Simulation of Economy-based Load Balancing in Computational Grids for Large-scale Scientific Applications

Relatore: Prof. Giovanni Pollarolo

Co-relatore: Prof. Albert Werbrouck

Candidato: Rosario Michael Piro

Preface

Brief Introduction to the Problem

The enormous demand of computing power by current developments in High-Energy Physics (HEP) experiments — like CMS, ALICE, ATLAS, and LHC-b at the Large Hadron Collider (LHC) at CERN\textsuperscript{1}, Switzerland — cannot be satisfied by traditional computing systems and hence requires the development of a distributed computing environment (called “DataGrid”) that allows to combine and share the heterogeneous resources of an elevated number of collaborating institutions and computing centers.

A globally distributed Grid computing environment, like the \textit{European DataGrid} (EDG), that combines enormous numbers of computing resources (in terms of processing power, disk space, etc.) and astronomical amounts of preliminary scientific data, requires measures for a balanced access of many thousands of simultaneous users. The economy-based \textit{DataGrid Accounting System (DGAS)}, that has been developed at the Section of Turin of the Italian National Institute of Nuclear Physics (INFN), is designed to support an economy-based approach to regulating the distribution of the resources among the authorized Grid users.

This thesis presents a modular and extensible Grid simulation tool that can be used to evaluate the impact of different resource pricing strategies on the balancing of the workload of a Computational Grid and thus on its overall throughput.

\textsuperscript{1}“Conseil Européen pour la Recherche Nucléaire”, the European Organization for Nuclear Research
Structure of the Thesis

The first part of this thesis may be considered an “introduction for pedestrians” to Grid Computing and the computing challenges in High-Energy Physics (Chapter 1), the European DataGrid Project (Chapter 2), the EDG Workload Management System (Chapter 3), and the DataGrid Accounting System (Chapter 4).

Those familiar with Grid Computing and scheduling in such a context, may want to start reading Part II of the thesis, that discusses the problem of resource allocation and load balancing (Chapter 5) and introduces computational economics as an approach to the problem (Chapter 6). It furthermore discusses strategies for resource pricing in a Grid economy (Chapter 7).

The original work of this thesis is described in Part III. In Chapter 8, DGAS-Sim(ulator) is presented — a modular and expandable simulation tool, developed for studying the impact of different resource pricing schemes and resource brokering strategies on workload balancing and thus on the overall throughput of the Grid. The simulation results concerning the Hybrid Pricing Model (HPM), combined with a price-sensitive brokering strategy, are presented and analyzed in Chapter 9. Furthermore, the results are compared with a “worst case” (jobs are submitted to a randomly chosen computing resource) and a “best case” scenario (jobs are submitted to the resource that offer the lowest job completion time) in order to evaluate the efficiency of the HPM. Finally, the thesis is concluded in Chapter 10.

Appendix A is a brief introduction to some fundamental economic concepts and terminology of the general equilibrium theory. A legend of the notation used for the Unified Modeling Language (UML) diagrams contained in this thesis, can be found in Appendix C. Appendix D contains a glossary for the most important abbreviations used throughout the text.
Acknowledgments

The research work for this thesis was done for the Section of Turin of the Italian National Institute of Nuclear Physics (INFN) and with the support of Eurix Group Torino.

I want to express my sincerest thanks to the following nerds and hoopy froods for contributing to this thesis:

- Alberto, Angela, Antonio, Antonio, Baldo, “Franka Floppy”, Giorgio, “Giorgione”, Giuseppe, Gloria, Luciano, Marco, Massimo, Stefano, Stefano and all others at the INFN computing center (centro di calcolo) and the University computing center (centro rete) for the unforgettable time “down in the bunker”, and in particular Andrea Guarise for his instructions and his patience,

- Prof. Pollaro and Prof. Werbrouck of the University and INFN of Turin for their time and patience,

- Prof. Gamba of the University and INFN of Turin for his continuous support,

- Dr. Walter Allasia and Ing. Paolo Decarlini of Eurix Group Torino for their valuable help and interest in my research,

- Christian Träger of the Interdisciplinary Institute for Environmental Economics of the University of Heidelberg (Germany) for his valuable aid,

- and all members of the Workload Management Work Package (WP1) of the European DataGrid Project.
Contents

I The European DataGrid 1

1 Grid Computing and HEP at LHC 3
  1.1 LHC Physics 3
  1.2 HEP Data Production Process 4
    1.2.1 Triggering and Data Recording 4
    1.2.2 Reconstruction 5
    1.2.3 Batch Analysis and Interactive Analysis 5
    1.2.4 Event Simulation 5
  1.3 HEP Computing Challenges 6
  1.4 Key Characteristics of Computational Grids 7

2 The European DataGrid Project 9
  2.1 The Globus Toolkit 11
  2.2 Grid Security Infrastructure 12
  2.3 Grid Monitoring Services 13
  2.4 Data Management 14
    2.4.1 Data Replication 14
    2.4.2 Replica Selection 15

3 The Workload Management System 16
  3.1 The User Interface 17
    3.1.1 The Job Description Language 18
  3.2 The Workload Manager 19
  3.3 The Resource Broker 19
    3.3.1 Resource Discovery 20
3.3.2 Resource Classification .................................... 20
3.3.3 Matchmaking .................................................. 21
3.3.4 Ranking ....................................................... 22
3.4 The Job Adapter .................................................. 22
3.5 Job Submission and Monitoring ............................. 22
3.6 Composed Jobs .................................................. 23
3.7 Logging and Bookkeeping ..................................... 23

4 The DataGrid Accounting System ............................. 24
4.1 Basic Services of the Accounting System ................. 25
4.1.1 Resource Pricing ............................................. 25
4.1.2 Resource Usage Accounting ............................... 27
4.2 Advantages of the DGAS Infrastructure .................. 32

II Load Balancing and Economic Resource Brokering ....... 33

5 Resource Allocation & Load Balancing ....................... 35
5.1 The Resource Allocation Problem ......................... 35
5.2 The Load Balancing Problem ............................... 36
5.3 Computation Scheduling and Data Movement .......... 37

6 Economic Scheduling .............................................. 40
6.1 Advantages of an Economic Approach .................... 40
6.2 Computational Economies .................................... 42
6.2.1 Requirements of Computational Economies .......... 43
6.2.2 Market Models .............................................. 44
6.2.3 Commodities Markets for Resource Allocation ...... 46
6.2.4 Utility Function for Resource Ranking ................. 48
6.3 Market Equilibrium in Computational Grids ............. 50

7 Pricing Strategies .................................................. 53
7.1 Global vs. Local Resource Pricing ......................... 54
7.1.1 Global Pricing .............................................. 55
7.1.2 Local Pricing .............................................. 56
7.2 Local Pricing Algorithms .................................... 56
CONTENTS

7.2.1 Derivative Follower ........................................ 57
7.2.2 Hybrid Pricing Model ................................. 58
7.3 Performance Criteria ...................................... 59

III Simulation of Economy-based Load Balancing 61

8 DGAS-Sim(ulator) ............................................. 63
  8.1 Architecture of DGAS-Sim .................................... 63
  8.1.1 Resource Broker, Matchmaker and Utility Modules .... 64
  8.1.2 Computing Elements and Job Queues ..................... 65
  8.1.3 Price Authorities and Pricing Modules ................... 66

9 Simulation of the Hybrid Pricing Model .................... 67
  9.1 Preliminary Results .......................................... 67
  9.2 Simulation Framework ....................................... 73
  9.3 Simulation Runs ............................................. 76
  9.4 Analysis of Simulation Results ......................... 77
    9.4.1 Computational Energies and Request Denial Rates .... 77
    9.4.2 Queue Wait Times (QWTs) .............................. 78
    9.4.3 Relative Standard Deviation of QWTs .................... 80
    9.4.4 “Worst Case” and “Best Case” Scenarios ............... 81
  9.5 Evaluation of the Hybrid Pricing Model .............. 99

10 Conclusions .................................................. 102
  10.1 Simulation Tool and Simulation Results ............... 103
  10.2 Local Scheduling Policies and Per-job Pricing ....... 104

A General Equilibrium Theory .................................. 106
  A.1 Economic Model ........................................... 106
    A.1.1 Commodities and Price Vectors ....................... 106
    A.1.2 Economic Agents ...................................... 107
    A.1.3 Economies ............................................ 109
  A.2 Market Equilibrium ........................................ 110
    A.2.1 Excess Demand ....................................... 110
    A.2.2 Competitive Equilibrium ........................... 112
A.2.3 Existence of Competitive Equilibria ......... 115
A.2.4 Uniqueness of Competitive Equilibria ......... 116
A.3 The Tâtonnement Process ............................. 117
   A.3.1 Stability ......................................... 118
A.4 Nash Equilibrium in Game Theory ............... 119

B Pareto-Optimality and Total Utility .............. 120

C Notation of UML diagrams ........................... 123
   C.1 Class Diagrams .................................... 124
   C.2 Component and Collaboration Diagrams ........... 125

D Glossary .............................................. 126
Part I

The European DataGrid
The *European DataGrid (EDG)* [16, 36], that was funded by the European Commission under contract IST-2000-25182, is one of several “Computational Grid” projects developed to enable a large community of scientists to perform computationally intensive and data-intensive analysis by supporting a coordinated use of geographically distributed and heterogeneous computing, storage, and network resources, allowing them to be used as an integrated computing system.

The sharing of computational resources on a large scale — that has always been of interest for the INFN that has several sites all over Italy — has the potential to deliver order-of-magnitude increases in available computation, since many computational resource tend to be underutilized. The heterogeneity of these resources, the dispersion of their users, the size of scientific datasets, the limited network bandwidths, and the need for transparency, however, are the causes of novel technical challenges and require the development of new policies and new approaches to problems like scheduling of applications, authentication of users, and monitoring of system performance.

Although Grid computing might lead to new strategies for scientific exploration, as it allows for global access to fundamental scientific data, there is still a substantial gap between the promises of Grid computing and its current state. Further research and development will be necessary to deploy a working large-scale DataGrid before the LHC scheduled starting time in the year 2007.
Chapter 1

Grid Computing and
High-Energy Physics at LHC

1.1 LHC Physics

The Large Hadron Collider (LHC), that is currently being built at CERN, is a superconducting magnet accelerator ring of about 27 km circumference, designed as a \( pp \)-collider with a center-of-mass energy \( E_{\text{cms}} \) of 14 TeV (7 TeV per beam) available for new particle creation, but also for heavy ion collisions, which may evidence quark-gluon physics. The maximum collision energy for heavy ions (Pb-Pb) is 1148 TeV. The full peak luminosity of \( 10^{34} \text{ cm}^{-2}\text{s}^{-1} \) for \( pp \)-collisions and \( 1.95 \times 10^{27} \text{ cm}^{-2}\text{s}^{-1} \) for Pb-Pb collisions, will be reached after three years of LHC running. In the high luminosity phase about 24 minimum bias events are expected at each bunch crossing. Considering a bunch crossing frequency of 40.08 MHz, LHC will have a high interaction rate of up to \( 10^9 \) Hz. Its main purpose is the exploration of new physics such as the Higgs boson and possible Super-Symmetry (SUSY) particles like sleptons, charginos or neutralinos, but it may also yield new insight into CP violation and top quark physics. Events involving the Gauge bosons \( W \) and \( Z \), the possible heavier \( W' \) and \( Z' \) bosons in the TeV range, and their leptonic decays, e.g. the possible Higgs decay modes \( H \rightarrow ZZ^* \rightarrow 4l^\pm \) and \( H \rightarrow ZZ \rightarrow 4l^\pm \), are of primary importance for the LHC experiments, as well as are events containing \( b \)-jets originating from the decay of the heavy
Higgs and SUSY particles [11]. Each year the LHC will be operated for ten months for $p$-$p$ interactions and for one month for Pb-Pb interactions. At present the LHC experiments are expected to have lifetimes of 15 to 20 years.

The reconstruction of events requires an efficient identification and reconstruction of isolated tracks, an efficient tagging of jets, and the ability to unambiguously reconstruct and distinguish the interaction vertices from multiple events in the same bunch crossing. All this requires millions of read-out channels. The central tracker region of the Compact Muon Solenoid (CMS), for example, contains 10 million silicon strips for the Silicon Strip Tracker, and over 45 million pixels for the Pixel Vertex Detector, without even considering the calorimeters and the muon chambers of the outer regions [11, 12].

1.2 HEP Data Production Process

The data production and analysis process of the four LHC experiments — CMS, ALICE, ATLAS, and LHC-b — consists of several stages that produce different types of data.

1.2.1 Triggering and Data Recording

The high interaction rate of up to $10^9$ Hz produces a flow rate of about 40 TB/s of compressed data for CMS and ATLAS, of which most however is background. The first stage is the first level trigger, based on dedicated hardware, that reduces this flow rate to about 75 GB/s. It is further reduced to about 5 GB/s by a second level trigger, that consists of a brief, general analysis of the event data. A dedicated event builder software reconstructs the single event records, that have about 1 MB each. These event records are stored in a FIFO\textsuperscript{1} buffer, before a third level trigger, based on a cluster of personal computers, partially reconstructs the physical phenomena in order to decide which events to record on the permanent storage (usually tertiary storage like tapes). For CMS and ATLAS about 100 events will be recorded.

\textsuperscript{1}First In First Out
per second (flow rate of 100 MB/s). ALICE, that is designed for heavy ion collisions, will have a final flow rate of about 1 GB/s due to the much more complex Pb-Pb collision events. LHC-b will have a lower flow rate than the other three LHC experiments. Each year about 1 PetaByte of raw data will be recorded for each of the four experiments (for ALICE even an amount of up to 2.5 PetaByte per year can be expected [9]). The performance of an experiment and the data recording process are constantly monitored by accessing the raw data that is still buffered on disk, thus requiring an extended lifetime of this data. Additionally, parts of the data residing on permanent storage are accessed to verify its readability. Furthermore an independent stream of calibration data must be stored in order to allow a correct analysis of the raw data provided by the detector.

1.2.2 Reconstruction

The single events are reconstructed from the millions of signals of the detector’s read-out channels in order to determine physical information, such as the tracks, the energies and the momenta of particles and jets. The reconstructed physics data is called Event Summary Data (ESD) and has about one tenth to one half of the extent of the raw data. The ESD will be written to permanent storage as well.

1.2.3 Batch Analysis and Interactive Analysis

The analysis of the reconstructed ESD has basically two phases. In a first batch analysis phase an experiment summary, a so called Analysis Objects Dataset (AOD), is computed from the single events in the ESD. In a second phase the AOD can be analyzed interactively to visualize particle tracks, compute statistics, etc.

1.2.4 Event Simulation

The simulation of thousands of well-known events and the simulation of the raw data such events will produce is important for the development of correctly working analysis tools, such as real-time event filtering algorithms, for the previously described stages of the data processing and analysis process.
CHAPTER 1. GRID COMPUTING AND HEP AT LHC

This includes the tracing of simulated particles and jets through the detector geometry and their interactions with electromagnetic fields and detector materials. For ALICE, for example, at least $10^4$ central Pb-Pb events, each containing about $8.4 \times 10^4$ primary tracks at the interaction vertex, have been simulated with the HIJING event generator [2]. Furthermore, simulation can also help to evaluate the performance of a detector in terms of efficiency, acceptance, mass resolution, and momentum resolution with respect to different physics signals. For ATLAS about $10^7$ single muon events, $10^4$ background events, and $10^5$ physics events have been generated and analyzed in order to finalize the study of the Barrel Muon Trigger [2].

1.3 HEP Computing Challenges

The processing and analysis of the data produced by future HEP experiments represent large-scale data-intensive problems that pose new challenges to scientific computing.

Most HEP applications process large sets of experimental data in the form of single independent events that have only statistical correlations. Such applications can substantially benefit from parallel processing\(^2\), since the application can be executed in parallel on several computing systems with different input datasets. This requires a computing environment with a high throughput, i.e. that delivers large amounts of computational power over long periods of time.\(^3\)

The enormous amount of scientific data to be processed (10 PetaBytes and more of raw data, ESD and AOD per year only for the four LHC experiments) implies that the analysis has to be executed at geographically distributed centers.

Furthermore these large data collections are emerging as an important globally shared resource. The scientific communities that need to access and analyze this data are growing more and more and are almost always

\(^2\)In this case “parallel processing” should not be mistaken for processing on Symmetric Multi Processing (SMP) systems.

\(^3\)High Throughput Computing (HTC) — in contrast with High Performance Computing (HPC), that requires enormous amounts of computing power to be delivered over relatively short periods of time.
geographically distributed. Hundreds or even thousands of simultaneous users might need to access and process the data in an unpredictable way.

We thus require a new computing infrastructure beyond the physical infrastructure already present (laboratories, computing and storage facilities, networks) in order to effectively process the massive data resources produced by future HEP experiments. The only cost-effective way to do so is to deploy a worldwide DataGrid by integrating the computing resources of the institutions and computing centers that participate in the experiments. The computing requirements for LHC will be fulfilled by a number of large regional computing centers spread across Europe, North America and Japan, and of course the facilities at CERN, as proposed by the MONARC\textsuperscript{4} project [27]. For this purpose the European Commission funded the European DataGrid project, that successfully terminated at the end of March 2004, to develop the necessary middleware and deploy a prototype Grid infrastructure suitable for scientific computing.

1.4 Key Characteristics of Computational Grids

A number of key characteristics distinguish Computational Grids from traditional clusters and symmetric multiprocessors (SMP) and require new approaches to fundamental problems such as scheduling and data movement.

Due to the geographical distance between computing resources and due to their slow wide-area network (WAN) connections, data transfer times and network latencies become much more significant than in computing environments that can rely on high-speed local-area networks (LANs).

The potentially enormous number of entities (the number of participating sites and of simultaneous users) in large-scale Grid environments make the scalability\textsuperscript{5} of Grid architectures and policies a major concern of research and development efforts. Therefore most research focuses on decentralized approaches (in contrast to the mainly centralized management of clusters and SMPs).

\textsuperscript{4}Models of Networked Analysis at Regional Centers for LHC Experiments.

\textsuperscript{5}Scalability is the possibility to expand (scale) the dimension and the capacity of an information system, without intervention on its structure.
CHAPTER 1. GRID COMPUTING AND HEP AT LHC

Computational Grids are expected to have highly *dynamic characteristics* in terms of availability and capability of computing resources. Some resources might be shared only when not being utilized by local users, while others may be temporarily unavailable due to network or system failures. Hence Computational Grids require mechanisms that allows for runtime adaption to system conditions.

The computing resources of a distributed multi-institutional Grid environment belong to *multiple administrative domains*, therefore *distributed control* is an inherent property of Computational Grids and global policies should not interfere with site autonomy. Furthermore the resource sharing between multiple institutions implies additional policy issues regarding authentication and authorization of Grid users and requires an efficient *accounting* of their resource usage.

Additionally, Grids environments are *heterogeneous in nature* with respect to their computing resources (in terms of processing power, storage capacity, memory size, bandwidth, but also operating systems and installed software).

In the case of DataGrids further complexity arises from the fact that most applications are not only computationally intensive, but also *data-intensive*, in contrast to Computational Grids that usually deal with computationally intensive applications and small datasets. Thus in a DataGrid environments additional attention to data movement and data replication is important for a high performance.

Despite the large complexity of such computing environments, Computational Grids are supposed to couple the different computational resources and to present them as one *unified integrated resource*, such that “applications ‘plug’ into a ‘power grid’ of computing resources when they execute, dynamically drawing what they need from the global supply” [44]. This requires transparency for the user, that ideally should be unaware of the underlying infrastructure.
Chapter 2

The European DataGrid Project

The primary goal of the European DataGrid (EDG) project was the development of the middleware\(^1\), necessary to deploy a prototype of a DataGrid suitable for data-intensive scientific computing. The project has 21 contractual partners from all over Europe, of which the principal partners are the European Organization for Nuclear Research (CERN), the French National Center for Scientific Research (CNRS), the European Space Agency (ESA), the Italian National Institute of Nuclear Physics (INFN), the Dutch National Institute of Nuclear and High Energy Physics (NIKHEF), and the British Particle Physics and Astronomy Research Council (PPARC).

The project is organized in twelve work packages (WP1-12) that cover different areas of software development, deployment, applications, and project management. Figure 2.1 shows the organization of the technical work packages.

The development of the EDG middleware is divided into five main tasks concerning the management and scheduling of the workload (WP1), the data management (WP2), the monitoring and information provisioning (WP3), the management of the “computational fabric” (WP4), and the mass storage management (WP5).

\(^1\)“The services needed to support a common set of applications in a distributed network environment” [1].
Applications:
High Energy Physics (WP8)
Earth Observation (WP9)
Biology (WP10)

Applications build on EDG Middleware to access resources

Middleware:
Workload Management (WP1)
Data Management (WP2)
Monitoring Services (WP3)
Fabric Management (WP4)
Mass Storage Management (WP2)

EDG Middleware provides access to distributed and heterogeneous resources

Infrastructure:
Integration Testbed (WP6)
Network Services (WP7)

Figure 2.1: Organization of the technical work packages in the EDG project.

The responsibilities of the work packages for the integration testbed (WP6) and the network services (WP7) are the planning, the organization and the operation of a testbed of production quality.

