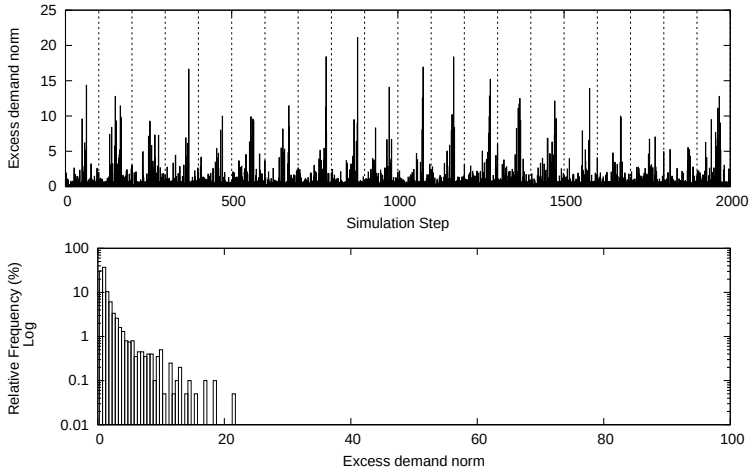

Figure 5.6: Analysis of the norm of the excess demand ($|\xi(\mathbf{p})|$)

Equation 5.9 defines a second metric, which we refer to as the mismatch factor or the relative norm.

$$ED\_relNorm = \frac{\sum_{i=1}^{n} |S_i - D_i|}{\sum_{i=1}^{n} (S_i + D_i)} * 100 \tag{5.9}$$

The mismatch factor is a relative indication of the number of goods for which supply and demand did not match. It varies between 0 and 100 where 0 means a

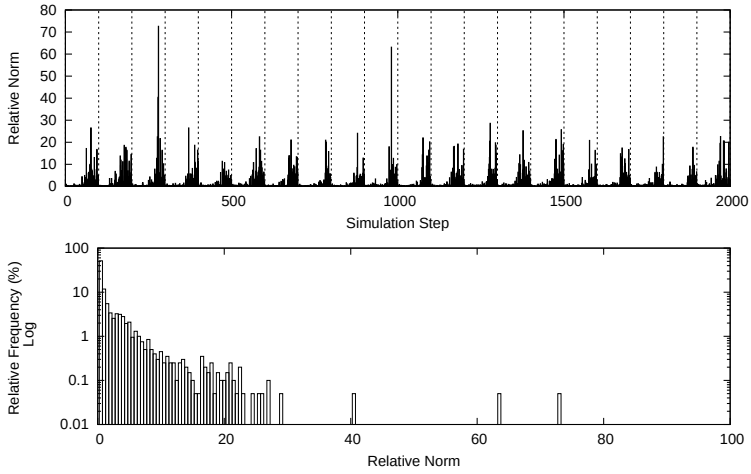

Figure 5.7: Analysis of the relative norm

Table 5.2: Statistical analysis of $|\xi(\mathbf{p})|$ and the relative norm

	$\lvert\xi(\mathbf{p})\rvert$	**Relative Norm**
min	0.00	0.00
25th percentile	0.43	0.21
50th percentile	0.68	0.46
75th percentile	1.31	2.54
90th percentile	2.87	5.82
95th percentile	5.13	9.98
max	21.12	72.6

perfect match and 100 means that supply and demand are fully unmatched. The mismatch factor allows one to assess whether the magnitude of the excess demand is high, relative to the amount of goods demanded from and supplied to the market. As such, the excess demand norm and the mismatch factor should be considered together for a complete interpretation. A peak in the norm of 25 for example, does point to an inefficiency, but if the mismatch factor indicates that this accounts for only 0.1% of the total demand and supply volume, we actually have a very good match. Conversely, a mismatch of 50% looks unacceptable, but if the norm is very small, then this indicates that, during this simulation step, there were only very few goods traded anyway.

Figure 5.7 shows the mismatch factor for each simulation step and its relative frequency distribution over the whole simulation. Key statistical figures are given in Table 5.2. From this data we can derive that in more than half of the simulation steps, the mismatch is less than 0.5%. In 90% of all cases, the mismatch is less than 5.82%.

5.8 Evaluation of the Market Mechanism

In this section, we present an evaluation of the functioning of our resource market. We focus on the following market properties:

1. Prices should be set in such a way that market equilibrium is reached.
2. The ratio between price levels for different goods should reflect the mean valuation ratio of those goods by the consumer population as a whole.
3. The resource market should lead to fair allocations.
   (a) In an oversupply situation, this means that every consumer should be able to acquire, on average, an equal share of the infrastructure if the job spawning rates over all consumers are equal.
   (b) In an overdemand scenario wherein consumers are not limited by a shortage of jobs in their queues, this means that a consumer i with budget share BS_i should be able to obtain a share of the total utility US_i with $US_i = BS_i$. In a market where consumers value the computational resources according to their nominal worth, this means that for every consumer, $BS_i = AS_i$ must hold. Where AS_i is the share of compute resources allocated by the consumer during the simulation.
4. The measured resource utilization levels should be equal or close to the maximum achievable utilization.
5. Prices should be stable in the sense that limited, short-term changes in supply and demand lead to limited, short-term price responses in the market.

In the context of point 4, note that all of our providers share the strategy of maximizing revenue instead of maximizing the utilization of their infrastructure.

Experimental data on the real-world execution of economic grid resource management systems is scarce, as such systems still need to find their way to grid middlewares that are deployed on a large scale. In the absence of such reference data for configuring our market, we investigate aforementioned properties under two distinct and typical scenarios. The first represents a market in *oversupply* which means that, on average, the total available CPU time in the market is greater than the total computational time requested by all consumer jobs. Following [104], we inject a periodical characteristic into the job flow in order to evaluate market efficiency and stability under sudden demand changes. The second scenario simulates a market in constant *overdemand*.

In order to evaluate the fairness of our market, we introduce four consumer groups with distinct allowance levels. A consumer c in allowance group AG_i is allocated a budget $B_c = A * AF_i$ with A the base allowance for all consumers and AF_i the allowance factor for AG_i. We evaluate market fairness by establishing that the average budget share of consumers in AG_i equals the average share of the CPU allocations that these consumers are able to acquire over the course of the simulation.

Table 5.3: Parameters specific to the oversupply simulations.

Parameter	Value
Simulation steps	450
$ValuationFactor_{\texttt{fast}}$	[1.0,2.0]
$ValuationFactor_{\texttt{slow}}$	1.0
AF_1	1.0
AF_2	1.5
AF_3	2.0
AF_4	2.5
Allowance period	100
Base allowance	500,000

5.8.1 Oversupply scenario

The simulation parameters specific to the oversupply scenario are given in Table 5.3, all other parameters are as in Table 5.1. For parameters that are specified with a range, we use a uniform random distribution. This scenario simulates a market with 100 consumers and 50 providers. Every provider hosts between one and eight `fast` CPUs and between two and fifteen `slow` CPUs. Jobs have a normalized runtime between 2 and 10 simulation steps and every consumer has a 10% probability of spawning a new job in a simulation step. At the job induction steps, between 1 and 200 jobs are spawned per consumer. In the presented instance of this scenario, the market contained 456 `slow` CPUs and 222 `fast` CPUs. The jobs that were spawned during the 450 simulation steps represented a total normalized compute time of 256,098 simulation steps. The infrastructure provides 900 normalized computation steps per simulation step. Therefore, the highest achievable utilization rate is 63.3%, which corresponds to an oversupply situation.

As shown in Figure 5.8, utilization levels start at 100% during the first job peak and slowly decrease as we move into oversupply. At the following job peak, the utilization levels rise again. The utilization at the end of the simulation is 61.7% for `fast` CPUs and 64.6% for `slow` CPUs. This means that our average utilization for the infrastructure is 63.15%. The market thus achieves a high level of resource utilization, even in scenarios in which the provider population strives for maximization of revenue instead of infrastructural utilization. This means that property 4 is fulfilled for this oversupply scenario. Figure 5.8 also shows that prices follow supply and demand closely. At the job induction steps, the prices of both goods are immediately adjusted to bring the market to equilibrium. After a price peak, prices decline as demand drops below the supply level due to a shortage of jobs and the market returns to the steady state in which there is oversupply. Because providers do not set minimum prices for resource usage, prices converge to zero at that point. This indicates that prices are stable in the sense that after disturbances due to the job induction, price levels gradually return to their steady state. This fulfills property 5.

As shown in Table 5.4, the median of the norm has a value of 0.64 with a 95th

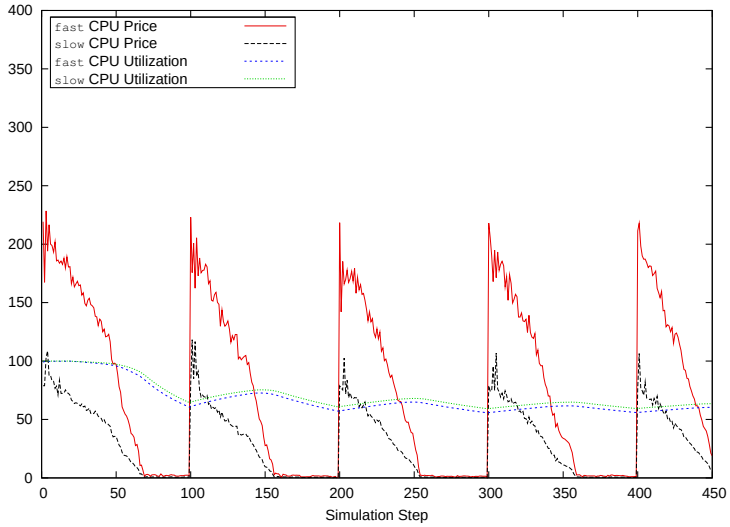

Figure 5.8: Price and utilization levels of `fast` and `slow` CPUs in the oversupply scenario.

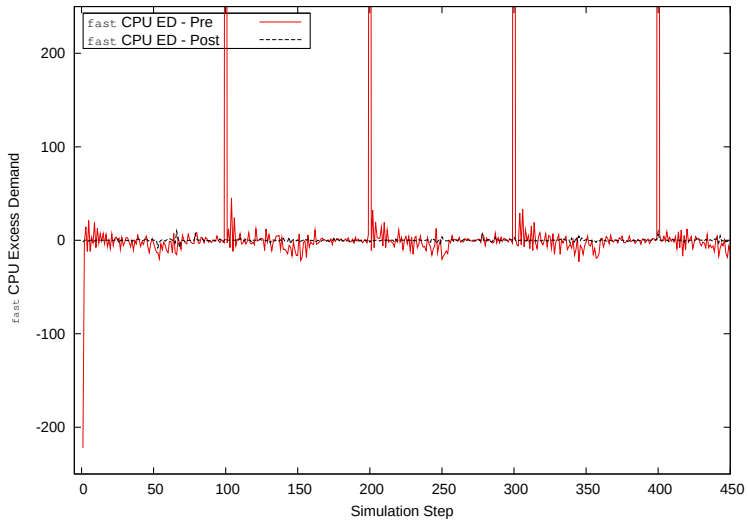

Figure 5.9: Excess demand levels in the oversupply scenario for `fast` CPUs at step i for the market prices computed at step i and step *i-1*

percentile of 5.8. The prices set by our pricing scheme thus approach the market equilibrium very closely, thereby fulfilling property 1.

The graph in Figure 5.9 shows the effect of our price alterations on the excess demand levels. The red curve displays the excess demand levels that would arise if we used the price calculated in step $i-1$, for step i. The black curve displays the excess demand levels for the prices calculated by our pricing scheme. We clearly see that the peaks in excess demand at the job induction steps are neutralized by

Table 5.4: Results from the oversupply and undersupply scenarios

	Oversupply	**Overdemand**
Median ED_Norm	0.64	0.29
95th Percentile ED_Norm	5.8	1.21
Median `fast` CPU price	78.0	217.04
Median `slow` CPU price	29.3	81.96
`fast` utilization	61.7%	99.6%
`slow` utilization	64.6%	99.7%
Maximal utilization	63.3%	100%

our price adjustments. At step zero, we bootstrapped the pricing algorithm with a default price vector of $(100000, 100000)$. As we can see from the graph, the default price vector needed correction by our pricing scheme because it would have induced a full oversupply situation, leaving all of the 222 `fast` CPUs unallocated. Note that the peaks in excess demand are not fully shown in the graph, as this would require an Y axis range of [-250, 5000].

The mean valuation factor for `fast` CPUs over the entire population is 1.5 and the performance factor is 2. Therefore, `fast` CPUs should be valued at three times the price of `slow` CPUs. Table 5.4 shows that prices approximate this relative valuation, but are not in perfect correspondence with it. This can be explained by the fact that consumers favoring `fast` CPUs for a particular price point, can still acquire `slow` CPUs with the budget they have left after buying their `fast` CPUs. The acquisition of these `slow` CPUs however, takes place regardless of their price. This elevates the aggregate demand for `slow` CPUs and their price level. If we disable this behavior in the consumer's spending logic, the mean valuation factor is correctly reflected in the price levels through a median price of 85.05 for `fast` CPUs and 28.13 for `slow` CPUs. We therefore conclude that property 2 is fulfilled.

Figure 5.10 shows the budget share (BS) and allocation share (AS) of the four consumer groups in the simulation. We notice that the allocation shares oscillate between the job induction pulses. This behavior is to be expected; a consumer group with a high budget share is only able to allocate its affordable resource share fully when enough jobs are available in the consumer queues. As the job queues shrink, prices drop and other consumer groups are able to allocate resources. As prices converge to zero, the allocation shares remain approximately constant. When a new price peak arises, the allocation shares for wealthy consumers gradually rise again at the expense of the poorer consumers. The oscillatory effects are damped by averaging the allocation shares over all simulation steps. We see that allocation shares converge to the 25% mark. This corresponds with the notion that in a market with oversupply, differences in consumer budgets do not affect their allocation share in the long term, i.e. every consumer is able to allocate an equal share of the grid's resources. This fulfills property 3a.

Although the market's behavior can be intuitively analyzed and evaluated when the demand peaks of the consumers are synchronized, such synchronization cannot

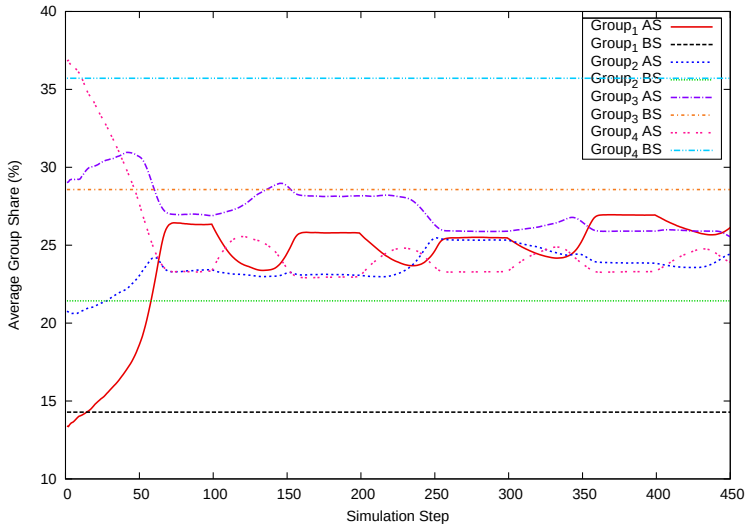

Figure 5.10: Mean budget and allocation shares per budget group for the oversupply scenario.

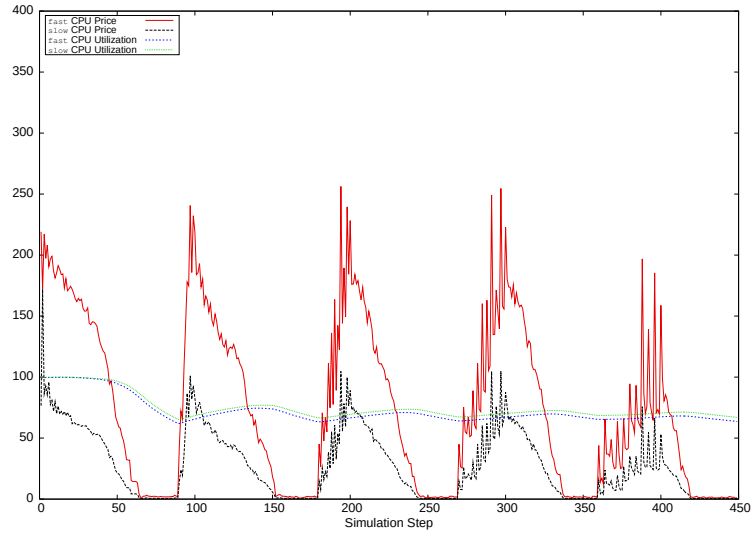

Figure 5.11: Price and utilization of `fast` and `slow` CPUs in the oversupply scenario with non-synced demand peaks.

be expected in a real-world scenario. Figure 5.11 shows the evolution of the market prices when the demand peak period for each consumer is uniformly distributed between 90 and 100 simulation steps. As the simulation advances, the demand spikes of the consumers drift further apart from each other, resulting in price peaks earlier on in each 100 time step period.

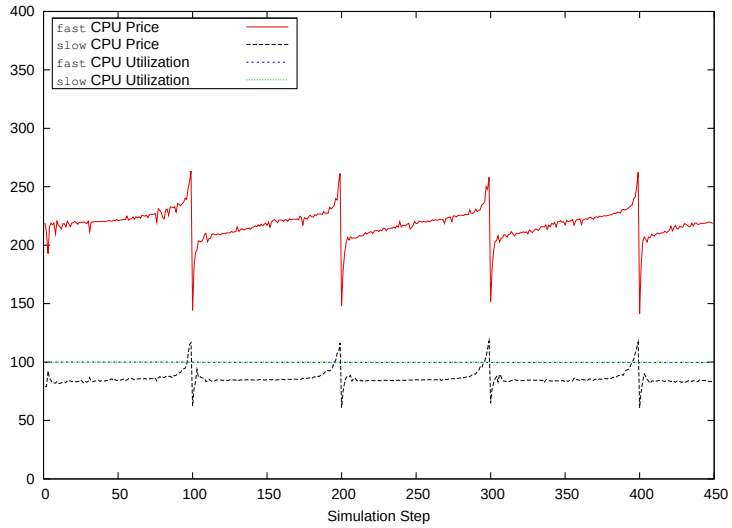

Figure 5.12: Price and utilization levels for `fast` and `slow` CPUs in the overdemand scenario with allowance periods of 100 steps.

5.8.2 Overdemand scenario

For the overdemand scenario, we keep the number of jobs in the consumer queues at the level obtained during the first job induction at simulation step zero. All other parameters are equal to those of the oversupply scenario.

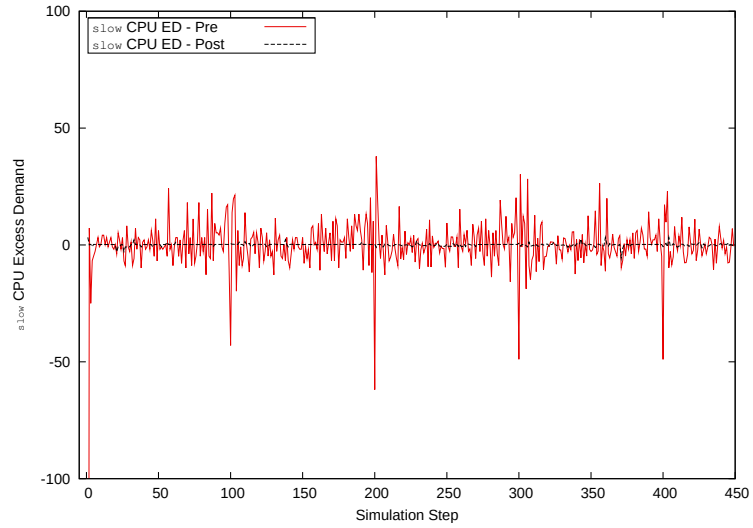

Figure 5.13: Excess demand levels in the overdemand scenario for `slow` CPUs at step i for the market prices computed at step i and step *i-1*

Figure 5.12 shows the price and utilization levels for `fast` and `slow` CPUs in the overdemand scenario. Although one might expect prices to remain constant,

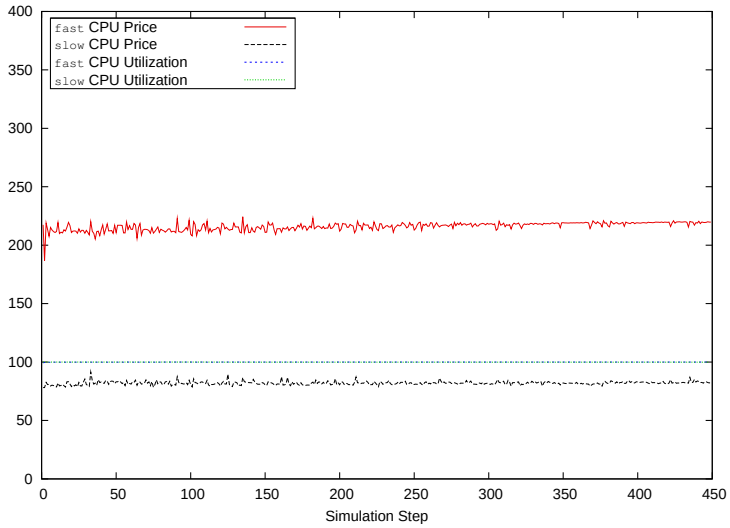

Figure 5.14: Price and utilization levels for `fast` and `slow` CPUs in the overdemand scenario with allowance periods of 1000 steps.

they rise as we approach the allowance replenishment steps (at intervals of 100). This can be explained by the fact that consumers always have an expenditure rate within the limits of their allowed spending rate per simulation step. Because of the unit price per CPU, consumers are not generally able to spend all of their budget. This way consumers accumulate budget leftovers from previous steps within one allowance period. Because the available spending rate for a particular simulation step ($rate_{available}$) is calculated in function of the remaining steps until the next allowance period, higher spending rates are possible near the end of the period. As we reach the allowance replenishment step, all leftovers are discarded and prices drop, after which this price evolution repeats.

The graph shows that our utilization level remains constant at approximately 100%. The mean utilization level for `fast` CPUs is 99.6% and 99.7% for `slow` CPUs. Property 4 is thus fulfilled in the overdemand scenario.

The data in Table 5.4 shows that we achieve price correctness with the median of the norm at 0.29 and a 95th percentile of 1.21, thereby fulfilling property 1.

Although supply and demand remain fairly constant in the overdemand scenario, Figure 5.13 demonstrates that it is still crucial to perform small price adjustments in order to keep excess demand levels low. This time we show the excess demand levels for the `slow` CPU category, when using the price level of the previous simulation step (red curve) and when using the adjusted prices (black curve) for the current simulation step.

If we increase the allowance period to 1000 simulation steps and compensate with a tenfold increase of allowance for all consumers, we obtain stable and fairly constant price levels, fulfilling property 5. This is shown in Figure 5.14.

Figures 5.12 and 5.14, as well as Table 5.4 show that the price levels of both CPU categories approximate the relative valuation of these types by the consumer

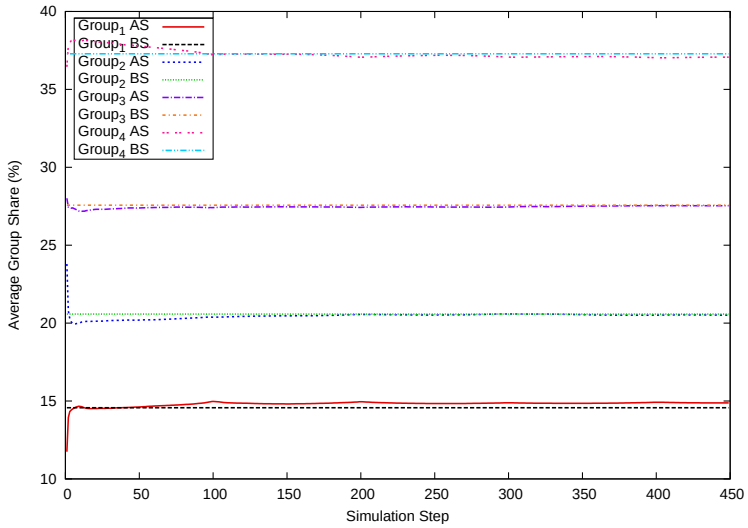

Figure 5.15: Mean budget and allocation shares per budget group for the overdemand scenario.

population. The relation does not perfectly reflect the valuation factor of 3, and the same explanation holds as in the oversupply scenario.

In the overdemand scenario, all consumers are able to fully exert their budgetary capabilities to allocate their share of the grid infrastructure. As shown in the graph in Figure 5.15, the mean allocation share for a particular budget group, corresponds to its budget share. This fulfills property 3b.

Two important premises exist for obtaining this fair allocation, that are outside the control of our pricing scheme and market operation. The first is that a consumer's spending rate should not be limited by a shortage of jobs in the consumer queue. Because we keep the number of jobs in the consumer queues constant at a high enough level, this premise is fulfilled. The second premise is that consumers have to value the CPU categories according to their real nominal worth. This is because we compare the budget share to the share of allocated resources, and not to the share of total perceived utility. Note that in the context of an individual consumer, this premise does not necessarily hold as the consumer may add extra relative value to a CPU category through its $ValuationFactor$. However, because the valuation factors are uniformly distributed over the consumer population, the premise does hold when we approach the fairness issue from the perspective of the individual budget groups.

5.9 Equilibrium pricing with multiple CPU categories

As demonstrated by sections 5.7 and 5.8, the extension of Smale's algorithm performs well for markets with two substitutable commodities. When we scale up the number

Table 5.5: Mean, 95% confidence interval and maximum of the ED_Norm for the ESGN algorithm, the pattern search algorithm, and the combination of both algorithms, for different numbers of CPU categories in the commodity market (n).

n	**ESGN**			**ESGN + Pattern Search**			**Pattern Search**		
	Mean	95% CI	Max	Mean	95% CI	Max	Mean	95% CI	Max
1	0.21	[0.15,0.26]	2	0.21	[0.15,0.26]	2	0.15	[0.10,0.20]	2
2	3.92	[3.37,4.47]	23.22	3.49	[2.93, 4.05]	19.39	103.83	[52.01,155.64]	2592.66
4	14.33	[8.64,20.01]	369.93	6.96	[6.3, 7.62]	26.4	351.5	[268.05,434.95]	2900.0
6	49.17	[26.57,71.76]	1114.15	10.66	[9.76, 11.56]	34.14	367.6	[260.9,488.31]	6397.07
8	75.31	[46.48, 104.15]	2023.9	32.11	[27.63, 36.59]	189.46	1966.93	[1716.75,2217.11]	7717.12

of commodities, we found that the algorithm's performance cannot maintain the same level of price correctness.