The EDG middleware has primarily been developed for applications of three exemplary scientific fields. High-Energy Physics applications, as already described in the earlier Sections, are represented by WP8. The earth observation work package (WP9) and the molecular biology and genomics work package (WP10) focus on a uniform access to large, distributed databases and on the integration of these archives with already existing national testbeds, but also on new Grid-based algorithms for data mining, sequencing of genomes, and graphical interface tools.
CHAPTER 2. THE EUROPEAN DATAGRID PROJECT

The two non-technical work packages are concerned with dissemination (WP11) and the management of the EDG project (WP12).

The following sections briefly describe some core services required as basic infrastructure to build higher-level Grid middleware, and introduces specific parts of the EDG middleware that are essential for the scheduling of the workload, and thus for an effective load balancing. The Workload Management System itself and its DataGrid Accounting System, given their importance for this thesis, will be described in more detail in Chapter 3 and 4 respectively.

2.1 The Globus Toolkit

The Globus Toolkit [19], that has been adopted by various scientific Grid projects, is an open-source, community-based set of basic low-level services and software libraries that address issues of remote resource allocation, data management, the retrieval of information on characteristics and state of Grid resources, authentication, and communication security.

The Grid Resource Allocation and Management (GRAM) module and its “gatekeeper” service provide a common user interface for the allocation of a specific Grid resource that may be used with different Local Resource Management Systems (LRMS) such as LSF, PBS, and Condor. It interfaces between the higher-level middleware that is responsible for job\(^2\) submission and the LRMS.

Another important basic service provided by Globus is the Meta Directory Service (MDS) — also called Grid Information Service (GIS) — that helps to collect, organize and publish information on Grid resources. It is based on the Lightweight Directory Access Protocol (LDAP), that provides a well-defined model for accessing arbitrary data objects on directory servers.

The Grid Security Infrastructure (GSI), that shall be described in the following Section, is responsible for user and host authentication, single sign-on, and the delegation of user credentials.

\(^2\)An atomic application or execution task is often called a “job”.
CHAPTER 2. THE EUROPEAN DATAGRID PROJECT

2.2 Grid Security Infrastructure

A multi-institutional Grid environment for resource sharing requires security policies and mechanisms that can guarantee authentication and authorization across different administrative domains. All communication between entities (users and resources) has to be authenticated. Transmissions of confidential data (such as HEP analysis results) may have to be encrypted as well.

For this purpose, the EDG relies on the Globus Grid Security Infrastructure (GSI) [21] that is based on an implementation of the Generic Security Service Application Program Interface (GSS-API) defined in RFC 2078 and RFC 2743. Each Grid entity can be authenticated by means of an X.509 PKP certificate (RFC 2459) that has been signed by a gridwide trusted Certification Authority (CA). The signing CA has to guarantee the association between the public key contained in a certificate and the identity of the user or the host, that owns the certificate.\(^4\) Such an approach allows a secure authentication of all Grid entities, relying on trust relationships between a limited number of CAs.

An X.509 certificate is usually composed of several fields, of which the most important are the serial number, the issuer X.500 name (the name of the CA), the validity period, the subject X.500 name (the name of the certificate owner), the subject’s public key, the issuer unique identifier (the CA’s ID), and of course the CA signature. The signature is a hash of the certificate itself, that is encrypted with the CA’s private key, such that the certificate’s authenticity may be controlled using the CA’s public key.

Since most Grid operations (such as the submission of a job) involve multiple Grid components and resources, an efficient security infrastructure has to allow for “single sign-on”, a mechanism that requires users to authenticate (“log on”) only once to create a proxy certificate (signed by the user with his private key). A proxy credential may then be used by a job to authenticate itself to all Grid components and resources [20]. This pro-

\(^3\)Public Key Infrastructure. An infrastructure based on (asymmetric) cryptography with public and private key pairs.

\(^4\)The corresponding private key is known only to the user or the host.
procedure, called credential delegation, may also be used to create a restricted user proxy certificate and communicate it to a remote service, such that this service may act on the user’s behalf.

Security practices, however, call for proxy credentials with limited validity. Since it is possible that a long running analysis application does not complete before its proxy expires, a proxy renewal mechanism has to guarantee that valid credentials are available for the entire lifetime of the application.

2.3 Grid Monitoring Services

The Grid Monitoring work package (WP3) implemented a geographically distributed GIS based on the tools provided by Globus. The service is composed of one root Grid Information Index Server (GIIS) at CERN, one national GIIS in each country and one local GIIS at each site. Additional GIIS may contain information of Grid resources belonging to specific collaborations, such as the LHC experiments mentioned in Chapter 1. Each of the GIIS, with exception of the root GIIS at CERN, will be responsible for merely a subset of resources and will eventually direct incoming requests to the single Grid Resource Information Servers (GRIS) that have to be deployed on each Grid resource.

Various types of information can be published through the GIS using the Lightweight Directory Access Protocol (LDAP). The static resource information includes the physical characteristics and the configuration of a Grid resource, such as the number of processors and their architecture, the amount of physical memory, the operating system, the local resource management system, access policies, a list of services, protocols, and installed libraries (the runtime environment), etc. The dynamic resource information reflects its current status\(^5\), such as the number of idle processors and the number of pending jobs in its queue. Additionally, the GIS may be used for user information (user description and groupings for authentication purpose), network information (characteristics and status), and repository information (software and data repositories).

\(^5\)or its recent status, depending of the update interval of the GRIS.
2.4 Data Management

Addressing large scientific datasets — that in the case of EDG are mostly read-only (data stored once and rarely modified) — has an important impact on application scheduling, since large transactions over WANs, that have a relative low bandwidth compared to LANs, can influence the response time of the entire system. The issue of data location — data should, if possible, be located in the same LAN as the application — can be addressed by different approaches: data migration, application migration, or a combination of both strategies, as in EDG.

The primary goal of the Data Management (WP2) is the reduction of the data access overhead by actively replicating popular datasets towards sites that show a high frequency of access requests and by passively selecting the replica with the lowest access time for a given request.

2.4.1 Data Replication

Data replication can be compared to the caching of often used memory contents, as is done by modern processors. The Replica Optimizer developed by WP2 is a Grid service responsible for the optimization of data access and replication by triggering both replication and deletion of data files [4, 8].

A high degree of replication does not necessarily imply a low overall response time of the system, since the update overhead due to the synchronization of remote data copies increases with the number of replicas, causing an immoderate network traffic [39]. A partially replicated system, however, requires a data replication strategy based on quantity and quality constraints — data that contain the Higgs boson, for example, are supposed to be rare (low quantity), but will have a major importance for analysis applications (high quality) [39].

The Replica Optimizer uses a replication strategy based on an economic model, dynamically replicating or deleting datasets based on the demand (the number of requests) for them. Simulation results show that, even without significant tuning, an auction-based economic model can be as effective

\textsuperscript{6}The time that elapses between the submission of an application and its completion, also called completion time.
as the best simple file replication strategies and that in most cases it does
even bring performance improvements [4, 5]. Computational economies shall
be described in more detail in Section 6.2.

2.4.2 Replica Selection

For the purpose of *replica selection* a hierarchy of Replica Catalogues main-
tains a mapping between unique Logical File Names (LFN) — that are
abstract references to files, independent of their locations and their number
of replicas — and Physical File Names (PFN), that are URLs indicating the
physical locations of specific replicas.

Replicas may have access times ranging from seconds to hours (if located
on a tape that is not yet mounted). Therefore the replica selection process is
based on an access cost estimator that has been developed in collaboration
with the European CrossGrid\(^7\) project [40]. The *access cost* is defined as the
combined access time of the best replicas for retrieving all of the jobs input
files, taking into account the current network status [7, 40].

\(^7\)The CrossGrid Project. http://www.crossgrid.org
Chapter 3

The Workload Management System

An efficient distribution of the entire workload of the DataGrid — including job scheduling, submission, and the management of submitted jobs — is fundamental for an efficient execution of computing and data-intensive HEP applications with high throughput. For this purpose WP1\(^1\) has developed and implemented a Workload Management System (WMS)\(^2\) that addresses the particular requirements of a heterogeneous and distributed DataGrid (see Section 1.4). Figure 3.1 shows the current\(^3\) architecture of WMS and some of its dependencies on other EDG components, not including the WMS Logging and Bookkeeping Service.\(^4\) The following Sections briefly introduce the WMS components that are crucial for job submission and job scheduling, and thus for load balancing and throughput optimization. The DataGrid Accounting System will be described in more detail in the next Chapter.

\(^1\)WP1 includes members from INFN (Italy), DATAMAT Ingegneria dei sistemi S.p.A. (Italy), CESNET (Czech Republic), and the Imperial College London (United Kingdom).

\(^2\)The WMS architecture has significantly changed for EDG release 2.0.

\(^3\)For a legend of the used UML notation, see Appendix C.
Chapter 3. The Workload Management System

3.1 The User Interface

The User Interface (UI) provides users and applications with access to the functionalities offered by the WMS. The most important functionalities include the submission of an application for execution on remote Grid resources\(^1\) (this includes the transfer of the executable and an eventually necessary set of input files — called the “Input Sandbox” — to the executing node), the cancellation of one or more submitted jobs, the retrieval of the output files of a completed job (also called its “Output Sandbox”), the retrieval of logging and bookkeeping information of a submitted job (see Section 3.7), and the registration of a local listener for interactive jobs.

The UI assigns a unique identifier \(\text{edg}_\text{jobId}\) to each job or subjob.

\(^1\)The user may explicitly specify a particular resource or may leave the automatic resource discovery to the WMS.
CHAPTER 3. THE WORKLOAD MANAGEMENT SYSTEM

This identifier is a URI\(^5\) that includes information about the Logging and Bookkeeping server that is responsible for keeping the job state, the Workload Manager that should schedule it, and the UI machine itself.

3.1.1 The Job Description Language

Since resource management should be as simple and transparent as possible for the user, most interactions with the WMS are limited to the specification of resource requirements for a submitted job by means of a high-level Job Description Language (JDL). This highly flexible and extensible language is based on Condor Classified Advertisements (ClassAds) [13, 31].

A ClassAd is composed of a list of attribute/value pairs of the format “attribute = expression;”. Attributes may be assigned values of different types, such as integers, floating-point values or alphanumeric strings, or they may be mapped to more complex expressions, constructed with arithmetic and logical operators. The whole semi-structured\(^6\) description has to be included between square brackets, i.e. “[<job description>]”. Both resource requests and resource characteristics, can be described by similar JDL expressions. This symmetry simplifies the matching of requests to suitable resources (see Section 3.3.3). ClassAds can also be nested to represent aggregates of resources or jobs.

Although ClassAds may contain arbitrary attribute names, the WMS components take into account only a defined set of “supported attributes” [29]. Of these, two attributes are of particular interest for job scheduling and thus for load balancing. The requirements attribute is a boolean expression that specifies resource characteristics that have to be matched for a submitted job (see Section 3.3.3). The rank attribute is a floating-point expression that is used to compute a level of suitability of a resource for the given job (see Section 3.3.4). For this purpose, both the requirements and the rank expression usually refer to attributes of other ClassAds by means of the prefix \texttt{other}.” (see Section 3.3.3).

\(^5\)Uniform Resource Identifier, as defined by RFC 2396.
\(^6\)Attributes do not have to be ordered according to a well-defined schema.
3.2 The Workload Manager

Most user requests are forwarded from the UI to a Network Server, a generic network daemon that merely verifies the requests’ validity and eventually passes them to the Workload Manager that resides on the same node.

The Workload Manager (WM) is the core component of the WMS, whose purpose is the processing of valid user requests. This may require calling one or more specific Helpers, according to the type of the request. WM Helpers are components that, given a job description, take a well-defined action and return a modified JDL expression. The Resource Broker (see Section 3.3), for example, returns the original job description extended by information on the specific resource that has been selected for job submission.

3.3 The Resource Broker

The particular challenges of Grid Computing discussed in Section 1.4 make traditional schedulers inappropriate for distributing and executing submitted applications in an effective and transparent way. Although the Globus Toolkit (see Section 2.1) offers some basic functionalities for resource management, it does not provide a grid scheduler (sometimes called metascheduler or superscheduler). The optimization of the system’s performance, however, requires the enforcement of appropriate scheduling policies, based on the heterogeneous characteristics of Grid resources and their fluctuating availability and state. The highly dynamic characteristics of the EDG environment call for dynamic resource discovery and “just in time scheduling” [32] where the single jobs are scheduled as and when they are submitted to the metascheduler.\(^7\)

Scheduling policies may be system-oriented — aiming at optimizing system performance metrics, such as throughput or system utilization (High Throughput Computing, HTC) — or application-oriented — aiming at minimizing the turnaround time of a single given application (High Performance Computing, HPC). EDG, due to its importance for large-scale parallelizable

\(^7\)In contrast to an “offline” context in which all jobs are known before scheduling begins, and the objective is to minimize the makespan, i.e. the time to complete all jobs.
scientific applications (see Section 1.3), focuses on HTC, nonetheless trying
to satisfy the users' needs for Quality-of-Service (QoS) for single applica-
tions.

For this purpose WP1 has implemented a component called Resource
Broker (RB) or Matchmaker\(^8\). The RB is called as a Helper by the WM if a job's description does not specify a particular resource for submission. Deploying the RB as a pluggable WM Helper increases the flexibility of the entire system by providing different institutions the possibility to adopt highly specialized scheduling policies, and allows for interoperability with other Grid frameworks [3]. Given a job's JDL expression, the RB chooses the most appropriate Computing Element (CE)\(^9\) and returns a modified JDL expression containing the identifier (CEId) of the selected resource. The currently implemented resource selection process can be divided into different phases and is described in the following Sections.

### 3.3.1 Resource Discovery

In the first phase the RB interacts with the Grid Information Service (see
Section 2.3) to accumulate capability-, performance-, and eventually price-
information about available resources. The set of possible candidates, for
which the user has access permission, is determined by a query to a Grid In-
formation Index Server (GIIS), after which the RB contacts the corre-
ponding Grid Resource Information Servers (GRIS) to obtain more detailed and
up-to-date information. The resource characteristics are translated from
LDAP into JDL expressions that are required for the matchmaking phase.

### 3.3.2 Resource Classification

If the JDL expression of a submitted job specifies input files from remote
Storage Elements (SEs)\(^10\), the RB, before performing the actual match-

---
\(^8\)The term "Matchmaker" indicates that the EDG RB is not responsible for submitting
the job to remote resources, as is true for some other Grid projects.
\(^9\)An EDG resource that provides computational power. It may be a single-processor
computer, an SMP, or a cluster.
\(^10\)EDG resources that provide storage for data replicas, e.g. for Event Summary Data
of HEP experiments.
making, interacts with the Replica Management services (WP2) in order to classify the available CEs according to the number of input files residing on SEs that are “close” to the CE. This takes into account the impact of large data transfers on the system performance (see Section 2.4).

3.3.3 Matchmaking

In the requirements checking or matchmaking [31] phase the RB matches the attributes of the job’s JDL expression to syntactically equivalent attributes in the descriptions of the available CEs, in order to determine a set of suitable resources that meet user and job requirements (e.g. runtime environment, processing power, physical memory, local disk space, resource price, etc.) — considering, however, local policies and local resource management constraints.

The matchmaking algorithm is a generic evaluation of one ClassAd with respect to another one, in this case the job’s description and a resource’s descriptions. The two ClassAds match if both have an attribute named “requirements” that evaluates to true in the context of the other ClassAd. Attributes of the ClassAd that have to be matched can be accessed via the pseudo-attribute other, e.g. the expression “other.NumberCPUs > 16” evaluates to true if the other ClassAd contains an attribute named “NumberCPUs” having an integer value greater than 16.

If necessary, the single bilateral match may be replaced by an explicit list of multiple bilateral matches (“gangmatching”) in order to include both computational and storage resources in the matchmaking. This, for example, allows to consider only CEs that are “close” to an SE that has “enough” free space for the job’s output.

If the discovered resources have been classified according to the number of input files on close SEs, as described in the previous Section, only the class with the highest number of close input files will be considered for matchmaking. If the class doesn’t contain any matching resource, the class with the second highest number of input files will be evaluated, and so on, until at least one matching resource has been found.
3.3.4 Ranking

The final resource selection is based on the job description’s rank attribute, that evaluates to a floating-point value and represents the quality of the match. The rank expression should include the resource’s price information and current workload in order to allow for load balancing and to optimize the system utilization.

The current RB implementation allows to use the data access time estimation provided by the Replica Manager (see Section 2.4.2) for ranking (“rank = other.dataAccessCost;”), instead of the previously described classification process. In this case, however, the ranking expression may not contain any other attribute, and thus the current workload of the single resources cannot be included in the scheduling decision.

3.4 The Job Adapter

The Job Adapter, or Job Wrapper, is responsible for the final adjustments of the job’s JDL expression, before it can be submitted to the executing node. It also wraps the job with a script, that creates the appropriate execution environment on the CE worker node, including the download of the Input Sandbox (the input files residing on the submitting computer), the setting of environment variables, and the upload of the Output Sandbox (the job’s output files that have been specified for retrieval by the user) to the user’s computer after job completion.

3.5 Job Submission and Monitoring

The actual job submission is performed by the Job Controller, built on top of Condor-G [14, 22], a module responsible for performing job management operations, such as job submission, job removal and job status monitoring on remote Globus resources via GRAM. Condor-G keeps logging information in a persistent log file, that is regularly checked by a Log Monitor that perceives important events affecting the job state machine and triggers

\footnote{Condor-G is not part of the EDG project, but is developed by the University of Wisconsin, that collaborates with WP1 to simplify its integration into the WMS.}
appropriate actions (such as resubmission in case of a system crash of the executing CE).

3.6 Composed Jobs

The WMS also provides some basic support for composed jobs, such as Directed Acyclic Graph (DAG) jobs that establish a predefined order of single batch jobs. A DAG Manager resolves the dependencies between the batch jobs and submits them in the appropriate order, executing first parent jobs that supply the necessary input for their respective child jobs.

Since HEP applications often process large sets of independent events, a Job Partitioner may decompose them into smaller subjobs, each subjob being applied to a subset of the entire dataset (“trivial parallelization”) [3].

Due to the nature of most DataGrid applications, however, for the purpose of load balancing we can make the simplifying assumption of all jobs being atomic.

3.7 Logging and Bookkeeping

Most WMS components use a Logging and Bookkeeping (LB) service to store important information regarding their own status and the jobs they manage.

The volatile (short-term) bookkeeping information refers to currently active jobs. This kind of information is mainly required to maintain a job state machine and may also be accessed by the user via appropriate UI commands.

The persistent (long-term) logging information instead concerns mainly the WMS components and their status. It is used above all for debugging purpose and statistical analysis of job executions.

\[\text{based on a "push" model} \]
Chapter 4

The DataGrid Accounting System

A distributed multi-institutional grid computing environment requires mechanisms for a balanced access of many thousands of simultaneous users. Instead of allowing for unrestricted access, mechanisms for controlled sharing of Grid resources have to be enforced. This includes both local policies (such as access constraints for specific resources according to group membership) and global policies (only applications of authorized users will be executed, see Section 2.2) in order to avoid abuses of the resources.

The objective of resource exchange fairness — services rendered at one time can be recovered later without significant loss or gain\(^1\) — additionally requires an accurate accounting of the resource usage of all authorized Grid users.

In order to facilitate the fair and balanced exchange of computing resources, the Section of Turin of the Italian National Institute of Nuclear Physics (INFN) has designed and implemented the DataGrid Accounting System (DGAS) \([23, 30]\), an infrastructure for the accounting of resource usage in computational grids, developed for the EDG Workload Management System (WP1).

DGAS is designed to support an economy-based approach to regulating

\(^1\)This does not necessarily imply price stability, since from an economic point of view a price is fair if it reflects the value of a good that in turn depends on demand and supply.
the distribution of the resources among the authorized users of the Virtual Organizations that participate in the Grid. These VOs may correspond to “physical” organizations (INFN, CERN, etc.) or “logical” ones (e.g. researchers participating in a specific project or experiment).

4.1 Basic Services of the Accounting System

The fundamental services of which DGAS is composed — responsible for resource usage accounting and resource pricing — and their interactions with other EDG components are shown in a simplified diagram in Fig. 4.1. DGAS services are based on the assumption of distributed and decentralized control, to assure scalability, and the existence of trust-relationships between all VOs that are collaborating in the Grid.

![Simplified DGAS interaction diagram](image)

Figure 4.1: Simplified DGAS interaction diagram.

4.1.1 Resource Pricing

The resource prices are determined by a number of Price Authority (PA) servers, each of them being responsible for a subset of Grid resources. We propose to have one PA per Virtual Organization, but DGAS can be deployed with an arbitrary number of PAs. Resource prices are expressed in
a virtual currency, called *Grid Credits*, per *Unit of Computational Energy* (UCE).

**Computational Energy**

In analogy to physics, we define the *computational energy* “consumed” by a job, as the product of a performance factor or power $p$ (e.g. a benchmark of CPU speed) and the amount of usage $u$ (e.g. the CPU time), which ideally should be independent of the resource’s characteristics [30]. The resolution of the units of computational energy is variable, but as noted by Wolski et al. [45], a Computational Grid will need to work with large-scale aggregations of resources for reasons of efficiency. The concept of computational energy may be applied to different resource types, such as computing, storage and network resources. In case of computing resources, an accurate benchmark for processing power is required.

**Price Quotations**

The single PA servers can be queried and furnish valid price quotations for the resources within their administrative domain. The current implementation of the PA service adopts a *per-resource pricing*, where resource prices depend only on the current state of the resource — in contrast to *per-job pricing* where prices may also vary according to the job for which a price quotation is requested.

The pricing algorithm is implemented as a dynamically linked library (DLL), such that pricing strategies may be changed without restarting the PA server. Theoretically, different pricing schemes might be adopted by the single VOs.

Since prices are valid for a certain time interval (the current implementation foresees a minimum time-to-live for each resource price), each PA server keeps a history of resource prices, including their respective GMT timestamps, such that for long-executing jobs the prices that were valid at job submission time can be retrieved even after they have expired.

The described pricing infrastructure provides the possibility to dynamically adjust resource prices and thus enable a Resource Broker (RB) to base
scheduling decisions on economic models. An appropriate pricing scheme can help to improve workload balancing by providing self-regulation mechanisms. As in real market economies the prices for idle resources may be lowered to attract “consumers” while those of overloaded resources may be raised, in order to reasonably balance average utilization and computational efficiency of the Grid as a whole. This aspect is most important for the overall throughput of the Grid and will be discussed in more detail in Chapters 6 and 7.

4.1.2 Resource Usage Accounting

The accounting of the resource usage by all authorized DataGrid users is realized by a set of geographically distributed servers, named Home Location Registers\(^2\) (HLRs). Each of these “bank” servers manages a subset of Grid users and resources\(^3\), maintaining their respective account and transaction information in a relational database.

The proposed mechanism is depicted in detail in the component diagram in Fig. 4.2 and the collaboration diagram in Fig. 4.3.\(^4\) It is based on economic transactions between a User HLR, i.e. the HLR that manages the account of the Grid user that submits a job (“consumer”), and a Resource HLR, i.e. the HLR that manages the account of a Grid resource that executes the job (“producer”). All services are implemented on a client/server basis. Clients exist as standalone command line programs, but may also be directly integrated into other applications due to the provided C++ API. The communication protocols are based on XML, chosen for its flexibility and ease of use. Since security is a major objective of a reliable accounting mechanism, the HLR servers, as all other DGAS components, rely on the Globus GSI (see Section 2.2) in order to ensure that all communication is mutually authenticated and encrypted.

\(^2\)The name is inspired by the GSM mobile phone network.
\(^3\)As for Price Authority servers we propose to have one HLR per VO.
\(^4\)For a legend of the used UML notation, see Appendix C.
Figure 4.2: DGAS component diagram.
DataGrid Accounting System

1: <<submit>> {subject=job}
   <<client>>:jobAuth_client
   <<server>>:jobAuth_engine
2.1: <<notification>> {subject=transaction}
2.2: <<register>> {subject=transaction}
3.1: <<request>> {data=price}
   <<client>>:user_auth_client
   <<server>>:user_auth_engine
3.2: <<query>> {data=price}
3.3: <<compute>> {subject=job cost estimation}
4.1: <<request>> {subject=economic authorization}
   <<client>>:ATM_client
   <<server>>:ATM_engine
4.2: <<query>> {data=price}
5.1: <<submit>> {data=usage records}
5.2: <<send>> {data=usage records}
5.3: <<request>> {data=price}
5.4: <<query>> {data=price}
6.1: <<query and remove >> {subject=transaction}
6.2: <<process>> {subject=transaction}
6.3: <<end>> {subject=payment check}
6.4: <<update>> {action=debiting} {subject=account status}
6.5: <<insert>> {subject=transaction}
6.6: <<update>> {action=crediting} {subject=account status}

Title: DGAS collaboration diagram  Author: Rosario Piro, piro@to.infn.it  Date: March 4, 2004  Rev: 0  PUBLIC
CHAPTER 4. THE DATAGRID ACCOUNTING SYSTEM

Job Submission

Upon job submission the user specifies the contact information of the HLR server, that keeps his account information, through an appropriate JDL attribute (HLRLocation = "<host>:<port>:<X.509 contact string>";) in the job description (see Section 3.1.1). This information is necessary for a correct accounting of resource usage, since in a large-scale Grid environment users may belong to several VOs (e.g. INFN and the CMS experiment) and thus might have several accounts on different HLRs.

Job Authorization by the User

In order to authorize the payment transaction in advance, the UI uses the user's proxy credentials to send a notification containing the subject of the user's X.509 certificate and the job identifier (edg_jobId) to the corresponding User HLR (steps 2.1 and 2.2 in Fig. 4.3). This user authorization informs the accounting system of the existence of a job of a particular user, assuring that the job will be debited to its real owner. For this purpose the HLR uses the edg_jobId as a transaction identifier.

Job Cost Estimation

The authorization to use a specific resource should depend not only on the authorization policy of the resource owner, but also on the availability of user credits. Therefore the Resource Broker should estimate the overall cost of a job before the job can be sent to the Computing Element (CE). The job cost estimation, that is represented by steps 3.1 to 3.3 in Fig. 4.3, depends on the resource price that is requested from the responsible PA, and of the resource usage expectations in UCE. These are provided by the user's job description or eventually by predictions based on historical job information, as for example described by Smith et al. [38] and Li et al. [25].

Economic Authorization by the HLR

The RB then initiates a "fund adequacy" check by querying the User HLR in order to determine whether the user can afford the job to be computed or not.
(steps 4.1 and 4.2 in Fig. 4.3). The single VOs may adopt different economic authorization policies for their HLRs and users. We propose that if the job cost is covered by the user’s funds, the requested amount is reserved, hence allowing the user to submit multiple jobs only if their payment is covered.

Thus accounting and authorization are tightly bound in our model. The authorization mechanism can be split in two distinct layers: a user authorization per resource depending on the resource’s local policies and a user authorization per job depending on the user’s funds.

Job Cost Computation

An adequate job cost computation (steps 5.3 to 5.5) requires the Computing Element to determine the exact resource usage by a job, and to communicate it to the User HLR\textsuperscript{5} (steps 5.1 and 5.2). The User HLR requests the historical price information (according to the submission GMT timestamp) from the responsible PA in order to compute the job cost based on the resource prices valid at the time of job submission.

In the current implementation the light-weighted Job Monitor reports the job usage records only after job completion. This may cause problems in the case of an underestimation of the job cost, since the user account will be debited much more then expected. Instead, the Job Monitor and the ATM client should periodically submit the usage records to the User HLR, such that the latter may intervene by causing the job execution to be suspended if the user should run out of credits.

Credit and Debiting

The crediting and debiting process (steps 6.1 to 6.4) is initiated by Automatic Transaction Managers (ATMs) that are deployed on every HLR server. The economic transaction information is queued to a Transaction Queue such that it may be processed asynchronously (step 5.6), increasing the fault-tolerance of the system. In case of transaction failures it remains in the queue without being lost.

\textsuperscript{5}This reflects the current implementation, in the future the crediting and debiting process may as well be initiated by the Resource HLR instead of the User HLR.
The Transaction Manager processes each transaction stored in the queue by forwarding it to an HLR client that sends a payment check to the Resource HLR and waits for a receipt (step 6.3). Only then does the HLR client debit the user account (step 6.4). Reserved funds that have not been spent (overestimation of the job cost) are unlocked.

The accounting procedure is described in more detail in [23].

4.2 Advantages of the DGAS Infrastructure

The primary scope of the described accounting process is the tracing of resource usage by Grid users, independent of the way in which resource prices are determined and thus of the underlying economic model. It may be applied in both centralized co-operative and decentralized competitive distributed computing environments.

Each authorized Grid user should receive the amount of credits he needs for the execution of his applications from the management of his VO. The single VOs redistribute the credits earned by their resources among the users belonging to that VO, such that these may “buy” resources of other VOs in return. This approach may allow for fair resource exchange — given a relative price stability — since VOs, or their respective users, may acquire an amount of resources that is similar to the amount they provide for the benefit of other Grid users. Additionally, a relation between Grid Credits and real currencies might be established in order to make it feasible for large computing centers to share their resources even if their users don’t require Grid services in return, as well as allowing small laboratories or research groups to utilize these services even if they cannot contribute their own resources.

DGAS does not require a specific topology, allowing for an arbitrary and scalable number of HLRs and PAs, and may thus be adopted for computing environments with centralized control as well as for decentralized Computational Grids.

An appropriate pricing scheme can furthermore help to improve workload balancing by dynamically adjusting resource prices according to the current workload, as described in the following Chapters.
Part II

Load Balancing and Economic Resource Brokering
The utilization of the DataGrid for large-scale applications from different scientific domains, the relatively large numbers of users, and the dynamically changing performance characteristics and availability (due to system and network failures) of the distributed heterogeneous resources imply the generation of an unpredictable and chaotic workload.

Thus the optimization of resource usage is most important in order to guarantee acceptable execution times of user applications and fairness among them. This requires dynamic scheduling mechanisms for the choice of execution location that easily adapt to the current status and performance characteristics of the single computing, data and network resources and the Grid as a whole.

Most traditional dynamic load distribution systems have been developed for a centralized homogeneous context, such as SMP or homogeneous computer clusters and hence are inapplicable to the scheduling problem in a Grid environment, especially when considering issues like scalability, policy differences of the multiple administrative domains, and the necessity to avoid a single point of failure and to provide reliability by re-optimizing the scheduling configuration to account for system and network failures.

For this purpose scheduling algorithms based on economic principles are being studied for adoption to a Grid environment. The following Sections concern the improvement of workload balancing by including price information in the resource selection process to ensure both high throughput and stability of the grid environment, but also to satisfy design criteria such as scalability, low computation overhead and compatibility with different local policies. For an introduction to the relevant economic theory and terminology, see Appendix A.
Chapter 5

Resource Allocation and
Load Balancing

Due to the scarcity of resources and their dynamic characteristics, the issue of resource allocation and the distribution of the workload is of fundamental importance for an effective multi-agent (or multi-entity, including users, applications and resources) computing environment.

5.1 The Resource Allocation Problem

Mathematically, a general resource allocation problem can be described as a maximization problem (see for example [46, 47]):

Definition 5.1 (Resource Allocation Problem) Let \( \mathcal{N} = \{1, \ldots, N\} \) be a set of resource types and \( \mathbf{R} \in \mathbb{R}^N \) the vector of total resources in the system. Let further \( \mathcal{M} = \{1, \ldots, M\} \) be a set of resource users and \( f_m(\mathbf{r}) : X_m \to \mathbb{R}, m \in \mathcal{M} \) the (monetary) value of allocating \( \mathbf{r} \) to resource user \( m \), with \( X_m \subseteq \mathbb{R}^N \) being the set of possible allocations for \( m \). The maximization problem \( MP \) is given by

\[
\max_{\mathbf{r}} \sum_{m \in \mathcal{M}} f_m(\mathbf{r}_m) \tag{5.1}
\]

s.t. \( \sum_{m \in \mathcal{M}} \mathbf{r}_m = \mathbf{R}, \forall (\mathbf{r}_m)_{m \in \mathcal{M}} \) and \( r^l_m \leq r_{mn} \leq r^u_m \).
where \( r_m = [r_{m1}, r_{m2}, \ldots, r_{mN}] \in X_m \) is an allocation of resources with \( r_{mn}^l \) and \( r_{mn}^u \) being lower and upper bounds for allocating resource \( n \) to user \( m \). The aggregate resource allocation is given by \( (r_m)_{m \in M} = [r_1, r_2, \ldots, r_M] \).

Note the similarity to the concept of utility functions, utility maximization and Pareto-optimal allocations in economic theory, described in Appendices A.1.2 and A.2.2. An aggregation of the preference functions of individual resource users as in Def. 5.1, however, makes sense only if the functions \( f_m(r) \) have a common unit (e.g. if they describe monetary values based on a common currency) — a restrictive assumption that in general is not required by the general equilibrium theory.

In a Grid environment, the resource allocation process is characterized by the requirement of co-allocation, i.e. the allocation of “bundles” of resources of eventually different types. The execution of typical applications may require a collection of CPUs and may involve large amounts of data storage for input or output. Consequently, the requests for different resource types are often correlated and Grid resources can often be considered gross complements (see Def. A.5 in Appendix A.2.1).

The two major objectives associated with multi-user resource allocation are fairness among users and their applications and the balance between the overall throughput of the system (“High Throughput Computing”) and Quality of Service (QoS) for the single applications (“High Performance Computing”), thus seeking high utilization, but not at the expense of QoS and vice versa.

### 5.2 The Load Balancing Problem

In homogeneous parallel computing (such as SMP) with predictable workloads, where all tasks are known at least approximately before scheduling begins, most load balancing strategies seek a minimization of makespan, that is, the total time required to finish all tasks. In contrast, the main goals of load balancing systems in less predictable Grid environments are the maximization of the throughput — i.e. the average number of tasks executed per time unit —, the reduction of the average job wait time — i.e. the inter-
val between job submission and and execution start — and the average job turnaround time (or job completion time) — i.e. the time interval between job submission and job completion — through an appropriate distribution of the total system workload across all available resources.

Note that different sites may have different local scheduling policies and local resource management systems. In a distributed environment the main problem, however, is the distribution of workload among the different sites. Therefore, only issues of global load balancing will be considered for this thesis.

In general, load balancing systems can be distinguished according to the following two criteria:

- **state-based** load balancing strategies are based on the actual state of the single resources and the system as a whole,\(^1\) while **model-based** strategies are based on load prediction models and may thus be less accurate and less flexible, but easier to adopt.

- **mobile** load balancing strategies allow for a migration of tasks from one resource to another, if the latter should be considered more suitable at a later stage, while in **static** load balancing environments tasks do not migrate to another host once they have been submitted for execution.

Due to the nature of the EDG Workload Management System and the particular issues raised by DataGrids (see Section 1.4) static and status-based load balancing seems to be most promising. In a large-scale environment with thousands of loosely coupled atomic jobs,\(^2\) an effective load balancing will most probably not require the migration of single tasks/jobs.

### 5.3 Computation Scheduling and Data Movement

As outlined in Section 2.4, the data-intensive nature of most DataGrid applications requires to consider data location when scheduling jobs to the

\(^1\)Note, that the information about system and resource status is not necessarily up-to-date, since it may be expensive to obtain.

\(^2\)Most HEP applications process large amounts of experimental data in the form of single independent events.
executing sites, in order to reduce the impact of data transfer times and network traffic on the overall performance.

Thus, dynamic replication of popular datasets is a fundamental part of the load balancing problem. Scheduling strategies that merely focus on maximizing processor utilization, without considering the access cost of fetching data from remote Storage Elements, may be inefficient [32, 33, 7].

However, as shown by Ranganathan and Foster [32, 33], the tight relation between job scheduling decisions and data replication may be “decoupled” if data is dynamically replicated and jobs are scheduled to sites on which the required input data is already present. Hence, job scheduling and data replication strategies may be implemented and optimized separately, if the job scheduling algorithm chooses among the sites that offer local access to the input data required by a given job.

Since some scientific applications may as well require multiple input files located at different sites, it will not be possible to schedule all jobs such that they may access all data locally. Moreover, as pointed out by Cameron et al. [7], job scheduling algorithms that merely seek to minimize the access cost for remote data may lead to a low utilization of Storage Elements and the processing power of Computing Elements, since they clearly favor sites with high network connectivity due to the fact that these sites generally offer lower data access times. Furthermore, the impact of future network capacity, leading to continually improving bandwidths, may reduce the negative effects of data access times on the overall performance [32]. Similarly, with increasing job run times, computational power, and not bandwidth, becomes the bottleneck in a distributed computing environment, since input data can already be fetched from remote sites while jobs are still pending in the Computing Element’s queue [32].

An efficient workload management thus requires scheduling strategies that, while taking into account data location, seek to maximize system

---

3 That is, the combined access time of the best replicas for retrieving all of the jobs input files.

4 As is done in EDG, see Section 2.4.1.

5 The simulation studies of Ranganathan and Foster, however, considered only task that require a single input file.

6 As is done by the EDG Resource Broker in the matchmaking phase, see Sections 3.3.2
utilization and throughput by considering the characteristics and current workload of the single resources. One of the most promising and flexible approaches — since it may include different criteria such as data location, resource characteristics and their current workload, and even personal preferences of users — is based on economic principles, as described in the following Chapter.
Chapter 6

Economic Scheduling

The congestion control in a large-scale Grid environment requires a dynamic adaptation to the rapidly changing number of users and resource conditions, which can be based on a “virtual economy” in which users pay (with virtual credits) for the Grid resources they utilize. This Chapter is mainly concerned with the application of economic concepts to the resource allocation and load balancing problem in Computational Grids. For an introduction to general equilibrium theory and economic terminology, see Appendix A.

6.1 Advantages of an Economic Approach

The massive complexity of the simultaneous allocation of multiple resources and resource types for a highly dynamic set of users and applications requires an integrated flexible strategy that easily adapts to different system conditions and resource request patterns.

It is widely believed that an economic approach offers natural self-regulating mechanisms that can help with allocating resources to Grid users in a fair and effective manner by balancing demand and supply (reaching market equilibrium as long term optimization), and by balancing the incoming workload among the participating Grid resources (“just-in-time” or online scheduling as short term optimization). A long term optimization by bringing the “resource market” in equilibrium\(^1\) can help to lower the impact

\(^1\)Or at least approximate equilibrium, since the existence of an equilibrium is not guaranteed, as described in Section 6.3 and Appendix A.2.3.
of the demand’s periodic cycles (day and night, weekdays and weekends, ...) by providing incentives (through price differences) to delay the submission of less urgent jobs to periods in which the overall workload on the Grid is low, at the same time guaranteeing reasonable execution times for users or applications with more stringent requirements by paying more in periods of high system load.\(^2\)

Additionally, an economic approach is a natural motivation for resource contribution, for the benefit of the contributing Virtual Organization and its users, as well as for external users. As described in Section 4.2 the Grid Credits earned by sharing resources can later be utilized in order to benefit from resources offered by other participants. If necessary, a relation between Grid Credits and real currencies might be established in order to make it feasible for large computing centers to share their resources even if their users don’t require Grid services in return, as well as allowing small laboratories or research groups to utilize these services even if they cannot contribute own resources.

One of the most important advantages of most (but not of all, as we will see later) economic resource allocation models is the possibility to perform complex multidimensional\(^3\) optimizations in a decentralized manner, while respecting local control of the shared resources, which is of major importance in a computing system with multiple administrative domains. A decentralization of allocation and scheduling decisions is crucial for the scalability — in terms of both the number of resources and the number of tasks that can be managed by the system — required for a large-scale environment such as EDG.

Furthermore, having user accounts with a limited budget offers a simple way of assigning different priorities, i.e. different amounts of Grid Credits, to the single users. In an overloaded Grid, that will usually imply higher resource prices, users with a higher priority level (and thus more funds) should still be able to submit jobs, while users with a lower priority level

\(^2\)Even a small portion of adaptive users may bring a significant performance benefit and thus better service for all users, both adaptive and non-adaptive ones [43].

\(^3\)The planning domain is usually composed of a large number of system and resource characteristics that have to be taken in consideration.
CHAPTER 6. ECONOMIC SCHEDULING

may have to delay job submissions. Thus, as described in Section 4.1.2, in an economic model for allocation of Grid resources, accounting and authorization are tightly bound. Budget limits also help to force users to responsibly utilize the scarce resources, e.g. by preventing them from saturating the system with job-trials.

Finally, the intuitive familiarity with basic economics of most users (and administrators) simplifies the understanding of the system by hiding complex allocation and performance details behind a “computational cost”.

6.2 Computational Economies

Wolski et al. [45] define a computational economy as an artificial economy, set up under a certain set of constraints in order to make it obey a certain set of economic principles, in which consumers are represented by users and their applications (or the Resource Brokers on their behalf) and suppliers by the various Grid resources (or the Price Authorities on their behalf). In contrast to the idealized economies described by the general equilibrium theory (see Appendix A), however, the set of agents (both consumers and producers) in a Grid economy is not fixed, but dynamically changing.

The commodities, i.e. resource types, that can be accounted and charged may include, but are not limited to, computing power, memory usage, storage space, network activity, and access to particular software libraries and data repositories.

Markets may be horizontally differentiated — an article that is worthless to one consumer may be priceless to another, such as Computing Elements running under a particular run time environment (in terms of operating system, installed software libraries, etc.) — and/or vertically differentiated — commodities have different “quality”, such as processor speed, physical and virtual memory, or the bandwidth of network links. Due to their natural heterogeneity, Grid resource markets can be considered both horizontally and vertically differentiated.

4The term “computational economy” refers to the utilization of computers to solve complex financial problems, as well. In this thesis, however, it is intended exclusively to refer to artificial economies.
CHAPTER 6. ECONOMIC SCHEDULING

Note that in general “quality” can be a multidimensional concept, as is also true in our case. A Computing Element, for example, usually has many different characteristics (processor performance, local disk space, physical and virtual memory, etc.) that define its quality. This can be important if considering that not only resources are heterogeneous, but also user applications and their requirements. Therefore it is difficult to benchmark computing resources in a Grid context.