In order to further improve the algorithm's performance in situations with more goods, we have integrated a pattern search algorithm for nonlinearly constrained minimization problems with simple bounds [204]. This algorithm is tailored towards optimization problems with nonlinear objective functions or constraints. It does not rely on accurate derivatives and is therefore more robust with respect to discontinuities in the excess demand function.

The resulting optimization process first invokes the ESGN algorithm of section 5.4.3. When ESGN does not arrive at an equilibrium price, and the associated ED_Norm exceeds a preconfigured threshold, the pattern search algorithm is invoked to further improve the ED_Norm. We allow this optimization process to iterate, as long as constraints are fulfilled with respect to:

- The absolute value of the norm associated with the current price vector
- The number of times the optimization process has already iterated
- The size of the improvement made by invoking the pattern search algorithm

The threshold values for these constraints can be configured on a case by case basis.

We have used the implementation of the algorithm presented in [204] that is delivered by the Matlab Genetic Algorithm and Direct Search Toolbox (the `patternsearch` routine). An overview of the integration of Matlab optimization routines within GES is presented in section 4.6.3.

Table 5.5 shows the performance of the combined algorithm in function of the number of CPU categories in the commodity market, compared to the ESGN algorithm and the pattern search algorithm. The data is shown for the market scenario given in Table 5.6. We spread out the total processing power in the system evenly across all providers and CPU categories. Categories were configured with linearly increasing speed factors, i.e. the third category had a speed of 3. The ED_Norm threshold used for invoking the pattern search algorithm after ESGN was set to 8. The constraints for allowing an iteration of the optimization process were:

- ED_Norm < 80
- Number of iterations < 10
- ED_Norm improvement > 1

Table 5.6: Simulation parameters

Parameter	Value
Number of consumers	100
Number of providers	50
Total normalized processing power	2000
Valuation factors	[1.0,1.5]
Job runtime in time steps	$\{4, 5, \cdots, 8\}$
Jobs injected at peak	$\{50, 51, \cdots, 200\}$
step_peak	40
Base Allowance	100,0000 * [1.0,1.5]
$\{AF_1, AF_2, AF_3\}$	{1.0, 2.0, 3.0}
Allowance period in time steps	800

Table 5.7: Average number of excess demand queries and runtime per simulation step for the ESGN algorithm, the pattern search algorithm, and a combination of both algorithms, for different numbers of CPU categories in the commodity market.

n	**ESGN**		**ESGN + Pattern**		**Pattern**	
	Queries	Runtime (ms)	Queries	Runtime (ms)	Queries	Runtime (ms)
1	59.73	3	59.73	3	91.84	79
2	149.2	32	213.79	66	178.09	141
4	302.83	115	621.61	383	345.61	314
6	392.2	228	1286.8	1098	495	558
8	587.69	468	2281.96	2466	556.17	1205

Depending on these settings, a tradeoff can be made between price correctness and runtime performance of the equilibrium pricing algorithm.

As Table 5.5 illustrates, the ED_Norm obtained by the ESGN algorithm increases as we introduce more CPU categories in the market. This can be attributed to the fact that each additional CPU category increases the dimensionality of the equilibrium price optimization problem. This increase in dimensionality makes it more difficult for the algorithm to obtain a price level that brings the market (sufficiently close) to equilibrium.

Although the table shows that the pattern search algorithm does not deliver better performance compared to the ESGN algorithm by itself, the combination of both can significantly improve the norm of the excess demand for markets with more than two CPU categories. Table 5.5 shows a reduction of the average ED_Norm with more than 75% for a market with six CPU categories. As indicated by the confidence intervals, the combined algorithm results in less fluctuation of the ED_Norm over the different time steps in the simulation. This is also illustrated by the large reduction in the maximum value of the ED_Norm that is recorded over the entire simulation. Note that there is no difference between the results for the combined algorithm and ESGN in the case of a single CPU commodity. The pattern search algorithm is not invoked in this setting as the ED_Norms obtained by ESGN do not exceed the threshold value of 8.

However, these improvements come at the cost of more evaluations of the aggregate excess demand function as shown in Table 5.7. The data in Table 5.7 was generated on a desktop with an Intel Quad Core processor running at 2.83 GHz, and 8GB of RAM. As the table shows, the number of function evaluations per simulation step does not scale linearly with the number of commodities in the market for the combined algorithm. This also translates into an increase in the average time required to obtain a market price in each simulation step.

In the remainder of this work we will use the combination of the ESGN algorithm with the `patternsearch` routine to obtain equilibrium prices. The resulting price optimization process has also been used by other authors. Their studies establish that it also operates well in face of vast and dynamic user and provider populations [205].

5.10 Commodity Markets versus Single-Unit Vickrey Auctions

In this section, we compare the performance of the commodity market with a market organization employing single-unit Vickrey auctions. The price formation process is fundamentally different in both organizations. In the commodity market one performs global optimization to establish an equilibrium price and, for this purpose, polls all market participants for their supply and demand levels at a particular price. Participants are required to communicate these supply and demand levels to a central process performing the optimization. The auction market organization, on the other hand, is fully decentralized and lets prices emerge from the local interactions of the market participants in single-unit Vickrey auctions. The goal of this section is to investigate whether these two approaches lead to different outcomes in terms of established prices, fairness of allocations and communicative and computational requirements for establishing these allocations.

Limited work has been done on directly comparing both organizations on these grounds. The study in [7] compares them with respect to price stability and infrastructural utilization. The authors postulate that "auctioneering is attractive from an implementation point of view but that it does not produce stable pricing or market equilibrium, and that a commodity market performs better from the standpoint of a grid as a whole". Similar remarks on the stability of prices in a single-unit Vickrey auction market are made in [167].

For the purpose of this study we resort to simulation in order to efficiently analyse both market organizations on a large scale. Therefore, modeling decisions have to be made concerning the type of grid resources that are simulated and the behavior of the market's participants. The model adopted here is similar to the one described in section 5.3.

5.10.1 Consumer Model

Commodity Market

For the commodity market, we use the same consumer model as the one discussed in section 5.3.2.

Auction Market

For the auction market, consumers have to decide on the amount of credits they are prepared to bid for a particular CPU, the amount of CPUs they will bid on, and the specific auctions they will participate in. They calculate a base level for their bids which depends on their remaining budget and adjust this bid with the characteristics of the CPU resource as shown in equation 5.10.

$$Bid_{cpu} = (Base\_Bid * PerformanceRatio_{cpu}) * ValuationFactor_{cpu} \tag{5.10}$$

The calculation for the base bid level also takes into account a target *parallelization degree d* the consumer wishes to realize, which denotes the amount of jobs the consumer wants to run in parallel. Consumers thereby adopt the strategy of achieving the highest possible degree of parallelization, within their budgetary limitations. In order for the consumer to learn a parallelization degree which is realistic given the current market conditions and its personal budgetary endowment, we use a reinforcement learning approach. At the start of the simulation, all consumers try to launch all of their available jobs in parallel and determine their bids accordingly. After a trading round, consumers will receive feedback from the environment through the amount of bids that won. The consumer then considers the number of jobs currently running and tries to improve on its current parallelization degree using equation 5.11.

$$d_{new} = min(n_{jobs}, (d_{current} + 1) * f_{aggressive}) \tag{5.11}$$

with n_{jobs} the jobs remaining in the consumer's job queue, $d_{current}$ the current parallelization degree (i.e. the total number of jobs that are currently running), and $f_{aggressive}$, a factor which determines how aggressive the consumer tries to be in increasing its degree. In the following we have set $f_{aggressive} = 2$.

Consumers adopt the simple heuristic of participating in the auctions which currently host the lowest number of bidders, in order to optimize their chances of being confronted with limited bidding competition.

5.10.2 Provider Model

For the analysis presented in this section, providers will not set minimum prices for their resources and will supply all of their available resources to the market.

5.10.3 Market pricing

Commodity Market

In the commodity market, prices for the different CPU categories are dynamically set in every simulation step by an optimizer which adjusts the price in order to

bring the market to equilibrium. For the results presented in this section, we have used the combination of the algorithm presented in section 5.4 with a pattern search algorithm [204], as outlined in section 5.9.

Auction Market
In the auction market, each provider hosts a number of single-unit Vickrey auctions, one for each CPU that is available at that point in time. All CPU resources are thus individually auctioned which allows for a more accurate and diverse valuation by consumers. This is an advantage over the more abstract resource model used in the commodity market. Nevertheless, we will adopt the same single-attribute characterization from the commodity market in the form of the performance ratio, in order to be able to compare both approaches.

Consumers submit their sealed bids to the auctioneers of the CPUs they are interested in. The Vickrey auction allocates the CPU to the consumer with the highest bid, at the transaction price of the second highest bid (or zero if there is only one bidder). The fact that the consumer's transaction price does not depend on its own bid forms the basis for the *incentive compatibility* of the Vickrey auction. This means that a consumer has no incentive to place a bid which differs from its true value for the CPU, because no strategic advantage can be gained from this act.

5.10.4 Simulation and Analysis

The parameters of the scenario that we will use as the basis for our analysis are shown in Table 5.8. For parameters that are specified with a range, we draw values from a uniform random distribution. Three groups of consumers with different budget levels are created by multiplying a consumer's base allowance with the respective *allowance factor* AF_i of its group. Note that in the context of this analysis, we keep the number of jobs in the consumer queues constant at the initial level by reinjecting a new job in the consumer's queue for every finished job. This results in a stable demand level which should lead to stable market prices.

We simulate the scenario using GES with a discrete-time core. For the commodity market, we establish new equilibrium prices in each simulated time step. In the auction market, auctions are also held every simulated time step for all resources that are available in that step.

Dynamic Pricing
As shown in Figure 5.16, the average prices paid by the consumers for the different categories of resources are similar in both markets. The auction market shows a higher fluctuation in the price levels over the course of the simulation with a relative standard deviation [206] of 5.86% versus 1.62% for the commodity market. The deviation for the auction market prices does not include the unstable price levels of the first 10 steps, if these are included the deviation increases to 9.62%. Whereas the commodity market's pricing mechanism immediately brings the market to equilibrium through global optimization, the participants in the auctions still have to optimize their *target parallelization degree* and discover the amount of resources to bid for. This results in the extensive adjustments of the average CPU price paid

Table 5.8: Simulation parameters

Parameter	Value
Number of consumers	100
Number of providers	50
Number of `fast` CPUs per provider	$\{1, 2, \cdots, 7\}$
Number of `medium` CPUs per provider	$\{3, 4, \cdots, 11\}$
Number of `slow` CPUs per provider	$\{9, 10, \cdots, 17\}$
s_{fast}	3.0
s_{medium}	2.0
s_{slow}	1.0
$ValuationFactor_{fast}$	[1.0,1.5]
$ValuationFactor_{medium}$	[1.0,1.5]
$ValuationFactor_{slow}$	1.0
Job runtime in time steps	$\{4, 5, \cdots, 8\}$
Number of jobs per consumer (constant)	$\{150, 151, \cdots, 500\}$
Base Allowance	1.0e6 * [1.0,1.5]
$\{AF_1, AF_2, AF_3\}$	{1.0, 2.0, 3.0}
Allowance period in time steps	800

at the beginning of the simulation.

Although prices are slightly less stable in the auction market, they do follow the trend of supply and demand in the simulated environment as shown in Figure 5.17. This scenario is similar to the one used in section 5.8.1. Periods of overdemand are followed by periods of oversupply as peaks of job arrivals occur at intervals of 45 simulation steps. In addition, jobs are not reinjected in the consumer queues on completion. The study presented by Wolski et al. [7] has shown that such a scenario leads to very erratic pricing behavior for an auction market. These results are shown in Figure 5.18 for reference. Whereas the graph in Figure 5.18 indeed exhibits price levels that do not reflect the overall supply and demand trends in the market, our tests indicate very different results. Although some aspects of the simulation differ, one of those being the fact that consumers have to coallocate disk and CPU resources in [7], our results do show that it is *possible*, using fairly simple bidding logic, to obtain meaningful and fairly stable average prices in an auction market organization.

We note that the two price peaks for the `slow` CPU category in the commodity market scenario are caused by the fact that no resources are available for trade in that category at these time instances. The equilibrium optimizer generates a high price level for these resources in order to remove all demand for them in the market and minimize excess demand. As such, no resources are actually traded at these prices.

We note that the average transaction price for resources is lower in the auction market. The difference in total revenue generated for the providers is 15.79%. We have verified that this difference is caused by the fact that the consumers' aggressiveness factors are relatively high which results in consumers bidding at lower

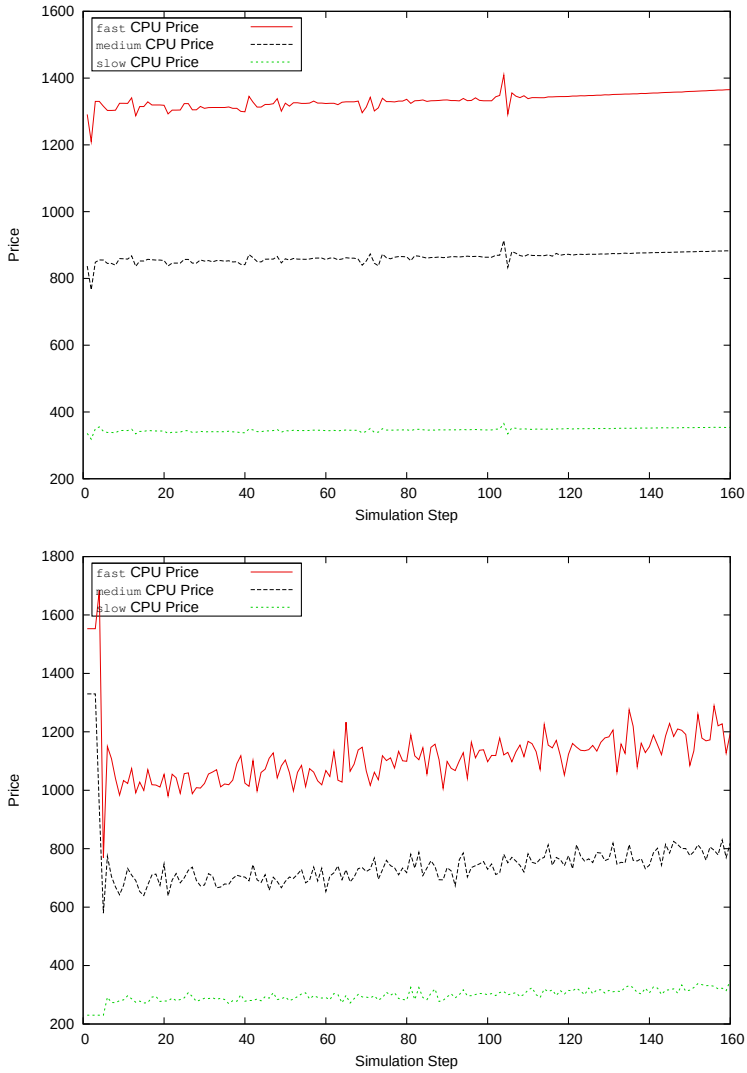

Figure 5.16: CPU Prices for the commodity market (top) and auction market (bottom) under the constant load scenario

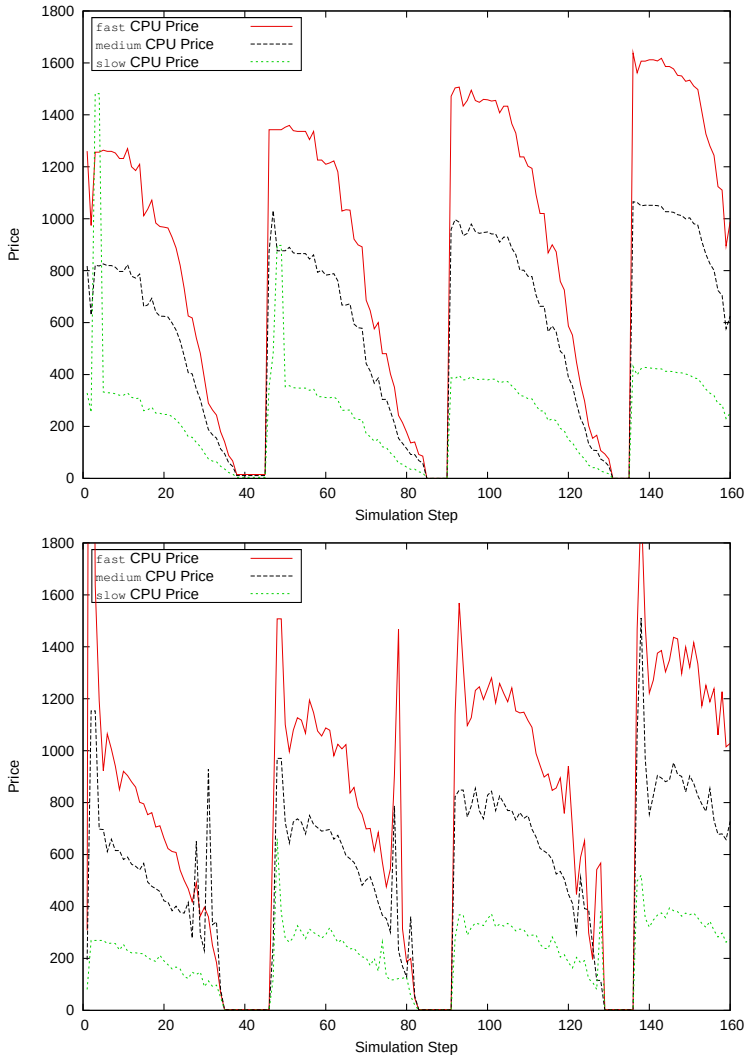

Figure 5.17: CPU Prices in the varying load scenario for the commodity market (top) and auction market (bottom)

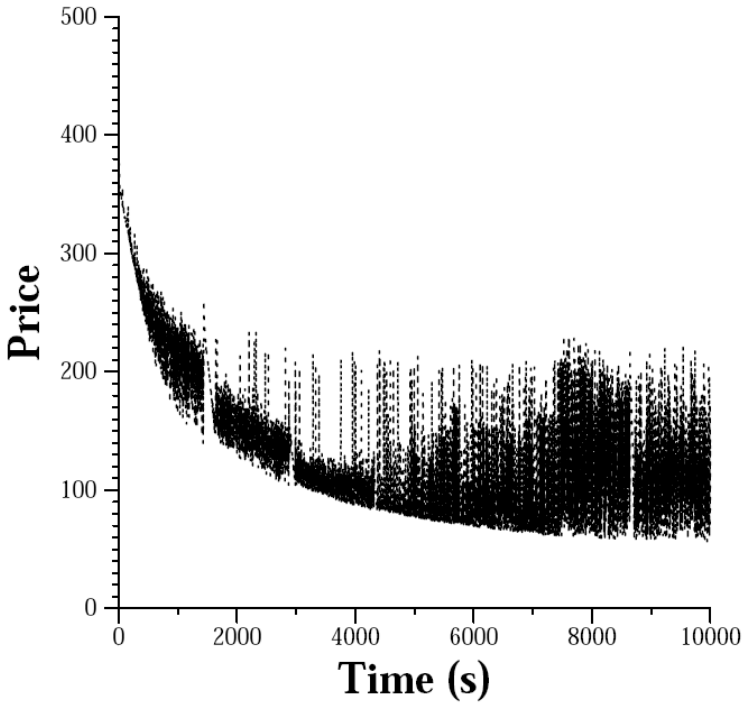

Figure 5.18: Results for a periodic overdemand scenario under an single-unit auction environment as presented in [7]

prices than the equilibrium prices found in the commodity market. If we reduce the aggressiveness factor to a randomly distributed value between 1.0 and 1.5, the difference decreases to 4%.

Fairness

The fairness of the allocations in an economic resource management system, denotes whether the level of budgetary endowment of a consumer is correctly translated into a corresponding share of the infrastructure allocated to that consumer. The graphs in Figure 5.19 show that in both market organizations the average allocation shares of the three consumer groups converge to the budget shares of those groups. In the auction market, correspondence is not achieved in the first simulation steps. This can be explained by the fact that consumers are still learning the parallelization degree that is obtainable by them in the current market situation. The commodity market does not require such judgment from its consumers and immediately brings about fair allocations.

To investigate whether shares quickly adapt to sudden changes in the market, we swap the budget levels of the different consumer groups at step 80. As shown in Figure 5.20, both market organizations are able to quickly adapt the allocations to reflect the new market situation. Instead of the cumulative average, Figure 5.20 shows the instantaneous shares at each simulation step, which are somewhat less stable for the auction market scenario.

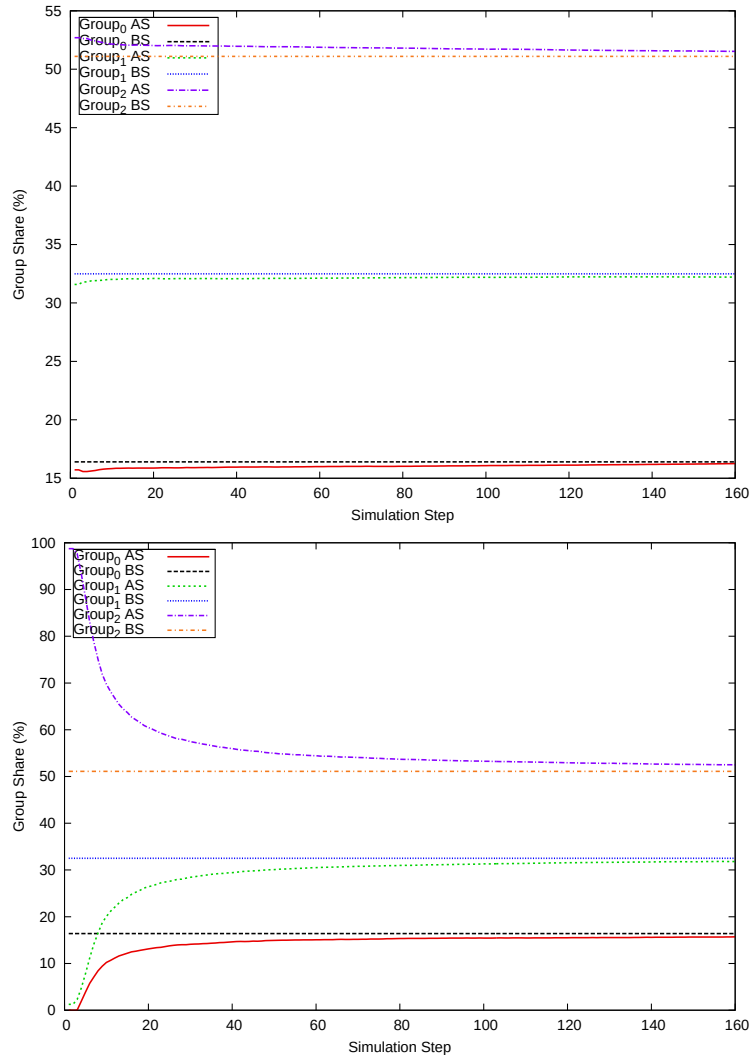

Figure 5.19: Budget shares (BS) and allocation shares (AS) as a cumulative average over the simulation for the commodity market (top) and auction market (bottom)

From the providers' point of view, a fair market operation should imply similar revenues for providers selling similar amounts of CPU resources. In the commodity market, we measured an average relative standard deviation of 2.09% on the normalized transaction price paid per CPU cycle in a single time step. This price is obtained by dividing the earned revenue on a set of CPUs by the performance factors of these CPUs. The origin of this deviation lies in the different valuation factors consumers have towards different CPU categories, which is not factored out in the normalized transaction price, and the differences in the number of CPUs each provider has of a particular category.

In the auction market, prices emerge from the local interactions among the bidders and since a limited number of bidders participate in each individual auction,

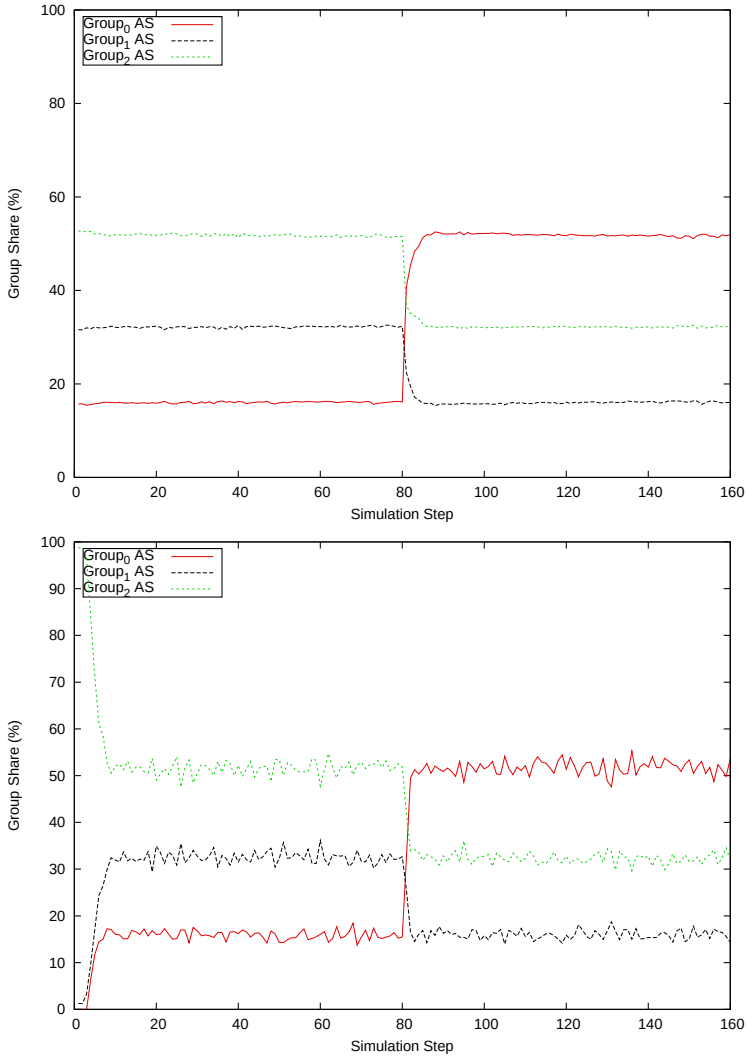

Figure 5.20: Budget shares (BS) and allocation shares (AS) under a sudden budget change for the commodity market (top) and auction market (bottom)

Categories	CM Messages	CM Runtime (ms)	CM ED_Norm	AM Messages
1	4440	80	1.0	2996
2	6276	314	1.41	2541
4	47033	1187	5.48	1962
6	90812	2355	6.86	1675
8	157608	4539	8.94	1625

Table 5.9: Number of messages sent, average runtime of the price determination process per simulation step, and median of the ED_Norm for the commodity market (CM), with respect to the number of CPU categories in the market (for the constant load scenario). The table also includes the number of messages per simulation step in the auction market (AM).

this results in less stable revenue streams for the providers. The average deviation was 6.67% in the auction market. Although a greater variance in revenue was observed on the transactions made at a single step of the simulation, the deviation on the total normalized revenue accrued by the providers at the end of the simulation was only 2.33%. For the commodity market this relative standard deviation was 2.06%. This is logical, as the effect of the specific composition of the set of bidders in the auctions on the total revenue accrued by the provider, levels out as more auctions are held over the course of the simulation.

In the less stable peaked demand scenario, the differences in revenue stability between the two market organizations increase. For the auction market, we measured a deviation of 43.74% on transaction prices for a single step and a deviation of 6.79% on the total accrued revenue. The respective deviations for the commodity market under the peak demand scenario were 4.03% and 2.31%. The significantly greater variance in the auction market is mainly due to the fact that reserve prices will be charged to some providers in times of low demand, while other providers do not need to charge a reserve price as they still receive sufficient competition in their auctions. In addition, the demand shocks in the market require the bidding agents to rediscover their obtainable degree of parallellization which leads to greater variances in bids made among the consumer population.

Computational and Communicative requirements

As already mentioned in section 5.9, the number of resource categories introduced in the commodity market is a determining factor for its communicative and computational requirements. We further investigate this relation and its implications in the context of a choice between both market organizations.

Tables 5.9 and 5.10 show the effect of introducing more categories for the constant load and peaked load scenarios. The number of messages and runtime are given per simulated time step, as well as the median of the excess demand norm over all simulation steps. The tables also include the average number of network messages sent in the auction market per simulation step for comparison. All results were obtained on a AMD64 desktop computer with a single 2.4 GHz core and 2 GB of memory.

Categories	CM Messages	CM Runtime (ms)	CM ED_Norm	AM Messages
1	3482	50	1.0	1646
2	33257	314	3.16	1576
4	93177	1084	7.55	1359
6	163750	2629	12.79	1312
8	261181	4808	72.02	1250

Table 5.10: Number of messages sent, average runtime of the price determination process per simulation step, and median of the ED_Norm for the commodity market (CM), with respect to the number of CPU categories in the market (for the peaked load scenario). The table also includes the number of messages per simulation step in the auction market (AM).

Introducing more categories allows the market participants to express their valuations for the different levels of CPU performance in a more fine-grained manner. As a result, resource allocations will be better adapted to the real needs of the users. The computational costs would allow us to determine prices for 8 CPU categories in a five second timeframe, which does not in itself inhibit a deployment of this market organization. However, the communicative requirements of the price optimization process might. If multiple rounds of polling are necessary to achieve the equilibrium price, care must be taken to limit the effects of network delays on the price adjustment process. This is especially true for large scale, wide-area deployments with higher communication delays, lower network bandwidth and higher network usage costs. Nevertheless, the communication burden can be reduced significantly through a dynamic deployment of the consumer bidding logic into the local environment of the price optimization process.

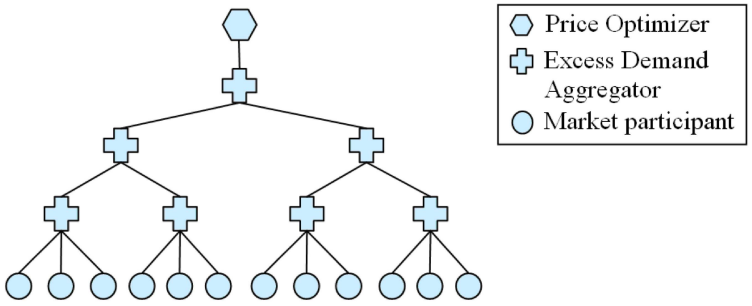

Figure 5.21: Hierarchical organization for the information flow of supply and demand data in the commodity market

In a Java-based environment, we foresee the use of dynamic classloading to transfer bidding agents and their associated Java code to the JVM of the equilibrium optimizer when new prices are to be formed. The agent then reports the participant's supply or demand levels to a local component which builds up aggregated supply and demand levels and passes them on to the optimizer. As a result, polling

will not introduce any network delays and result in a local method call on the bidding agent. Java's support for dynamic classloading allows the agent's code to be automatically downloaded when needed. This model has already been validated through a real-world deployment of the commodity market logic using the CoBRA framework [207]. Depending on the memory requirements of the bidding agents, this approach might still suffer from scalability issues. Such issues can be addressed by hierarchically organizing and aggregating demand and supply information as shown in Figure 5.21.

The amount of network messages sent in the auction market organization is significantly lower, especially for the scenarios with a higher number of categories. It diminishes as we introduce more categories because we keep the total processing capacity of the grid constant, while introducing more types with higher performance factors. This leads to a lower amount of discrete resources that need to be auctioned. Note that other types of auction, e.g. an English auction, can lead to significantly higher communication costs as a result of iterative overbidding in such auctions. Because of their strategy-proofness, single-unit Vickrey auctions require only one round of bidding, resulting in the minimum amount of communication necessary to establish a trade. We do note that Vickrey auctions are not incentive compatible if they are deployed in a repeated setting such as the one presented here. In a repeated setting, a bidder has an incentive to correctly time its bid in order to obtain a better price. As a result, the use of the presented auction mechanism in real-world settings might introduce more messages as a result of this. We have not taken such strategic behavior with respect to bid timing into account in this study. Another important factor for the lower amount of communication is the fact that a consumer only participates in a limited number of auctions (according to its target parallelization degree). On average, each auction attracted approximately five consumers in the simulated scenarios.

5.11 Summary and Conclusions

In this chapter, we have considered the use of spot markets in order to integrate economic principles into grid resource management. We have presented spot market organizations in the form of commodity and single-unit Vickrey auction markets. While fixed-pricing schemes offer a low-overhead and simple solution to resource pricing, they cannot realize optimal efficiency. This is especially the case in the presence of strong demand and supply fluctuations. In addition, establishing an initial fixed price for grid resources that delivers satisfactory results is problematic.

We have shown how both a commodity and single-unit Vickrey auction markets can dynamically price resources in response to varying demand and supply levels over time. These price adjustments ensure that economically efficient allocation of resources to grid users is achieved. Both markets result in users obtaining a share of the infrastructure that reflects their valuations relative to other competing users in the environment. In addition, prices in both markets are shown to quickly adapt

to changing levels of congestion in the grid's infrastructure.

A market model in which grid resources of a similar type are considered to be interchangeable commodities, corresponds closely with the utility vision put forward by the grid computing paradigm that is inspired by the power grid analogy. In line with this analogy, the commodity market organization requires consumers and providers to report their supply and demand levels for the different classes of commodities in the market, given a set price for each commodity. Through direct optimization of the price vector linked to the different commodities by a central process, supply and demand are balanced. This results in the market achieving equilibrium, which in turn guarantees that the optimal economic efficiency in the market is attained. Intuitively, this means that the maximal value is created by the grid's infrastructure through an allocation of resources to users that value them the most. The global nature of this optimization process results in price stability and offers limited incentive to users to act strategically. A downside of the commodity approach lies with the level of abstraction at which it operates. Although resources can be considered as commodities on a high level of abstraction, in reality they can have very different characteristics that impact application performance. The aspect of resource locality for example, is very important for data-driven grid applications, or applications that require intensive inter-job communication. Aspects such as available RAM memory also impact the suitability of resources for grid jobs. In light of such specific resource characteristics, it is difficult for a commodity market model to offer sufficient differentiation while at the same time obtaining dynamic pricing with low latency.

We have taken the state of the art in commodity-based grid resource management one step further by extending it to allow for trading and pricing of substitutable goods. This more closely models grid markets where multiple resources are available to process the jobs, but with varying performance characteristics. The proposed pricing algorithm is based on Smale's algorithm and extends this with a number of enhancements that allow it to operate in typical grid market settings. We have broken down the evaluation of the proposed model into two parts. The stability and efficiency of the algorithm, and the correctness of the market mechanism. Simulations have shown that the proposed algorithm is stable and efficient across a wide range of conditions. Furthermore, we have demonstrated that the proposed market model achieves the desirable properties of price correctness and stability, allocative efficiency and fairness.

In addition, we have combined this algorithm with existing Matlab optimization routines in order to further improve its performance for markets with a larger number of resource categories. Although the amount of supply and demand queries required to directly optimize the price can be high, we have motivated that such queries do not need to imply substantial network transmission costs across a wide-area network. We have presented a distributed and tree-based organization that is combined with Java classloading facilities in order to efficiently establish excess demand levels in the context of large-scale grids.

At the other end of the spectrum, single-unit Vickrey auctions allow consumers to formulate their valuation for an individual resource, taking into account all the

specific characteristics of the resource. No global and direct optimization of prices is performed. In contrast, prices emerge from the interactions of the bidders in the market within individual auctions. The distributed nature of this market organization requires more complex decision making from users. This involves the user to decide in which auctions to bid, for how many resources to bid and at what price. We have evaluated the market dynamics of this organization under bidding strategies that use reinforcement learning. Through initial interactions in the market, the bidding agents learn to adapt their expectations in terms of the number of resources that can be acquired on the spot market for their limit price. We compared the outcome of using both market organizations, showing that less stable prices and allocation shares are achieved in the auction market. This is caused by the distributed nature of this market organization in which local information and interactions can have a large influence on the resulting price levels. The fact that the market organization operates on the level of individual resources also poses problems for users that need to co-allocate multiple resources.

CHAPTER 6

Futures Markets

6.1 Introduction

The challenge that we propose to tackle in this chapter is the co-allocation of multiple resources in the context of service requests that require deadline-based QoS guarantees.

In the previous chapter, computational resources are traded on a spot market according to dynamically established prices. These prices however are based on the market situation at a particular point in time, and do not take into account the future supply and demand situation. As a result, users that need to allocate resources over an extended timeframe, face the *exposure* problem [131] in the sense that they have to commit to trades at time t, without guarantees for allocation opportunities or price levels at a later time $u > t$[1]. In effect, users are striving for long term QoS requirements at the level of the execution of their entire application, but have to commit to piecewise allocations early on without future guarantees. Therefore it is hard for users to formulate their valuations for allocations at a particular time instance. As mentioned by Cheliotis et al. [208], the use of futures markets and resource reservations is an important step in dealing with these issues. Futures markets allow market participants to establish contracts for delivering goods at a certain date in the future at a pre-set price (the *futures price*).

In this chapter, we propose and evaluate a resource management system that employs a futures market to trade usage rights on co-allocated and reserved CPU resources. We will compare our approach in terms of generated consumer value to

[1]Note that the model presented in chapter 5 in which jobs are allowed to run to completion for a price determined at time t, already protects users from exposure for all jobs that they launch at time t. However, users can still suffer from exposure at the level of their application, if they cannot allocate resources for all of their application's jobs at time t.

other recent work that uses market-based resource allocation for applications with deadline requirements, as well as to a non-economic approach that takes deadline requirements into consideration.

6.2 Related Work

Futures markets are also adopted in [78, 93, 209, 162] in order to deal with the exposure problem. Chun [78] presents an economic resource manager which uses *combinatorial auctions* [94] for allocating time slots on different motes in a sensor net testbed. The system supports advance reservations and bidder competition takes place over the planning horizon of the resource management system. A similar but richer approach (in terms of bid expressiveness) is adopted by Schnizler [93]. From a usage model point of view, it is clear that combinatorial auctions, in which a participant can submit a single bid for a combination of goods, provide one of the most attractive market organizations. They enable consumers to accurately define their valuations for specific combinations of grid resources that are required by their applications. As such, they allow for expressing valuations that are conditional on the co-allocation of a set of resources. This eliminates the exposure problem that users face when they need to participate in multiple auctions for acquiring the constituent parts of an allocation bundle. However, this approach suffers from high computational complexity which can mostly be attributed to the NP-completeness of determining the optimal set of winners in such an auction [161]. In addition, the lower bounds on the communicative complexity of the value elicitation process in combinatorial auctions also inhibit their applicability for large scale economies, certainly in the case of general bidder valuations and when aiming for optimal efficiency [210].

Bapna [209] has presented a centralized economic RMS that schedules preemptive workloads across multiple resources under deadline constraints. The RMS uses the `tsfGrid` heuristic that divides the workload over providers that have the lowest load in the application's execution window. A greedy approach is used that accepts workloads in order of increasing associated bid levels, in order to generate as much consumer value as possible. Because of the assumptions of zero-cost workload preemption and migration, and the absence of constraints on the parallellizability of the application, an efficient implementation of the scheduling heuristic is proposed that scales well. Our work differs from this approach in that we do not make these assumptions as we believe that unconstrained workload preemption and migration would result in significant performance losses in grid environments. This is especially the case for jobs with substantial memory or local storage requirements. When running multiple jobs that have to communicate (using MPI [211] for example), or jobs with some degree of interaction with users or peripheral devices, preemption and migration become even more difficult and costly. In addition, we believe that constraints on the degree of parallelization that applications can support is an important consideration, and that the absence of such constraints hinders the instantiation of

the proposed schedules in real-world settings.

Stößer [162] presents a centralized exchange that uses a greedy algorithm for clearing the market. The market's bidding language allows resource providers and consumers to formulate demand and supply in a future timeframe. The approach is shown to offer superior scalability in terms of the number of accepted bids, compared to a combinatorial exchange that uses CPLEX for solving the winner determination problem. Unfortunately, the exchange's bidding language requires a bidder to specify the total number of time slots a job will take, as well as the start and end time of the job. Although users may be able to specify a deadline for their job, and estimate the job's duration, the specification of a start and ending time is a difficult task and severely constrains the solution space. More specifically, these values depend on the overall system load and timing requirements of other jobs. Therefore, a scheduler typically determines the exact execution time of a job within the constraints of the request (release time and deadline) as to maximize the overall welfare in the system. Consequently, we believe this specification constrains the solution space for the scheduling problem too much and requires information that users typically cannot provide unless the execution timeframe of their application is strictly linked to a real-world process with a fixed timeframe[2]. In addition, although constraints on the degree of parallelization in the application are supported, jobs are assumed to be preemptible and migratable to other execution nodes, leading to drawbacks similar to the `tsfGrid` heuristic.

In [212] a trading system is presented with cognitive agents participating in a market based on multiple ascending English auctions with support for advance reservations. Different bidding strategies are evaluated in a simulated environment and the effects of local semi-cooperative interaction among agents are shown on realized bidder satisfaction and cost. As this study assumes that each bidding agent only has one indivisible task to execute, it is unclear how the system would perform in the context of parallel applications that benefit from co-allocation of multiple resources.

6.3 Market model

Grid resource management is complex since it has to deal with a set of heterogeneous resources of different types, such as network links, CPUs, memory space, and special purpose hardware. In addition, a set of widely varying user requirements and application models needs to be supported. Adding the notions of costs, budgets and prices induces another level of complexity related to the bidding strategies or *profiles* of the users. Current research efforts are not yet able to tackle the complexity of (economic) grid resource management in its general form. Therefore, each study needs to place a number of modeling constraints on the system in order to deal with the problem's complexity.

[2]Consider for example the case in which CPU resources are required between 10am and 11am to support a live demonstration of a distributed application during a conference talk.

The model that we will outline in the following subsections focuses on environments with heterogeneous computational resources, and with so called *trivially parallel*[3] and CPU bound applications with hard deadlines. *Parameter sweep* applications that require limited amounts of network I/O, and to which the user attributes a strict completion deadline are a commonly occurring class of applications that fit our model.

6.3.1 Consumer Agents

Our market environment includes a set of n consumer agents C_i $(i = 1, ..., n)$ that act on behalf of users. An agent C_i is characterized by the quad-tuple $\{A_i, B_i, V_i, d_i\}$. The agent's budget B_i can be used to acquire resources from the providers in the market for running its application A_i. The agent's goal is to finish A_i which consists of n_i atomic jobs J_{ij} with $1 \leq j \leq n_i$ by deadline d_i. The execution window W_{A_i} for the application is defined as $W_{A_i} = [now, d_i]$.

Every job J_{ij} has a processing requirement p_{ij} which represents the job's running time on a reference machine M_{ref}. We denote the total workload induced by one agent request on the system by $L_i = \sum_{j=1}^{n_i} p_{ij}$. Users are not always able to accurately predict the runtimes of their jobs. In such cases, techniques based on symbolic analysis [213], statistical analysis [214] or analytic benchmarking [215] can be used for prediction. Here, we assume runtimes to be estimated correctly and leave a study on the effects of wrongly predicted job runtimes on the algorithms for future work.

If the application A_i finishes at $e_i \leq d_i$, the generated value is $v_i = V_i$, otherwise $v_i = 0$. We thus employ a user model with *hard* deadline constraints. In a model with *soft* deadline constraints, the value that results from successfully finishing the application's workload is determined by a reward function $r(e_i)$, decreasing in e_i. We assume that consumers have quasi-linear utility functions which means that if C_i pays P_i for the servicing request, the generated utility is $U_i = V_i - P_i$. The total consumer value generated by the allocations made by the RMS is denoted by $V = \sum_{i=1}^{n} v_i$. Also let $V_{MAX} = \sum_{i=1}^{n} V_i$.

We assume there are no precedence relations between the set of jobs of A_i. Although we deem it appropriate to consider individual jobs as being non-preemptible and atomic, we will relax these restrictions in parts of the evaluation to compare our approach to algorithms already proposed in the literature, and to investigate the effect of these constraints on the generated consumer value.

The consumer agent will formulate its service request as a bid $R_i = \{A_i, b_i, d_i\}$, specifying that C_i bids b_i credits per time unit of workload executed, to finish the jobs of application A_i before d_i.

6.3.2 Provider Agents

The m provider agents P_k $(k = 1, ..., m)$ in the market environment each host a set of m_k uniform parallel [216] machines $\{M_{k,1}, ..., M_{k,m_k}\}$. Machines are said to be

[3] Applications are called trivially parallel when the jobs that form the application are fully decoupled and have no precedence constraints

uniform parallel if all jobs can be processed on all machines and if the runtime of a job J_{ij} on a machine M_{kl} is $\frac{p_{ij}}{s_{kl}}$, with s_{kl} the speed of machine M_{kl}. The total processing capacity of the grid's infrastructure is denoted by $S = \sum_{k=1}^{m} \sum_{l=1}^{m_k} s_{kl}$.

Each provider has its own scheduling window $[now, now + h_k]$, which determines how far into the future the provider allows resources to be reserved for executing jobs.

Providers have the possibility to set a reservation price r_k which denotes the minimal price for which they are willing to sell one unit of processing capacity.

6.3.3 Market Operation

In this section, we discuss the market's operation under both a centralized and a decentralized algorithm for economic resource management.

Centralized Brokering System

The *Centralized Brokering System* (`CBS`) consists of a *broker* to whom a consumer can direct its service request R_i. The broker acts as an intermediary and negotiates with the market's providers to fulfill the request within the QoS constraints put forward by the consumer. In doing so, the broker makes sure that it protects consumers from the exposure problem by co-allocating resources across different providers at different points in time, thereby ensuring that R_i is either entirely fulfilled or entirely rejected. The broker prioritizes requests in decreasing order of their bid level. It then handles requests one by one in sorted order, trying to co-allocate resources for the individual jobs of application A_i under the constraint of meeting the request's deadline.

If multiple providers are able to schedule in a job, the broker can use a number of different *provider selection* policies. We will consider the following policies :

- Selecting the provider which can plan in the job closest to its deadline (`CTD`). This policy allows the broker to keep the most opportunities open to schedule in lower-valued jobs with shorter deadlines.
- Selecting the provider with the greatest remaining capacity in the execution window of the job (`MAX`), which allows the broker to load-balance jobs over all providers.
- Selecting the provider with the lowest remaining capacity in the execution window of the job (`MIN`). This heuristic tends to use up resources one by one. As a result, this increases the likelihood that scheduling fragments that are created early on, can be used at a later point in time.
- Random selection (`RND`). This policy is fast and load balances over all providers without requiring any ordering on the set of providers.

The effect of these policies will be investigated in the evaluation section.

When jobs are assigned to providers, the providers face a similar decision to determine on which local resources to schedule. As such, the policies presented above can also be used as *local resource selection* policies. Note that in the following, we

will denote the CBS market with e.g. RND provider selection policy and MIN local resource selection policy as $\texttt{CBS}_{\texttt{RND-MIN}}$.

After selection of a local resource, a local scheduler for the resource will schedule the job workload according to a CTD policy. This maximizes the likelihood that a highly ranked job does not interfere with lower ranked jobs that have an earlier deadline, but increases the degree of schedule fragmentation. Another option is to schedule in the workload according to an earliest start time policy, which is characterized by the inverse of the CTD policy's properties. Note that the use of heuristics is mandatory since the problem of minimizing the weighted tardiness of a set of deadline constrained jobs with arbitrary running times, is already strongly NP hard for the single processor case [217].

We do not consider hard job requirements such as minimal amount of memory or scratch storage in this study. Such constraints are typically fulfilled by the broker through a *matchmaking* process which filters the candidate set of resources using the grid's information system. This process precedes the scheduling activities and does not interfere with it. We can therefore omit such requirements from our model.

After the broker has determined the set of acceptable requests and associated resources, it sets up servicing contracts with the providers and consumers involved, describing their rights and duties. A trusted accounting and charging system needs to be in place that deals with the payment transactions between the trading parties and the follow-up of contract compliance by the involved parties. The design of such a system is outside the scope of the work presented here. For payment, a Web standards-compliant service such as the one presented by Cohen [218] could be adopted. The service supports a market model in which real currencies are used by handling payments through PayPal [219].

Decentralized Auctioning System

A centralized approach has the advantage of the broker having full access to the valuation information of all bidding consumers. It is therefore able to rank all requests in order of the expected value they will generate and use this information as a lead for making allocation decisions that maximize the aggregate value generated by the system. Depending on the scale of deployment however, a centralized approach might not offer sufficient scalability. We therefore also investigate a *Decentralized Auctioning System* (DAS), in which each provider agent P_i hosts one auction for selling its resources. Each consumer agent C_i will attempt to fulfill the requirements for A_i, by placing bids for the n_i jobs of A_i at these auctions. The market operates in *rounds*, each of which consists of the following three *phases*:

- **Bidding Phase** For each job in application A_i, consumer C_i selects a random auction to bid in. Upon arrival of a bid, the provider checks whether the bid is acceptable, irrespective of the bid's level in comparison to other bids received by the provider, and notifies the consumer of the outcome. This involves a planning phase in which the provider checks whether it is possible to schedule in the workload that would be induced upon acceptance of all of C_i's current bids at its auction while honoring confirmed bids from previous rounds. If the outcome is positive, the bid is accepted. Upon a negative outcome, the

consumer will attempt to contact other providers to resubmit his bid.

If consumer C_i is unable to issue n_i accepted bids, it retreats from the bidding process as it has no possibility of winning enough auctions to run A_i within its QoS constraints. This involves a cancellation of all of its accepted bids. The consumer agent will then become *inactive*. Note that consumers bid for their jobs in decreasing order of job runtime. Firstly, this limits the amount of bids made by consumers. Secondly, this limits the risk that a consumer cannot schedule its jobs due to the fact that these are bid for in the wrong order, i.e. bidding for larger jobs first decreases the chances that previously made bids impede the acceptance of subsequent bids.

- **Closing Phase** Each provider P_i closes its auction and performs a *winner determination* procedure. This involves a greedy algorithm which ranks all bids in decreasing order of their level, and subsequently tries to schedule in the workloads of as many bids as possible in this order. If multiple bids have the same bid level, they are ranked according to the unique ID of their bidder. The provider uses the `MIN` policy for selecting a local scheduler to plan in a job. As in the centralized case, workloads are scheduled in on an individual resource according to the `CTD` policy.

- **Confirmation Phase** Consumers check whether all of their bids have won. If this is the case, the consumer has a full guarantee of finishing its application by deadline and will confirm its bids at the given providers. The winning consumers then become inactive and providers will reserve resources for their workloads.

The market mechanism organizes bidding rounds until no active consumers remain. At that point, payments are determined, resources are allocated and the market is said to *close*.

Note that in the closing phase, the selected bids are not guaranteed to maximize total revenue for the provider. Computing the optimal set of winning bids however, would involve solving a combinatorial allocation problem which is not tractable. In addition, preferring a number of lower bids over higher bids in an attempt to maximize total revenue would disalign the bid selection procedure over the different providers. This will result in significantly more rounds since the bidder needs to find a resource for all of the jobs in its application. It would also no longer guarantee a finite running time for the planning phase. Finally, finding the optimal outcome would require full information on all bids that will arrive in future bidding rounds.

We have opted to use the `RND` provider selection policy for the `DAS` market since the other policies establish an ordering among the providers. If all consumers were to create such an ordering, this would result in too much communication and computation overhead. Moreover, these policies do not lead to significantly better results for the `CBS` market, as we will show in the evaluation section.

Note that the market is guaranteed to *close* in a finite number of rounds as there will always be one highest ranked bidder during each bidding round. If this bidder is able to place bids for all of its n_i jobs, he is guaranteed to win and therefore become

inactive. If the bidder is unable to place n_i bids (due to scheduling constraints at providers), he will become inactive as well. Because at least one bidder will exit the bidding process during each round, the process is guaranteed to terminate.

Scheduling time frames

Within the context of this chapter, we will focus on resource allocation and pricing when managing grid resources for one set of consumer requests that are given at the beginning of a scheduling time frame. As such, the time frames for resource reservation and market competition are assumed to be large enough to accommodate the requests' deadline requirements and to allow for sufficient competition.

In a real-world deployment however, there will be a trade-off between using large time frames in order to minimize the exposure problem and maximize the degree of market competition, versus maximizing the flexibility of the RMS in terms of fulfilling online user requests that are allowed to arrive at any time. If multiple planning phases are used, there is also an opportunity for *defragmenting* the schedules between two consecutive planning phases by shifting workloads to the front of the schedule. The frequency of planning and the length of the adopted time frames can vary dynamically in accordance with the QoS requirements of the consumer population and properties such as market size and bidding delay.