6.2.1 Requirements of Computational Economies

Wolski et al. [45] define three fundamental requirements for a resource allocation mechanism to be economic in nature:5

Assumption 6.1 (Relative worth of a resource) The relative worth of a resource is not given by its price, but by its supply and the demand for it.

From a mathematical point of view, for a sufficiently large price the supply of a commodity must exceed the demand for it (provided the price is allowed to increase without bound).

Assumption 6.2 (Price and currency) The price of a given resource can be considered to be its worth with respect to the unit of a particular commodity called “currency”.

The currency (in our case Grid Credits) is thus a separate commodity that has utility for all agents (consumers and producers). As a consequence of both Assumption 6.1 and 6.2, the price as well as the worth is determined by demand and supply. Wolski et al. [45] emphasize that the mere use of a currency does not necessarily bring about an economic behavior of the system if the price is not determined by demand and supply. For the purpose of this thesis, however, we somehow loosen this restriction by allowing the price to be determined according to the workload of a resource (that in turn depends on demand and supply).

5Note that the following discusses a special case, since from an economic point of view economic models are not necessarily based on prices and currencies.
Assumption 6.3 (Equilibrium price and value) The relative worth of a resource is accurately measured only if the market is in equilibrium. Hence, the price of a resource is an accurate measure of its value only if the demand for it equals its supply (see Appendix A.2.2 for a more detailed definition of market equilibrium).

6.2.2 Market Models

An economic model is determined not only by a set of resources and a set of economic agents (consumers and producers), but also by a set of rules specifying the interaction between them and influencing the collective behavior emerging from individual actions. Economic models can be distinguished by their price-setting mechanism. Two models are of major interest for distributed computing and are studied by various researchers: auction-based models and models based on multi-commodity markets. The following is a brief introduction to both models, describing their differences and limitations.

Auctioning

In auction markets consumers place bids to purchase Grid resources. Although there are different types of auctions — English auction, Dutch auction, First-price-sealed-bid auction, Second-price Vickery auction, just to name a few without going into detail —, all of them are based on the purchase of specific resources, that is, consumers place bids for a previously specified Grid resource.

The application of an auction model usually comes with a significant messaging overhead, due to the information exchange that is necessary between the bidders and the vendor (that might be the resource itself or a designated auctioneer). Depending on the type of auction it might also lead to a considerable time delay associated with running the auction.

Another important drawback of an auction-based model is its adaptation to the necessity of co-allocation in a Grid environment — that is, the need to allocate multiple resources, eventually even of different types (e.g. computing and storage resources), for executing user applications. Requiring
applications (or a Resource Broker on their behalf) to place simultaneous bids in multiple auctions may lead to inefficiency, since it is rarely possible to succeed in all auctions. This may cause the execution to be delayed until all required resources have been purchased. The possibility of unallocated resources due to failed bidding, may lead to poor fairness among different applications. Although it is possible to hold combinatorial auctions in which resource “bundles” are traded, as for example proposed in [41] and [10], such an approach comes with additional problems, such as determining the sizes and types of resource bundles, and eventually limiting resource combinations to the single administrative domains in order not to interfere with the autonomy of local policies.

The simulation results of Wolski et al. [45] show that auctions do not produce stable pricing or market equilibrium. Scheduling decisions based on auction models will most probably share the same instability and lack of fairness. Therefore, auctioning will not be considered in this thesis.

Multi-Commodity Markets

In contrast to auctions, in a multi-commodity market, or simply commodities market, equivalent resources\(^6\) (even if owned by different suppliers) are regarded as interchangeable, and consumers do not purchase specific resources, but simply purchase a certain amount of commodities from unspecified suppliers.

Although a commodities market is often combined with posted pricing (suppliers announce their prices), most theoretical models require both consumers and producers to be price-takers, that is, a single agent does not represent a large enough market share to affect prices significantly. Instead, prices are determined by collective behavior and the actions of consumers and suppliers are mere responses to the established prices, a setting that is called “perfect competition”. In large-scale markets, such as we can expect it for an international scientific DataGrid, this assumption usually holds.

\(^6\)Note that, as already mentioned, we have a vertically differentiated market with different qualities. Thus equivalent resources are commodities of the same type and (at least roughly) the same quality.
approximately [45].

Considering that Computational Grids are supposed to couple the different computational, storage and network resources and to present them as one “unified integrated resource”, such that “applications ’plug’ into a ’power grid’ of computing resources when they execute, dynamically drawing what they need from the global supply” [44], it seems appropriate to model the Grid as a commodities market instead of using auctions where consumers place bids to purchase specified Grid resources.

6.2.3 Commodities Markets for Resource Allocation

The relationship between price, value and competitive equilibrium is of major importance for the efficiency of an economic approach for solving the maximization problem of Definition 5.1. The following definition decomposes the general resource allocation problem into a specific market configuration consisting of consumers and producers. In order to simplify the understanding, without loss of generality, the definition considers the currency to be separate from the other resource types (and thus the monetary wealth of an agent is not a component of its allocation vector).

Definition 6.1 (Market) Using the notation of the maximization problem MP of Definition 5.1 and the notation of Appendix A, in the resource market all agents \( j \in J \subseteq \mathcal{M} \) for which \( \exists n \in \mathcal{N} \) such that \( r_{jn}^l < 0 \) are modeled as profit maximizing producers:

\[
\max_{s_j} \left( \mathbf{p} \cdot \mathbf{s}_j + f_j^p(s_j) \right),
\]

where \( \mathbf{p} = [p_1, \ldots, p_N] \in \mathbb{R}^N \) is the price vector and \( \mathbf{s}_j = -\mathbf{r}_j \) is the supply vector of producer \( j \) (the allocation vector \( \mathbf{r}_j \) corresponds to the action vector

\footnote{A direct application of economic theory is difficult anyway, since our Grid environment is highly dynamic — with a rapidly changing set of agents (both users and resources) of heterogeneous requirements and characteristics — and the economic theory described in Appendix A concerns economies with static conditions, such as fixed numbers of agents and utility functions that are time independent.}

\footnote{As described in Appendix A.1.2 a negative allocation (or consumption) is considered to be a supply.}
$x_j$ in Appendix A). Thus, a producer $j$ tries to maximize the difference between its revenue $\mathbf{p} \cdot \mathbf{s}_j$ and the production cost $f_j^s(\mathbf{s}_j) = f_j(-\mathbf{s}_j) = f_j(\mathbf{r}_j)$.\footnote{Remember that for producers the components of the allocation vector $\mathbf{r}_j$ are negative, and so is the value $f_j(\mathbf{r}_j)$ of allocating $\mathbf{r}_j$ to them. Thus, $f_j(\mathbf{r}_j)$ can be considered the negative value, or cost, of supplying the resources.}

All agents $i \in I \subseteq M$ for which $\exists n \in N$ such that $x^n i < 0$ are modeled as utility maximizing consumers having a utility function defined by:

$$u_i(\mathbf{r}_i, \mathbf{m}_i) = f_i(\mathbf{r}_i) + \mathbf{m}_i,$$

thus the utility is the monetary value $f_i(\mathbf{r}_i)$ of allocating $\mathbf{r}_i$ to consumer $i$ plus the remaining funds $\mathbf{m}_i$.

Since it is possible to rewrite the maximization problem (6.1) of a producer as

$$\max_{\mathbf{r}_j} \left( f_j(\mathbf{r}_j) - \mathbf{p} \cdot \mathbf{r}_j \right),$$

both consumers and producers have an equivalent behavior, i.e. profit can be considered as the producer’s utility.

Let $m_j = -\mathbf{p} \cdot \mathbf{r}_j = \mathbf{p} \cdot \mathbf{s}_j$ be the (positive) revenue of producer $j$, then

$$\sum_{\mathbf{m} \in M} m_m = M \in \mathbb{R} \quad \forall \mathbf{m} \in \mathbb{R}^M, \mathbf{m} = [m_1, \ldots, m_M],$$

that is, the total funds in the market remain constant.\footnote{From an economic point of view, this is a restrictive assumption, but in an economy of grid resources the total available funds should of course be constant.} The total resources are given by

$$\sum_{\mathbf{m} \in M} \mathbf{r}_m = \mathbf{R} \quad \forall (\mathbf{r}_m)_{\mathbf{m} \in M} \in \mathbb{R}^{N \times M}, (\mathbf{r}_m)_{\mathbf{m} \in M} = [\mathbf{r}_1, \ldots, \mathbf{r}_M]$$

Ygge et al. stated in [46, 47] that any Pareto-optimal allocation (see Definition A.9) in the described market is a solution to the resource allocation problem $MP$ of Definition 5.1. Since according to the First Theorem of Welfare Economics (Theorem A.1), a competitive equilibrium is Pareto-optimal, they conclude that it is a solution to $MP$ as well, demonstrating the applicability of an economic approach to resource allocation problems.
The proof given by Ygge et al., however, is incorrect,\footnote{Their proof is based on the assumption that for each allocation that is not a solution to $MP$, there exists a Pareto-improvement, an assumption that however does not hold.} as is demonstrated in Appendix B with a simple example.

A competitive equilibrium — provided it exists — can still be considered a solution to $MP$, since, as defined in Def. A.8, it is an allocation $(r_m^*)_{m \in M}$ (and a price vector $p^*$) such that demand equals supply and all agents maximize their utility (subjects to their budget constraints). Hence, it maximizes the total utility by definition.

6.2.4 Utility Function for Resource Ranking

Mathematical utility functions, defined in Def. A.1, are commonly used to model the level of satisfaction a decision-maker receives as a result of its actions or choices. As in actual economics, the decision-making in a Grid environment is characterized by a strong heterogeneity due to widely varying preferences of users and heterogeneous requirements of applications. Some application, such as interactive applications, for example, may be required to execute immediately, while for less urgent tasks a user may as well want to execute them as cheaply as possible.

Defining a utility function requires the identification of the preference relations that are specific to the problem. In the case of a Computational Grid, a utility function should reflect the desired tradeoff between the parameters involved in the resource selection, such as price, data access time or the resource performance characteristics (resource “quality”).

The following proposed Resource Broker utility function\cite{30} can be customized by the user by specifying a few parameters:

$$U_i = \alpha_T \cdot \frac{T^* - T_i}{T^*} + \alpha_P \cdot \frac{P^* - P_i}{P^*} + \alpha_Q \cdot \frac{Q_i}{Q^*}$$

\hspace{1cm} (with $Q_i \geq Q_{\text{min}}$ and $P_i \leq P_{\text{max}}$)

Where $\alpha_T$, $\alpha_P$ and $\alpha_Q$ are user-defined utility weights; $P_{\text{max}}$ and $Q_{\text{min}}$ are user-defined maximum prices and minimum qualities\footnote{Where instead of one quality parameter, most probably several will have to be considered, e.g. CPU performance, queue wait time, etc.}; $T^*$, $P^*$ and $Q^*$ are
the highest data access time, price and quality found in the set of matching resources, while $T_i$, $P_i$ and $Q_i$ are time, price and quality of the $i$-th resource.

The utility weights should have adequate default values, but may also be set according to the requirements of the job. A data-intensive High Energy Physics analysis, for example, will focus on low data access times, while a computing-intensive Monte Carlo simulation will probably focus on high resource performance (in this case CPU performance).

Note that it is theoretically possible to specify such a utility function upon job submission by using the JDL rank attribute (see Section 3.3.4). However, the current implementation of the Resource Broker supports only the specification of resource attributes that are published through the Grid Information Service in the ranking expression, while prices are published through the DGAS Price Authorities. Moreover, it may be of advantage to implement the utility function directly into the RB in order to have an appropriate default behavior for cases in which the user does not specify different ranking criteria.

The advantage of using such a utility function is that all scheduling criteria may be considered in one framework, instead of using the current approach that first classifies the matching Computing Elements according to the data location and then ranks them according to other user-defined criteria (see Section 3.3). The current approach may be even suboptimal since it considers only the number of input files, not their size, and hence may classify Computing Elements that offer significantly shorter queue wait times as less suitable, if they have less input files on “close” Storage Elements. Considering that even short queue wait times may be sufficient to fetch several small files from remote SEs, in this case the CE of lesser class might be much more suitable for the given job. Therefore, in the future the different scheduling criteria should be unified into one framework.

The proposed utility function (or any comparable scheduling algorithm), however, has to be considered problem-specific and not user-specific as the

---

13 I propose to publish the currently valid resource prices on the responsible GRIS and to query the PA only for expired prices if necessary for the job cost computation. This also eliminates a significant communication overhead for the resource selection process, since the RB has to contact the GRIS anyway.
utility function that is considered by the general equilibrium theory. In other words, a user doesn’t have a characteristic utility function but adapts his utility function according to the problem to solve (application). As described in the following Section, this is not the only problem when trying to apply general equilibrium theory to the scheduling problem in Computational Grids.

6.3 Market Equilibrium in Computational Grids

The application of general equilibrium theory to the actual resource allocation in Computational Grids (exchange of computational energy) is somehow problematic, although it may also be beneficial since at market equilibrium — the computational energy demanded by the users equals the computational energy supplied by the Grid resources — a high throughput and service quality may be maintained (with some restrictions as explained later). It may also help to avoid undesirable effects such as virtual inflation or virtual finance speculation.

The particular characteristics of the traded commodities simplify the producers’ profit functions, since the usually negligible maintenance and production cost\(^\text{14}\) imply that profit maximization is equivalent to revenue maximization. The nonstorability of the traded services, however, might in the worst case lead to continuously decreasing prices\(^\text{15}\), since producers cannot benefit from storing their resources until prices are higher. The desired resource exchange fairness, however, requires prices to be at least approximately stable at market equilibrium, such that rendered services may be recovered later without significant gain or loss.\(^\text{16}\)

The proof of existence of a competitive equilibrium — a market equilibrium\(^\text{17}\) that also maximizes all agents’ utilities — requires the fulfillment
of very restrictive conditions (see Appendix A.2.3), that usually are not met by real markets. The demands resulting from competitive behavior are discontinuous — especially if the economy contains discrete commodities (including the currency) —, and so is the excess demand. Hence the existence of an equilibrium in a Grid economy cannot be guaranteed (which however does not imply that its existence has to be excluded).

In case of an existence of a competitive equilibrium, it most probably will not be unique, since the commodities in a Grid economy are usually highly complementary [44], such as computational power and storage space, that are both required for most scientific applications (see Definition A.5 for gross complements and Appendix A.2.4 for the uniqueness of competitive equilibria). The general equilibrium theory offers no rational basis for choice among different (often isolated) equilibria [44].

Furthermore, reaching a market equilibrium — if considering current user request per time unit and current resource supply per time unit — without taking into account the workload already present on the Grid might even lead to a suboptimal state if the resources are currently overloaded. In such a case a market equilibrium would keep the overloaded state instead of reducing it by reducing user demand. Hence, in contrast to much related work, we consider the workload (in DGAS-Sim represented by the queue wait times) as a basis for resource pricing.

Most of the mentioned drawbacks, however, are valid for actual markets as well. Nonetheless these markets effectively balance demand and supply and distribute resources in a more or less fair manner.\footnote{Although this may also be a point of ideological dispute.} Therefore it is still worth to investigate an application to Grid resource markets.

A major concern when trying to optimize the efficiency of a virtual economy, is to define a price-setting mechanism that can bring about market equilibrium at a reasonable level of overall workload (as a long term optimization) and helps to distribute the incoming workload in order to improve the system’s throughput (as a short term optimization), without relying too much on restrictive theory, since it “does not deal with an economic mech-
anism that makes most actual economies function” [28]. The following Chapter describes only a few possible pricing strategies.

\[10\text{Nakai and Van Der Wijngaart [28] remark that what makes an actual economy function is the force that moves the economy toward an equilibrium, not the state of equilibrium itself.}\]
Chapter 7

Pricing Strategies for Commodities Markets

Economic theory can be divided in two basic categories: *macroeconomics* examines the aggregate behavior of groups of agents, while *microeconomics*, or *price theory*, examines the behavior of individual agents and their interaction with the market.

This Chapter is concerned with a microeconomic view of the problem of economic resource allocation and describes some possible pricing strategies for the commodities in a Grid economy, i.e. computing power, memory usage, storage space, network activity, access to particular software libraries and data repositories, etc. Prices are expressed in a virtual currency, called *Grid Credits*, per *Unit of Computational Energy* (UCE), as described in Section 4.1.1.

The major goals when trying to optimize the efficiency of a virtual economy via resource pricing, are to bring about (or at least to approximate) market equilibrium — if not competitive equilibrium — at a reasonable level of overall workload (as a long term optimization) and to help to distribute the incoming workload in order to improve the system’s throughput (as a short term optimization), while avoiding negative effects such as price wars, inflation or speculation.

It has to be pointed out that, in contrast to most existing economies, in a virtual environment the cost for changing prices is virtually zero. It thus is
not only technically possible, but also economically feasible, to adjust prices more frequently.\textsuperscript{1}

Since user requirements are heterogeneous and the resource ranking strategy used by the Resource Broker is highly customizable (see Sections 6.2.4 and 3.3.4) in order to be flexible enough to support arbitrary application requirements, only pricing schemes that do not model consumer behavior are being considered.

The described pricing strategies focus on per-resource pricing, where resource prices depend only on the current state of the resource, in contrast to per-job pricing where prices may also vary according to the job for which a price is determined (although they might also be adapted to support per-job pricing). This is somehow suboptimal, since due to the local scheduling policies of certain Computing Elements some jobs might start execution immediately although the queues of these CEs may still have pending jobs [25]. But for the purpose of evaluation of pricing schemes we can simplify and use per-resource pricing. Furthermore, the goal of a Grid economy is not necessarily to determine the best Grid resource for execution of every single job (High Performance Computing), but to improve the distribution of the overall workload in order to maximize the number of applications that can simultaneously achieve a certain Quality-of-Service (QoS) objective (High Throughput Computing), while limiting the complexity and the computational overhead of the scheduling process.

\section{Global vs. Local Resource Pricing}

Pricing models can roughly be divided in two categories: global pricing schemes, that focus on determining global equilibrium prices for the single commodities, and local pricing schemes that determine resource prices "locally", i.e. independently for the single resource suppliers.

\textsuperscript{1}This may be one of the reasons for the success of e-commerce. Price adjustments can be announced without significant loss of time and without requiring to pay for advertisement in other media.
7.1.1 Global Pricing

Much related work on economic models for Computational Grids is mainly concerned with reaching a market equilibrium (user demand equals resource supply) by computing or approximating global prices that balance supply and demand over the entire Grid. Such an approach may satisfy performance criteria in terms of price stability and efficiency; it however has some important drawbacks.

A possible approach is the tâtonnement process that searches over a set of prices (price vector) until demand meets supply (see Appendix A.3 for a description of the algorithm). Another algorithm proposed by Smale [37] and adapted by Wolski et al. [45] computes equilibrium prices by means of the partial derivatives of the excess demand (as a function of price, see Appendix A.2.1).

In general, global approaches require a vast knowledge of the entire market, that is costly to obtain. Wolski et al. admit that in a Grid economy, individual users are likely unable to state their own demand functions reliably [45], thus the determination of the global demand (and hence the excess demand) is very difficult, if not impossible, especially if considering the hierarchy of Resource Brokers that process requests on a First-Come-First-Served (FCFS) basis. The determination of the partial derivatives of the excess demand is even more difficult, since it requires a polling for consumer reactions to possible price adjustments or a model-based estimation [45].

Hence the computation of global equilibrium prices usually comes with a high computation overhead and eventually also with a high communication overhead. An approximation still needs to model user behavior that, however, is highly dynamic and thus implies a frequent adjustment of model parameters. It most probably requires a constant monitoring of the consumer population.

Furthermore, a global approach requires centralized control mechanisms, causing problems of scalability, security and reliability (due to a single point of failure).

The major drawback, however, is the fact that the computation or estimation of global equilibrium prices — paid by all users and excepted by
all resources — does help in balancing demand and supply, but it does not help in balancing the incoming workload in an effective manner.

7.1.2 Local Pricing

A more promising approach adjusts prices locally, i.e. independently for the single resources,\(^2\) possibly leading the system towards a global optimum through emergent marketplace behavior. This allows to base prices on factors like the current workload or the request arrival rate of the single resources. The resulting price differences can be used to aid the resource selection process, eventually leading to balanced workloads and consequently balanced prices.\(^3\)

The distributed responsibility for setting resource prices eliminates the need for a central controlling entity and thus improves the scalability and the reliability (fault-tolerance) of the Grid economy.

Moreover, if Price Authorities are associated to Virtual Organizations, as proposed, a local pricing is more compatible with the principle of local policy autonomy than an imposed global pricing model, since in theory each VO might apply a different pricing strategy.

Finally, a local approach offers a much reduced computation and implementation complexity. We therefore prefer local pricing strategies over global ones.

7.2 Local Pricing Algorithms

The following Sections briefly describe two possible algorithms, that do not require detailed information about the market situation, for local resource pricing and discuss their applicability for our scope.

\(^2\)The term "local" pricing is justified, although in practice the prices are set by the DGAS Price Authorities.

\(^3\)Which however does not mean that prices cannot be differentiated according to different resource “qualities”.

7.2.1 Derivative Follower

The game-theoretic equilibrium prices may be approximated by the derivative follower (DF) pricing algorithm \cite{24, 34, 6}. A DF iteratively adjusts prices by incrementally increasing or decreasing them until the observed profitability level falls, then the direction of price adjustment is reversed, thus seeking a local maximum of profitability. The improved Adaptive Step-size Derivative Follower (i-ADF), proposed by van Bragt et al \cite{6}, dynamically alters the step-size of price increments to allow the prices to converge to their asymptotic values, instead of oscillating with a fixed step-size.

In their simulations, Sairamesh and Kephart \cite{34} compared the DF, among others, to two pricing strategies that in general require perfect knowledge about the entire market and thus are not applicable in a heterogeneous large-scale Grid economy: a myoptimal (or “myopically optimal”) strategy\footnote{A myoptimal seller tries to maximize its profit by computing its price based on perfect knowledge of consumer behavior and assuming that other sellers maintain their current prices.}, and a game-theoretic strategy.\footnote{Their simulations, however, have been executed with a very small population of only five sellers.} In a population of quality-sensitive consumers — consumers prefer the good of higher quality as long as its price does not exceed a certain price-ceiling — the prices of all strategies converged to the game-theoretic equilibrium prices. In a population of price-sensitive consumers — consumers prefer the cheaper good as long as it fulfills a minimum quality requirement —, however, the outcomes were considerably different. Most strategies (especially the myoptimal pricing) led to cyclical price wars with large amplitudes. Only the derivative-following strategy was able to converge to the game-theoretic prices without showing any price-war behavior.\footnote{Note that a derivative-follower searches a local profit maximum and thus its price does not necessarily converge to the game-theoretic optimum if its profit function has multiple maxima.} In a Grid economy both quality-sensitive users and price-sensitive users will be present. Sairamesh and Kephart showed that a population of derivative-followers can be out-performed by a single myoptimal seller that grasps the major part of the market share. However, in a heterogeneous large-scale Grid economy a myoptimal pricing agent is
CHAPTER 7. PRICING STRATEGIES

not reasonably implementable, since it requires to predict the behavior of the entire user population.

Despite these positive results, a pricing strategy based on derivative-followers has an important drawback that has to be addressed for an efficient application to a Grid economy and its particular characteristics: The DF algorithm considers only the immediate profit\(^7\), when deciding whether to lower or raise the price. Such an approach does help to balance the incoming workload, but it is not capable of balancing a pre-existing uneven workload distribution, that can occur due to network failures ("splits") or if new resources enter the Grid. Since our aim is not profit maximization but load balancing, in our future work we intend to investigate the use of the queue wait time and its variation as a stimulus for price adjustments.

7.2.2 Hybrid Pricing Model

In the *Hybrid Pricing Model* (HPM), already presented in [30], Price Authorities *dynamically* adjust prices within static limits to balance the workload on the basis of the queue wait times (QWT). The computation of a resource’s price requires only local information about the state of the given resource (state-based load balancing). Prices are expressed in Grid Credits/UCE and determined as follows:\(^8\)

\[
price = P_0 + \Delta P \frac{W - \frac{1}{2} W_{\text{max}}}{\frac{1}{2} W_{\text{max}}} \quad 0 \leq W \leq W_{\text{max}}
\]

Where \(W\) is the current QWT and \(W_{\text{max}}\) the queue’s characteristic maximum QWT. \(P_0\) is a fixed base price for the specific queue and \(\Delta P\) its fixed variation limit.

In contrast to much related work, the HPM explicitly addresses the current workload of the single computing resources, so that price differences may reflect the imbalances of the system’s overall workload.

The main advantage of this pricing strategy, in addition to its simplicity, is that it guarantees relative stable prices that do not degenerate in the long

\(^7\)More precisely, it compares the profit earned within the last two time intervals of prefixed length.

\(^8\)The pricing function does not necessarily need to be linear, but the first simulation results presented in Chapter 9 show that even such a simple approach can be effective.
CHAPTER 7. PRICING STRATEGIES

term. The relative price stability depends on the ratio of $\Delta P$ to $P_0$, since prices vary between $P_0 - \Delta P$ and $P_0 + \Delta P$.

Furthermore, the algorithm doesn’t adjust prices stepwise, such as DF or tâtonnement, and thus reaction times to the dynamic workload are limited to a prefixed price adjustment interval.

Reaching market equilibrium (balancing demand and supply), however, will require the ratio of $\Delta P$ to $P_0$ to be large enough to allow flexible prices that can guide user behavior by incentivizing to delay submissions to a congested system or to submit additional jobs to an idle system.

The HPM will work best in case of agreement on common base prices and variation limits and a limited set of queue classes (e.g. short, medium, and long).

7.3 Performance Criteria

The distributed nature of the workload management and the unprecedented complexity of the resource allocation and load balancing problem complicate the definition of appropriate performance metrics. A system-wide performance evaluation requires an amount of information that usually is costly, if not impossible, to obtain.

Since prices reflect the true worth of the resources only at market equilibrium, a comparison of global demand and global supply are desirable. The determination of the global demand, however, is extremely difficult, due to the decentralized hierarchy of Resource Brokers and due to the scheduling of applications as and when they are submitted (FCFS). It is furthermore complicated by the fact that the resource requirements of applications can only be estimated.⁹

Other economic performance criteria, such as total utility maximization (utilitarian criterion), equating the level of utility (equality criterion), or reaching a Pareto-optimal allocation (see Definition A.9), have above all theoretical value, since in practice users don’t have mathematical utility

⁹Estimations can be provided by the users' job descriptions or eventually by predictions based on historical job information, as for example described by Smith et al. [38] and Li et al. [25].
functions.\textsuperscript{10}

From a computer engineering point of view, performance metrics can be divided into two categories: system metrics and user metrics.

A user usually attempts to optimize its individual QoS requirements, i.e. the performance of his applications (High Performance Computing) — and is not concerned with system-wide performance. A possible metric might be the average response time or turnaround time (time from job submission to job completion) for predefined job categories.

Due to the heterogeneous job requirements, a system-wide average response time has less significance for users, and can be considered a system metric. Other system metrics might be the average queue wait time (eventually divided in predefined queue types, such as short, medium and long queues) and its standard deviation. The latter may be significant above all for evaluating the efficiency of pricing strategies for the important goal of load balancing (the lower the standard deviation, the better balanced is the workload). Moreover, the resource utilization (average percentage of time each resource is occupied) and the system utilization — the ratio of the computational energy utilized and available computational resources — are often considered as system metrics, to determine how efficiently the system resources are being used.

Since EDG is focused on maximizing the system-wide performance (High Throughput Computing) and this thesis is concerned above all with load balancing, the simulations presented in the following Chapters consider the queue wait time (QWT) — that is, the time a job has to wait in a resource’s queue before starting execution — and its standard deviation as most important performance criteria. Furthermore, the fraction of requests that cannot be served due to a lack of matching resources (request denial rate) may indicate the efficiency of system usage.

\textsuperscript{10}Even if a utility function is implemented on the RB-level, the willingness of users to submit jobs cannot be easily expressed in mathematical functions.
Part III

Simulation of Economy-based Load Balancing
Simulation is an important aid in designing an effective scheduling framework for distributed computing, since it allows to evaluate potential scheduling strategies by controlled and repeatable experimentation without requiring to deploy them on a large basis.

This thesis presents *DGAS-Sim(ulator)*, a simulation tool for the evaluation of different pricing strategies and utility (ranking) functions, and their effect on economy-based load balancing. It furthermore presents and analyzes the first simulation results concerning the Hybrid Pricing Model (HPM), described in Section 7.2.2.

Since the EDG Resource Broker does already consider data location in it’s matchmaking phase — before price information might be included into the resource selection process — and popular datasets are dynamically replicated, the executed simulations focus on job scheduling without considering data location or network topology (see Section 5.3 for a discussion of the importance of data movement for the load balancing problem in a data-intensive computing environment).

The presented simulation tool, although based on the architecture of the EDG Workload Management System, and the results regarding the HPM may have importance for similar Grid environments as well.
Chapter 8

DGAS-Sim(ulator)

DGAS-Sim(ulator) is a modular and expandable simulation tool of the Workload Management System architecture of the European DataGrid project, written in Java. It simulates mainly the components involved in the job scheduling process, including parts of the DataGrid Accounting System. Its purpose is to study the impact of different resource pricing schemes and resource brokering strategies on workload balancing and thus on the overall throughput of the Grid. DGAS-Sim simulates static\(^1\) state-based\(^2\) load-balancing.

To simplify the analysis, at present only processing power is being priced.

8.1 Architecture of DGAS-Sim

Figure 8.1 is a simplified class diagram of DGAS-Sim, showing mainly classes, data members and methods that are relevant for this thesis.\(^3\)

The core class used for the presented simulation (AutoJobSubmitSimCore, not shown in Fig. 8.1) is implemented as a thread and periodically sends random jobs to a single instance of the Resource Broker (RB) class by invoking the submitJob() method. The different job types (see Section 9.2) and the time interval between job submissions may be defined in the DGAS-

---

\(^1\)Tasks do not migrate to another host once they have been submitted for execution.

\(^2\)The workload is balanced according to the actual system state, in contrast to model-based load balancing.

\(^3\)For a legend of the adopted UML notation, see Appendix C.
ComputingElement (CE)
- cpuPower: int
- maxQWT: int
- currQWT: int
- qTypeIdString: String
- pa: PriceAuthority
- profit: int
+ computeJobsInQueue(): void
+ queueJob(job: Job): boolean

PriceAuthority (PA)
- paID: int
- pmod: PricingModule
- ces: Map (<int, CE>)
- prices: Map (<int, int>)
+ getPrice(ceId: int): int
+ registerCE(ce: CE): void
+ registerCE(ce: CE, price: int): void

<<interface>>
PricingModule
+ computePrice(ce: CE): int

HybridPricingModule
- basePrice: int
- deltaP: int
+ computePrice(ce: CE): int

GIS (Grid Information System)
- ces: Map (<int, CE>)
- pas: Map (<int, PA>)
- jobTypes: Map (<int, Job>)
- maxQWTs: Map (<String, int>)
+ getCEList(): List
+ getCE(ceId: int): CE
+ registerCE(ce: CE): void
+ getPA(ceId: int): PA
+ getCEsPA(ceId: int): PA
+ getPrice(ceId: int): int
+ registerPA(pa: PriceAuthority): void
+ getMaxQWT(qTypeStr: String): int

Job
- edgJobId: int
- compEnergy: int
- minCPUPower: int
- requiredQTypes: List

<<interface>>
UtilityModule
+ selectCE(ces: List, job: Job): CE

PriceSensitiveUtilityModule
+ selectCE(ces: List, job: Job): CE

ResourceBroker
- gis: GIS
- utmod: UtilityModule
- matchmaker: Matchmaker
- submit(job: Job): boolean
+ submit(job: Job): boolean

<<interface>>
Matchmaker
+ getMatchingCEList(job: Job): List

CEAttribMatchmaker
- gis: GIS
+ getMatchingCEList(job: Job): List

<<implements>>
LeastLoadedUtilityModule
- RELATIVE_LOAD: int = 0
- ABSOLUTE_LOAD: int = 1
- loadType: int
- accuracy: double
+ selectCE(ces: List, job: Job): CE

CompletionTimeUtilityModule
+ selectCE(ces: List, job: Job): CE

<<interface>>
RandomUtilityModule
+ selectCE(ces: List, job: Job): CE

RandPriceSensitiveUtilityModule
+ selectCE(ces: List, job: Job): CE

<<implements>>
RandomUtilityModule
+ selectCE(ces: List, job: Job): CE

<<implements>>
PriceSensitiveUtilityModule
+ selectCE(ces: List, job: Job): CE

Figure 8.1: Simplified DGAS-Sim class diagram.

Sim configuration file. The EDG architecture does not have a single central RB, but a distributed hierarchy of RBs in order to improve scalability. For simulation, however, a single RB is sufficient.

8.1.1 Resource Broker, Matchmaker and Utility Modules

The Resource Broker class queries the GIS (Grid Information System) to obtain a list of all registered Computing Elements (CEs) and uses the CEAttribMatchmaker class, that implements the Matchmaker interface⁴, to determine a subset of CEs that match the job requirements. It finally calls

⁴Defining a Matchmaker interface theoretically allows to define other criteria for matchmaking, additionally to the job requirements.
the Utility Module's method `selectCE()` to choose a specific CE for job submission. Of the different Utility Modules that have been implemented, the following are relevant for the simulations presented in this thesis:

- The **Price-Sensitive Utility Module** simply selects the cheapest resource. If multiple resources have the same cheap price, the resource for job submission will be randomly chosen among them.

- The **Random Utility Module** randomly selects a resource among all matching resources.

- The **Completion Time Utility Module** selects the resource that offers the shortest completion time (queue wait time plus execution time) for the given job.

The Random Utility Module and the Completion Time Utility Module do not use price information for the resource selection process. They are rather used to simulate “worst case” and “best case” scenarios for comparison reasons.

### 8.1.2 Computing Elements and Job Queues

The *Computing Element* class represents a single-processor grid resource with an FCFS (first-come-first-serve) queue. The current queue length corresponds to a queue wait time (QWT) and is measured in time units. Maximum QWTs, initial QWTs and CPU performance of the single resources can be set in the configuration file, as well as the duration of a time unit. The computing power furnished by a CE is measured in *Units of Computational Energy (UCE)* per time unit, where we define computational energy as the product of a performance factor or power $p$ and resource usage $u$ (e.g. product of a benchmark for CPU performance and CPU time) in analogy to physics (see Section 4.1.1). The computational energy required by a job should ideally be independent of the resource that is executing it.\(^6\) In

---

\(^5\)For simplicity we assume that time is discrete.

\(^6\)The resource usage expectations of an EDG job might be provided by the user’s job description or eventually by predictions based on historical information, as for example described in [38] and [25]. For simplicity, however, DGAS-Sim assumes the exact consumption of computational energy to be known.
CHAPTER 8. DGAS-SIM(ULATOR) 66

EDG, CEs are often clusters and may have several queues based on more sophisticated scheduling policies than FCFS. Such a complexity however is not required for a general understanding of the efficiency of different pricing schemes.

8.1.3 Price Authorities and Pricing Modules

Each CE is registered with a specific Price Authority (PA), that is responsible for pricing one or more associated CEs. Several PA instances may be defined. DGAS foresees one or more PAs for each Virtual Organization (VO) that is part of the EDG community, such that the participating VOs retain control of the pricing of their resources. Each simulated PA instance is a thread that periodically computes new prices for its registered CEs. The time interval between price adjustments is set in the configuration file.

Each PA uses a specific Pricing Module. The following simulations consider only a module, that implements the Hybrid Pricing Model, presented in [30] and in Section 7.2.2. Each instance of the pricing module class, however, may have a different configuration (see Section 9.2). In the DGAS implementation (that contrarily to DGAS-Sim is written in C/C++) the different pricing modules can be implemented as dynamically linked libraries, such that it is possible for VOs to change their pricing strategy without the need to restart the PA.
Chapter 9

Simulation of Price-Sensitive Brokering with the Hybrid Pricing Model

In the Hybrid Pricing Model, described in Section 7.2.2, Price Authorities (PAs) dynamically adjust prices within static limits to balance the workload on the basis of the queue wait times (QWT). The computation of a resource’s price requires only local information about the state of the given resource (state-based load balancing); see Section 7.1 for a discussion on global and local pricing strategies.

In the following some preliminary simulation results are given, in order to gain a general understanding of the impact of the pricing strategy on workload balancing. Prices are expressed in Grid Credits/UCE. The workload of the single-queue, single-processor Computing Elements (CEs), is represented by their current QWTs.

9.1 Preliminary Results

The HPM may be applied with a non-linear pricing function, but as shown in Fig. 9.1 even a simple linear approach is able to balance the QWTs of initially unbalanced queues, at least in a very simplified Grid setting with five CEs of equal characteristics and a common base price $P_0$. The initial
Figure 9.1: QWTs and resource prices at market equilibrium with equal base prices and equal CPU performance.
QWTs of the single CEs were arbitrarily chosen and jobs required nearly the same amount of computational energy that was furnished by the CEs, thus simulating market equilibrium\(^1\) where demand equals offer (zero excess demand). Since price adjustments are proportional to the variations in queue length — as can be deduced from the formula given in Section 7.2.2 —, the resource prices show the same behavior as the corresponding QWTs, as can be seen in Fig. 9.1. Therefore only results concerning QWTs will be presented in this thesis.

Figure 9.2 refers to a similar setting, with exception of CPU performances that were not equal, but chosen between 10 and 40 UCE per time unit (furnishing the same total computational energy as before). The lower performant CE0 and CE1 show fluctuations in their QWTs that can be explained due to the fact, that jobs take longer to execute and price adjustments are done only every 12 seconds. Thus, whenever the low performant CEs are cheapest they receive the next job submissions and their QWTs grow more than those of high performant CEs would, until new prices are

---

\(^1\)Not to mistake for competitive equilibrium, that also maximizes all agents utility, see Appendix A.2.
Figure 9.3: QWTs in the over-demand case with equal base prices and equal CPU performance.
QWT of HPM, under-demand case; equal base price; equal CPU power

submitted jobs required 96.5 UCE/t.u. (mean value)
maximum QWT is 2000 time units (1000 s)
broker utility function: price sensitive
pricing module: hybrid
base price: 1000, variation limit: 50

QWT of HPM, under-demand case; equal base price; equal CPU power

submitted jobs required 72 UCE/t.u. (mean value)
maximum QWT is 2000 time units (1000 s)
broker utility function: price sensitive
pricing module: hybrid
base price: 1000, variation limit: 50

Figure 9.4: QWTs in the under-demand case with equal base prices and equal CPU performance.
Figure 9.5: QWTs at market equilibrium with different base prices and equal CPU performance.

computed. Apart from these fluctuations the CEs QWTs remain balanced.

The behavior in case of over-demand (positive excess demand) or under-demand (negative excess demand) are demonstrated in Fig. 9.3 and 9.4. QWTs become balanced and then increase or decrease together until the maximum QWTs are reached or queues are emptied.

These encouraging results, however, can be applied only to cases in which all CEs have the same queue type and a common base price $P_0$ and variation limits $\Delta P$. Figure 9.5 shows a slightly more complicated setting with three different queue types (“short”, “medium” and “long”), different base prices for each queue type and three CEs of each queue type. The submitted jobs are randomly chosen from three job types that differ only in the queue type that they can be submitted to. Although the market is nearly in equilibrium — the total amount of computational energy required each time unit is only 0.4% above the furnished amount —, CEs having short and long queues are less requested (weak under-demand) than medium queues (weak over-demand).\textsuperscript{2} This may lead, in a worst case scenario where there is a

\textsuperscript{2}Note: QWTs of long queues become balanced at a higher level due the fact that the initial QWTs are higher. Short queues have the lowest initial QWTs and thus become
strong under- or over-demand for certain CE or queue types, to partially idle resources while others are overloaded.

Since queues still become balanced within the three different types, Grids incorporate many different resource types — that cannot easily be classified — and submitted jobs have very different requirements, it is necessary to examine the efficiency of the Hybrid Pricing Module in more realistic Grid settings. The following Sections present and discuss results of more realistic simulations.

9.2 Simulation Framework

The results presented in Section 9.4.1 were obtained by simulating a small grid environment with a total of 50 CEs. Each CE has one of the following 5 queue types. Maximum Queue Wait Times (QWT) are expressed in time units (1 time unit = 1 minute):

- **very short queues**
  maximum QWT of 120 time units, corresponding to 2 hrs.

- **short queues**
  maximum QWT of 240 time units, corresponding to 4 hrs.

- **medium queues**
  maximum QWT of 600 time units, corresponding to 10 hrs.

- **long queues**
  maximum QWT of 1080 time units, corresponding to 18 hrs.

- **very long queues**
  maximum QWT of 1440 time units, corresponding to 24 hrs.

balanced at a lower level.
CHAPTER 9. SIMULATION OF THE HYBRID PRICING MODEL 74

The simulation was done with 10 CEs for each queue type, that furnish the following computing power per time unit:

- two CEs furnishing 800 UCE/t.u. (simulating an 800MHz CPU).
- two CEs furnishing 1200 UCE/t.u. (simulating a 1.2GHz CPU).
- two CEs furnishing 1600 UCE/t.u. (simulating a 1.6GHz CPU).
- two CEs furnishing 2000 UCE/t.u. (simulating a 2.0GHz CPU).
- two CEs furnishing 2400 UCE/t.u. (simulating a 2.4GHz CPU).

For simulation purpose we can assume the CPU clock speed to be a sufficiently good measure for CPU performance and thus define the UCE as 1 MHz × 1 time unit = 1 MHz×min. Hence a total computational energy of 80000 UCE per time unit is furnished by the 50 CEs.

The integer-valued prices of the single CEs are set by a total of 15 Price Authorities (PAs). For each queue type there are three corresponding PA types: “cheap” for CEs with low CPU performance, “medium” for CEs with medium CPU performance, and “expensive” for CEs with high CPU performance. Prices are adjusted — according to the HPM — only every 10 time units (10 minutes) and only if the price difference is superior or equal to a threshold of 2 Grid Credits, otherwise no price adjustment is done. Such a price adjustment threshold can help to limit the number of records in the PAs’ databases3 in times when price variations are insignificant. Since in the HPM price adjustments are proportional to the variations in queue length, the QWTs shown in Figures 9.6 to 9.10 are updated only when prices are adjusted.