6.4 Pricing algorithm

There is a wide range of pricing mechanisms available and at present it is unclear which mechanism(s) would be best suited to resource pricing in grid environments. A desirable property of a pricing mechanism is *incentive compatibility* [220] as such a mechanism will elicit true valuations of bidders. This in turn, is a requirement for the RMS to obtain efficient allocations [93]. One of the most simple incentive compatible mechanisms is a Vickrey auction for a single good, which is an application of the general Vickrey-Clark-Groves (VCG) mechanism [101].

While the VCG mechanism is incentive compatible it is in many cases only interesting from a theoretical perspective, as it is often intractable to calculate VCG prices in complex settings such as the one presented here. This is a consequence of the fact that it involves recomputing the solution to the *winner determination problem* (WDP) m times if there are m winners. Moreover, the VCG mechanism is built on the assumption that the algorithm used to solve the WDP delivers the optimal solution [221]. In our case, this premise is not fulfilled as the WDP presented in this work is strongly NP hard (as determining the optimal set of winning bids in our market involves an NP hard scheduling problem), and we therefore need to resort to approximations and heuristics in order to maintain computational tractability. Also, in repeated settings such as the `DAS`, the VCG mechanism is not incentive compatible [222].

Note that consumers might also untruthfully report information on their deadline constraints in an attempt to obtain a lower price, as prices can be expected to decrease over time. This is especially the case if they can make an accurate

estimation of the expected load on the system and if the RMS returns job results to consumers as soon as they are ready. The approach taken by Kang [223] can be used to rule out such misreports by postponing the delivery of results until the scheduled completion time of the job. Other limitations to the practical use of the VCG mechanism and its premises for reaching incentive compatibility are given by Rothkopf [224], Ausubel [225] and Sandholm [226].

We also note that while a VCG mechanism is incentive compatible for both buyers and sellers, it is not (weakly) budget-balanced for markets with multiple buyers and sellers [92]. This means that the mechanism's outcome can result in a situation in which the payments to the sellers are not guaranteed to be covered by a redistribution of the payments made by the buyers to the mechanism. Therefore, the market's operation needs to be subsidized from an external source. As a general result, Myerson and Satterthwaite have shown that no Bayes-Nash incentive compatible mechanism can simultaneously obtain the properties of efficiency (i.e. result in the optimal outcome), individual rationality and budget-balance for agents with quasi-linear utility functions [92].

As an alternative to the VCG approach, we have implemented a pricing algorithm that encourages users to bid truthfully through the use of second-pricing principles. Note that we cannot provide a fully incentive compatible algorithm as determining the optimal set of winning bids in our setting involves a NP hard problem and therefore requires the use of heuristics. The algorithm iterates over the losing bids from low to high and for each losing bid R_l, it assigns the bid level b_l to the time period given by the execution window of A_l. This results in market-wide prices for each point in time that equal the bid level of the highest losing bid at that time. As such, this algorithm applies the core principles of a Vickrey auction to a more general setting involving multiple sellers and a time dimension.

In the centralized case, the broker knows the set of losing bids and calculates the transfer prices accordingly. In the decentralized case, providers are contacted by losing consumers at market closing time so they are able to identify these bids. For periods not covered by the execution window of a losing bid, each provider P_k uses its reserve price r_k.

Note that a consumer's bid might lose due to the fact that a provider is unable to plan in the consumer's job, irrespective of the consumer's bid level. In that case, a lower bid covering a job with an overlapping execution window could win. Therefore, some jobs might be scheduled in a period that has a set price above the bid made by the consumer. When necessary, our pricing mechanism will cut off the total cost of the application so as to make sure that the average price per unit of time stays below b_i. We thereby limit the payment that a consumer has to make for a reservation such that the price paid per unit of time for the reservation does not exceed b_i. This ensures that our mechanism has the property of individual rationality as the consumer can never be confronted with a cost level that exceeds his bid level. Therefore, the consumer does not run the risk that participating in the mechanism, results in negative utility. The `tsfGrid` [209] algorithm uses a similar

pricing scheme but purges all capacity in a rejected bid's execution window[4] in order to deal with this problem and to obtain *fair* allocations.

The `tsfGrid` algorithm is not incentive compatible in theory. In practice however, *posterior regret-free* allocations [227] were obtained in more than 85% of the cases under study in [209]. For the remainder of this chapter, we will focus on the efficiency of the proposed market mechanisms and assume that our pricing mechanism elicits true valuations such that $B_i = V_i$ and $b_i = \frac{V_i}{L_i}$.

6.5 Evaluation

In this section, we will evaluate both the centralized CBS and decentralized DAS markets and compare their performance to a non-economic resource manager and to an existing market-based resource management algorithm.

6.5.1 tsfGrid

In order to compare the performance of our algorithms to an existing market-based algorithm for resource allocation in grids, we have implemented the `tsfGrid` heuristic [209]. This heuristic adopts a centralized approach in which the user requests are directed to a broker which plans in the workloads so as to maximize user value. A difference with our approach is that workloads are assumed to be infinitely parallellizable, preemptible and migratable at zero cost. Bapna et al. have shown that in 87% of their trails, the heuristic obtains the optimal efficiency [209]. Therefore, the results of this algorithm can serve as an indication for the maximal efficiency that can be delivered under these relaxed assumptions with respect to the user's workload.

Algorithm

Every consumer C_i submits its request R_i to a central broker which hosts an auction. The request specifies a bid level b_i, the total workload induced by the request L_i, and the scheduling window during which the workload can run $[x_{C_i}, y_{C_i}]$. The bid level b_i is the amount of money C_i is willing to pay in order to allocate one standardized unit of CPU time. Because of the previously mentioned assumptions related to the users' workload, there is no differentiation between individual jobs within a request and we can represent the full workload of the request by a single job J_i.

Providers submit sell-offers to the broker specifying their available processing power S_k, as well as the time-interval $[x_{P_k}, y_{P_k}]$ over which resources are made available. The entire processing capacity of a provider is considered to be delivered by a single machine M_k.

The algorithm divides time into discrete *time periods* $[t_z, t_{z+1}]$ in such a way that reported supply and demand in each period remain constant. This is done by considering the set $T = \{\bigcup_i \{x_{C_i}, y_{C_i}\} \cup \bigcup_k \{x_{P_k}, y_{P_k}\}\}$ and imposing an ordering on T such that $t_z < t_{z+1}$ for $z \in \{0, |T| - 2\}$.

[4] I.e. it removes all remaining supply in the market within this timeframe.

After receiving the bids and offers, the bids are sorted in decreasing order of b_i and are then processed in this order. For each bid R_i, the algorithm checks whether there is enough remaining CPU capacity in the bid's scheduling window to successfully execute the workload. If this is not the case, the whole scheduling window of R_i is purged so that workloads of the remaining requests with a lower bid level, cannot allocate resources in $[x_{C_i}, y_{C_i}]$. This is done in order to maintain fairness.

If there is enough CPU capacity available in $[x_{C_i}, y_{C_i}]$ to execute J_i, the interval $[t_z, t_{z+1}]$ with the highest remaining CPU capacity is chosen. In this interval, the provider with the highest remaining CPU capacity is selected to deliver up to L_i capacity for J_i. This process iterates until the full processing requirements for J_i are fulfilled.

After all requests have been processed, the price for each period $[t_z, t_{z+1}]$ is set to the highest losing bid in that period. If there is no losing bid in $[t_z, t_{z+1}]$, an exogenously determined reserve price is used.

6.5.2 Simulation Configuration

Since no real-life data records are available concerning valuation levels of grid users with deadline constraints, we are obliged to construct synthetic valuation functions. We define consumer agent C_i's valuation as

$$V_i = V_{base} * VF_{load} * VF_{deadline} \tag{6.1}$$

with V_{base} a chosen base valuation for the execution of one unit of workload, VF_{load} a valuation factor tied to the total load a consumer plans to induce, and $VF_{deadline}$ a factor related to the urgency of the request. Three consumer groups are created. Each group has its own deadline range and uses a different function for calculating $VF_{deadline}$, as shown by the graph in Figure 6.1. The VF_{load} factor is defined as

$$VF_{load} = 1 + (\frac{L_i}{\frac{1}{n} * \sum_{j=1}^{n} L_j} - 1) * LF \tag{6.2}$$

with LF a factor which introduces a valuation surplus for all consumers that have a larger than average load.

We generate 100 scenarios according to the scenario parameters depicted in Table 6.1 for each value of the independent variable, and determine the average values over these runs for the metrics under study. Parameters that are specified with a range are drawn from a uniform random distribution, using the random number generator delivered by Sun's JDK. Note that we have used the same value for the scheduling horizon h_k over all providers. While this is not required by the presented algorithms, we decided to rule out the effect of heterogeneous scheduling horizons in favor of a clear interpretation of the results.

Also note that we have considered providers to be price takers and have therefore used a zero reservation price. In scenarios with high resource contention, reservation prices do not play a role in the price formation process since actual prices will be formed by competing user bids. When installing a provider-side strategy for

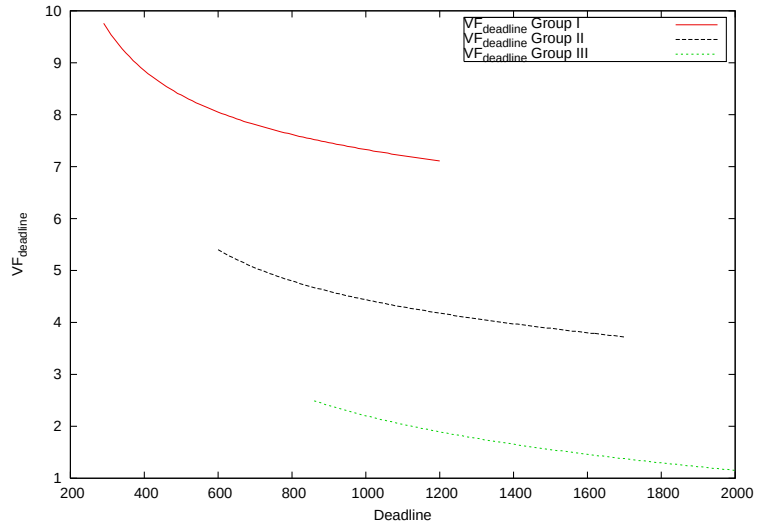

Figure 6.1: $VF_{Deadline}$ for the different consumer groups in function of deadline

Table 6.1: Scenario parameters for the futures market scenario

Parameter	**Value**
h_k	2016
n	300
m	20
p_{ij}	{1,...,80}
n_i	{210,...,390}
V_{base}	10000
LF	1.05
d_i $(Group_I)$	{288,...,1152}
d_i $(Group_{II})$	{576,...,1728}
d_i $(Group_{III})$	{864,...,2016}
s_{kl}	{0.5,...,1}
S	1250
r_k	0

setting non-zero reservation prices, care must be taken that providers do not price themselves out of the market. This would lead to underutilization of resources which is generally not in the interest of either providers or consumers.

Since the runtimes of jobs are generated synthetically, we will use unitless time in the remainder of this section for all time-related variables. We will assume that both processing requirements and processing capacity can be expressed in agreed upon standardized units. More specifically, a machine M_{kl} with $s_{kl} = 1$ will require p_{ij} time units to execute job J_{ij}.

6.5.3 Consumer Value

In this section, we compare the total consumer value generated by the presented resource management systems. One of the key aims of economic resource management is to maximize this value. This differs from non-economic resource management systems that do not take consumer valuations into account. The *Deadline-Oriented Non-Economic* (DONE) RMS that we compare against uses a central broker that aims to fulfill a maximum number of consumer requests. It follows a greedy approach, processing requests in order of increasing workload. When planning in an individual job, the CPU with the largest available remaining processing capacity is selected and the job is planned in according to the CTD policy.

The graph in Figure 6.2 shows the resulting consumer value generated for different levels of aggregate processing capacity in the system. The graph in Figure 6.3 shows the percentage-wise increase in realized consumer value compared to the DONE RMS. Note that we have omitted the results for the $\text{CBS}_{\text{MAX-MAX}}$ and the DAS markets in Figure 6.2 for the sake of readability. Relative standard deviations (RSD) for the market-based RMS's were below 2.2% for all sample points considered. The results for the DONE RMS showed an RSD up to 6.8% for the scenario with the highest contention and 2.12% for the scenario with the lowest contention. The average RSD over all sample points was 1.66%.

The system-centric optimization goal of the non-economic broker leads to suboptimal results with regard to the realized value. An increase of up to 57% can be observed for the tsfGrid heuristic and 53% for the $\text{CBS}_{\text{MIN-MIN}}$ market. The difference decreases with increasing processing capacity that is available in the system. This is caused by the fact that under congested settings, only few requests can be fulfilled. A careful selection of the high value requests therefore has a relatively large impact on the percentage of user value that is realized. In non-congested settings, the impact of such careful selection is relatively smaller which leads to a less significant difference between the performance of the DONE RMS and the market-based approaches in such settings.

As shown by the graph in Figure 6.3, ordering providers and CPUs according to the CTD-CTD and MAX-MAX heuristics does not offer significant performance gains compared to the fully random RND-RND heuristic. The adoption of the MIN heuristic for CPU selection however, does yield substantial performance gains over RND-RND, as shown by the performance of the RND-MIN and MIN-MIN heuristics, as well as the DAS market. The MIN-MIN heuristic performs slightly better than RND-MIN, but requires a potentially costly ordering of the entire list of providers. In a grid setting, this might induce too much communication overhead. In addition, the ordering might (quickly) become invalid as a result of other brokers or schedulers using the same grid resources. In the decentralized DAS market these problems are further exacerbated since each bidder would have to query all providers to obtain a sorted set of providers for each job. Therefore, we did not evaluate the non-randomized provider selection policies for DAS.

Figures 6.2 and 6.3 demonstrate that the decentralized DAS market organization can obtain similar efficiency notwithstanding the fact that there is no central coordi-

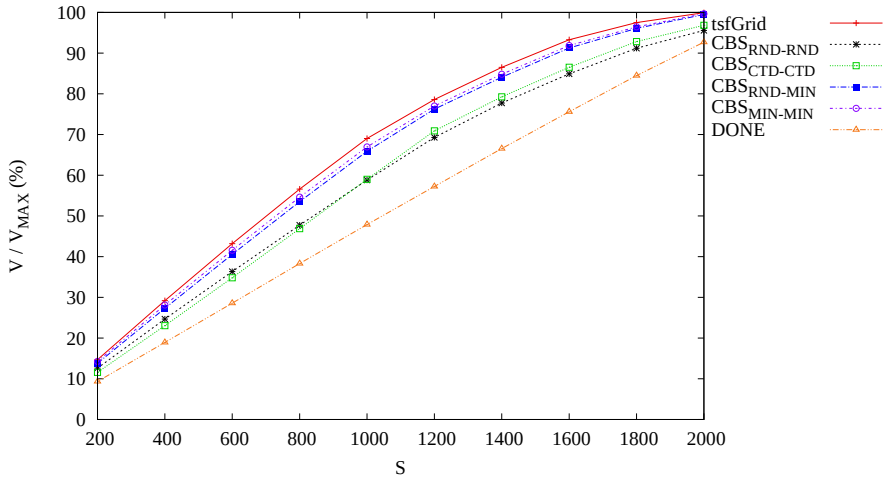

Figure 6.2: Percentage of consumer value realized for varying processing capacity S

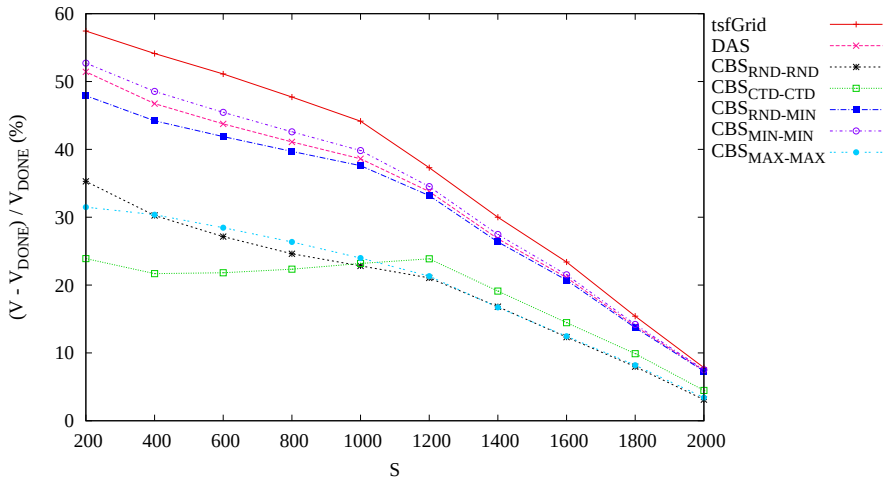

Figure 6.3: Percentagewise increase of consumer value realized under varying processing capacity S compared to the `DONE` RMS.

nating entity that has full access to all valuation information. In the `DAS` market this coordination is achieved in a decentralized manner through the market mechanism.

The `tsfGrid` heuristic purges all remaining processing capacity in the execution window of an unfulfilled request, which leads to *fair* allocations. Fair in the sense that a bidding agent can never allocate resources in the time frame of an unfulfilled request covered by a higher bid. Although purging is shown to have a small effect on realized utility and utilization by Bapna [209], Table 6.2 indicates that this is not the case for the scenario under study here, especially under conditions of high congestion. Therefore we have turned off purging in order to obtain an even-handed comparison.

Table 6.2: Effect of purging on utilization and realized consumer value

S	**Purging**		**Non-Purging**	
	Util. (%)	V/V_{MAX} (%)	Util. (%)	V/V_{MAX} (%)
2000	91.13	100	91.13	100
1800	90.24	96.26	92.08	96.89
1600	89.86	91.97	93.2	93.08
1400	90.49	85.15	94.07	86.83
1200	90.31	76.08	95.04	78.6
1000	91.94	66.37	96.32	68.33
800	88.05	51.37	97.24	55.63
600	82.08	36.57	97.74	42.12
400	72.6	21.89	99.02	28.2
200	54.79	8.36	98.13	13.99
100	32.69	2.79	98.53	6.44

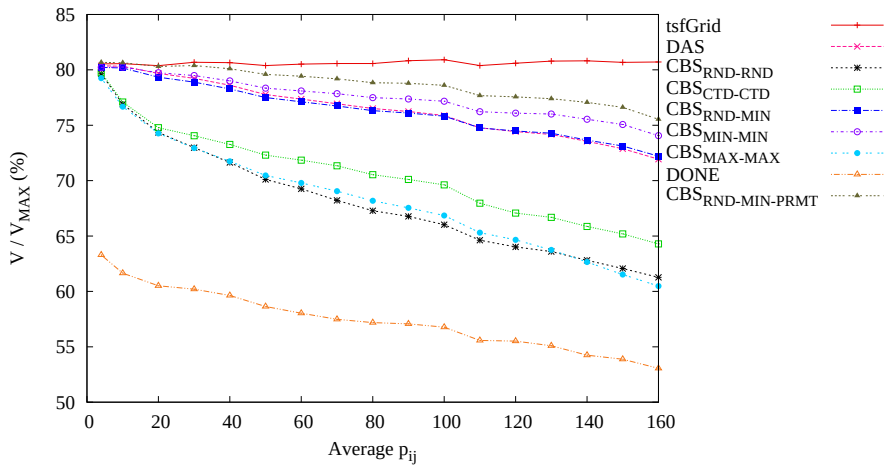

Figure 6.4: Effect of average job size (p_{ij}) on generated consumer value

Figures 6.2 and 6.3 present the results for the `tsfGrid` heuristic, assuming that preemption and migration will cause no overhead costs. Because the `tsfGrid` heuristic is not constrained by the runtime of individual jobs, it is able to use the infrastructure more efficiently, which results in a slightly higher generated value compared to the `CBS` and `DAS` markets. The graph in Figure 6.4 with the effect of the average job size on the generated consumer value, illustrates that this is especially true for large average job sizes. Note that we have kept the total load in the system constant by introducing more jobs when lowering the average job size. For the data shown in Figure 6.4, relative standard deviation was below 3.5% for all sample points. The average RSD over all sample points was 2.01%. The fact that our algorithms do not outperform the `tsfGrid` heuristic in terms of realized consumer value is not

entirely surprising, as this heuristic has been shown to realize the optimal outcome in 87% of the trails investigated in [209]. In the other trails, an efficiency close to the optimal outcome was obtained. However, it should be kept in mind that the `tsfGrid` heuristic operates under the assumption that workloads can be migrated across CPUs hosted at different providers, at zero cost.

In addition, `tsfGrid` cannot incorporate constraints on the degree of parallelization that is supported by the workload. As a result, it might propose schedules which are not executable on a real grid infrastructure, and admit workloads to the system under QoS constraints that cannot be satisfied in reality. These constraints have been defined in our model through the composition of an application out of a set of atomic jobs. For the parameters given in Table 6.1 for example, the average size of a time period defined by the heuristic (cfr. section 6.5.1), was 7.35. This means that even on the fastest machines in the scenario with $s_{kl} = 1$, the average job with $p_{ij} = 40$ would be parallelized and executed concurrently over more than 5 machines, in the schedule built by the `tsfGrid` heuristic[5], thereby violating the atomicity constraint. For an average job size of 160, over 21 machines would need to be used concurrently. This is caused by the fact that the heuristic selects the time period with the largest remaining processing capacity and then selects the provider with the largest amount of available processing capacity in that period. It then assigns the maximum amount of workload to this provider which is to be completed within this time period, thereby requiring parallelization of the job over multiple CPUs.

The assumption of zero migration costs (in terms of runtime performance) leads to similar problems when the tsfGrid schedules need to be executed on real grid systems. Finally, current production-level grids offer limited support for live job migration and it is unclear to what extent migration will be supported by future grid environments.

Considering the higher consumer value realized by the `tsfGrid` heuristic (especially for large job sizes), it is interesting nevertheless to assess to what degree *local job preemption*, which allows jobs to be preempted and resumed on the same host, would improve the performance of the algorithms that we propose. This involves minimal adaptations to our algorithms. The provider maintains the same local scheduling policy but can now use different fragments in the schedule to plan in the job's workload. Allowing local preemption without job migration makes it easier for the RMS to deal with schedule fragmentation, without incurring the high migration costs when transferring jobs from one host to another. The cost for local preemption depends mainly on the overhead caused by the context switches that transfer control to a different job. Depending on the memory intensiveness of the applications under consideration, this cost might be acceptable.

The effect of allowing local preemption is illustrated in Figure 6.4 by the performance of the $\texttt{CBS}_{\texttt{RND-MIN-PRMT}}$ market. The graph indicates that local preemption can significantly improve on the realized user value compared to the $\texttt{CBS}_{\texttt{RND-MIN}}$ market.

[5]Provided that the provider that is selected by the tsfGrid heuristic, has enough processing capacity available to run the entire job, and hosts at least 5 machines

Table 6.3: Average values and standard deviation s of cost levels per unit of workload for the three consumer groups under varying processing capacity S in the $\texttt{CBS}_{\texttt{RND-MIN}}$ market

S	Group_{I}	s	Group_{II}	s	$\text{Group}_{\text{III}}$	s
2000	0.98	0.66	0.82	0.56	0.56	0.39
1500	3.72	0.13	2.47	0.14	0.64	0.18
1000	5.98	0.10	2.53	0.5	0.73	0.27
500	6.69	0.08	3.23	0.09	1.00	0.09
200	6.97	0.07	3.34	0.05	1.05	0.03

The effect of adding local preemption capabilities to the other markets was similar to the one observed for the $\texttt{CBS}_{\texttt{RND-MIN-PRMT}}$ market.

6.5.4 Pricing

In a correctly operating market, prices should reflect the value users attribute to resources and the scarcity of those resources. Table 6.3 confirms that this is the case for the prices generated in the $\texttt{CBS}_{\texttt{RND-MIN}}$ market. Under diminishing infrastructure size and higher resulting congestion levels, higher prices need to be paid by consumers. The scarcity of resources with which the RMS has to plan in the workload, causes a larger number of consumers to be rejected. This increase in rejections leads to higher prices because of our second pricing approach. High value consumers also pay more to get their workloads planned in because of their higher bid levels and the larger external effects they introduce through their workloads. The results for the other markets exhibit similar trends, which is to be expected as they use the same pricing algorithm.

As can be observed from Table 6.3, the sample standard deviation denoted by s in the table, is high for the setting with the largest processing capacity. We have verified that this is caused by the fact that this scenario constitutes a boundary case. More specifically, in some runs all consumers can be serviced which results in zero costs because providers do not impose reservation prices. In other runs, not all requests from consumers in the third group were accepted which results in non-zero prices. This creates a higher variation across runs in terms of prices paid when compared to scenarios with lower capacity.

6.5.5 Allocation shares

Tables 6.4 and 6.5 show the shares of the total compute time allocated to the different consumer groups in the system for the $\texttt{CBS}_{\texttt{RND-MIN}}$ market and `DONE` RMS respectively. This data shows how the use of an economic resource manager translates the valuations of the consumers in the different groups into allocations on the grid's infrastructure.

The high-value consumer groups are allotted a greater share in the $\texttt{CBS}_{\texttt{RND-MIN}}$ market because of their larger budgetary endowment and valuations. In the non-economic approach, the low-value group is given the largest share because it has the execution window with the least amount of resource competition.

Table 6.4: Average and standard deviation s of allocations shares for the three consumer groups under varying processing capacity S for the $\texttt{CBS}_{\texttt{RND-MIN}}$ market

S	Share_{I} (%)	s	Share_{II} (%)	s	$\text{Share}_{\text{III}}$ (%)	s
2000	33.75	0.81	33.86	0.66	32.39	1.23
1500	43.65	1.12	37.36	2.53	18.98	2.7
1000	57.43	1.21	27.61	2.11	14.96	2.14
500	57.1	0.94	29	1.16	13.9	1.16
200	56.8	1.37	29.01	1.62	14.2	1.45

Table 6.5: Average and standard deviation s of allocations shares for the three consumer groups under varying processing capacity S for the DONE RMS

Capacity	Share_{I} (%)	s	Share_{II} (%)	s	$\text{Share}_{\text{III}}$ (%)	s
2000	32.08	0.73	33.63	0.74	34.28	0.69
1500	30.33	1.45	33.08	1.5	36.59	1.29
1000	29.28	2	31.78	2.28	38.94	2.17
500	28.82	2.82	30.58	3.43	40.6	3.12
200	27.76	5.33	30.99	6.14	41.25	5.07

We note that this anomaly does not occur for all non-economic scheduling approaches. A priority or share-based approach for example, can perform better in this regard [228]. For example, if the users from group one were allowed to express a priority level for their jobs that is higher than the level expressed by the users in groups two and three, they would get full access to the infrastructure at the beginning of the simulation. As a result, consumers from groups two and three would not be able to allocate resources during these first steps. Similarly, if equal shares are assigned to users in a share-based approach, it can be ensured that a user group cannot allocate more than its fair share.

Such priorities and shares are typically defined by committee through real-world negotiations. Since these negotiations only occur periodically, they result in rather static priorities and shares over time. Because of the number of stakeholders these negotiations are so complex that they take up a substantial amount of time and are not likely to happen frequently. As a result, the assigned priorities/shares will start to deviate from the real importance of the users. The whole negotiation process also restricts the openness of the system because in between negotiations, no new users or providers can participate in the system. An economic approach on the other hand, can accommodate new participants in a flexible manner.

The occurrence of congestion has no effect on priorities and shares, so the user will not have an incentive to change its behavior and congestion will endure. In a priority-based system this means that high priority users can monopolize the system, leaving no room for other users. Even when a lower priority user has an application it considers very important, it cannot force the execution of the application. This happens to a lesser extent in a share-based system when the share of such low priority users is not big enough to execute the application within deadline. When

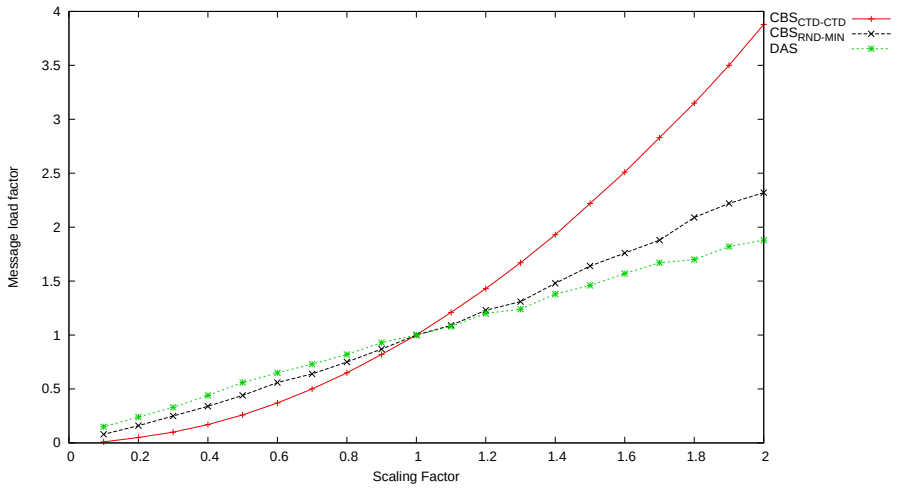

Figure 6.5: Effect of scaling factor on message load factor

using an economic approach this situation would not occur as the user can simply pay enough to have its important application executed in times of congestion, increasing flexibility. Moreover, high price levels give incentive to users to defer the execution of their applications if possible, allowing for economical efficiency.

One of the main issues with fixed priorities or shares resulting from a negotiation process, is the failure to recognize the importance of dynamicity in priorities or shares. When openness, flexibility and responsiveness are less important, a non-economic approach will suffice. In our opinion however, these are the main characteristics of a grid system.

Tables 6.4 and 6.5 indicate that in scenarios with high contention, the economic approach shows considerably lower sample standard deviations compared to the results for the `DONE` RMS. This is a result of the consistent manner in which the economic RMS gives priority to high value consumers.

6.5.6 Scalability

We investigate the scalability of the proposed market mechanisms with respect to communicative requirements. We scale the number of consumer requests, the available processing capacity and the number of providers in the system linearly. The graph in Figure 6.5 shows the effect of this scaling on the relative increase in the number of messages during the planning phase of the RMS. For the variants of the `CBS` market, we consider the number of messages sent and received by the centralized broker. For the decentralized `DAS` market, we consider the maximum number of messages sent out and received on a single provider or consumer agent over all bidding rounds. The message load in both the `DAS` and `CBS` market with random provider selection scales almost linearly with the market size. Since the `CTD` heuristic generates a full ordering on the set of providers, it scales quadratically. This is also true for all the other variants of the `CBS` market that use a non-randomized provider selection policy.

In order to empirically test the scalability of a combinatorial auction-based approach such as the one described in [93], we have also translated the scenario under study into the `MACE` [93] bidding language that is supported by the jCase [93] tool. We used the open source LPSolve library for solving the WDP of the combinatorial auction used by jCase. While `MACE` allows for co-allocation of multiple resource types for a single job, it does not permit the co-allocation of multiple jobs. The language also disallows the distribution of A_i over a flexible number of time steps in the planning window of the RMS as well as variations in the capacity of allocations over these steps. Because of these restrictions, the load of application i (L_i), has to be distributed over a chosen fixed number of steps l_i with a fixed capacity $\frac{L_i}{l_i}$. We tried to sidestep the limitations of this approach by including a limited number of XOR bids for each application, varying l_i. When scaling down our scenario with a factor of 20 however, jCase was still unable to deliver results after 12 hours. Recently, work by Stößer [162] has also pointed out the limited scalability properties of the `MACE` scheme. While the restrictions imposed by `MACE` may be needed to keep the algorithm tractable, we believe they currently limit its general applicability.

6.6 Deployment

Although the results presented in this chapter have been obtained by means of simulation, it is important to assess to what extent our algorithms can be deployed on real-world grid infrastructures. More specifically, the algorithms rely on a number of capabilities that the grid middleware must provide. This section briefly discusses the most important requirements in this regard.

6.6.1 Accounting

The middleware should provide for accounting mechanisms that track resource usage and enable the entities participating in the market to be informed about contract compliance. Dedicated monitoring services can inform consumers, brokers and providers whether parties live up to their contractual obligations. In addition, a bank component should offer transactional support for handling payments and responding to account balance queries. The brokers or providers might want to check whether a user's account balance is not depleted before launching the user's workload. Implementations of banking and accounting components can be found in the GridBus toolkit [229].

Both DGAS [54] and SGAS [55] also provide these components. However, they tend more towards usage metering and pay-as-you-go schemes with limited support for implementing price negotiation and market mechanisms. While SGAS is aimed at the management and enforcement of grid-wide capacity allotments [55], DGAS does offer the possibility to implement more dynamic pricing models to obtain load balancing [230]. Both in DGAS and SGAS, the accounting amounts to checking whether the user has enough budget to pay for a job before it is accepted into the system, reserving the necessary amount on the user's account and charging it when

the job is finished. This fits well into the traditional job submission process, but it means that the user or its agent is not actively involved in price negotiation.

The development of an economic framework more in line with our contribution is being undertaken by the European SORMA [231] project. This project aims to create a framework for realizing self-organizing resource management through the use of market-based approaches.

6.6.2 QoS guarantees at the local resource management level

In the presented market mechanisms, resources are traded under the contractual obligations of adhering to a deadline for completing a workload. If the market is to operate successfully, local resource management systems must provide for mechanisms that offer sufficient QoS to enable deadline adherence and contract compliance. More specifically, the $\texttt{CBS}_{\texttt{RND-*}}$ and `DAS` algorithms require the local resource management system to be able to:

- Respond correctly to the following question; Can a workload with expected running time p_{ij} be executed before time d_i?
- Ensure that upon acceptance of new workloads, no running contracts are violated.

The `CBS` markets with `CTD`, `MIN` or `MAX` provider selection policy additionally require the local resource management system to be able to:

- Answer queries related to the amount of free capacity in specific time intervals, to create an ordering among providers.

As such, our algorithms respect the fact that grid resources are under the local control of site administrators and do not impose a specific local resource management policy, or require full control over local resources.

Advanced local resource management systems such as MAUI [72] include advance reservation functionality in support for these requirements.

6.6.3 UI configuration and job description

A number of extensions have to be made to client-side tools such that users can communicate their bid information to economic-aware providers and brokers. This can be achieved by an extension of the JDL language with job attributes for cost and deadline constraints. User interface nodes in the grid must be reconfigured to be able to submit jobs to a QoS and market-aware broker or provider. In addition, client-side tools should be provided that allow a user to obtain information on the (historical) prices formed in the market in order to support the user's valuation process.

6.7 Summary and Conclusions

Introducing economic principles in grid resource management is a promising way for building grid systems that maximize value for individual users. Although spot resource markets are suitable to resource management systems that operate without advance reservations, they are not applicable if the RMS is to support deadline-based QoS requirements. We have developed a centralized and a decentralized algorithm for economic resource management that use advance reservation and co-allocation in order to deal with the problem of bidder exposure. Both algorithms have been shown to compare favorably to a non-economic RMS that supports deadline requirements. Several existing approaches to deadline-based economic resource management require workload preemption and migration for their operation. Because workload preemption, and especially workload migration, can incur high overhead costs in a grid setting, we have chosen not to impose this requirement. We have quantified the consequence of this choice on the realized consumer value. In addition, we have investigated the impact of various policies for scheduling in the application workload. For the centralized market organization, a random provider selection policy delivers the best results, especially when taking messaging complexity into account. In the decentralized market organization, messaging complexity is an even stronger motivation for preferring a randomized selection strategy.

CHAPTER 7

Hybrid Markets

7.1 Introduction

As we discussed in chapter 2, one of the key challenges in realizing efficient grid systems is the development of well-performing resource management and scheduling systems. Many researchers have focused on the development of resource management systems that adopt system-centric optimizations. Through such optimizations, one tries to maximize the total throughput of the grid system or minimize the overhead caused by transporting job input and output data over the network for example [50]. In times of resource contention however, such approaches offer limited means for maximizing the aggregated value that the grid system delivers to its participants. This is a direct consequence of the fact that they have limited means for allowing users to express their valuations for service, and therefore cannot act upon this information. As a result, the process of selecting which loads to accept for execution in times of congestion is suboptimal and the full potential of the grid's infrastructure is not realized.

As many grid applications can inject significant (and potentially unbounded) loads in the system, resource contention is expected to be the norm in grid systems rather than the exception. This is especially the case if no mechanism is in place that incentivizes users to limit their volume of requests to the system. Accounting systems that have recently been deployed on large scale systems such as EGEE's grid are a first step in posing such limits, provided that they are coupled to access control mechanisms. Although such an approach might limit users in their ability to overload the system, it also requires difficult decision making in terms of assigning rights to individual users or user groups. In addition, users have no incentive to back off their loads during periods of congestion as their credits are not valued dynamically by the resource management system depending on the current load

situation or the valuations of other users.

As we motivated in chapter 3, the use of market mechanisms that dynamically price grid resources based on the demands formulated by the user population offer means to deal with these issues. Early works focused on the creation of spot markets such as the ones presented in chapter 5, where currently available resources are traded among resource consumers and providers through a market mechanism. These systems allow the RMS to allocate resources based on the reported value they generate for consumers and result in the formation of dynamic prices based on the current demand and supply situation. They also incentivize users to use the infrastructure conscientiously and to formulate their QoS constraints both in line with their own requirements and those of others using the system. However, their limited support for co-allocating multiple resources over an extended time window hinders the valuation process of users that want to reach an application-wide performance goal[1]. This is a result of the exposure problem [131] users face when they have to commit to trades at some time, without guarantees for allocation opportunities or price levels at a later time.

The futures markets that we presented in chapter 6 offer means to deal with this problem. A number of futures markets such as those developed by AuYoung [232] and Chun [78] use *combinatorial auctions* [94] in order to deal with the exposure problem. Combinatorial auctions allow bidders to express valuations that are conditional on the co-allocation of a set of resources over time. Inevitably, such approaches introduce a significant delay in the allocation process. Firstly, this is caused by the inherently complex optimization process necessary to maximize the reported value delivered by the set of accepted requests. Secondly, in light of consumer bids that arrive over time, the bidding process itself needs enough time in order for the mechanism to select the most promising bids. Consider for example Bellagio [232] that allocates resources on the PlanetLab [233] testbed, which is one of the few operational systems that support economic resource management with advance reservations and co-allocations. It uses a scheduling horizon of one week and a combinatorial auction is cleared every four hours to select the optimal set of accepted bids. The CBS market discussed in chapter 6 is another example of a market that clears periodically. While it does not suffer from the computational complexity of solving a winner determination problem in a combinatorial auction, it nevertheless deliberately introduces a delay in order to be able to select the most promising bids.

Although such futures market organizations allow for value maximization in the context of workloads that can be reserved well ahead of their deadline, they fail to capture the value formulated by high priority requests that need immediate execution. Typically, such requests need a smaller amount of resources but have more stringent turnaround requirements. In addition, some resource management systems offer support for advance reservations while others do not. This brings about additional complexity when trading resources that are not under the control of a resource manager that supports advance reservations, in a futures market.

[1] e.g. finishing a simulation which consists out of 10,000 individual jobs before tomorrow morning

The key contribution of this chapter is the presentation and evaluation of a hybrid market approach that integrates spot and futures market mechanisms to tackle these problems. We also present simulation results that demonstrate how the combination of these two mechanisms leads to a significant increase in the value realized by the grid system.

7.2 Related Work

Limited work has been done to date on developing and evaluating hybrid, multi-market approaches. Neumann [234] motivates the adoption of a multi-market approach, advocating a two-tiered market structure. The theoretical framework includes a first market tier that is targetted towards trading commodity resources on a raw resource market, while a second tier is directed towards negotiations for access rights to higher-level services. Services negotiate on the first tier market to obtain raw resources for fulfilling their requests. Our approach differs from this work in that we propose a multi-market approach for a *single* market tier in order to support a wider range of user requests efficiently. In this respect, both approaches can coexist.

Recently, Courcoubetis et al. [235] presented an adaptation of a continuous double auction for matching bids and asks that incorporate constraints on timely delivery and provisioning of resources. Bids and asks that are sent to the auctioneer for (close to) immediate delivery of resources are called *spot* bids/asks, while *future* bids and asks relate to service delivery at a later point in time. Both types of bids and asks are put in two distinct queues for matching. Similarly to our model in chapter 5, Courcoubetis et al. also consider applications with hard deadline constraints.

However, they do not show how well the presented auction mechanism performs. In addition, the application workload is considered to either have a fixed degree of parallellization throughout the entire execution timeframe, or is taken to be fully preemptible and migrateable at zero cost. Finally, consumer bids cannot be matched with multiple provider bids at the same point in time, i.e. the application workload cannot be parallellized over multiple providers. Unlike the model that we will present in this section, provider strategies for maximizing revenue by dynamically adjusting supply to the spot and futures market, are not considered.

7.3 Hybrid market model

In this section we present a hybrid market model that integrates a spot and a futures market. The first subsection outlines the consumer and provider models used in this chapter. We focus on consumers that formulate requests with hard deadlines and CPU bound computational tasks. Consequently our resource model is mainly aimed at computational processing power. The next two subsections discuss the individual

mechanisms for the futures and spot markets. The integration of both in a hybrid market is presented in the last subsection.

7.3.1 Consumer and provider model

The model that we adopt here is similar to the model put forward in chapter 6. We only repeat the key elements here. A set of n consumer agents $C_i (i = 1, ..., n)$ that act on behalf of users will enter the market to obtain resources for the user's workload. Agent C_i is characterized by the tuple $\{A_i, B_i, V_i, d_i, a_i\}$. The agent's budget B_i can be used to acquire resources from the providers in the market for running its application A_i. The agent's goal is to finish A_i which consists of n_i jobs $\{J_{i,1}, ..., J_{i,n_i}\}$ by deadline d_i. Every job has a processing requirement p_{ij} which represents the job's running time on a reference machine M_{ref}. We denote the total workload induced by one agent request on the system by $L_i = \sum_{j=1}^{n_i} p_{ij}$.

We employ a user model with *hard* deadline constraints; if A_i finishes at $e_i \leq d_i$, the generated value $v_i = V_i$, otherwise $v_i = 0$. The total consumer value generated by the allocations made by the RMS is denoted by $V = \sum_{i=1}^{n} v_i$. Also let $V_{MAX} = \sum_{i=1}^{n} V_i$. In the context of this chapter we will equate the valuation the consumer attributes to his request with the amount of budget that he assigns to his bidding agent for acquiring resources, i.e. $V_i = B_i$. Consumer agents will enter the market at their *activation time* a_i.

We assume the agent's application is CPU bound, trivially parallel, and that the execution of a job cannot be preempted.

The m provider agents P_k $(k = 1, ..., m)$ in the market environment each host a set of m_k uniform parallel [216] machines $\{M_{k,1}, ..., M_{k,m_k}\}$. The runtime of a job J_{ij} on a machine M_{kl} is $\frac{p_{ij}}{s_{kl}}$, with s_{kl} the speed of machine M_{kl}. For the remainder of this chapter we assume s_{kl} to be expressed relative to the speed of M_{ref}. The total processing capacity of the grid's infrastructure is denoted by $S = \sum_{k=1}^{m} \sum_{l=1}^{m_k} s_{kl}$.

7.3.2 Spot Market

For the spot market, we adopt a commodity market model in line with the model presented in chapter 5. In general, several classes of commodity resources can be defined in the market. In chapter 5, CPU resources are assigned to different commodity classes based on their peak performance. The class with index i is then associated with a performance ratio pr_i that expresses the average increase in performance (relative to M_{ref}) a consumer can expect from a CPU resource in that class. Given price vector P, the consumers have to decide on the volume of demand they are willing to express for each class. For the remainder of this chapter we will use one commodity class which will cover all CPU resources. The price P_0 set for this class fixes the price per time unit of a CPU with a speed equal to M_{ref}, i.e. $pr_0 = 1$. The actual price charged for a CPU M_{kl} upon allocation is $P_0 * s_{kl}$.

In the context of this chapter, consumers in the spot market are characterized as having a short hard deadline and a limited amount of workload to run. In view of this, we use a strategy for formulating demand that minimizes the consumer's exposure risk. Equation 7.1 describes the demand response of consumer agent C_i at

time t as a function of P_0.

$$D_i(P_0) = \begin{cases} n_i & \iff B_i \geq P_0 \times \frac{L_i}{pr_0}, \forall k \in \{1, \cdots, n_i\} : t + \frac{p_{ik}}{pr_0} < d_i, \\ 0 & \text{otherwise} \end{cases} \tag{7.1}$$

The agent will thus only express demand if the suggested price level allows it to buy enough resources for all of its jobs immediately and if jobs are expected to finish within deadline. This bidding strategy virtually eliminates the risk of exposure and differs from the rate-based spending strategy used in chapter 5. Note however, that this model differs from the one in chapter 5 in that a user must be able to estimate the runtime of its workload in order to apply equation 7.1.

An advantage of the commodity market model in the context of user requests with deadline constraints is that users can signal the optimizer that they need a minimum amount of resources for adhering to their deadline. In case a suggested spot level price is too high to achieve the necessary degree of advancement in the computation, the bidding agent can formulate zero demand. In spot market organizations such as single-unit auctions or one-on-one bargaining, all trades are individual which increases the risks for exposure.

7.3.3 Futures Market

The futures market operates through the *Centralized Brokering System* presented in chapter 6, with the `MIN` policy for both provider and local resource selection ($\texttt{CBS}_{\texttt{MIN-MIN}}$). In this chapter, we consider a more general setting in which the CBS market clears periodically over time, instead of only considering one clearing phase for requests that are formulated at a given point in time.

The futures market clears with a periodicity of c time units. A consumer therefore has to wait until the next market clearing time in order to receive feedback on the acceptance of its request. At clearing time, the broker prioritizes all outstanding requests in decreasing order of b_i. It then greedily handles requests one by one in sorted order, trying to co-allocate resources for the individual jobs of A_i under the constraint of meeting the request's deadline. A rolling window is used that allows the broker to allocate resources up to H time units in the future, starting from the time of market clearing t_c. We call the interval $[t_c, t_c + H]$ the broker's *planning window*. If the interval $[t_c, t_c + c]$ is not fully occupied on some resources in the resulting schedule, the local resource manager will shift loads to the front of the window in order to minimize underutilization of resources. The use of a limited planning window is key to the efficient operation of a market in which requests arrive continuously over time. Otherwise the resource management system is exposed to the risk of not realizing the maximal value, as early accepted loads with lower value might block high value loads that arrive at a later point in time. In practice, a balance needs to be found between allowing enough flexibility in terms of being able to accept large workloads and late deadlines versus mitigating the risks associated with premature allocation of resources due to incomplete information. Here, we consider the size of the window to be fixed at H. A useful extension to this work would be to tune the window size automatically based on current market conditions and job characteristics.

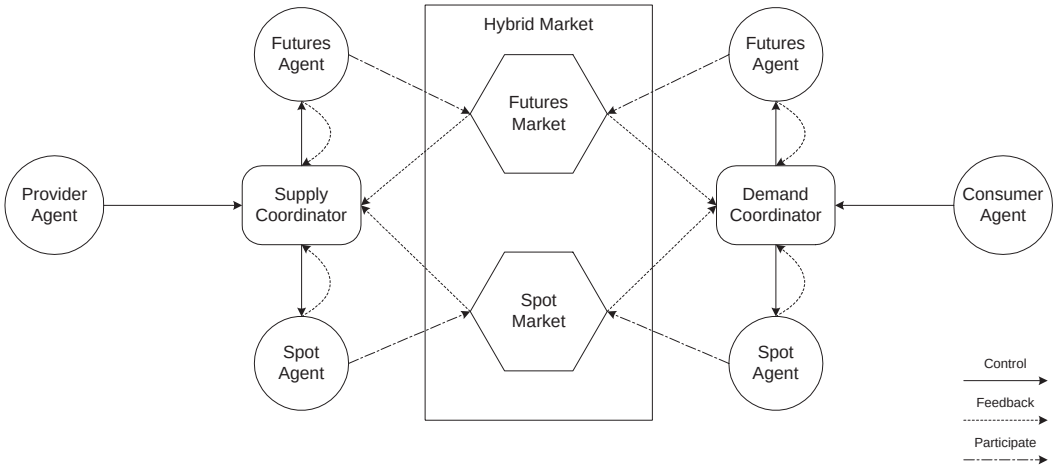

Figure 7.1: Overview of the hybrid market organization

In order to establish prices for the accepted workloads, we use the second pricing approach presented in chapter 6. Let $R_{acc,t}$ denote the set of all accepted requests at time t and $R_{rej,t}$ the set of rejected requests at time t. Let R_i be an accepted request, i.e. $R_i \in R_{acc,t}$. For a CPU resource M_{kl} allocated by C_i at time t, C_i will pay a price $p_{t,i}$ given by equation 7.2.

$$p_{t,i} = \min(\max_{j:R_j \in R_{rej,t} \text{ and } t \in [a_j,d_j]}(b_j), b_i) * s_{kl} \tag{7.2}$$

This second pricing approach motivates users to bid their true value for a request as the cost charged for fulfilling that request does not depend on their own bid level. Note that we take the price to be the minimum of the consumer's bid level and the highest rejected bid. A request R_j with a bid level $b_j > b_i$ and with $[a_j, d_j] \cap [a_i, d_i] \neq \emptyset$ may arrive after R_i is accepted due to our rolling window approach. If R_j is rejected we need to cap $p_{t,i}$ at b_i as to ensure that C_i does not go over budget. This is necessary in order to ensure individual rationality of the market mechanism.

7.3.4 Integration

Figure 7.1 indicates how both market mechanisms are integrated in a hybrid market environment. Providers register agents with both markets for selling their resources. Provider agents delegate bidding and trading responsibilities to futures and spot market agents. The provider's supply coordination service dynamically adjusts the amount of resources both agents control, based on inputs from both market environments and feedback from the agents. Likewise, consumers also host a demand coordination service that decides on the amount of workload that is to be processed through resource acquisitions in both markets. In the remainder of this chapter however, we will focus on the dynamic provisioning of the spot and futures markets and assume that each consumer directs its full workload to either the spot or the futures market.

Dynamic division of supply

The provider's supply coordination service will use the following rules to determine the level of resource supply to both agents:

- At the start of a futures market round, the provider calculates the average revenue earned on the spot market per normalized CPU time sold over the previous $w_{revenue}$ time units. It then puts all the CPU resources under the control of the futures market agent for selling the resources in the futures market. The provider requires that the agent sells off the resources for a reserve price which equals this average price.

- If at the end of the futures market round there are resources under the futures market agent's control that have not been sold, the spot market agent gains control over these resources.

- In between futures market rounds, resources are moved from the future to the spot market if market prices indicate higher revenues can be earned on the spot market. In this regard, providers consider the revenue earned on the futures market (rev_{fut}) and the spot market (rev_{spot}) over the last two time units. The probability that one CPU is shifted towards the most profitable market is given in 7.3.

$$p_{shift} = \frac{max(rev_{fut}, rev_{spot})}{rev_{fut} + rev_{spot}} \tag{7.3}$$

  This probabilistic shift mitigates the supply volatility that results when agents uniformally shift all their supply to the market with the highest current revenue. In the event that the revenue in one of the markets is zero, we substitute the revenue in equation 7.3 with the equilibrium price on the spot market for rev_{spot}, or the highest rejected bid on the futures market for rev_{fut}. This is done to ensure that supply to that market does not shut down permanently after a period of zero revenue.

Resource transfers

After supply levels to both agents have been determined, the transfer of an individual resource proceeds by the following rules:

- **Spot to futures market** Control is immediately transferred to the futures market agent. If the resource is executing a job at the time of futures market clearing, the futures market agent is allowed to accept allocations on the resource as long as they do not interfere with the currently running job. Note that this requires an estimate on the job's running time.

- **Futures to spot market** If there are no workloads scheduled in on the resource, control is transferred immediately to the spot market agent. In case a job is currently running on the resource but no future workloads are scheduled in, the spot market agent gains control over the CPU but will wait for the currently executing job to finish before selling the resource on the spot market. In

the event that future workloads are planned in on the resource, the provider's resource manager attempts to reschedule these to other local resources at his disposal. If it is not possible to reschedule these loads, the transfer will not succeed.

With respect to the second rule we need to ensure that there is enough liquidity in terms of available CPU resources in order to be able to react to the price dynamics in both markets. Therefore, the provider's resource manager will use an extended planning horizon $H_{ext} > H$ when reorganizing loads to provide for more rescheduling opportunities.

7.4 Evaluation

In this section, we investigate the market's operation and the value increase that can be realized by using a hybrid market approach compared to a market organization that only uses a futures market.