The job submission requests are satisfied by a single Resource Broker, that uses a price-sensitive brokering strategy by simply submitting jobs to the cheapest CE that matches the job requirements. Data location and network traffic are not considered.

Every 30 seconds (half of a time unit) one of the twenty job types listed in Table 9.1 is randomly chosen for submission to the Resource Broker. The table shows the requested computational energy in UCE4, the minimum re-

3The PA needs to keep track of the price history [23].
4a 1000 UCE job takes about 1 time unit (1 minute) on a 1.0 GHz CPU.
requirements of CPU performance\textsuperscript{5}, and the queue types to which the jobs can be submitted\textsuperscript{6}. Restricting job submissions to specific queue classes and requiring a minimum CPU performance simulates a minimum of matchmaking by the Resource Broker\textsuperscript{7}. Since all job types are equiprobable, some of them are similar in order to increase their probability of being selected.

<table>
<thead>
<tr>
<th>type</th>
<th>UCE\textsuperscript{4}</th>
<th>minCPU\textsuperscript{5}</th>
<th>queue types\textsuperscript{6}</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2840-4260</td>
<td>none</td>
<td>very short, short</td>
</tr>
<tr>
<td>1</td>
<td>2840-4260</td>
<td>1.6GHz</td>
<td>very short, short</td>
</tr>
<tr>
<td>2</td>
<td>9230-12070</td>
<td>none</td>
<td>very short, short</td>
</tr>
<tr>
<td>3</td>
<td>9230-12070</td>
<td>none</td>
<td>short, medium</td>
</tr>
<tr>
<td>4</td>
<td>9230-12070</td>
<td>1.2GHz</td>
<td>short, medium</td>
</tr>
<tr>
<td>5</td>
<td>9230-12070</td>
<td>2.0GHz</td>
<td>short, medium</td>
</tr>
<tr>
<td>6</td>
<td>9230-12070</td>
<td>none</td>
<td>short, med., long</td>
</tr>
<tr>
<td>7</td>
<td>17750-24850</td>
<td>none</td>
<td>short, medium</td>
</tr>
<tr>
<td>8</td>
<td>17750-24850</td>
<td>none</td>
<td>medium, long</td>
</tr>
<tr>
<td>9</td>
<td>17750-24850</td>
<td>none</td>
<td>medium, long</td>
</tr>
<tr>
<td>10</td>
<td>17750-24850</td>
<td>1.2GHz</td>
<td>medium, long</td>
</tr>
<tr>
<td>11</td>
<td>17750-24850</td>
<td>2.0GHz</td>
<td>medium, long</td>
</tr>
<tr>
<td>12</td>
<td>39050-46150</td>
<td>none</td>
<td>medium, long</td>
</tr>
<tr>
<td>13</td>
<td>39050-46150</td>
<td>1.6GHz</td>
<td>medium, long</td>
</tr>
<tr>
<td>14</td>
<td>39050-46150</td>
<td>none</td>
<td>long, very long</td>
</tr>
<tr>
<td>15</td>
<td>39050-46150</td>
<td>1.2GHz</td>
<td>long, very long</td>
</tr>
<tr>
<td>16</td>
<td>39050-46150</td>
<td>2.0GHz</td>
<td>long, very long</td>
</tr>
<tr>
<td>17</td>
<td>99400-113600</td>
<td>none</td>
<td>long, very long</td>
</tr>
<tr>
<td>18</td>
<td>106500-113600</td>
<td>1.6GHz</td>
<td>long, very long</td>
</tr>
<tr>
<td>19</td>
<td>198800-227200</td>
<td>none</td>
<td>long, very long</td>
</tr>
</tbody>
</table>

Table 9.1: Job types and their respective requirements.

\textsuperscript{5}1.0 GHz corresponds to a computational energy of 1000 UCE/t.u.
\textsuperscript{6}The queue types should be specified by the users.
\textsuperscript{7}The different queue classes may be interpreted as different CPU architectures, operating systems, etc.
CHAPTER 9. SIMULATION OF THE HYBRID PRICING MODEL 76

Note that there are only a few job types (0-2) that can be submitted exclusively to short or very short queues, and that most job types (6 and 8-19) can be submitted to long or very long queues. Thus we can expect an over-demand for the longer queues and an under-demand for the shorter queues.

9.3 Simulation Runs

Three simulation runs using the Hybrid Pricing Model have been executed for a total time of 18 hrs each. Each simulation was started with arbitrarily chosen initial QWTs for the single CEs in order to verify the capability of the Hybrid Pricing Model approach to balance initially unbalanced queues.

<table>
<thead>
<tr>
<th>PA</th>
<th>queue type</th>
<th>price category</th>
<th>$P_0$ [GC]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>very short</td>
<td>cheap</td>
<td>1175</td>
</tr>
<tr>
<td>1</td>
<td>very short</td>
<td>medium</td>
<td>1200</td>
</tr>
<tr>
<td>2</td>
<td>very short</td>
<td>expensive</td>
<td>1225</td>
</tr>
<tr>
<td>3</td>
<td>short</td>
<td>cheap</td>
<td>1075</td>
</tr>
<tr>
<td>4</td>
<td>short</td>
<td>medium</td>
<td>1100</td>
</tr>
<tr>
<td>5</td>
<td>short</td>
<td>expensive</td>
<td>1125</td>
</tr>
<tr>
<td>6</td>
<td>medium</td>
<td>cheap</td>
<td>975</td>
</tr>
<tr>
<td>7</td>
<td>medium</td>
<td>medium</td>
<td>1000</td>
</tr>
<tr>
<td>8</td>
<td>medium</td>
<td>expensive</td>
<td>1025</td>
</tr>
<tr>
<td>9</td>
<td>long</td>
<td>cheap</td>
<td>875</td>
</tr>
<tr>
<td>10</td>
<td>long</td>
<td>medium</td>
<td>900</td>
</tr>
<tr>
<td>11</td>
<td>long</td>
<td>expensive</td>
<td>925</td>
</tr>
<tr>
<td>12</td>
<td>very long</td>
<td>cheap</td>
<td>775</td>
</tr>
<tr>
<td>13</td>
<td>very long</td>
<td>medium</td>
<td>800</td>
</tr>
<tr>
<td>14</td>
<td>very long</td>
<td>expensive</td>
<td>825</td>
</tr>
</tbody>
</table>

Table 9.2: PA types and base prices (simulation run no.1).

Simulation run no.1 was done with assigning different base prices $P_0$ to the single PAs in order to reflect the different “quality” of the Computing Elements (in terms of processing power and maximum QWT). The variation
CHAPTER 9. SIMULATION OF THE HYBRID PRICING MODEL  77

limit $\Delta P$ was 50 Grid Credits (GC) for all PAs. Table 9.2 shows the base prices assigned to the single PAs. As mentioned in Section 9.2 the price categories “cheap”, “medium”, and “expensive” reflect the CPU performance of the CEs.

Simulation run no.2 was done with all PAs having a base price $P_0$ of 1000 Grid Credits and a price variation limit $\Delta P$ of 50 Grid Credits.

Simulation run no.3 was similar to simulation run no.1, having the same base prices as listed in Table 9.2, but the variation limit $\Delta P$ was 250 Grid Credits. Hence, the price domains of the different queue classes overlap significantly\(^8\), in contrast to simulation run no.1 with a variation limit of only 50 GC.

9.4 Analysis of Simulation Results

The following is a qualitative and quantitative analysis of the simulation results for the Hybrid Pricing Model obtained with a merely price-sensitive brokering strategy — the Resource Broker simply chooses the cheapest matching Computing Element for a given job.

9.4.1 Computational Energies and Request Denial Rates

Figures 9.6 to 9.10 show the results of all three simulation runs. In simulation run no.1 a total of 2032 job submissions were requested. Although the requested total computational energy was only 85,268,585 UCE, corresponding to 78,952.4 UCE/t.u. (about 98.7% of the 80,000 UCE furnished per time unit by all 50 CEs) not all job submission requests could be satisfied. Only 1944 jobs (95.7%) could be executed on the CEs, with a total of 70,670,302 UCE (65,435.5 UCE/t.u. = 81.8% of 80,000 UCE). The high request denial rate of 4.3% can be explained by the fact that the single jobs have requirements in terms of queue types and minimum processing power that could not be satisfied if all matching CEs had full queues.

In simulation run no.2 a total of 2034 job submissions were requested. All jobs could be submitted to Computing Elements (no request denials).

\(^8\)In other words, the possibility that queues of distinct classes have similar prices is much higher.
In simulation run no.3 only one of 2037 job requests could not be satisfied (request denial rate of 0.05%).

9.4.2 Queue Wait Times (QWTs)

Figure 9.6 shows the QWTs of the very short queues for all three simulation runs. Since there are only a few jobs that may run on very short queues (see Table 9.1), an under-demand of the computational energy furnished by the 10 CEs with very short queues could already be expected. In simulation runs no.1 and no.3 not a single job was submitted to these queues due to the fact that the Resource Broker uses a price sensitive brokering strategy and the very short queues are the most expensive ones. Thus the arbitrarily chosen initial QWTs are continuously reduced, but no new jobs are submitted. Simulation run no.2 shows some job submissions (although there is still an under-demand) due to the fact that very short queues have the same base prices as other queue types.

Figure 9.7 shows the QWTs of the short queues. It can be seen that — in spite of the expected under-demand — in simulation runs no.2 and no.3 substantially more jobs were submitted to short queues than in simulation run no.1. In the latter many jobs “preferred” medium queues instead, since they were usually cheaper. Note that in simulation run no.1 jobs were submitted only to the three cheapest CEs (CE10-12).

Figure 9.8 shows the QWTs of the medium queues. In simulation runs no.2 and no.3 the demand for computing resources with medium queues was similar to the offer (see also Fig. 9.11b and 9.11c). After an initial balancing phase the mean QWT was more or less constant. It can be seen that the CEs with lower CPU power show higher fluctuations. This can be explained by the fact that a job with a certain requirement of UCE will increase the QWT of a low performance CE more than the QWT of a high performance CE (remember that the QWT is measured in time units, jobs obviously take longer to execute on CEs with low performance). In simulation run no.1 the low performant and thus cheaper CEs (CE20-22) received most of the jobs that were submitted to medium queues. No job was submitted to the high performance CEs (CE27-29). The mean QWT first decreases
(see Fig. 9.11a), until after about 35000 seconds (about 9.7 hrs) the QWTs of the medium queues (simulation run no.1) start to increase because the long and very long queues have reached their maximum QWTs (see also Fig. 9.9a, 9.10a and 9.11a) and thus no cheaper resources are available for job submission.

Figure 9.9 shows the QWTs of the long queues. In all simulation runs there was an over-demand of computing resources with this queue type (see Table 9.1). On the figure from simulation run no.2 the mean QWT slowly increases (see also Fig. 9.11b). The same is valid for simulation run no.3 where, however, the gradient of the mean QWT significantly increases after about 55000 seconds (15.3 hrs), since several very long queues have reached their maximum QWTs (see also Fig. 9.10c) and thus cannot satisfy all job requests that require a certain minimum CPU performance and consequently are submitted to the corresponding CEs with long queues. In simulation run no.1 the maximum QWTs of long queues are reached even faster. This can be easily explained by the fact that CEs with long queues are cheap and thus receive most of the jobs that might also run on CEs with medium queues. After about 20000 seconds (about 5.6 hrs) the gradients of the QWTs increase because the very long queues have reached their maximum QWTs (Fig. 9.10a). Furthermore it can be seen that the QWT of the three cheaper CEs (CE30-32) initially increases faster than that of the more expensive CEs (CE33-39). The QWT of the most expensive CEs (CE37-39) initially increases slowly until the queues of the cheaper CEs have reached the maximum QWTs and thus cannot satisfy further job requests.

Figure 9.10 shows the QWTs of the very long queues. In simulation run no.2 and no.3 the very long queues show more or less the same behavior as the long queues (see Fig. 9.9b and 9.9c). In simulation run no.1, however, the very long queues grow even faster than the long ones and soon reach their maximum (remember that very long queues are the cheapest ones in simulation run no.1).

Figures 9.9a and 9.10a show the main reason for the high request denial rate in simulation run no.1 (see Section 9.4.1): The maximum QWTs of both long queues and very long queues are reached and not all incoming job requests may be satisfied if their requirements do not allow a submission
CHAPTER 9. SIMULATION OF THE HYBRID PRICING MODEL

... to shorter queues. Although the PAs in simulation run no.3 have the same base prices as in run no.1, the QWTs of the long and very long queues grow less fast, since the price domains widely overlap due to the relative high variation limit $\Delta P$ of 250 GC, and thus the incoming jobs get better distributed without being submitted mainly to CEs with longer queues. In simulation run no.2 the QWTs of long and very long queues grow the slowest (after all there is still an over-demand), since — having equal base prices for all PAs — these queue types are not generally preferred by the price-sensitive brokering strategy.

9.4.3 Relative Standard Deviation of QWTs

The efficacy to balance the workload, i.e. the QWTs, within the different queue classes can be represented by the relative standard deviation (RSD), that is, the ratio of the standard deviation to the mean QWT.\(^9\) Figure 9.12 shows the relative standard deviations for all three simulation runs. The initial RSD is quite high for all queue types and simulation results, given by the fact that initial QWTs are chosen arbitrarily in order to verify the capability of the HPM to effectively balance initially unbalanced workloads.

In case of differentiated base prices (simulation run no.1 and no.3) the RSD of very short queues drops to zero as soon as the queues are emptied, since no new jobs are submitted to CEs with this queue type. The short queues (and the very short queues in simulation run no.2) are mostly idle, due to the high under-demand for short queues. Therefore even the few jobs that are submitted to these queues cause very high standard deviations. Nonetheless, the RSD of short queues is significantly lower if using equal base prices for all PAs (simulation run no.2).

Since the under-demand case worsens RSDs significantly\(^10\) and is thus less appropriate for comparing the general efficiency of different configurations for load balancing purpose, Fig. 9.13 shows only relative standard

---

\(^9\)The relative standard deviation is being used instead of the standard deviation in order to be able to compare the behavior for the different queue types.

\(^10\)For the case in which all queues are empty (e.g. very short queues) the RSD is set to zero, although mathematically the RSD should be 1 (zero standard deviation divided by zero mean QWT).
deviations of the medium, long and very long queues on a larger scale.

The RSD of medium queues is highest for simulation run no.1 (differentiated base prices \( P_0 \) and low price variation limit \( \Delta P \)), which can simply be explained by the fact that the low performant and thus cheaper CEs (CE20-22) received most of the jobs that were submitted to medium queues, and no job was submitted to the high performance CEs (CE27-29), as can be seen in Fig. 9.8a. The medium queues become balanced best in simulation run no.2 (equal base prices), but the results of simulation run no.3 show a similar performance for differentiated base prices and widely overlapping price domains.

The performance for long and very long queues seems to be best for simulation run no.1, since the RSD drops to zero after about 35000 seconds (9.7 hrs) and after about 20000 seconds (5.6 hrs) respectively. This however is not the result of a balanced distribution of the workload, but simply depends on the fact that these queues reach their maximum, as described earlier (see Fig. 9.9a, 9.10a and 9.11a), and thus their QWTs do not vary anymore. The same holds for very long queues after about 60000 seconds (16.7 hrs) in simulation run no.3 (see Fig. 9.10c). Apart from that, the results for simulation run no.2 and no.3 are comparable.

### 9.4.4 “Worst Case” and “Best Case” Scenarios

For comparison purpose a “worst case” and a “best case” scenario have been simulated using two Utility Modules that are not based on resource pricing: a Random utility — each given job is submitted to a randomly chosen matching resource\(^{11}\) — and a Minimum Completion Time utility — each given job is submitted to the matching resource that offers the shortest completion time (queue wait time plus job execution time).\(^{12}\) The simulations were executed with the same base configuration used for the previously discussed simulation runs (described in Section 9.2), except the

---

\(^{11}\)Choosing the resource randomly is a “worst case” in the sense that no optimization takes place.

\(^{12}\)A pricing strategy might also be based on the completion time, but this would require a per-job pricing rather than the per-resource pricing that is applied by the DGAS Price Authorities.
fact that the minimum completion time was determined "just in time" for each single job, thus avoiding the fluctuations resulting from an adjustment interval of 10 minutes as for the pricing-based simulations (see Section 9.5 for an explanation). The resulting QWTs of these two brokering strategies are shown in Fig. 9.14 to 9.18. The mean QWTs and relative standard deviations are shown in Fig. 9.19 to 9.21.

The comparison of Fig. 9.8b with Fig. 9.16b, Fig. 9.10b with Fig. 9.18b and Fig. 9.13b with 9.21b shows that, given an appropriate configuration, a price-sensitive brokering combined with the HPM may approximate the "best case" (completion time-based scheduling) while having the advantage of an economic approach by providing an incentive for Grid users to delay less urgent job submissions to periods with lower congestion (and thus lower prices).

Simulation run no.2 (equal base prices for all PAs) shows the best performance if compared to the "best case" scenario (completion time-based scheduling). The comparison of simulation run no.1 and no.3 with the best case, shows that, if differentiating the base prices $P_0$ according to the quality of the Computing Elements, the price variation limits $\Delta P$ have to guarantee widely overlapping price domains in order to offer an acceptable performance.
Figure 9.6: QWTs of very short queues for simulation run no.1 (a), no.2 (b) and no.3 (c).
Figure 9.7: QWTs of short queues for simulation run no.1 (a), no.2 (b) and no.3 (c).
Figure 9.8: QWTs of medium queues for simulation run no.1 (a), no.2 (b) and no.3 (c).
Figure 9.9: QWTs of long queues for simulation run no.1 (a), no.2 (b) and no.3 (c).
Figure 9.10: QWTs of very long queues for simulation run no.1 (a), no.2 (b) and no.3 (c).
Figure 9.11: Mean QWTs of all queue types for simulation run no.1 (a), no.2 (b) and no.3 (c).
Figure 9.12: Relative standard deviation of all queue types for simulation run no.1 (a), no.2 (b) and no.3 (c).
Figure 9.13: Relative standard deviation of medium, long and very long queues for simulation run no.1 (a), no.2 (b) and no.3 (c).
Figure 9.14: QWTs of very short queues for utility functions Random (a) and Minimum Completion Time (b).
Figure 9.15: QWTs of short queues for utility functions Random (a) and Minimum Completion Time (b).
Figure 9.16: QWTs of medium queues for utility functions Random (a) and Minimum Completion Time (b).
Figure 9.17: QWTs of long queues for utility functions Random (a) and Minimum Completion Time (b).
Figure 9.18: QWTs of very long queues for utility functions Random (a) and Minimum Completion Time (b).
Figure 9.19: Mean QWTs of all queue types for utility functions Random (a) and Minimum Completion Time (b).
Figure 9.20: Relative standard deviation of all queue types for utility functions Random (a) and Minimum Completion Time (b).
Figure 9.21: Relative standard deviation of medium, long and very long queues for utility functions Random (a) and Minimum Completion Time (b).
9.5 Evaluation of the Hybrid Pricing Model

Although it seems fair to price computing resources according to their “quality” parameters (e.g., processing power and maximum queue lengths), this might lead to a suboptimal resource selection and thus decrease the overall throughput of the grid, at least if a merely price-sensitive brokering strategy is used and the price domains of the different queue classes do not overlap significantly (as is the case for simulation run no.1).

The adoption of a single base price for all CEs (simulation run no.2) shows the better results. The medium queues (see Fig. 9.8b) are more or less balanced and show a constant mean QWT, since demand and offer for these queues are comparable. Although the longer queues show increasing mean QWTs (Fig. 9.9b and 9.10b) due to an over-demand, they still become more or less balanced, apart from the high fluctuations. These can be explained by basically two effects: First, the heavy jobs (see Table 9.1) that affect low performance CEs in a different way than high performance CEs. A simple example: an 80000 UCE job will increase the queue of a 800MHz CE by 100 time units (100 minutes), while it will increase the QWT of a 2.4GHz CE by only 33 time units (33 minutes). Second, the fact that new prices are calculated only every 10 minutes, means that the currently cheapest resources will be chosen for the next 10 minutes to satisfy the incoming job requests, leading thus to a much higher QWT on the next price computation. Hence the amplitudes of these fluctuations depend in a significant way on the time interval between price adjustments. A shorter interval will lead to better balanced QWTs, but also to a higher computational overhead and to an elevated number of records in the PAs’ databases. The fluctuations are furthermore intensified by the over-demand of long and very long queues, that causes much more jobs to arrive than on shorter queues.

It can be concluded that for a merely price sensitive brokering strategy a single base price, independent from CPU performance and maximum queue length, is the better choice. The interval between price adjustments has to be chosen according to the desired tradeoff between stable QWTs and low computational overhead. Since the DGAS Price Authorities allow to set different price adjustment intervals for the single resources, shorter intervals
can be set for low performant CEs that show higher fluctuations, in order to stabilize their QWTs, and longer intervals can be set for high performant CEs in order to reduce the number of records in the PA databases.