7.4.1 Simulation Configuration

In the base configuration for our simulation scenario we take $n = 2050$, $m = 20$, $S = 1250$, $s_{kl} \in [0.9, 1.1]$, $H = 100$, $H_{ext} = 200$ and $w_{revenue} = 50$. In the spot market, market prices will be adjusted by the optimizer in each simulated time step. The futures market will clear every $c = 25$ simulation steps. To illustrate the operation and value realization in a hybrid market setting, we inject a set of consumers with high volume requests in the futures market. The parameters for these consumers are given by the first three groups in table 7.1[2]. In addition, we inject 1750 high priority consumers into the market over a period of 450 simulation steps according to the parameter profile of the fourth group in table 7.1. Each consumer from this group submits between 15 and 45 jobs of a normalized execution time between 4 and 10 simulation steps. All jobs need to finish within 11 simulation steps of the consumer's activation time. The workloads of the consumers in the fourth group will be directed to the spot market because of their tight deadline constraints.

A fifth group of users are bargain hunters on the spot market. Their valuations are considerably lower than users from other groups at $b_i \in [175, 200]$. If there is excess supply on the spot market, these consumers will take advantage of this. Note that the high priority consumers do not have enough workload to fully load the system, hence oversupply to the spot market could lead to zero prices in absence of these bargain hunters.

[2]For parameters that are specified with a range, we draw values from a uniform random distribution

Table 7.1: Parameter profiles for the different consumer groups in the simulation that determine for each consumer C_i, the number of jobs n_i and the normalized runtime per job p_i, as well as C_i's bid level b_i, deadline d_i and activation time a_i.

Parameter	Group I	Group II	Group III	Group IV
Group Size	100	100	100	1750
n_i	$[140, 260]$	$[140, 260]$	$[140, 260]$	$[15, 45]$
p_i	$[2, 20]$	$[2, 20]$	$[2, 20]$	$[4, 10]$
b_i	$[200, 1400]$	$[200, 1400]$	$[200, 1400]$	$[400, 2000]$
d_i	$[100, 200]$	$[250, 350]$	$[350, 450]$	$a_i + 11$
a_i	$[10, 50]$	$[100, 150]$	$[200, 300]$	$[1, 450]$
Market	Futures	Futures	Futures	Spot

7.4.2 Market Operation

We evaluate whether the market responds correctly to input signals from the provider and consumer population. The top graph in Figure 7.2 shows the price evolution in both submarkets of the hybrid market. We have plotted the average price paid by consumers for their currently running jobs at each simulation step.

In the first steps of the simulation there is no congestion on the futures market. As a result prices remain at the zero level. However, as more consumers enter the futures market supply shifts towards this market (as shown in the bottom graph). This also results in a higher level of congestion and higher resulting price levels on the spot market. Consequently, market prices that reflect the valuations of the consumers in both markets are formed after the second clearing of the futures market in step 50, and are maintained up until simulation step 425. This is where demand starts to taper off quickly resulting in a significant price reduction in both markets. As demand in the futures market diminishes in step 425, non-utilized CPUs are automatically shifted to the spot market where bargain hunters pick them up at prices that reflect their valuations.

To demonstrate the reaction of the provider agents in response to sudden demand surges in one of the markets, we reconfigure the second group from table 7.1 with $b_i \in [2000, 4000]$ and $n_i \in [210, 390]$. The resulting price and supply levels are shown in Figure 7.3. As can be observed from the graphs, the provider agents respond to the demand surge by gradually shifting their supply to the futures market. As a result, the maximum number of requests from the second consumer group are executed successfully. After the high value demand on the futures market drops, supply is returned to the spot market and the system returns to the supply levels observed before the demand surge.

7.4.3 Value realization

We compare the performance of our hybrid market with a pure futures market approach in terms of realized user value. For the tests with the pure futures market, we retain the configuration from table 7.1, with the exception of redirecting the

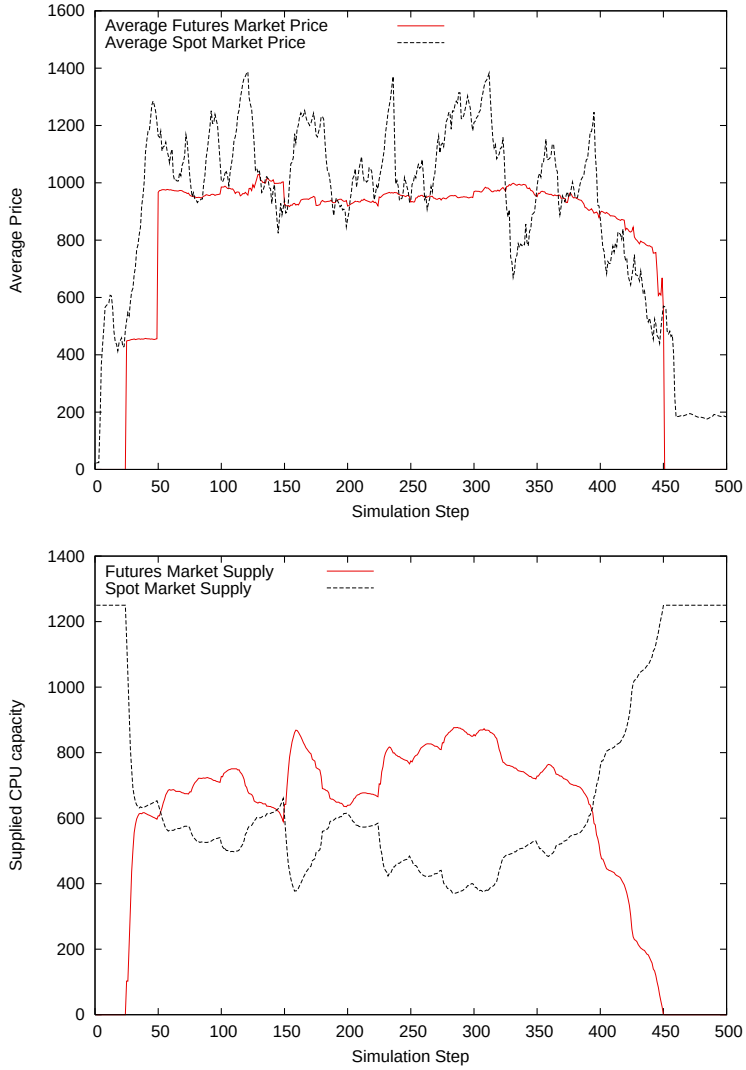

Figure 7.2: Spot and futures market prices (top) and supplied processing capacity (bottom)

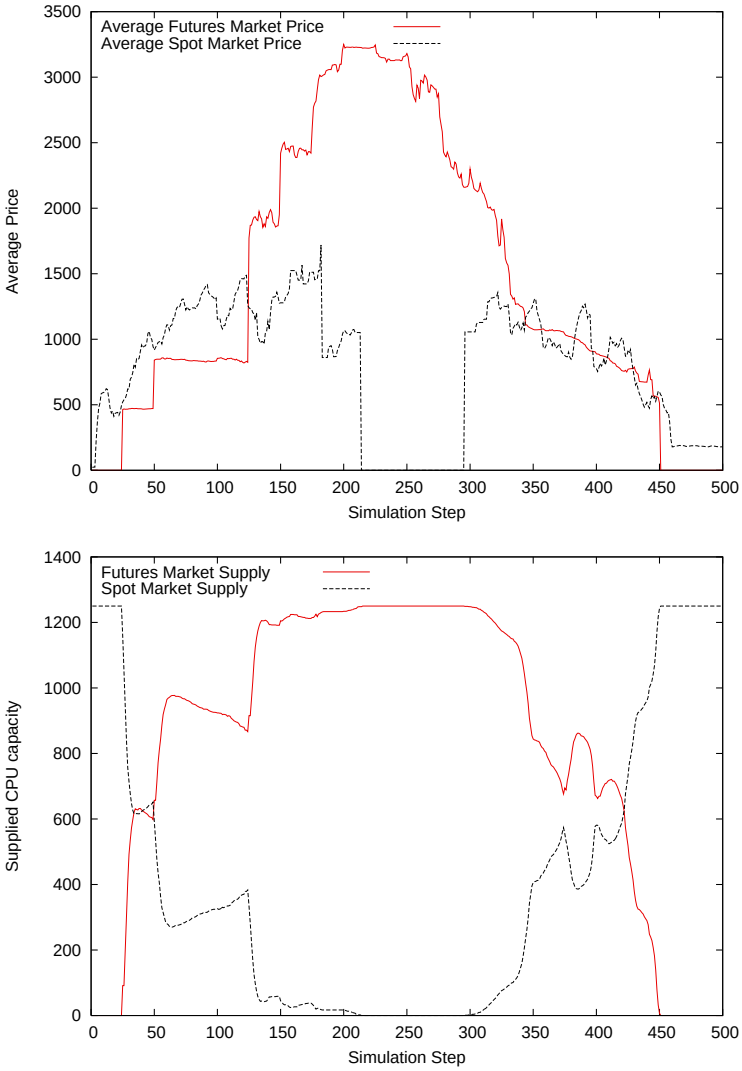

Figure 7.3: Prices (top) and normalized supply (bottom) in the spot and futures markets under a demand surge

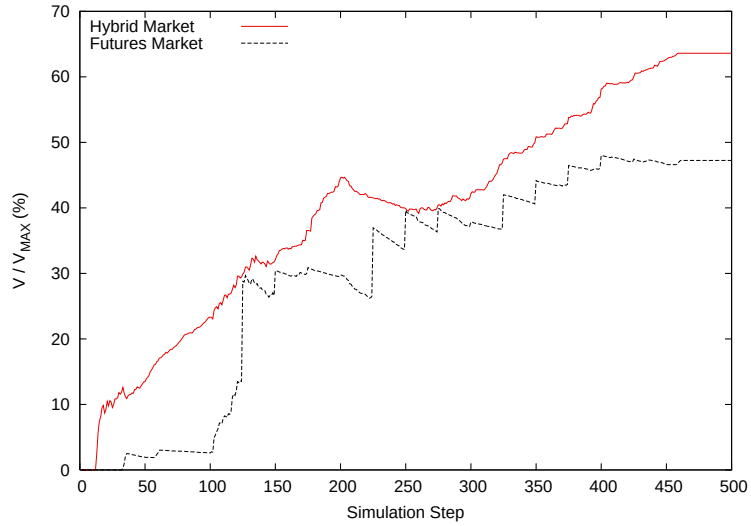

Figure 7.4: Percentage of consumer value realized over the course of the simulation in the hybrid and the futures market

loads from user group IV to the futures market instead of the spot market. We also do not include a set of bargain hunters. The contribution of user group IV in table 7.1 to the maximal aggregated user value that can be realized is 50%. The bargain hunters in the hybrid market are valued at zero to keep results comparable.

The graph in Figure 7.4 shows the percentage of maximal user value realized under the hybrid market and under the pure futures market over the course of the simulation. At the end of our simulation, a difference of 16.4% (47.2% vs 63.6%) can be observed for this scenario in favor of the hybrid market. The difference increases at the end of the simulation as the hybrid market takes the opportunity to accept high priority loads on the spot market. These loads can be accepted without having to drive out workloads on the now more lightly loaded futures market, resulting in a higher contribution to the realized user value.

Finally we point out that we have consistently observed that the hybrid approach results in additional value, but that the extent of the value increase depends on the setting at hand. In general, the valuations, workload characteristics and deadline constraints of consumer agents in both the high priority and low priority groups are determining factors in this regard.

7.5 Future Work

One of the main opportunities for future work is to investigate the possibility for consumer agents to dynamically direct their workload to the different markets based on current market conditions. In addition, we plan to evaluate whether more intelligent rules for directing supply to the two markets can further increase efficiency

and reduce price volatility on the spot market. In this respect, we are also considering the use of trading brokers that direct demand and supply to both markets on behalf of consumers and providers. Finally, we are considering further developments towards a two-tiered market structure in which agents that sell high-level services on the second tier need to negotiate for access rights in the first tier of the market.

7.6 Summary and Conclusions

Since user valuations for grid service delivery are closely linked to timing requirements and guarantees, the introduction of a futures market in which resources can be traded and allocated ahead of time is key to a smoothly operating market. Although the benefits of installing futures markets such as the ones described in chapter 6 are clear, their major drawback is the introduction of a significant delay between the formulation of a request by a user and the earliest time at which the request's execution can start. For users with short deadlines and limited workloads this is unacceptable. We have presented an integration of spot and futures market mechanisms in order to tackle this shortcoming. The resulting hybrid market organization combines low-latency resource allocation and trading for adhoc workloads with the ability to reserve resources for workloads that require longer term QoS guarantees. Providers adopt a revenue maximizing strategy to dynamically shift supply between both markets. Experimental results demonstrate that compared to a pure futures market approach, significant efficiency increases can be obtained as a result of the increased flexibility delivered by a hybrid market. In addition providers are shown to react appropriately to demand surges in one of the markets.

CHAPTER 8

Conclusions

Grid computing holds a major promise in transforming the Internet as we know it today to a platform for using and sharing a wide array of services. New challenges in science have driven software development projects to realize the vision of grid computing in order to enable the sharing of computational resources across organizational boundaries. These projects have delivered the technical foundations for grids to be deployed but have made limited considerations for developing a sustainable sharing proposition. As a consequence, current grid systems offer limited openness and cannot realize the envisioned on-demand computing model that would transform IT resources into utilities.

The introduction of economic principles in grid resource management through a market-based approach can deliver the necessary incentives for providers to share resources and for users to use these shared resources in a well-considered manner. The distributed nature of decision making that follows from this approach, can better account for diverse and changing preferences and goals compared to an approach that uses centralized control.

We have shown how spot markets can provide an efficient distribution of access rights among the user population based on the demand and supply of resources at a specific point in time. The low-latency and iterative nature of spot markets allow for an immediacy in establishing resource allocations. We have demonstrated how both a commodity and single-unit Vickrey auction market can set dynamic resource prices that are tailored to time-varying demand and supply levels, so as to obtain an optimal allocation of resources among the user population. For the commodity market, an extension of Smale's algorithm has been proposed and evaluated in order to deal with the specific properties of the presented grid resource market. We have also shown that the distributed decision making process in a spot market automatically leads to users obtaining a share of the infrastructure that reflects their valuations relative to other competing users in the environment.

In the development of a resource market, one can decide to model resources at an

abstract level and consider them as fully interchangeable commodities, or consider each resource to be unique. This results in either global commodity prices being set for resources, or prices being negotiated for individual resources respectively. The two spot market organizations that we have investigated are both at opposite ends of this spectrum. Although differentiated pricing can be obtained in the commodity market by introducing multiple substitutable goods, there are practical limitations with respect to the tractability of the market's price formation process in light of a large number of such goods. In practice, the inherent heterogeneity of resources which follows from their locality for example, cannot be taken into account through multiple categories due to the complexity of the equilibrium price optimization problem. The single-unit Vickrey auction market does allow for a valuation of the individual characteristics of a resource, at the cost of lower price stability and more complex bidding logic.

In order to support more advanced QoS requirements, we have developed a futures market organization that allows for resources to be coallocated over time. This protects users from the financial exposure of having to pay for a partial fulfillment of their requirements. We have discussed the limitations of using combinatorial auctions to achieve this goal and presented both a centralized and decentralized futures market organization that use advance reservation and co-allocation, as an alternative. A comparison with a non-economic scheduling algorithm highlighted the potential for value increase that economic forms of resource management can bring. A key characteristic of our approach is that we do not make the problematic assumptions of workloads being infinitely parallellizable or preemptable at zero cost, but still allow for the market mechanism to establish an allocation of resources to workloads in a tractable manner.

A major drawback of a futures market model is the introduction of a significant delay between request formulation and request acceptance and execution. This is unfortunate for users that have very short deadline requirements and limited workloads to run. We have presented a hybrid market organization that integrates spot and futures market mechanisms in order to combine low-latency resource allocation and trading for adhoc workloads with the ability to reserve resources for workloads that require longer-term QoS guarantees. Our experiments demonstrate that significant efficiency increases can be obtained as a result of the increased flexibility delivered by a hybrid market. Moreover, we presented a supply-side strategy for providers to dynamically determine their supply on the spot and futures markets. This strategy allows providers to react appropriately to demand surges in one of the markets.

Studying the performance of resource management schemes through real-world deployment is costly and offers limited flexibility. The bidding process in market-based resource management systems further complicates the execution of large-scale experiments in the real-world. In support for efficient simulation-based studies for market-based resource management schemes, we have developed the Grid Economics Simulator. It supports both discrete-time as well as discrete-event simulation and includes provisions for running experiments in a distributed manner, as well as real-time graphical analysis of the simulated grid environment through a wide range

of metrics. The design of the simulator's discrete-event core has shown how the combination of Aspect Oriented Programming, Java annotations and the use of continuations contribute to an efficient simulation core and a clean programming model. A performance study that compared the performance of the GES simulation cores with GridSim and SimGrid has demonstrated their efficiency for simulations that focus on resource management for computing resources.

Appendices

APPENDIX A

List of Symbols

Abbreviation	Meaning
C_i	Consumer i with $1 \leq i \leq n$
A_i	Application of C_i consisting of n_i jobs
B_i	Budget of C_i to execute A_i
V_i	Value associated with the successful execution of A_i
d_i	The deadline by which A_i needs to be executed to generate V_i
b_i	C_i's bid per time unit of execution of its application with $0 < b_i \leq \frac{B_i}{L_i}$.
R_i	The consumer's bid defined by the triplet $\{A_i, b_i, d_i\}$
J_{ij}	Job of application A_i with $1 \leq j \leq n_i$.
p_{ij}	Processing requirement of job J_{ij} on a reference architecture M_{ref}
L_i	The total workload introduced in the system by C_i defined as $\sum_{j=1}^{n_i} p_{ij}$
W_{A_i}	The execution window of A_i defined as $[now, d_i]$
e_i	The actual finish time of A_i
v_i	The actual generated value for C_i defined as V_i if $e_i \leq d_i$ or 0 otherwise
P_i	The actual price paid by C_i for the execution of A_i
U_i	The generated utility for C_i defined as $V_i - P_i$
V	The generated value for all consumers defined as $\sum_{i=1}^{n} v_i$

Abbreviation	Meaning
V_{MAX}	The potential generated consumer value defined as $\sum_{i=1}^{n} V_i$
P_k	Provider k with $1 \leq k \leq m$
M_{kl}	Uniform parallel machine of P_k with $1 \leq l \leq m_k$
s_{kl}	The speed of M_{kl}
h_k	The scheduling horizon of provider P_k
S	The total processing capacity of the system
V_{base}	A chosen base valuation attributed to a consumer's request
VF_{load}	A valuation factor tied to L_i
$VF_{deadline}$	A valuation factor related to d_i
AF_i	Allowance factor for consumer group i
AS_i	Allocation share of consumer group i
BS_i	Budget share of consumer group i

APPENDIX B

Glossary of Economic Terms

Term	Meaning
Vickrey Auction	An auction with sealed bids where the winner pays the price that the highest losing participant has bid.
Double Auction	An auction in which both buyers and sellers can submit bids.
The Exposure Problem	A problem that occurs when a participant has negotiated a contract for part of the goods it needs, but does not yet have a contract for the remaining part. As a result, the participant is exposed to the financial risk of having to pay for the former, while not being able to obtain the latter.
Incentive Compatible	A mechanism is incentive compatible if participants in the mechanism fare best when bidding truthfully.
Individually Rational	A mechanism is individually rational if its outcome cannot induce a negative utility for the participating agents in equilibrium.
Tractable	A mechanism is tractable if it is practical to compute its outcome in a reasonable timeframe. In complexity theory, a problem is deemed computationally tractable if it is possible to provide an algorithm that can solve each instance of the problem within a computation time that is bounded by a polynomial function of the size of the instance.

Term	Meaning
Budget Balanced	A mechanism is budget balanced if the mechanism's outcome is guaranteed to result in a situation in which the payments to the sellers are covered by a redistribution of the payments made by the buyers to the mechanism.
Vickrey-Clark-Groves mechanism	A generalization of the Vickrey auction in which each bidder pays the opportunity cost that their presence induces on the other bidders. The opportunity cost for bidder A is determined by subtracting the value of the outcome with bidder A excluded, from the value of the outcome under the full set of participants.
Nash equilibrium	A situation where no participant can benefit from changing its behaviour while knowing the strategies of all the other participants.
Bayes-Nash equilibrium	A situation where no participant is expected to benefit from changing its behaviour while making assumptions on the strategies of all the other participants.
Bayes-Nash incentive compatible	A mechanism is Bayes-Nash incentive compatible if any agent's expected utility maximizing strategy in equilibrium with every other agent is to bid thruthfully.
Posterior regret-free outcome	An outcome of a mechanism is posterior regret-free if after the computation of the outcome, all bids and allocations are communicated to the bidders, upon which no single bidder has an incentive to change its bid, provided that the other bidders would not change their bid.
Winner Determination Problem	The problem of determining the set of winning bids in an auction such that the auction's outcome results in a feasible allocation of items to bidders and revenue is maximized.

APPENDIX C

Dutch Summary

Het Internet heeft een revolutie ontketent in de manier waarop mensen communiceren, informatie delen en zaken doen. Deze revolutie wordt gevoed door wereldwijde standaarden die uitwisseling van informatie mogelijk maken op globale schaal. Naast het delen van informatie, heeft het ter beschikking stellen van diensten (of services) een belangrijke plaats ingenomen in de verdere ontwikkeling van het internet. Zo werd midden jaren negentig de visie geformuleerd van een grid waarin een brede waaier van services worden geïntegreerd in een gedistribueerd systeem, over de grenzen van organisaties en administratieve domeinen heen. In de academische wereld wordt dit model reeds toegepast in het kader van services die reken- en opslagcapaciteit van diverse universiteiten en onderzoeksinstellingen bundelen en ter beschikking stellen voor een selecte groep wetenschappers. Door deze bundeling van middelen (resources), kunnen grootschalige berekeningen die het wetenschappelijk proces ondersteunen, uitgevoerd worden binnen een voorheen niet realiseerbaar tijdsbestek.

De visie van een computationeel grid vindt zijn oorsprong in de analogie met elektriciteitsnetwerken (power grids). De idee is dat rekenkracht en opslagcapaciteit op aanvraag (on-demand) kunnen geleverd worden aan gebruikers over de gehele wereld. De levering van rekenkracht en opslagcapaciteit wordt zo een nutsvoorziening (een utility), net als water, gas en elektriciteit.

Het Large Hadron Collider Compute Project (LCG) is een actueel voorbeeld van een grid systeem. Het maakt het mogelijk voor wetenschappers om de gegevens die worden geproduceerd door de grootste en krachtigste deeltjesversneller ter wereld te beheren en te analyseren. Hiervoor stellen honderden instellingen hun infrastructuur ter beschikking door gebruik te maken van grid technologie en leveren zo rekenkracht en opslagcapaciteit aan duizenden wetenschappers over de gehele wereld.

Hoewel zulke academische grids in de wetenschappelijke context reeds hun nut hebben bewezen, is tot op heden de visie om IT infrastructuur om te zetten in een algemeen beschikbare nutsvoorziening voor een bredere waaier aan gebruikers niet

gerealiseerd.

De belangrijkste reden hiervoor is dat er onvoldoende incentives zijn om infrastructuur ter beschikking te stellen aan een derde partij. De incentives die in de academische wereld aanwezig zijn, zoals samenwerking op een gezamenlijk project, of een opgelegde eis tot delen van infrastructuur vanuit een gesubsidieerd project, ontbreken in de algemenere context. Afspraken omtrent welke gebruikers beroep mogen doen op de gedeelde infrastructuur en hoe ver toegangsrechten voor gebruikers reiken worden dan ook gemaakt op basis van onderling direct overleg tussen de betrokken partijen, wat leidt tot weinig flexibiliteit en openheid.

Verder is er onvoldoende aansporing om gebruikers er toe aan te zetten de beschikbare infrastructuur in een weloverwogen manier te gebruiken. Het ontbreken van een duidelijke notie van kost spoort gebruikers er toe aan om zoveel mogelijk resources te willen occuperen. De dienstverlening in huidige grids wordt dan ook op een first-come-first-serve manier benaderd. Hierdoor worden resources mogelijk niet toegekend aan gebruikers die ze het meest nodig hebben. Als gevolg hiervan wordt het optimale potentieel van de grid infrastructuur, in termen van algemene gebruikerstevredenheid, niet gerealiseerd.

In dit doctoraat wordt het potentieel van de toepassing van economische principes in het beheer van grid resources onderzocht om deze problemen aan te pakken. De aanpak wordt geïnspireerd door het feit dat het probleem om schaarse goederen optimaal te verdelen onder een groep belanghebbenden in onze maatschappij, reeds sinds duizenden jaren aangepakt wordt via een marktwerking. De idee is dat gebruikers hun waardeoordeel uitdrukken voor het verkrijgen van dienstverlening en kenbaar maken aan een marktmechanisme. Het mechanisme zorgt er vervolgens voor dat de gebruikers met de hoogste waarderingen toegang krijgen tot het systeem en berekent hiervoor de te betalen kostprijs. De uitdaging bestaat er verder in om tot een marktwerking te komen die geschikt kan bevonden worden voor de context van grid resource beheer.

In een eerste deel van het doctoraat worden mechanismen voor spot markten onderzocht. In spot markten worden toegangsrechten verhandeld voor resources die op een gegeven tijdstip beschikbaar zijn, met onmiddellijke ingang van de dienstverlening tot gevolg. We tonen hierbij aan dat de toepassing van een marktmechanisme gebaseerd op centrale optimalisatie van het prijsniveau, alsook een marktmechanisme gebaseerd op enkelvoudige veilingen, de prijs van resources correct aanpast aan de vraag- en aanbodverhoudingen in de markt. Dit leidt tot een quasi optimale verdeling van toegangsrechten onder de populatie gebruikers die beroep doen op de grid infrastructuur. Er zijn evenwel verschillen tussen beide markt organisaties, onder andere op niveau van prijsstabiliteit en communicatieve en computationele complexiteit, die in dit doctoraat in kaart worden gebracht.

Een nadeel van spotmarkten is evenwel dat zij onvoldoende ondersteuning kunnen bieden voor gebruikers die garanties willen met betrekking tot de beschikbaarheid van resources in de toekomst voor een bepaalde prijs. De vraag naar zulke garanties vloeit voort uit de wens om een volledige applicatie te kunnen beëindigen voor een bepaalde deadline in de toekomst, tegen een vooraf gekende kost. We hebben zowel een gecentraliseerd als een gedecentraliseerd mechanisme ontworpen

en geëvalueerd voor een termijnmarkt om tegemoet te komen aan zulke eisen.

De voorgestelde termijnmarkten brengen echter een zeker vertraging met zich mee tijdens het beheersproces van grid resources. Daarom werd ook een gecombineerd mechanisme uitgewerkt waarin spot- en termijnmarkten met elkaar coëxisteren. Deze combinatie heeft als voordeel dat ze de inherente vertraging die optreedt bij het gebruik van een termijnmarkt kan neutraliseren door gebruikers en leveranciers de mogelijkheid te bieden om met onmiddellijke ingang resources te verwerven op de spot markt, terwijl vraag naar dienstverlening met een garantie op niveau van de applicatie nog steeds kan voldaan worden op de termijnmarkt.

Het evalueren van resource beheer systemen voor grids op bestaande operationele systemen is uiterst moeilijk. Het grootschalig en productierijp karakter van zulke systemen laat experimentatie niet toe of zorgt voor een zeer hoge kost. Om tot een kost- en tijdsefficiënte evaluatie te komen van beheerssystemen voor grids werd in het kader van dit doctoraat simulatie software ontwikkeld in de vorm van de Grid Economics Simulator. Deze laat toe om wetenschappers op efficiënte en interactieve wijze de performantie en het gedrag van zowel economische als traditionele beheerssystemen voor grids te onderzoeken. Een vernieuwend programmeermodel en een performante simulatiekern laten een vlotte programmatie van gesimuleerde omgevingen toe die op een tijdsefficiënte manier tot uitvoering kan gebracht worden.

Bibliography

[1] "GÉANT2 topology maps." `http://www.geant2.net/server/show/nav.00d00b002005`, January 2008. [Accessed 01-10-2008].

[2] I. Foster and C. Kesselman, "The grid in a nutshell," in *Grid Resource Management - State of the Art and Future Trends* (J. Nabrzyski, J. M. Schopf, and J. Weglarz, eds.), ch. 1, pp. 3–15, Kluwer Academic Publishers, 2004.

[3] I. Foster, C. Kesselman, and S. Tuecke, "The anatomy of the grid," *Int. J. of High Performance Computing Applications*, vol. 3, p. 15, 2001.

[4] B. Sotomajor and L. Childers, "OGSA, WSRF, and GT4," in *Globus Toolkit 4 - Programming Java Services*, Elsevier Series in Grid Computing, ch. 2, pp. 13–17, Morgan Kaufmann / Elsevier Publishers, 2006.

[5] K. Lai, "Markets are dead, long live markets," *SIGecom Exch.*, vol. 5, no. 4, pp. 1 – 10, 2005.

[6] H. Casanova, A. Legrand, and M. Quinson, "SimGrid: A generic framework for large-scale distributed experiments," in *Proceedings of the 10th IEEE International Conference on Computer Modelling and Simulation (UKSIM/EUROSIM'08)*, (Washington, DC, USA), pp. 126 – 131, IEEE Computer Society, 2008.

[7] R. Wolski, J. Brevik, J. Plank, and T. Bryan, "Grid resource allocation and control using computational economies," in *Grid Computing: Making the Global Infrastructure a Reality* (F. Berman, G. C. Fox, and A. J. Hey, eds.), ch. 32, pp. 747–772, John Wiley & Sons, Ltd., 2003.

[8] I. Foster, C. Kesselman, J. M. Nick, and S. Tuecke, "The physiology of the grid: An open grid services architecture for distributed systems integration," in *Grid Computing: Making the Global Infrastructure a Reality* (F. Berman, G. C. Fox, and A. J. Hey, eds.), John Wiley & Sons, Ltd., 2002.

[9] F. Berman and T. Hey, "The scientific imperative," in *The Grid 2 : Blueprint for a New Computing Infrastructure* (I. Foster and C. Kesselman, eds.), ch. 2, pp. 13–24, San Francisco, CA: Morgan Kaufmann / Elsevier Publishers, 2 ed., 2004.

[10] G. Graham, R. Cavanaugh, P. Couvares, A. D. Smet, and M. Livny, "Distributed data analysis: Federated computing for high-energy physics," in *The Grid 2 : Blueprint for a New Computing Infrastructure* (I. Foster and C. Kesselman, eds.), ch. 10, pp. 135–145, San Francisco, CA: Morgan Kaufmann / Elsevier Publishers, 2 ed., 2004.

[11] W. Cottingham and D. Greenwood, *An Introduction To The Standard Model Of Particle Physics*. Cambridge: Cambridge University Press, 2007.

[12] A. Abbas, "Bussiness value of grid computing," in *Grid Computing: A Practical Guide To Technology and Applications*, ch. 3, Hingham, MA: Charles River Media, 2004.

[13] D. Sugimoto, "GRAPE: A parallel computer dedicated to astrophysical many-body problems," *Parallel Computing*, vol. 25, no. 14, pp. 1663–1676, 1999.

[14] "Homepage of the Top500 website." `http://www.top500.org/`, 2008. [Accessed 01-11-08].

[15] D. P. Anderson and K. Reed, "Celebrating diversity in volunteer computing," in *Proceedings of the 42th Annual Hawaii International Conference on System Sciences*, IEEE Computer Society, 2009. To Appear.

[16] L. F. G. Sarmenta, *Volunteer Computing*. PhD thesis, Massachusetts Institute of Technology, June 2001.

[17] D. P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer, "SETI@home: An experiment in public-resource computing," *Communications of the ACM*, vol. 45, no. 11, pp. 56–61, 2002.

[18] M. Shirts and V. Pande, "Screen savers of the world unite!," *Science*, vol. 290, pp. 1903–1904, 2000.

[19] D. P. Anderson, "BOINC: A system for public-resource computing and storage," in *Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing*, pp. 4–10, IEEE Computer Society, 2004.

[20] IBM, "World community grid - technology solving problems." `http://www.worldcommunitygrid.org/index.jsp`, 2008. [Accessed 01-10-08].

[21] G. Gilder, *Telecosm: How infinite bandwidth will revolutionize our world*. Free Press, 2000.

[22] G. Gilder, *The Gilder Technology Report*. Gilder Publishing, LLC, June 2000.

[23] D. F. McMullen and T. Devadithya, "Integrating instruments and sensors into the grid with cima web services," in *Proceedings of the Third APAC Conference on Advanced Computing, Grid Applications and e-Research (APAC05)*, 2005.

[24] G. Aloisio, G. Quarta, D. Conte, C. Elefante, G. Marra, and G. Mastrantonio, "SensorML for grid sensor networks," in *Proceedings of the 2006 International Conference on Grid Computing and Applications (GCA'06)*, pp. 147–152, CSREA Press, 2006.

[25] F. Breitling, T. Granzer, and H. Enke, "Grid integration of robotic telescopes," *Astronomical Notes*, vol. 329, no. 3, pp. 343–346, 2007.

[26] E. Frizziero, M. Gulmini, F. Lelli, G. Maron, A. Oh, S. Orlando, A. Petrucci, S. Squizzato, and S. Traldi, "Instrument element: a new grid component that enables the control of remote instrumentation," in *Proceedings of the Sixth IEEE International Symposium on Cluster Computing and the Grid Workshops (CCGridW '06)*, IEEE Computer Society, 2006.

[27] S. L. Goldman, R. N. Nagel, and K. Preiss, *Agile competitors and virtual organizations*. New York: Van Nostrand Reinhold, 1995.

[28] L. M. Camarinha-Matos, H. Afsarmanesh, C. Garita, and C. Lima, "Towards an architecture for virtual enterprises," *Journal of Intelligent Manufacturing*, vol. 9, no. 2, pp. 189–199, 1998.

[29] M. L. Bote-Lorenzo, Y. A. Dimitriadis, and E. Gomez-Sanchez, "Grid characteristics and uses: a grid definition," in *Proceedings of the First European Across Grids Conference (AG'03)*, vol. 2970 of *Lecture Notes in Computer Science*, (Heidelberg), pp. 291–298, Springer-Verlag, 2004.

[30] I. Foster, "What is the grid? a three point checklist," 2002.

[31] S. Segaller, *Nerds: A Brief History of the Internet*. New York: TV Books, 1998.

[32] T. A. DeFanti, I. Foster, M. E. Papka, R. Stevens, and T. Kuhfuss, "Overview of the I-WAY: Wide-area visual supercomputing," *International Journal of High Performance Computing Applications*, vol. 10, no. 2-3, pp. 123–131, 1996.

[33] S. Bhatt, M. Chen, J. Cowie, G. Fox, W. Furmanski, and A. Lenstra, "Factoring on the world-wide computer," February 1995. Supercomputing '95 Teraflop Challenge.

[34] "MPICH2." `http://www-unix.mcs.anl.gov/mpi/mpich/`, 2006. Argonne National Laboratory and University of Chicago.

[35] I. Foster, J. Geisler, W. Nickless, W. Smith, and S. Tuecke, "Software infrastructure for the I-WAY high performance distributed computing experiment," in *Proceedings of the 5th IEEE Symposium on High Performance Distributed Computing*, pp. 562–571, 1997.

[36] I. Foster and C. Kesselman., "Globus: A metacomputing infrastructure toolkit," *International Journal of Supercomputer Applications*, vol. 11, no. 2, pp. 115–128, 1997.

[37] D. D. Roure, M. A. Baker, N. R. Jennings, and N. R. Shadbolt, "The evolution of the grid," in *Grid Computing: Making the Global Infrastructure a Reality* (F. Berman, G. C. Fox, and A. J. Hey, eds.), ch. 3, pp. 65–100, John Wiley & Sons, 2003.

[38] I. Foster, H. Kishimotot, A. Savva, D. Berry, A. Djaoui, A. Grimshaw, B. Horn, F. Maciel, F. Siebenlist, R. Subramanian, J. Treadwell, and J. V. Reich, "The open grid services architecture, version 1.5," Tech. Rep. GFD-I.080, Open Grid Forum, July 2006.

[39] T. Erl, *Service-Oriented Architecture - Concepts, Technology, Architecture*. Prentice Hall Service Oriented Computing Series, Upper Saddle River, NJ, USA: Prentice Hall PTR, 6 ed., 2006.

[40] S. Weerawarana, F. Curbera, F. Leymann, T. Storey, and D. F. Ferguson, *Web Services Platform Architecture - SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging, and More*. Prentice Hall PTR, 2005.

[41] M. Li and M. Baker, "OGSA and WSRF," in *The Grid : Core Technologies*, pp. 34–35, John Wiley & Sons, 2005.

[42] T. Banks, "Web services resource framework (WSRF) - primer v1.2." `http://docs.oasis-open.org/wsrf/wsrf-primer-1.2-primer-cd-02.pdf`, May 2006. Accessed [01-10-2008].

[43] OASIS, "OASIS homepage." `http://www.oasis-open.org/home/index.php`, 2008. [Accessed 01-10-2008].

[44] M. Humphrey, G. Wasson, Y. Kiryakov, S.-M. Park, D. D. Vecchio, N. Beekwilder, and J. Gray, "Alternative software stacks for OGSA-based grids," in *Proceedings of the ACM/IEEE Supercomputing 2005 Conference*, 2005.

[45] I. Foster, C. Kesselman, J. M. Nick, and S. Tuecke, "Grid services for distributed system integration," *Computer*, vol. 35, no. 6, pp. 37–46, 2002.

[46] M. Marzolla, P. Andreetto, V. Venturi, A. Ferraro, A. Memon, M. Memon, B. Twedell, M. Riedel, D. Mallmann, A. Streit, S. van de Berghe, V. Li, D. Snelling, K. Stamou, Z. Shah, and F. Hedman, "Open standards-based interoperability of job submission and management interfaces across the grid middleware platforms glite and unicore," in *Proceedings of the IEEE International Conference on e-Science and Grid Computing*, (Washington, DC, USA), pp. 592 – 601, IEEE Computer Society for Industrial and Applied Mathematics, 2007.

[47] A. Streit, D. Erwin, T. Lippert, D. Mallmann, R. Menday, M. Rambadt, M. Riedel, M. Romberg, B. Schuller, and P. Wieder, "UNICORE - From project results to production grids," in *Grid Computing : The New Frontier of High Performance Computing* (L. Grandinetti, ed.), vol. 14 of *Advances in Parallel Computing*, pp. 357–376, Amsterdam: Elsevier, 2005.

[48] I. Foster, "Globus toolkit version 4: Software for service-oriented systems," in *Proceedings of the IFIP International Conference on Network and Parallel Computing* (H. Jin, D. Reed, and W. Jiang, eds.), vol. 3779 of *Lecture Notes in Computer Science*, (Heidelberg), pp. 2–13, Springer-Verlag, 2006.

[49] K. Ranganathan and I. Foster, "Simulation studies of computation and data scheduling algorithms for data grids," *Journal of Grid Computing*, vol. 1, no. 1, pp. 53–62, 2004.

[50] D. Cameron, A. Millar, C. Nicholson, R. Carvajal-Schiaffino, K. Stockinger, and F. Zini, "Analysis of scheduling and replica optimisation strategies for data grids using Optorsim," *Journal of Grid Computing*, vol. 2, no. 1, pp. 57–69, 2004.

[51] T. Wauters, J. Coppens, F. D. Turck, B. Dhoedt, and P. Demeester, "Replica placement in ring based content delivery networks," *Computer Communications*, vol. 29, no. 16, pp. 3313–3326, 2006.

[52] M. Antonioletti, M. Atkinson, R. Baxter, A. Borley, N. C. Hong, B. Collins, N. Hardman, A. Hume, A. Knox, M. Jackson, A. Krause, S. Laws, J. Magowan, N. Paton, D. Pearson, T. Sugden, P. Watson, and M. Westhead, "The design and implementation of grid database services in OGSA-DAI.," *Concurrency and Computation: Practice and Experience*, vol. 17, no. 2-4, pp. 357–376, 2005.

[53] J. M. Schopf, M. D'Arcy, N. Miller, L. Pearlman, I. Foster, and C. Kesselman, "Monitoring and discovery in a web services framework: Functionality and performance of the Globus Toolkit's MDS4," Tech. Rep. ANL/MCS-P1248-0405, Argonne National Laboratory, April 2005.

[54] R. M. Piro, A. Guarise, and A. Werbrouck, "An economy-based accounting infrastructure for the datagrid," in *GRID '03: Proceedings of the 4th International Workshop on Grid Computing*, (Washington, DC, USA), p. 202, IEEE Computer Society, 2003.

[55] P. Gardfjäll, E. Elmroth, L. Johnsson, O. Mulmo, and T. Sandholm, "Scalable grid-wide capacity allocation with the SweGrid accounting system (SGAS)," *Concurrency and Computation: Practice and Experience*, vol. 20, pp. 2089–2122, December 2008.

[56] M. Henning and S. Vinoski, *Advanced CORBA Programming with C++*. Addison-Wesley Professional, 1999.

[57] W. Grosso, *Java RMI*. Sebastopol, California: O'Reilly & Associates, Inc., 2002.

[58] W. K. Edwards, *Core Jini*. Upper Saddle River, New Jersey: Prentice-Hall, Inc., 1999.

[59] K. Vanmechelen, J. Broeckhove, G. Stuer, and T. Dhaene, "Jini and JXTA based lightweight grids," in *Engineering the Grid: status and perspective* (J. Dongarra, H. Zima, A. Hoisie, L. Yang, and B. D. Martino, eds.), ch. 13, pp. 503–519, Valencia, USA: American Scientific Publishers, 2006.

[60] S. Pota and Z. Juhasz, "The benefits of Java and Jini in the jgrid system," in *Proceedings of the IEEE International Parallel & Distributed Processing Symposium 2006 (IPDPS 2006)*, IEEE Computer Society, 2006.

[61] N. Furmento, W. Lee, A. Mayer, S. Newhouse, and J. Darlington, "ICENI: An open grid service architecture implemented with Jini," in *Proceedings of the 2002 ACM/IEE Supercomputing Conference (SC2002)*, (Los Alamitos, CA, USA), pp. 1 – 10, IEEE Computer Society Press, 2002.

[62] M. L. Kahn and C. D. T. Cicalese, "The CoABS grid," in *Goddard/JPL Workshop on Radical Agent Concepts (WRAC)*, vol. 2564 of *Lecture Notes in Computer Science*, (Heidelberg), pp. 125–134, Springer, 2003.

[63] A. Goldchleger, F. Kon, A. Goldmann, M. Finger, and G. Bezerra, "Integrade: object-oriented grid middleware leveraging idle computing power of desktop machines," *Concurrency and Computation: Practice and Experience*, vol. 00, pp. 1 – 12, 2000.

[64] A. S. Grimshaw and W. A. Wulf, "The legion vision of a worldwide virtual computer," *Communications of the ACM*, vol. 40, pp. 39–45, 1997.

[65] R. V. Nieuwpoort, J. Maassen, R. Hofman, T. Kielmann, and H. Bal, "Ibis: An efficient java-based grid programming environment," in *Proceedings of the Joint ACM Java Grande - ISCOPE 2002 Conference*, pp. 18–27, ACM, November 2002.

[66] J. Nabrzyski, J. M. Schopf, and J. Weglarz, eds., *Grid Resource Management : State of the Art and Future Trends*. Kluwer Academic Publishers, Dordrecht, The Netherlands, 2004.

[67] J. M. Schopf, "Ten actions when grid scheduling," in *Grid Resource Management - State of the Art and Future Trends* (J. Nabrzyski, J. M. Schopf, and J. Wglarz, eds.), pp. 15–23, Kluwer Academic Publishers, 2004.

[68] R. Raman, M. Solomon, M. Livny, and A. Roy, "The ClassAds language," in *Grid Resource Management - State of the Art and Future Trends* (J. Nabrzyski, J. M. Schopf, and J. Wglarz, eds.), pp. 255–270, Kluwer Academic Publishers, 2004.

[69] S. Andreozzi, S. Burke, F. Ehm, L. Field, G. Galang, B. Konya, M. Lithmaath, P. Millar, and J. Navarro, "GLUE specification v. 2.0," tech. rep., Open Grid Forum, May 2008.

[70] A. Anjomshoaa, F. Brisard, M. Drescher, D. Fellows, A. Ly, S. McGough, D. Pulsipher, and A. Savva, "Job submission description language (JSDL) specification, version 1.0," tech. rep., Open Grid Forum, November 2005.

[71] F. Heine, M. Hovestadt, O. Kao, and A. Streit, "On the impact of reservations from the grid on planning-based resource management," in *Proceedings of the 5th International Conference on Computational Science (ICCS 2005)* (V. S. Sunderam, G. D. van Albada, P. M. Sloot, and J. J. Dongarra, eds.), vol. 3516 of *Lecture Notes in Computer Science*, (Heidelberg), pp. 155–162, Springer-Verlag, 2005.

[72] D. B. Jackson, Q. Snell, and M. J. Clement, "Core algorithms of the Maui scheduler," in *JSSPP '01: Revised Papers from the 7th International Workshop on Job Scheduling Strategies for Parallel Processing*, (London, UK), pp. 87–102, Springer-Verlag, 2001.

[73] R. L. Henderson, "Job scheduling under the portable batch system," in *Job Scheduling Strategies for Parallel Processing* (D. Feitelson, E. Frachtenberg, L. Rudolph, and U. Schwiegelshohn, eds.), vol. 949 of *Lecture Notes in Computer Science*, pp. 279–294, Heidelberg: Springer, 1995.

[74] Platform, "Platform load sharing facility homepage," 2008. [Accessed 01-10-2008].

[75] W. Gentzsch, "Sun grid engine: Towards creating a compute power grid," in *Proceedings of the 1st International Symposium on Cluster Computing and the Grid (CCGrid'01)*, (Washington, DC, USA), IEEE Computer Society, 2001.

[76] J. M. Schopf and B. Nitzberg, "Grids: The top ten questions," *Scientific Programming, special issue on Grid Computing*, vol. 10, no. 2, pp. 103–111, 2002.

[77] K. Krauter, R. Buyya, and M. Maheswaran, "A taxonomy and survey of grid resource management systems for distributed computing," *Softw. Pract. Exper.*, vol. 32, pp. 135 – 164, 2002.

[78] B. N. Chun, P. Buonadonna, A. AuYoung, N. Chaki, D. Parkes, J. Shneidman, A. Snoeren, and A. Vahdat, "Mirage: A microeconomic resource allocation system for sensornet testbeds," in *Proceedings of the second IEEE Workshop on Embedded Networked Sensors*, pp. 19–28, IEEE Computer Society, May 2005.

[79] D. F. Ferguson, C. Nikolaou, J. Sairamsesh, and Y. Yemini, "Economic models for allocating resources in computer systems," in *Market-based Control - A*

Paradigm for Distributed Resource Allocation (S. H. Clearwater, ed.), World Scientific, 1996.

[80] G. Cheliotis, C. Kenyon, and R. Buyya, "Grid economics: 10 lessons from finance," Tech. Rep. GRIDS-TR-2003-3, Grid Computing and Distributed Systems Laboratory, University of Melbourne, Grid Computing and Distributed Systems Laboratory, 2004.

[81] C. Kenyon and G. Cheliotis, "Grid resource commercialization: Economic engineering and delivery scenarios," in *Grid resource management: State of the art and future trends* (J.Nabrzyski, J.Schopf, and J.Weglarz, eds.), ch. 28, pp. 465 – 478, Kluwer, 2004.

[82] J. Altmann, M. Ion, and A. A. B. Mohammed, "Taxonomy of grid business models," in *Proceedings of the 4th International Workshop on Grid Economics and Business Models (GECON 2007)* (J. Altmann and D. Veit, eds.), vol. 4685 of *Lecture Notes in Computer Science*, Springer, 2007.

[83] M. Risch and J. Altmann, "Cost analysis of current grids and its implications for future grid markets," in *Proceedings of the 5th International Workshop on Grid Economics and Business Models (GECON 2008)* (J. Altmann, D. Neumann, and T. Fahringer, eds.), no. 5206 in Lecture Notes in Computer Science, Springer, 2008.

[84] Amazon, "Elastic compute cloud." `http://aws.amazon.com/ec2`, 2008. [Accessed 22-12-08].

[85] J. Murty, *Programming Amazon Web Services - S3, EC2, SQS, FPS, and SimpleDB*. O'Reilly, 2008.

[86] J. Nakai, "Pricing computing resources: Reading between the lines and beyond," Tech. Rep. NAS-01-010, NASA Ames Research Center, Moffett field, CA, January 2002.

[87] Linden Research, Inc., "The Second Life market place." `http://secondlife.com/whatis/marketplace.php`, 2008. [Accessed 25-09-08].

[88] Linden Research, Inc., "LindeX market data." `http://secondlife.com/whatis/economy-market.php`, 2008. [Accessed 25-09-08].

[89] F. Hayek, "The use of knowledge in society," *American Enonomic Review*, vol. 35, pp. 519–530, 1945.

[90] D. Fudenberg and J. Tirole, *Game Theory*. Cambridge: MIT Press, 1991.

[91] J. C. Harsanyi, "Games with incomplete information played by Bayesian players. I. The basic model," *Management Science*, vol. 14, no. 3, pp. 159–182, 1967.

[92] R. B. Myerson and M. A. Satterthwaite, "Efficient mechanisms for bilateral trading," *Journal of Economic Theory*, vol. 29, no. 2, pp. 265–281, 1983.

[93] B. Schnizler, *Resource Allocation in the Grid – A Market Engineering Approach*. PhD thesis, University of Karlsruhe, 2007.

[94] P. Cramton, Y. Shoham, and R. Steinberg, eds., *Combinatorial Auctions*. Cambridge, MA: MIT Press, 2006.

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

[96] M. Shubik, "Auctions, bidding and markets: An historical sketch," in *Auctions, Bidding and Contracting: Uses and Theory* (R. Engelbrecht-Wiggins and R. Stark, eds.), pp. 33–52, New York: New York University Press, 1983.

[97] IAL, "The internet auction list," 2008. [Accessed 01-10-2008].

[98] R. P. McAfee and J. McMillan, "Auctions and bidding," *Journal of Economic Literature*, vol. 25, no. 2, pp. 699–738, 1987.

[99] P. R. Wurman, M. P. Wellman, and W. E. Walsh, "A parametrization of the auction design space," *Games and Economic Behavior*, vol. 35, pp. 304 – 338, 2001.

[100] P. Klemperer, *Auctions: Theory and practice*. New Jersey: Princeton University Press, 2004.

[101] W. Vickrey, "Counterspeculation, auctions, and competitive sealed tenders," *Journal of Finance*, vol. 16, no. 1, pp. 8–37, 1961.

[102] J. P. Quirk and R. Saposnik, *Introduction to general equilibrium theory and welfare economics*. McGraw-Hill Education, 1968.

[103] M. P. Wellman, "A market-oriented programming environment and its application to distributed multicommodity flow problems," *Journal of Artificial Intelligence Research*, vol. 1, pp. 1 – 23, 1993.

[104] R. Wolski, J. Plank, J. Brevik, and T. Bryan, "Analysing market-based resource allocation strategies for the computational grid," *Int. J. of High-Performance Computing Applications*, vol. 15, pp. 258–281, 2001.

[105] K. Vanmechelen, G. Stuer, and J. Broeckhove, "Pricing substitutable grid resources using commodity market models," in *Proceedings of the 3rd Int. Workshop on Grid Economics and Business Models (GECON 2006)* (H. Lee and S. Miller, eds.), (Singapore), pp. 103–112, World Scientific, 2006.

[106] M. Bichler, G. Kersten, and S. Strecker, "Towards a structured design of electronic negotiations," *Group Decision and Negotiation*, vol. 12, no. 4, pp. 311–335, 2003.

[107] H. Gimpel, N. R. Jennings, G. E. Kersten, A. Ockenfels, and C. Weinhardt, "Market engineering: A research agenda," in *Negotiation, Auctions, and Market Engineering*, vol. 2 of *Lecture Notes in Business Information Processing*, ch. 1, pp. 1–15, Springer Heidelberg, 2008.

[108] G. E. Kersten, S. J. Noronha, and J. Teich, "Are all e-commerce negotiations auctions?," in *Proceedings of the Fourth International Conference on the Design of Cooperative Systems (COOP'2000)*, IOS Press, 2000.

[109] P. Milgrom, "Auctions and bidding: A primer," *The Journal of Economic Perspectives*, vol. 3, no. 3, pp. 3–22, 1989.

[110] J. Bulow and P. Klemperer, "Auctions versus negotiations," *American Enonomic Review*, vol. 86, no. 1, pp. 180–194, 1996.

[111] R. P. McAfee and J. McMillan, "Competition and game theory," *Journal of Marketing Research*, vol. 33, no. 3, pp. 263–267, 1996.

[112] E. Kjerstad, "Auctions vs negotiations: a study in price differentials," *Health Economics*, vol. 14, no. 12, pp. 1239–1251, 2005.

[113] J. Banks, J. Carson, B. L. Neslson, and D. Nicol, *Discrete-Event System Simulation*. Prentice Hall International Series in Industrial and Systems Engineering, Pearson Prentice Hall, 2005.