The adoption of the Hybrid Pricing Model for a “resource market” differentiated by performance and queue parameters would require an accurate tuning of the PAs’ base prices, their price variation limits and the amount of computational energy that is offered by the resources of the single queue types. This however will mean a less flexible grid setting that cannot easily adapt to the highly dynamic user behavior. Therefore the Hybrid Pricing Model should be adopted mainly for resource markets that do not require a differentiation of the resources’ base prices. A single prefixed base price, however, has to be accepted by all participating VOs in order not to interfere with their freedom to define their own policies, one of the principles of grid computing.

Different prefixed base prices might be applied for different classes of comparable resources, if users are forced to specify a single resource class on which to run a specific application (see Fig. 9.5). This requires the definition of a limited number of resource classes and leads to “subgrids” that may be easier to manage. Different resource classes with different levels of workload might also be a desirable grid setting with multiple service classes, where users pay more for empty queues if their applications require an immediate processing (e.g. for interactive jobs).

In spite of the accurate tuning that is necessary for an efficient adoption of the HPM, this approach has the advantage of relative price stability, since the fixed price interval ensures that prices do not degenerate in the long term. Furthermore, if having equal base prices for all PAs, even small price variations lead to balanced queues (see Fig. 9.1: the variation limit $\Delta P$ is only 5% of the base price $P_0$). In order to achieve resource exchange fairness this model relies on fair parameter settings by the resource managers.

It is however important to remark that a low price variation limit $\Delta P$ will significantly reduce the advantages of an economic approach: If price variations are insignificant, the HPM can balance the workload only in the short term (by “just-in-time” scheduling of the incoming workload), but it will not offer sufficient incentives for users to delay job submissions in times
of high congestion on the system, thus failing to balance the workload in the long term (see Section 6.1).

The major advantages of the HPM are its minimal implementation complexity and the low computational overhead required for price adjustments. Moreover, the fact that it requires only local information significantly limits the communication overhead.
Chapter 10

Conclusions

In a large-scale computing environment with thousands of loosely coupled atomic jobs,\(^1\) an effective load balancing and resource allocation may be based on virtual economics that dynamically adjust resource prices according to the current workload (see Section 6.1 for advantages of an economic approach). The general equilibrium theory, as described in Appendix A however, cannot be directly applied to actual markets, since they do not fulfill the theory’s restrictive assumptions (see Section 6.3).

A major concern when trying to optimize the efficiency of a virtual economy, is to define a price-setting mechanism that can bring about market equilibrium at a reasonable level of overall workload (as a long term optimization) and helps to distribute the incoming workload in order to improve the system’s throughput (as a short term optimization).

As described in Section 5.3, the tight relation between job scheduling decisions and data replication may be “decoupled” if data is dynamically replicated, as is done in EDG (see Section 2.4.1), and jobs are scheduled to sites on which the required input data is already present. Hence, job scheduling and data replication strategies may be implemented and optimized separately.

Therefore, this thesis considers only economic job scheduling strategies, although an economic approach may as well provide a framework for inte-

---

\(^1\)Most HEP applications process large amounts of experimental data in the form of single independent events.
grating job scheduling and data replication concerns.\textsuperscript{2}

10.1 Simulation Tool and Simulation Results

In Chapter 8, \textit{DGAS-Simulator} — a modular and expandable simulation tool of the EDG Workload Management System architecture — is presented. Its purpose is to study the impact of different resource pricing schemes and resource brokering strategies on workload balancing and thus on the overall throughput of the Grid. DGAS-Sim simulates \textit{static\textsuperscript{3} state-based\textsuperscript{4}}

load-balancing.

The simulation results for the Hybrid Pricing Model (see Section 7.2.2) — that dynamically adjust prices within \textit{static} limits to balance the workload on the basis of the queue wait times (QWT) — are presented and discussed in Chapter 9.

The results indicate that the HPM — provided an appropriate configuration —, combined with a price-sensitive brokering strategy, may approximate the “best case” results that are obtained when basing the job scheduling strategy on a minimization of the job completion times\textsuperscript{5}.

The best performance, if compared to completion time-based scheduling, is obtained when considering equal base prices \(P_0\) for all Computing Elements. For sufficiently large price variation limits \(\Delta P\), that guarantee widely overlapping price domains with respect to the base prices, similarly effective load balancing may be obtained even when base prices differ to reflect the resources’ different “qualities” (in terms of CPU performance, maximum queue length, etc.).

Since the adoption of the Hybrid Pricing Model for a “resource market” differentiated by performance and queue parameters requires an accurate tuning of the PAs’ base prices, their price variation limits and the amount of computational energy that is offered by the resources of the single queue

\textsuperscript{2}The dynamic data replication strategies used for EDG are based on economic principles as well [4, 5, 7, 8, 39].

\textsuperscript{3}Tasks do not migrate to another host once they have been submitted for execution.

\textsuperscript{4}The workload is balanced according to the actual system state, in contrast to model-based load balancing.

\textsuperscript{5}The job completion time is given by the queue wait time plus the job execution time.
CHAPTER 10. CONCLUSIONS

types, it should be adopted mainly for resource markets that do not require a differentiation of the resources' base prices.

10.2 Local Scheduling Policies and Per-job Pricing

The single sites of which the Grid is composed usually adopt different local scheduling policies, that are generally more sophisticated than First-Come-First-Serve (FCFS) and may lead to different job start times for jobs from different Virtual Organizations (VOs) and users [25]. It may therefore be worth to investigate mechanisms that allow a *per-job pricing*, where prices may also vary according to the job for which a price is determined. If, for example, a VO or user has been assigned a maximum number of CPUs that may be utilized on a given multi-processor Computing Element, further incoming jobs from that VO or user will be queued while jobs from other VOs or users might start execution at once if there are still free processors. Therefore a single queue wait time may not be sufficient to reflect the real state and queue status of a Computing Element [25].

However, for the efficiency of the HPM it is not important if the price is computed according to the queue wait time (if it uses FCFS) or the job start time for the given job (if the local scheduling policy may lead to different job start times), since in both cases the goal is to minimize the average job turnaround times. Hence, the HPM may also be adopted to more realistic Grid settings that require to consider job start times instead of queue wait times.

---

The presented simulation results focus on *per-resource pricing*, where resource prices depend only on the current state of the resource.
Appendix
Appendix A

General Equilibrium Theory

The following is a brief introduction to some fundamental economic concepts and terminology. Although most of the presented theory cannot directly be applied to a “market” of Grid resources as we intend it for load balancing purpose, an understanding of the relationship between price, supply, and demand and their fluctuations as users “acquire” or release Grid resources is crucial to an economic approach.

Since this is only an introduction to the theory of economic equilibria, all proofs are omitted.

A.1 Economic Model

An economy consists of three essential components: a usually finite number of commodities — that can be either storable goods or non-storable goods (e.g. services, as in the case of a Grid economy) —, prices, and a usually finite number of economic agents.

A.1.1 Commodities and Price Vectors

Let $\mathcal{N} = \{1, \ldots, N\}$ be the set of commodities in the economy, and $\mathcal{L} = \mathbb{R}^N$ the commodity space. The total resources in the economy are represented by a vector $\mathbf{w} \in \mathcal{L}$ with $w_n \in \mathbb{R}_+$ for all $n \in \mathcal{N}$, where $w_n$ specifies the total amount of the $n$-th commodity. A price vector $\mathbf{p} \in \mathcal{P}$ is a vector with $p_n$ specifying the price of the $n$-th commodity and $\mathcal{P}$ being the $N$-dimensional
space of possible price vectors (for the following discussion on economic equilibria it is assumed that $\mathcal{P} = \mathcal{L} = \mathbb{R}^N$).

In a monetary market the currency itself can be considered a commodity, thus prices of the single commodities are only relative and we can set $p_N = 1$, taking commodity $N$ as the currency.

### A.1.2 Economic Agents

Economic agents are autonomous goal-seeking entities that are characterized by their initial endowments, preferences, decision rules and constraints.

Let $\mathcal{M} = \{1, \ldots, M\}$ denote a finite set of economic agents. An action taken by an agent (also called an allocation) can be represented by a vector $\mathbf{x}_m \in \mathcal{L}$, with the $n$-th component $x_{mn}$ specifying the amount of the $n$-th commodity demanded or supplied by agent $m$. More specifically, if $x_{mn} > 0$, then agent $m$ demands the given amount of the $n$-th commodity. If $x_{mn} < 0$, then agent $m$ supplies the given amount of the $n$-th commodity.

The preferences of an agent $m$ can be represented by the binary relation $\mathbf{x}_m \succ_m \mathbf{x}'_m$ ("$\mathbf{x}_m$ is preferred to $\mathbf{x}'_m$") defined on the action set $X_m \subseteq \mathcal{L}$ (set of all possible actions) of that agent.

Each agent $m$ has an endowment $\mathbf{e}_m \in \mathcal{L}$ defining its initial possessions, where $e_{mn} \geq 0$ is the amount of commodity $n$ the agent possesses.

Although in general economic agents can require certain commodities while furnishing other commodities, they are often classified as two distinct types: consumers (demanding goods or services) and producers (supplying goods or services).\footnote{Sometimes other classes of agents, such as speculators, are defined. For the purpose of this thesis, only consumers (Grid users) and producers (Grid resources) are considered.}

### Consumers and Demand

Let $\mathcal{I} \subseteq \mathcal{M}$ denote the set of consumers. The consumption, or demand, of a consumer $i \in \mathcal{I}$ is given by a vector $\mathbf{d}_i \in \mathcal{L}$. Summing over all consumers, the total demand is

$$\mathbf{d} = \sum_{i \in \mathcal{I}} \mathbf{d}_i = \sum_{i \in \mathcal{I}} \mathbf{x}_i. \quad (A.1)$$
Assumption A.1 (Convex consumption set) Let $D_i \subseteq \mathcal{L}$ be the set of all possible demand vectors or consumption set (corresponding to the consumer’s action set). Assume $D_i$ is convex, i.e., for all $d_i^1, d_i^2 \in D_i$ the weighted average $td_i^1 + (1-t)d_i^2$ is a possible demand vector $\forall t \in [0, 1]$ (and thus element of the consumption set).

Assumption A.2 (Continuous preference preordering) The preordering of consumer preferences $(D_i, \succ_i)$ is continuous and complete, i.e. for $d_i \in D_i$, the preference set $P_i(d_i) = \{d_i' \in D_i \mid d_i' \succ_i d_i\}$ of consumptions/demands, that are preferred to the given demand vector $d_i$, is closed in $D_i$.

The preferences of a consumer $i \in \mathcal{I}$ are represented by a utility function that measures the degree of consumer satisfaction (utility) with regard to all possible consumptions or demand vectors.\footnote{For a Grid economy it is important to note that utility functions are merely theoretical constructs, since user behavior is in general not easily observable. Still, the “ranking” of resources by the Resource Broker (see Section 3.3.4), might be compared to the concept of utility functions.} The Assumptions A.1 and A.2 imply the existence of utility functions with the following properties:

Definition A.1 (Utility function) A utility function $u_i$ is a continuous, monotonically increasing, real-valued function of the completely preordered set of preferences: i.e., $u_i : D_i \mapsto \mathbb{R}$.

Let $p \in \mathcal{P}$ be a price vector with $p_n$ being the price of commodity $n$. The value of (or expenditure for) a consumption $d_i$ with regard to that price vector is given by $p \cdot d_i$. Hence, the value of consumer $i$’s endowment $e_i$ with regard to the price vector is $p \cdot e_i$. The expenditure for a consumption $d_i$ by a consumer $i$ may not exceed the value of it’s initial endowment (budget or wealth constraint).

Therefore the goal of a consumer $i$ — choosing the most preferred consumption, subject to its budget constraint — can be represented by the following utility maximization problem:

$$\max \quad u_i(d_i) \quad \text{s.t.} \quad p \cdot d_i \leq p \cdot e_i \quad \text{(A.2)}$$
Producers and Supply

Producers or suppliers can be described similarly to consumers, but they are characterized by “technological” limitations (supply capacities) and seek profit maximization (their utility functions reflect the profit for supplied commodities).

Let $J \subseteq M$ denote the set of producers. The production, or supply, of a producer $j \in J$ is given by a vector $s_j \in L$. Summing over all producers, the total supply is

$$s = \sum_{j \in J} s_j = -\sum_{j \in J} x_j.$$  \hspace{1cm} (A.3)

A.1.3 Economies

Definition A.2 (Economy) Given a set of commodities $N = \{1, \ldots, N\}$, an economy $E = (X_m, u_m, e_m)_{m \in M}$ is a finite set of economic agents $M = \{1, \ldots, M\}$ (both consumers and producers), each agent $m \in M$ being characterized by

- an action set $X_m \subseteq L$,
- a utility function $u_m : X_m \mapsto \mathbb{R}$ (consumers) or a profit function (producers), and
- an initial endowment $e_m \in L$.

A pure exchange economy is an economy without production, in which only consumption of the initially present resources takes place.

Additionally to the assumption that economic agents act rationally and selfishly (by maximizing a personal utility as already described above), economic models are usually formulated under the assumption of perfect competition. That is, all economic agents are supposed to control sufficiently small segments of the market, such that their individual actions do not influence global prices, and thus are considered to be price-takers.
A.2 Market Equilibrium

In 1776, Adam Smith assumed in *The Wealth of Nations* that social systems tend towards a state of balance despite the individualistic goals pursued by selfish agents, giving birth to the theory of competitive equilibrium. The modern concept of competitive equilibrium, however, was introduced by Léon Walras (1834–1910), who defined an equilibrium as a set of prices (price vector) that equates supply and demand on all markets, that is, for all commodities [42].

A.2.1 Excess Demand

**Definition A.3 (Excess demand)** Given an economy $\mathcal{E}$ with consumers and producers, and given a price vector $p \in \mathcal{P}$, the **total or aggregate excess demand** $z$ is defined as the difference between total demand $d$ and total supply $s$:

$$z(p) = d(p) - s(p) = \sum_{i \in I} d_i(p) - \sum_{j \in J} s_j(p) = \sum_{m \in M} x_m(p).$$  \hspace{1cm} (A.4)

Note that the excess demand is a vector in the commodity space $\mathcal{L}$. The (single-valued) total excess demand for the $n$-th commodity is given by

$$z_n(p) = \sum_{m \in M} (d_{mn}(p) - s_{mn}(p)) = \sum_{m \in M} x_{mn}(p),$$  \hspace{1cm} (A.5)

where $d_{mn}(p)$ and $s_{mn}(p)$ denote the demand and supply of agent $m$ for the $n$-th commodity. Since each agent $m$ is supposed to demand or supply, this is equal to its allocation $x_{mn}(p)$.

An excess of supply is simply represented by a negative excess demand. Note that demand and supply, and thus excess demand, of a particular commodity $n$ are functions of the entire price vector $p$ and not only of its $n$-th component $p_n$. In general, price variations in one market (that is, for one commodity) may influence demand and supply in the other markets as well, as covered by the following two definitions.
Definition A.4 (Gross substitutes) Let \( n_i, n_j \in \mathcal{N}, n_i \neq n_j \), be two commodities of an economy \( \mathcal{E} \). The two commodities are called **gross substitutes** if their total excess demands \( z_{n_i}(p) \) and \( z_{n_j}(p) \) fulfill the following condition:\(^3\)

\[
\frac{\partial z_{n_i}(p)}{\partial p_{n_j}} > 0
\]

That is, two commodities are considered gross substitutes if an increasing (decreasing) price of one of the commodities does not only cause a decreasing (increasing) excess demand for that commodity, but also an increasing (decreasing) excess demand for the other one.

As an example, consider two commodities A and B of similar characteristics. If the price for commodity A increases, the demand for it obviously decreases. Since commodity B has similar characteristics, the consumers acquire B instead of A, and thus the demand for B increases.

Definition A.5 (Gross complements) Let \( n_i, n_j \in \mathcal{N}, n_i \neq n_j \), be two commodities of an economy \( \mathcal{E} \). The two commodities are called **gross complements** if their total excess demands \( z_{n_i}(p) \) and \( z_{n_j}(p) \) fulfill the following condition:\(^3\)

\[
\frac{\partial z_{n_i}(p)}{\partial p_{n_j}} < 0
\]

That is, two commodities are considered gross complements if an increasing (decreasing) price of one of the commodities does not only cause a decreasing (increasing) excess demand for that commodity, but for the other one as well.

As an example, consider two commodities A and B that are both required by the consumer population to satisfy its preferences. If the price for commodity A increases, the demand for it decreases, and so does the demand for commodity B, since without commodity A, commodity B is not necessary.

---

\(^3\) Note that in general the signs of the partial derivatives depend on the size of \( z(p) \), two commodities may be gross substitutes or gross complements only for certain values of \( z(p) \).
For \( n_1 = n_2 = n \) we assume that the excess demand has always the opposite sign of the increase in price:\(^4\)
\[
\frac{\partial z_n(p)}{\partial p_n} \leq 0
\]

### A.2.2 Competitive Equilibrium

**Definition A.6 (Market allocation)** Given an economy \( \mathcal{E} \), a market allocation \( \langle x_m \rangle_{m \in \mathcal{M}} \) is defined as an \( M \)-tuple of commodity vectors, that is, a specification of actions for all agents \( m \in \mathcal{M} \), s.t. \( x_m \in X_m \), where \( X_m \) is agent \( m \)’s action set. If the agents can be classified as consumers and producers, then a market allocation can be specified as
\[
\langle x_m \rangle_{m \in \mathcal{M}} = \left( \langle d_i \rangle_{i \in \mathcal{I}}, \langle s_j \rangle_{j \in \mathcal{J}} \right),
\]
where \( \langle d_i \rangle_{i \in \mathcal{I}} \) represents the demands of the consumers and \( \langle s_j \rangle_{j \in \mathcal{J}} \) represents the supplies of the producers. The set of all possible allocations for the economy \( \mathcal{E} \) is given by
\[
ALL(\mathcal{E}) \equiv \left\{ \langle x_m \rangle_{m \in \mathcal{M}} \mid x_m \in X_m, \forall m \in \mathcal{M} \right\}.
\]

A market allocation is considered to be a market equilibrium if the total demand \( d \) does not exceed the total supply \( s \), and thus the total excess demand is less than or equal to zero. In such a case the market is said to be cleared.

**Definition A.7 (Market equilibrium)** Given an economy \( \mathcal{E} \), a market equilibrium \( \langle x_m^* \rangle_{m \in \mathcal{M}}, p^* \) is a tuple consisting of a market allocation \( \langle x_m^* \rangle_{m \in \mathcal{M}} = \left( \langle d_i^* \rangle_{i \in \mathcal{I}}, \langle s_j^* \rangle_{j \in \mathcal{J}} \right) \in ALL(\mathcal{E}) \) and an equilibrium price vector \( p^* \in \mathcal{P} \), such that for all commodities \( n \in \mathcal{N} \)
\[
z_n(p^*) \leq 0.
\]

The set of all market allocations for the economy \( \mathcal{E} \), for which market equilibria exist, is given by
\[
ME(\mathcal{E}) \equiv \left\{ \langle x_m^* \rangle_{m \in \mathcal{M}} \in ALL(\mathcal{E}) \mid \exists p^* \in \mathcal{P}, \text{ s.t. } z_n(p^*) \leq 0 \ \forall n \in \mathcal{N} \right\}
\]

\(^4\)This assumption does not always hold, depending on whether the utility or the profit is kept constant by the partial derivation.
Note: In most cases market equilibrium is defined even more restrictively by requiring the total demand to be equal to the total supply, i.e.

\[ z_n(p^*) = 0 \quad \forall n \in \mathcal{N} \quad \text{or simply} \quad z(p^*) = 0. \quad (A.6) \]

An alternative definition — more suitable for an application to a Grid economy — is similar to a definition given by Ferguson et al. [18]: An economy \( \mathcal{E} \) is considered to be in equilibrium at a given price vector \( p^* \in \mathcal{P} \), if \( \forall n \in \mathcal{N} \) either

- \( z_n(p^*) = 0 \), or
- \( z_n(p^*) < 0 \) and \( p_n^* = \epsilon_n \),

where \( \epsilon_n \in \mathcal{P} \) is a minimum price for commodity \( n \) and thus the excess supply cannot be reduced by further decreasing \( n \)'s price.\(^6\)

**Definition A.8 (Competitive equilibrium)** Given an economy \( \mathcal{E} \) with consumers and producers, a market equilibrium \( (\mathbf{x}_m^*, \mathbf{p}^*) \) is called a competitive equilibrium, or Walrasian equilibrium, if all economic agents maximize their utility (or profit), i.e.

\[
(\mathbf{x}_m^* \mid m \in \mathcal{M}, \mathbf{p}^*) = (\langle \mathbf{d}_i^* \rangle_{i \in \mathcal{I}}, \langle \mathbf{s}_j^* \rangle_{j \in \mathcal{J}}, \mathbf{p}^*)
\]

s.t. \( \forall i \in \mathcal{I} : u_i(d_i^*) = \max u_i(d_i), \forall d_i \in D_i \) with \( p^* \cdot d_i \leq p^* \cdot e_i \)

and \( \forall j \in \mathcal{J} : s_j^* \in \left\{ s_j \in S_j \mid \arg \sup p^* \cdot s_j \right\} \),

where \( p^* \cdot d_i \leq p^* \cdot e_i \) is the budget constraint (see Equation (A.2)) and \( D_i \) the consumption set of consumer \( i \), and \( S_j \) is the production set (i.e. the set of all “technically” possible supply vectors) of producer \( j \). The profit for a supply vector \( s_j \) is given by \( p^* \cdot s_j \).