[114] A. Legrand, L. Marchal, and H. Casanova, "Scheduling distributed applications: the SimGrid simulation framework," in *3rd IEEE/ACM International Symposium on Cluster Computing and the Grid, 2003. Proceedings. CCGrid 2003.*, pp. 138–145, May 2003.

[115] 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*, vol. 14, no. 13-15, pp. 1175 – 1220, 2002.

[116] K. Vanmechelen, W. Depoorter, and J. Broeckhove, "A simulation framework for studying economic resource management in grids," in *Proceedings of the International Conference on Computational Science (ICCS 2008)*, vol. 5101, pp. 226–235, Springer-Verlag, Berlin Heidelberg, 2008.

[117] A. Sulistio, C. S. Yeo, and R. Buyya, "A taxonomy of computer-based simulations and its mapping to parallel and distributed systems simulation tools," *Softw. Pract. Exper.*, vol. 34, pp. 653 – 673, 2004.

[118] A. Takefusa, S. Matsuoka, H. Nakada, K. Aida, and U. Nagashima, "Overview of a performance evaluation system for global computing scheduling algorithms," in *Proceedings of the 8th IEEE International Symposium on High Performance Distributed Computing (HPDC)*, pp. 97–104, IEEE Computer Society, 1999.

[119] R. Wolski, N. T. Spring, and J. Hayes, "The network weather service: a distributed resource performance forecasting service for metacomputing," *Future Generation Computer Systems*, vol. 15, no. 5–6, pp. 757–768, 1999.

[120] A. Takefusa and H. Casanova, "A study of deadline scheduling for client-server systems on the computational grid," in *Proceedings of the 10th IEEE International Symposium on High Performance Distributed Computing (HPDC)*, pp. 406–415, 2001.

[121] H. J. Song, X. Liu, D. Jakobsen, R. Bhagwan, X. Zhang, K. Taura, and A. Chien, "The MicroGrid: a scientific tool for modeling computational grids," *Sci. Program.*, vol. 8, no. 3, pp. 127–141, 2000.

[122] A. Legrand and J. Lerouge, "MetaSimGrid: Towards realistic scheduling simulation of distributed applications," Tech. Rep. 2002-28, LIP, July 2002.

[123] "The network simulator (ns2)."

[124] J. Cowie, D. Nicol, and A. Ogielski, "Modeling the global internet," *Computing in Science and Engineering*, vol. 1, no. 1, pp. 42–50, 1999.

[125] G. F. Riley, "The georgia tech network simulator," in *Proceedings of the ACM SIGCOMM workshop on Models, Methods and Toosl for Reproducible Network Research*, pp. 5–12, 2003.

[126] M. Assuncao and R. Buyya, "An evaluation of communication demand of auction protocols in grid environments," in *Proceedings of GECON 2006*, pp. 24–33, World Scientific, 2006.

[127] C. O. Inc., "ILOG CPLEX." `http://www.ilog.com/products/cplex/`, 2008. [Accessed 01-10-2008].

[128] M. Berkelaar, K. Eikland, and P. Notebaert, "lp_solve : Open source (mixed-integer) linear programming system," 2008. [Accessed 01-10-2008].

[129] Sun Microsystems, Inc., "Java object serialization specification." `http://java.sun.com/javase/6/docs/platform/serialization/spec/serialTOC.html`, 2005. [Accessed 01-10-2008].

[130] J. Y.-T. Leung, ed., *Handbook of Scheduling: Algorithms, Models and Performance Analysis*. CRC Press, 2004.

[131] M. M. Bykowsky, R. J. Cull, and J. O. Ledyard, "Mutually destructive bidding: The FCC auction design problem," *Journal of Regulatory Economics*, vol. 17, no. 3, pp. 205–228, 2000.

[132] W. Depoorter, "Establishment of agency as an effective market based resource allocation method using GES," Master's thesis, University of Antwerp, 2007.

[133] K. Vanmechelen and J. Broeckhove, "A comparative analysis of single-unit Vickrey auctions and commodity markets for realizing grid economies with dynamic pricing," in *Proceedings of the 4th International Workshop on Grid Economics and Business Models (GECON 2007)* (J. Altmann and D. Veit, eds.), vol. 4685 of *Lecture Notes in Computer Science*, (Heidelberg), pp. 98–111, Springer-Verlag, 2007.

[134] G. Stuer, K. Vanmechelen, and J. Broeckhove, "A commodity market algorithm for pricing substitutable grid resources," *Future Generation Computer Systems*, vol. 23, no. 5, pp. 688–701, 2007.

[135] S. Smale, "A convergent process of price adjustment and global newton methods," *Journal of Mathematical Economics*, vol. 3, no. 2, pp. 107–120, 1976.

[136] M. Feldman, K. Lai, and L. Zhang, "A price-anticipating resource allocation mechanism for distributed shared clusters," in *Proceedings of the 6th ACM conference on Electronic Commerce (EC05)*, (New York, NY, USA), pp. 127–136, ACM, 2005.

[137] K. Vanmechelen, W. Depoorter, and J. Broeckhove, "Economic grid resource management for CPU bound applications with hard deadlines," in *Proceedings of CCGrid 2008*, pp. 258–266, IEEE Computer Society, 2008.

[138] J. M. Garrido, *Object-Oriented Discrete-Event Simulation with Java: A Practical Introduction*. Kluwer, 2001.

[139] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. V. Lopes, J.-M. Loingtier, and J. Irwin, "Aspect-oriented programming," in *Proceedings of ECOOP 1997* (M. Akit and S. Matsuoka, eds.), vol. 1241 of *Lecture Notes in Computer Science*, (Berlin, Heidelberg), Springer-Verlag, 1997.

[140] R. E. Filman, T. Elrad, S. Clarke, and M. Akit, eds., *Aspect-Oriented Software Development*. Addison-Wesley, 2005.

[141] D. Coward, "JSR 175: A metadata facility for the Java programming language." `http://www.jcp.org/en/jsr/detail?id=175`, 2004. [Accessed 03-10-2008].

[142] T. Curdt, K. Kawaguchi, and M. Cooper, "Apache Commons Javaflow." `http://commons.apache.org/sandbox/javaflow`, 2008. [Accessed 03-10-2008].

[143] G. Bevin, J. Boyens, K. Lea, P. Raoul, T. Pitchford, and S. Grimm, "The RIFE project homepage." `http://rifers.org/`, 2008. [Accessed 03-10-2008].

[144] S. Srinivasan, "A thread of one's own," in *Proceedings of the Workshop on New Horizons in Compilers*, (Bangalore, India), 2006.

[145] M. Lacage, C. Dowell, and G. Carneiro, "The ns-3 network simulator." `http://www.nsnam.org/`, 2008. [Accessed 01-10-2008].

[146] I. OPNET Technologies, "OPNET Modeler." `http://www.opnet.com/solutions/network\_rd/modeler.html`, 2008.

[147] A. Varga, "The OMNeT++ discrete event simulation system," in *Proceedings of the European Simulation Multiconference (ESM'2001)*, pp. 319–324, 2001.

[148] K. Fujiwara and H. Casanova, "Speed and accuracy of network simulation in the simgrid framework," in *Proceedings of the workshop on Network Simulation Tools (NSTools)*, 2007.

[149] D. Lu and P. A. Dinda, "GridG: Generating realistic computational grids," in *Proceedings of Supercomputing '03, Phoenix, Arizona*, pp. 1 – 14, Association for Computing Machinery (ACM), November 2003.

[150] Y.-S. Kee, H. Casanova, and A. A. Chien, "Realistic modeling and synthesis of resources for computational grids," in *Proceedings of Supercomputing 2004*, 2004.

[151] K. Calvert, M. Doar, and E. Zegura, "Modeling internet topology," *IEEE Communications Magazine*, vol. 35, no. 6, pp. 160–168, 2007.

[152] B. Waxman, "Routing of multipoint connections," *Journal of Selected Areas in Communications*, vol. 6, no. 9, pp. 1622–1671, 1988.

[153] J. Winick and S. Jamin, "Inet-3.0: Internet topology generator," Tech. Rep. CSE-TR-456-02, Department of EECS, University of Michigan, 2002.

[154] M. Faloutsos, P. Faloutsos, and C. Faloutsos, "On power-law relationships of the internet topology," in *Proceedings of SIGCOMM '99*, 1999.

[155] P. M. Papadopoulos, M. J. Katz, and G. Bruno, "NPACI Rocks: Tools and techniques for easily deploying manageable linux clusters," in *Proceedings of the IEEE International Conference on Cluster Computing (Cluster '01)*, pp. 258–270, 2001.

[156] J. O'Madadhain, D. Fisher, and T. Nelson, "Java universal network/graph framework (JUNG) homepage." `http://jung.sourceforge.net`, 2008. [Accessed 01-11-08].

[157] E. W. Dijkstra, "A note on two problems in connection with graphs," *Numerische Mathematik*, vol. 1, pp. 269–271, 1959.

[158] L. Walras, *Éléments d'économie politique pure, ou: Théorie de la richesse sociale*. Paris: Pichon et Durand-Auzias, 4 ed., 1952.

[159] A. Leijonhufvud, *On Keynesian Economics and the Economics of Keynes*. Oxford University Press, 1968.

[160] A. Mas-Colell, M. D. Whinston, and J. R. Green, *Microeconomic Theory*, ch. 16, pp. 545–578. Oxford University Press, 1995.

[161] D. Lehmann, R. Maller, and T. Sandholm., "The winner determination problem," in *Combinatorial Auctions* (P. Cramton, Y. Shoham, and R. Steinberg, eds.), ch. 12, pp. 297–317, Cambridge, MA: MIT Press, 2006.

[162] J. Stößer and D. Neumann, "GreedEx – A scalable clearing mechanism for utility computing," in *Proceedings of the Networking and Electronic Commerce Research Conference (NAEC) 2007*, 2007.

[163] D. J. Roberts and A. Postlewaite, "The incentives for price-taking behavior in large exchange economies," *Econometrica*, vol. 44, no. 1, pp. 115–127, 1976.

[164] M. Karlsson, F. Ygge, and A. Andersson, "Market-based approaches to optimization," *Computational Intelligence*, vol. 23, no. 1, pp. 92–109, 2007.

[165] M. Yang, "A survey of fuzzy clustering," *Int. J. of Mathematical and Computer Modelling*, vol. 18, no. 11, pp. 1–16, 1993.

[166] B. Volckaert, P. Thysebaert, M. D. Leenheer, F. D. Turck, B. Dhoedt, and P. Demeester, "Flexible grid service management through resource partitioning," *The Journal of Supercomputing*, vol. 38, no. 3, pp. 279–305, 2006.

[167] C. A. Waldspurger, T. Hogg, B. A. Huberman, J. O. Kephart, and W. S. Stornetta, "Spawn: A distributed computational economy," *IEEE Trans. Softw. Eng.*, vol. 18, no. 2, pp. 103–117, 1992.

[168] O. Regev and N. Nisan, "The POPCORN market – an online market for computational resources," in *Proceedings of the 1st International Conference on Information and Computation Economies*, (New York, NY, USA), pp. 148–157, ACM, 1998.

[169] T. Sandholm and K. Lai, "Prediction-based enforcement of performance contracts," in *Proceedings of the 4th International Workshop on Grid Economics and Business Models (GECON 2007)* (J. Altmann and D. Veit, eds.), vol. 4685 of *Lecture Notes in Computer Science*, (Heidelberg), pp. 71–82, Springer-Verlag, 2007.

[170] T. Sandholm, K. Lai, and S. H. Clearwater, "Admission control in a computational market," in *Proceedings of CCGrid 2008*, pp. 277–286, IEEE Computer Society, 2008.

[171] D. Grosu and A. Das, "Auctioning resources in grids: model and protocols," *Concurrency and Computation: Practice and Experience*, vol. 18, no. 15, pp. 1909 – 1927, 2006.

[172] B. Pourebrahimi and K. Bertels, "Auction protocols for resource allocations in ad-hoc grids," in *Proceedings of Europar 2008* (E. Luque, T. Margalef, and D. Benitez, eds.), vol. 5168 of *Lecture Notes in Computer Science*, (Heidelberg), pp. 520–533, Springer-Verlag, 2008.

[173] B. Pourebrahimi and K. Bertels, "Adaptation to dynamic resource availability in ad hoc grids through a learning mechanism," in *Proceedings of the 2008 11th IEEE International Conference on Computational Science and Engineering (CSE'08)*, (Washington, DC, USA), pp. 171–178, IEEE Computer Society, 2008.

[174] J. Gomoluch and M. Schroeder, "Market-based resource allocation for grid computing: A model and simulation," in *Int. Middleware Conference, Workshop Proceedings* (M. Endler and D. Schmidt, eds.), (Rio De Janeiro, RJ), pp. 211–218, PUC-Rio, 2003.

[175] R. Buyya, D. Abramson, and J. Giddy, "Economic models for resource management and scheduling in grid computing," *Concurrency and Computation: Practice and Experience*, vol. 14, pp. 1507–1542, 2002.

[176] M. Stonebraker, R. Devine, M. Kornacker, W. Litwin, A. Pfeffer, A. Sah, and C. Staelin, "An economic paradigm for query processing and data migration in Mariposa," in *Proc. of 3rd Int. Conf. on Parallel and Distributed Information Systems*, 1994.

[177] R. Buyya and S. Vazhkudai, "Compute power market: Towards a market-oriented grid," in *Proceedings of the 1st IEEE/ACM Int. Symp on Cluster Computing and the Grid (CCGRID '01)*, 2001.

[178] N. Stratford and R. Mortier, "An economic approach to adaptive resource management," in *7th Workshop on Hot Topics in Operating Systems*, pp. 142–147, 1999.

[179] R. Smith and R. David, "The contract net protocol: High-level communication and control in a distributed problem solver," *IEEE Transactions on Computers*, vol. C-29, no. 12, pp. 1104–1113, 1980.

[180] B. Chun and D. Culler, "Market-based proportional resource sharing for clusters," tech. rep., University of California, Berkeley, 1999.

[181] K. Lai, B. A. Huberman, and L. Fine, "Tycoon: A distributed, market-based resource allocation system," Tech. Rep. cs.DC/0412038, HP Labs, December 2004.

[182] J. McCoy, "Mojo nation." `http://sourceforge.net/projects/mojonation/`, 2001. [Accessed 01-11-08].

[183] A. Lazar and N. Semret, "Auctions for network resource sharing," Tech. Rep. TR 468-97-02, Columbia University, 1997.

[184] P. Gradwell and J. Padget, "Distributed combinatorial resource scheduling," in *Proceedings of the First International Workshop on Smart Grid Technologies (SGT05)*, 2005.

[185] B. N. Chun and D. E. Culler, "User-centric performance analysis of market-based cluster batch schedulers," in *Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID '02)*, pp. 22–30, IEEE Computer Society, 2002.

[186] J. Sherwani, N. Ali, N. Lotia, Z. Hayat, and R. Buyya, "Libra: a computational economy-based job scheduling system for clusters," *Softw. Pract. Exper*, vol. 34, no. 6, pp. 573–590, 2004.

[187] C. S. Yeo and R. Buyya, "A taxonomy of market-based resource management systems for utility-driven cluster computing," *Software Practice and Experience*, vol. 36, pp. 1381–1419, nov 2006. ISSN = "0038-0644".

[188] R. K. Dash, N. R. Jennings, and D. C. Parkes, "Computational-mechanism design: A call to arms," *IEEE Intelligent Systems*, vol. 18, pp. 40–47, November/December 2003.

[189] J. Bredin, R. Maheswaran, Çagri Imer, T. Başar, D. Kotz, and D. Rus, "Computational markets to regulate mobile-agent systems," *Autonomous Agents and Multi-Agent Systems*, vol. 6, pp. 235–263, May 2003.

[190] C. Courcoubetis and R. Weber, *Pricing Communication Networks: Economics, Technology and Modelling*. John Wiley & Sons, Ltd., 2003.

[191] B. Tuffin, "Charging the internet without bandwidth reservation: An overview and bibliography of mathematical approahes," *Journal of Information Science and Engineering*, vol. 19, no. 5, pp. 765–786, 2003.

[192] M. A. Gibney, N. R. Jennings, N. J. Vriend, and J.-M. Griffiths, "Market-based call routing in telecommunications networks using adaptive pricing and real bidding," in *Proc. of the 3th Int. Workshop on Intelligent Agents for Telecommunication Applications*, pp. 46–61, 1999.

[193] N. Haque, N. R. Jennings, and L. Moreau, "Resource allocation in communication networks using market-based agents," *International Journal of Knowledge Based Systems*, vol. 18, no. 4-5, pp. 163–170, 2005.

[194] L. Badia and M. Zorzi, "Radio resource management with utility and pricing for wireless LAN Hot-Spots," *Wireless Personal Communications*, vol. 34, pp. 127–142, July 2005.

[195] M. Stonebraker, P. M. Aoki, W. Litwin, A. Pfeffer, A. Sah, and J. Sidell, "Mariposa: A wide-area distributed database system," *Very Large Data Bases Journal*, vol. 5, no. 1, pp. 48–63, 1996.

[196] L. Joita, O. F. Rana, F. Freitag, I. Chao, P. Chacin, L. Navarro, and O. Ardaiz, "A catallactic market for data mining services," *Future Generation Computer Systems*, vol. 23, pp. 146 – 153, 2007.

[197] R. Buyya, D. Abramson, and S. Venugopal, "The grid economy," *Proceedings of the IEEE*, vol. 93, pp. 698–714, March 2005.

[198] J. Broberg, S. Venugopal, and R. Buyya, "Market-oriented grids and utility computing: The state-of-the-art and future directions," *Journal of Grid Computing*, vol. 6, no. 2, pp. 255–276, 2008.

[199] N. Dube and M. Parizeau, "Utility computing and market-based scheduling: Shortcomings for grid resources sharing and the next steps," in *Proceedings of the 22nd International Symposium onHigh Performance Computing Systems and Applications (HPCS 2008)*, pp. 59–68, 2008.

[200] K. J. Arrow and F. H. Hahn, *General Competitive Analysis*. North Holland, 1971.

[201] V. Ginsburgh and M. Keyzer, *The structure of applied general equilibrium models*. Cambridge, Massachusetts: MIT Press, 2002.

[202] J.-J. Herings, "Universally converging adjustment processes – a unifying approach," *Journal of Mathematical Economics*, vol. 28, pp. 341–370, 2002.

[203] M. Hirsch and S. Smale, "Algorithms for solving f(x)=0," *Communications on Pure and Applied Mathematics*, vol. 32, pp. 281–312, 1979.

[204] R. M. Lewis and V. Torczon, "A globally convergent augmented lagrangian pattern search algorithm for optimization with general constraints and simple bounds," *SIAM J. on Optimization*, vol. 12, pp. 1075 – 1089, 2002.

[205] K. Abdelkader, J. Broeckhove, and K. Vanmechelen, "Commodity resource pricing in dynamic computational grids," in *Proceedings of the 6th ACS/IEEE International Conference on Computer Systems and Applications (AICCSA-08)*, no. CFP08283-CDR in ISBN: 978-1-4244-1968-5, (Doha, Qatar), pp. 422–429, IEEE Press, March31 - April 4 2008.

[206] G. Snedecor and W. Cochran, *Statistical Methods*. Ames, IA: The Iowa State University Press, 6 ed., 1967.

[207] P. Hellinckx, G. Stuer, W. Hendrickx, F. Arickx, and J. Broeckhove, "Grid-user driven grid research, the CoBRA grid," in *CCGRID '06: Proceedings of the Sixth IEEE International Symposium on Cluster Computing and the Grid (CCGRID'06)*, (Washington, DC, USA), p. 49, IEEE Computer Society, 2006.

[208] G. Cheliotis, C. Kenyon, and R. Buyya, "10 Lessons from finance for commercial sharing of IT resources," in *Peer-to-Peer Computing: Evolution of a Disruptive Technology* (R. Sumabramian and B. Goodman, eds.), ch. 11, pp. 244 – 264, Hershey, PA, USA: Idea Group Publishing, 2004.

[209] R. Bapna, S. Das, R. Garfinkel, and J. Stallaert, "A market design for grid computing," *INFORMS Journal on Computing*, vol. 20, no. 1, pp. 100–111, 2007.

[210] I. Segal, "The communication requirements of combinatorial allocation problems," in *Combinatorial Auctions* (P. Cramton, Y. Shoham, and R. Steinberg, eds.), ch. 11, pp. 265–294, Cambridge, MA: MIT Press, 2006.

[211] J. Dongarra and D. Walker, "MPI: A standard message passing interface," *Supercomputer*, vol. 12, no. 1, pp. 56–68, 1996.

[212] Z. Huang and Y. Qiu, "Resource trading using cognitive agents: A hybrid perspective and its simulation," *Future Generation Computer Systems*, vol. 23, no. 7, pp. 837–845, 2007.

[213] R. A. V. Engelen, K. A. Gallivan, and B. Walsh, "Parametric timing estimation with Newton-Gregory formulae," *Concurrency and Computation: Practice and Experience*, vol. 18, no. 11, pp. 1435–1463, 2006.

[214] M. A. Iverson, F. Özgüner, and G. J. Follen, "Run-time statistical estimation of task execution times for heterogeneous distributed computing," in *Proceedings of 5th IEEE International Symposium on High Performance Distributed Computing*, pp. 263–270, IEEE Computer Society, 1996.

[215] M. A. Iverson, F. Özgüner, and L. C. Potter, "Statistical prediction of task execution times through analytic benchmarking for scheduling in a heterogeneous environment," *IEEE Transactions on Computers*, vol. 48, no. 12, pp. 1374–1379, 1999.

[216] P. Brucker, *Scheduling Algortihms*. Berlin Heidelberg: Springer-Verlag, 2004.

[217] J. K. Lenstra, A. H. G. R. Kan, and P. Brucker, "Complexity of machine scheduling problems," *Annals of Discrete Mathematics*, vol. 1, pp. 343–362, 1977.

[218] J. Cohen, J. Darlington, and W. Lee, "Payment and negotiation for the next generation grid and web," *Concurrency and Computation: Practice and Experience*, vol. 20, no. 3, pp. 239–251, 2008.

[219] L. Paypal (Europe), "PayPal Home Page." `http://www.paypal.co.uk`, 2008. [Accessed 25-09-08].

[220] L. Hurwicz, "On informationally decentralized systems," in *Decision and Organization: a Volume in Honor of Jacob Marshak* (C. B. McGuire and R. Radner, eds.), vol. 12 of *Studies in mathematical and managerial economics*, ch. 14, pp. 297–336, North-Holland, 1972.

[221] A. Ronen, "Incentive compatibility in computationally feasible combinatorial auctions," in *Combinatorial Auctions* (P. Cramton, Y. Shoham, and R. Steinberg, eds.), ch. 15, pp. 369–394, Cambridge, MA: MIT Press, 2006.

[222] A. I. Juda and D. C. Parkes, "Mechanisms for options: Solving the composability problem," in *Proceedings of the AAMAS Workshop on Agent Mediated Electronic Commerce (AMEC VI)*, 2004.

[223] L. Kang and D. C. Parkes, "A decentralized auction framework to promote efficient resource allocation in open computational grids," in *Proceedings of the Joint Workshop on The Economics of Networked Systems and Incentive-Based Computing (NetEcon-IBC)*, ACM SIGCOMM, 2007.

[224] M. Rothkopf, "Thirteen reasons why the Vickrey-Clarke-Groves process is not practical," *Operations Research*, vol. 55, no. 2, pp. 191–197, 2007.

[225] L. M. Ausubel and P. Milgrom, "The lovely but lonely Vickrey auction," in *Combinatorial Auctions* (P. Cramton, Y. Shoham, and R. Steinberg, eds.), ch. 1, pp. 17–40, Cambridge, MA: MIT Press, 2006.

[226] T. Sandholm, "Limitations of the Vickrey auction in computational multiagent systems," in *Proceedings of the First International Conference on Multi–Agent Systems* (V. Lesser, ed.), (Cambridge, MA), pp. 299–306, MIT Press, 1996.

[227] M. O' Hara, *Market Microstructure Theory.* Oxford: Blackwell Publishers, 1995.

[228] E. Elmroth and P. Gardfjäll, "Design and evaluation of a decentralized system for grid-wide fairshare scheduling," in *Proceedings of the First International Conference on e-Science and Grid Computing*, (Washington, DC, USA), pp. 221–229, IEEE Computer Society, 2005.

[229] R. Buyya and S. Venugopal, "The gridbus toolkit for service oriented grid and utility computing: An overview and status report," in *Proceedings of the First IEEE International Workshop on Grid Economics and Business Models (GECON 2004)*, (New Jersey, USA), pp. 19–36, IEEE Computer Society, 2004.

[230] R. M. Piro, A. Guarise, and A. Werbrouck, "Simulation of price-sensitive resource brokering and the hybrid pricing model with DGAS-Sim," in *WETICE '04: Proceedings of the 13th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, (Washington, DC, USA), pp. 325–330, IEEE Computer Society, 2004.

[231] D. Neumann, J. Stößer, A. Amandasivam, and N. Borissov, "SORMA - building an open grid market for grid resource allocation," in *Proceedings of the 4th International Workshop on Grid Economics and Business Models (GECON 2007)* (J. Altmann and D. J. Veit, eds.), vol. 4685 of *Lecture Notes in Computer Science*, (Heidelberg), pp. 194–200, Springer-Verlag, 2007.

[232] A. Auyoung, B. N. Chun, A. C. Snoeren, and A. Vahdat, "Resource allocation in federated distributed computing infrastructures," in *Proceedings of the 1st Workshop on Operating System and Architectural Support for the On-demand IT InfraStructure*, October 2004.

[233] Y. Fu, J. Chase, B. Chun, S. Schwab, and A. Vahdat, "Sharp: An architecture for secure resource peering," in *SOSP '03: Proceedings of the nineteenth ACM*

symposium on Operating systems principles, (New York, NY, USA), pp. 133–148, ACM, 2003.

[234] D. Neumann, J. Stößer, and C. Weinhardt, "Bridging the adoption gap - Developing a roadmap for trading grids," *International Journal of Electronic Markets*, vol. 18, no. 1, pp. 65–74, 2008.

[235] C. Courcoubetis, M. Dramitinos, T. Rayna, S. Soursos, and G. D. Stamoulis, "Market mechanisms for trading grid resources," in *Proceedings of the 5th International Workshop on Grid Economics and Business Models (GECON 2008)* (J. Altmann and D. J. Veit, eds.), vol. 5206 of *Lecture Notes in Computer Science*, (Heidelberg), pp. 58–72, Springer-Verlag, 2008.