\(^{\text{Ferguson et al., however, did not consider the excess demands for different commodities, but for the single links in an economy for flow control, and described only one minimum charge } \epsilon \text{ for all links.}}\)

\(^{\text{Note that, if having gross substitutes (see Def. A.4), the excess demand } z_n \text{ of commodity } n \text{ may also be raised by increasing the price of another commodity } n'. \text{ This however will cause the excess demand } z_{n'} \text{ of commodity } n' \text{ to decrease, thus leading away from market equilibrium with regard to } z_{n'}.\)
Although the principal goal of an economy is an efficient (but not necessarily homogeneous) provisioning and distribution of resources, an optimal allocation may also address issues of welfare and fairness, as outlined by the following definition and theorem.

**Definition A.9 (Pareto-optimality)** Given an economy $\mathcal{E}$, a market allocation $\{x_m\}_{m \in \mathcal{M}}$ is said to be Pareto-optimal if the utility of no agent can be increased without decreasing the utility of at least one other agent.

**Theorem A.1 (First Theorem of Welfare Economics)** Given an exchange economy $\mathcal{E}$, if $\left(\{x'_m\}_{m \in \mathcal{M}}, \mathbf{p}^*\right)$ is a competitive equilibrium, then $\{x'_m\}_{m \in \mathcal{M}}$ is a Pareto-optimal market allocation.

Note the similarity to the concept of Nash equilibrium in game theory, that is briefly described in Appendix A.4.

The proof of Theorem A.1 relies on the important Walras’ Law. Assume the consumer demand in an exchange economy to be insatiable, i.e. $\forall m \in \mathcal{M}$ there is no empty preference set $P_m(x'_m) = \{x_m \in X_m \mid x_m \succ_m x'_m\} = \emptyset$ and thus no allocation vector $x'_m$ is preferred to all others. This implies that agents spend their entire wealth in consumption, which also means that their budget constraints (see Equation (A.2)) are binding. Mathematically, $\mathbf{p} \cdot x_m = \mathbf{p} \cdot \mathbf{e}_m \quad \forall m \in \mathcal{M}$. Under this assumption Walras’ Law can be formulated as

**Theorem A.2 (Walras’ Law)** Given an exchange economy $\mathcal{E}$ and a price vector $\mathbf{p} \in \mathcal{P}$,

$$\mathbf{p} \cdot \mathbf{z}(\mathbf{p}) = \mathbf{p} \cdot \mathbf{d}(\mathbf{p}) - \mathbf{p} \cdot \mathbf{s}(\mathbf{p}) = 0,$$

that is, the value of the total demand equals the value of the total supply.

Note that Walras’ Law is valid even if the exchange economy is not in equilibrium and thus for each possible price vector $\mathbf{p}$.

---

5Named after Vilfredo Pareto (1848–1923).

6An economy without real production, in which only consumption of the initially present resources takes place, i.e. the vector $\mathbf{w}$ of total resources in the economy is always equal to $\sum_{m \in \mathcal{M}} \mathbf{e}_m$, the sum of all agents’ endowments.

7An assumption that does not hold for a Grid environment where the “consumption” of users depends on the workload they need to introduce to the system.
A.2.3 Existence of Competitive Equilibria

Assumption A.3 (Free disposal) Assume that an economy allows for free disposal, thus prices are never negative:

\[ p_n \geq 0 \quad \forall n \in N \]

No agent would supply a commodity at a negative price, if the commodities can be simply disposed.

Furthermore, competitive equilibrium prices are not unique, but relative, as stated by the following lemma:

Lemma A.1 Given an exchange economy \( \mathcal{E} \), let \( (x^*_m)_{m \in M}, p^* \) be a competitive equilibrium. Then \( (\lambda x^*_m)_{m \in M}, \lambda p^* \) is also a competitive equilibrium \( \forall \lambda \in \mathbb{R}_+ \).

The price vector \( \lambda p^* \) still satisfies the budget constraints of all agents and maximizes their utility. Furthermore, taken into account the assumption of insatiable consumer demand for at least one commodity, it can be easily shown that the price vector \( p \) never becomes zero and \( p_n > 0 \) for at least one \( n \in N \).

It is thus convenient to normalize prices. The existence of competitive equilibria can be proved, if restricting prices to the \( n \)-dimensional unit simplex \( S_n \):

\[ S_n = \left\{ p \in \mathcal{P} \mid \sum_{n \in N} p_n = 1 \text{ and } p_n \geq 0 \quad \forall n \in N \right\} \]

Theorem A.3 (Arrow-Debreu Theorem) In an exchange economy \( \mathcal{E} \), let the excess demand vector \( z(p) \) satisfy the following conditions:

1. \( z(p) \) is a continuous and bounded function.

2. \( z(p) \) is homogeneous of degree 0, i.e. scaling of all prices or all initial endowments by the same factor does not alter the choices of the agents.

3. \( p \cdot z(p) = 0 \) (Walras’ Law).

Then there exists always a price vector \( p^* \) and a market allocation \( (x^*_m)_{m \in M} \) such that a competitive equilibrium is achieved.
APPENDIX A. GENERAL EQUILIBRIUM THEORY

The proof, that is out of scope of this thesis, is based on the Brouwer Fixed Point Theorem\textsuperscript{10}, showing the existence of a price vector $p^\ast$ (in analogy to the fixed point in a topological space) for which $z(p^\ast)$ satisfies Definition A.7.

In practice, however, the assumptions required for the Arrow-Debreu Theorem do not hold, since excess demands resulting from competitive behavior are often discontinuous.\textsuperscript{11} Hence, the existence of a competitive equilibrium is not guaranteed in the general case. Nonetheless, the larger the number of agents, and the more heterogeneous their decision-making and thus their utility functions, the larger the possibility that a competitive equilibrium exists, since the resulting aggregate excess demand may be “continuous and smooth enough” in the sense that its granularity and its discontinuities are small compared to its magnitude (pp. 627-630 of [26]). Furthermore, the fact that the existence proof requires restrictive conditions to be met, does not imply that there exists no equilibrium if the conditions are not met. Deng et al. [15] showed that if the competitive equilibrium exists, in some cases it may be possible to closely approximate it with a polynomial-time algorithm, even if the allocations (amounts) of commodities are integer-valued, and thus utility functions and demands necessarily discontinuous.\textsuperscript{12}

A.2.4   Uniqueness of Competitive Equilibria

As already stated by Lemma A.1, equilibrium price vectors $p^\ast \in P$ are not unique. The corresponding market allocation $(x^\ast_m)_{m \in M}$, however, is not affected by the multiplication of $p^\ast$ by a factor $\lambda \in \mathbb{R}_+$. Let therefore

$$(x^\ast_m)_{m \in M} \equiv (x^\ast_m)_{m \in M}, \lambda p^\ast$$

\textsuperscript{10}Every continuous mapping of a compact, topological space into itself has a fixed point, i.e. if $f : P \mapsto P$ continuous, then $\exists x^\ast \in P$ s.t. $f(x^\ast) = x^\ast$.

\textsuperscript{11}Utility functions are usually discrete, as are prices and allocations (amounts) of commodities.

\textsuperscript{12}Deng et al., however, require the number of commodities to be bounded and the utility functions to be linear of form $u_m(x_m) = \sum_{n \in N} v_{mn} x_{mn}$ with $v_{mn} \in \mathbb{N} \cup \{0\}, m \in M$, which is too restrictive for our purpose.
be considered a single competitive equilibrium. It can be shown that under the assumption of gross substitutability of all commodities (see Definition A.4) in an economy $E$, a competitive equilibrium is “unique”, i.e. there exists no other market allocation $\langle x'_m \rangle_{m \in M}$ and (relative) price vector $p^s \in \mathcal{P}$ that satisfy the conditions for competitive equilibrium in that economy.

In an economy that contains gross complements (see Definition A.5) as well, there may exist multiple competitive equilibria.

### A.3 The Tâtonnement Process

Assume to have an economy $E$ that fulfills all conditions for the existence of a competitive equilibrium. To achieve this equilibrium, Walras suggested a tâtonnement ("groping") process that iteratively adjusts prices until excess demand is zero on all markets, that is, for all commodities [42]. In this process prices for commodities with positive (negative) excess demand are raised (lowered) in order to deter (stimulate) consumer demand, such that

$$\Delta p_n \propto z_n(p), \quad n \in \mathcal{N}.$$  

Thus, price adjustment for commodity $n$ is a function of the corresponding excess demand $z_n$, which in turn is a function of the entire price vector $p$.

**Algorithm A.1 (Tâtonnement algorithm)** Ferguson at al. [18] describe the tâtonnement algorithm as follows:

1. Choose an initial price vector $p$.

2. Determine the excess demand $z(p)$ at price $p$.

3. If $\forall n \in \mathcal{N}, z_n(p) = 0$, or if $\forall n \in \mathcal{N}, z_n(p) \leq 0$ and $p_n = \epsilon_n$ (with $\epsilon_n$ being the minimum price for commodity $n$), then market equilibrium\textsuperscript{13} is reached and the iteration is stopped\textsuperscript{14}.

\textsuperscript{13}Remember that there are other definitions for market equilibrium, as described in Appendix A.2.2.
\textsuperscript{14}This assumes an economy in which the number of agents and their utility functions are not functions of time, and thus the equilibrium price vector is not. In a dynamic context, such as a Grid economy, the iteration would continue.
4. Else, \( \forall n \in \mathcal{M} \) the new price \( p'_n \) is computed according to the following function:

\[
p'_n = \max \left[ p_n + p_n \frac{z_n(p)}{s_n(p)}, \epsilon_n \right], \tag{A.8}
\]

where \( s_n(p) \) is the total supply of commodity \( n \) at price \( p \).

5. Continue at step 2.

The price adjustment \( \Delta p_n \) is thus the current price \( p_n \) multiplied by the relative excess demand \( z_n / s_n \), that depends on the prices of all commodities.

### A.3.1 Stability

In case of multiple equilibria, that are possible if some of the commodities are gross complements, the tâtonnement process gives no guarantees about which of the equilibria will be selected. In such a case the behavior of the algorithm depends heavily on the choice of the initial price vector.

Even worse, the interactions between price adjustments of different commodities may have a negative effect on the dynamic properties of the system, due to destabilizing cross-effects. An adjustment of the price for commodity \( n \) will also affect the excess demands of other commodities and thus their prices, which in turn affect the excess demand for commodity \( n \). Scarf [35] demonstrated that the tâtonnement process is unstable for an open set of economies having a unique equilibrium.

For an economy with only two commodities and an excess demand function \( z(p) \in C^1 \), i.e. \( z(p) \) is differentiable and has a continuous derivative, the process is guaranteed to converge to some competitive equilibrium.

Although in realistic economic settings there is no guarantee that the tâtonnement process will converge, it may well be able to approximate a market equilibrium.\(^{15} \)

\(^{15}\)In a highly dynamic environment, such as our Grid, this may be all we can hope for anyway.
A.4 Nash Equilibrium in Game Theory

The purpose of this Section is to outline some similarities of the described general equilibrium theory with game theory\textsuperscript{36}; no detailed description is given. The following discussion therefore uses the same nomenclature as in the previous sections.

Consider a game with $M$ players that can choose among a set of strategies $X_m$, $m \in \{1, \ldots, M\}$. Each player $m$ has a function $u_m : X_1 \times \cdots \times X_M \rightarrow \mathbb{R}$ that assigns a real-valued payoff to that player, based on the combined choice of all players.

The predominant (but not the only) concept of rational behavior in game theory is represented by the so called Nash equilibrium:

**Definition A.10 (Nash equilibrium)** Let $x_1 \in X_1, \ldots, x_M \in X_M$ be a combination of strategies, such that $\forall m, i \in \{1, \ldots, M\}$ and $\forall x'_i \in X_i$

\[
    u_m(x_1, \ldots, x_i, \ldots, x_M) \geq u_m(x_1, \ldots, x'_i, \ldots, x_M),
\]

That is, a combination of strategies from which no player has an incentive to deviate. Then the game is said to be in **Nash equilibrium**.

Nash proved that Nash equilibrium is guaranteed to exist under certain conditions, including convex strategy sets $X_m$ (for a definition of a convex set, see Assumption A.1) and continuous $u_m$. As for economic equilibria there may be multiple Nash equilibria in a game.

\textsuperscript{36}In fact, the Nash equilibrium is one of the concepts of equilibrium used by economists.
Appendix B

Pareto-Optimality and Total Utility

Ygge et al. stated in [46, 47] that any Pareto-optimal allocation (see Definition A.9) in the market described by Definition 6.1 is a solution to the resource allocation problem $MP$ of Definition 5.1. The proof given by Ygge et al., however, is based on the assumption that for each allocation that is not a solution to $MP$, there exists a Pareto-improvement, an assumption that however does not hold, as shall be demonstrated with a simple example.

The resource allocation problem $MP$ is solved by an allocation $\langle r_m \rangle_{m \in M} = [r_1, \ldots, r_M]$ that maximizes $\sum_{m \in M} f_m(r_m)$. Since

$$\sum_{m \in M} u_m(r_m, m_m) = \sum_{m \in M} f_m(r_m) + \sum_{m \in M} m_m,$$

and $\sum_{m \in M} m_m = M$, independent of the resource allocation to the single agents, for two allocations $\langle r_m^{(a)} \rangle_{m \in M} = [r_1^{(a)}, \ldots, r_M^{(a)}]$ and $\langle r_m^{(b)} \rangle_{m \in M} = [r_1^{(b)}, \ldots, r_M^{(b)}]$ we can define:

$$\Delta f_{a \rightarrow b} = \sum_{m \in M} f_m(r_m^{(b)}) - \sum_{m \in M} f_m(r_m^{(a)}) =$$

$$= \sum_{m \in M} u_m(r_m^{(b)}, m_m^{(b)}) - \sum_{m \in M} m_m^{(b)} - \left( \sum_{m \in M} u_m(r_m^{(a)}, m_m^{(a)}) - \sum_{m \in M} m_m^{(a)} \right) =$$

$$= \sum_{m \in M} u_m(r_m^{(b)}, m_m^{(b)}) - \sum_{m \in M} u_m(r_m^{(a)}, m_m^{(a)}) = \Delta u_{a \rightarrow b}.$$
Hence, a solution to $MP$ is an allocation $(r_m^{(a)})_{m \in M}$, such that

$$\Delta f_{a \to b} = \Delta u_{a \to b} \leq 0 \quad \forall \text{possible } (r_m^{(b)})_{m \in M}. \quad (B.1)$$

The maximization of $\sum_{m \in M} f_m(r_m)$ is therefore equivalent to the maximization of the total utility

$$\sum_{m \in M} u_m(r_m, m_m).$$

Consider a very simple market with only two agents 1 and 2 having a utility possibility set, i.e. a set of possible utility vectors $u = [u_1, u_2]$ whose components are the single utilities, as depicted in Fig. B.1.

![Utility possibility set and Pareto-optimal frontier for two agents (example).](image-url)
APPENDIX B. PARETO-OPTIMALITY AND TOTAL UTILITY

The utility possibility set is the region below the Pareto-optimal frontier, that is, the region below the set of all Pareto-optimal allocations.\(^1\) The utility possibility set is limited due to the fact that the market contains only a limited amount of resources (including the funds). The Pareto-optimal frontier has a negative slope since a Pareto-optimal allocation does not allow to increase the utility of one of the agents without decreasing the utility of at least one other agent (see Definition A.9).\(^2\) It is important to note that in general, the Pareto-optimal frontier is less smooth than shown in Fig. B.1; it often is not even continuous, since most real markets usually contain discrete commodities and utility functions (if they exist) often do not fulfill the requirements for an ideal behavior.

A Pareto-improvement for an allocation \(\langle r_m \rangle_{m \in M} \) is any allocation that increases some agent’s utility while not decreasing any of the other agents’ utilities. The allocations \(\langle r_m^{(b)} \rangle_{m \in M} \) and \(\langle r_m^{(c)} \rangle_{m \in M} \) in Fig. B.1, for example, are Pareto-improvements of \(\langle r_m^{(a)} \rangle_{m \in M} \), it is also said that they Pareto-dominate \(\langle r_m^{(a)} \rangle_{m \in M} \).

Now consider allocation \(\langle r_m^{(d)} \rangle_{m \in M} \) in Fig. B.1. It is not a solution to MP, since it does not maximize the total utility. The allocation \(\langle r_m^{(c)} \rangle_{m \in M} \), for example, brings about an only slightly lower utility for agent 1 while increasing the utility of agent 2 significantly, thus having a higher total utility than \(\langle r_m^{(d)} \rangle_{m \in M} \).\(^3\) Mathematically, \(\Delta u_{d \rightarrow c} > 0\), in contradiction to equation (B.1). In contrast to what assumed by Ygge et al. [46, 47], however, \(\langle r_m^{(d)} \rangle_{m \in M} \) is Pareto-optimal, i.e. there are no possible Pareto-improvements, since no agent’s utility can be increased without decreasing the other agent’s utility.

This simple example shows that Pareto-optimal allocations are not necessarily solutions to MP.

---

\(^1\)In general, in an economy with \(M\) agents, the points of Pareto-optimal allocation form a hypersurface of dimension \(M - 1\).

\(^2\)If the frontier had a positive slope, it would be possible to contemporaneously increase the utilities of both agents.

\(^3\)Note that a comparison of the two allocations is possible only if a total utility can be defined, as has been done in Def. 5.1, assuming that the agents’ preference functions are based on a common unit.
Appendix C

Notation of UML diagrams

The following (not exhaustive) legend briefly explains the notations used for the Unified Modeling Language (UML) diagrams in Chapters 3, 4 and 8. Figure C.1 depicts the UML notation for class diagrams, while Fig. C.2 shows the notation of component and collaboration diagrams (using some notation of deployment diagrams as well).
C.1 Class Diagrams

The purpose of a UML class diagram is to depict the classes of an object-oriented application and the relationships between them. Attributes (member variables) are specified with their respective types, and methods (member functions) with their parameters and return values. The visibility of class members is indicated by a plus sign (+) for public members or a minus sign (-) for private members. Relationships between classes may be of different types. Associations (in this case directed associations) indicate references to other classes. Aggregations indicate that classes contain other classes without necessarily having a tight relationship between their lifecycles. The multiplicities of associations and aggregations determine the numerical relationship between instances of the classes. In Fig. C.1, for example, each CE is associated to one PA, but each PA is responsible for several CEs.

Note that calls are usually parts of interaction diagrams. In this case, they are used to at least outline the program flow.

![Class Diagram](image_url)

Figure C.1: Notation of UML class diagrams.
C.2 Component and Collaboration Diagrams

Component diagrams show dependencies between different parts of the software and different nodes, while collaboration diagrams focus on the interactions between them. Note that the numeration used in the collaboration diagram Fig. 4.3 does not respect the UML specification. The numeration used indicates the different steps from job submission to the payment transactions with the process being divided into six major parts that are described in Section 4.1.2. The dependencies in the component diagrams and Fig. 3.1 and Fig. 4.2 should be specified only between nodes, not between the single software components. For reasons of clarity however the dependencies between the single components are included.

![Diagram of Grid Resource Information Server (GRIS) and Resource Broker](image)

![Diagram of HLR/Bank Server and ATM engine](image)

Figure C.2: Notation of UML component and collaboration diagrams.
Appendix D

Glossary

ALICE One of the four major LHC experiments.

AOD Analysis Objects Dataset.

API Application Programming Interface.

ATLAS One of the four major LHC experiments.

ATM Automatic Transaction Manager. A DGAS component responsible for the crediting and debiting of resource usage.

CE Computing Element. A DataGrid resource that offers computational power.

CERN Conseil Européen pour la Recherche Nucléaire, the European Organization for Nuclear Research.

ClassAd Condor Classified Advertisement. Used to specify job requirements as well as resource characteristics.

CMS Compact Muon Solenoid. One of the four major LHC experiments.

DF Derivative Follower. An algorithm for resource pricing, see Section 7.2.1.

DGAS DataGrid Accounting System.

EDG European DataGrid.

ESD Event Summary Data.
**FIFO** First-In-First-Out.

**FCFS** First-Come-First-Served.

**GC** Grid Credit. A virtual currency unit for resource exchange markets.

**GIIS** Grid Index Information Server. A GIIS is an important part of the GIS. It stores information on Grid resources in a relational database and publishes it through an LDAP server.

**GIS** Grid Information Service. A service that collects, organizes and publishes information on the state of Grid resources. The GIS is identical to the MDS.

**Globus** A toolkit that furnishes basic functionalities required by higher-level Grid middleware, such as a security infrastructure or resource access management. See [http://www.globus.org/](http://www.globus.org/)

**GMT** Greenwich Mean Time.

**GRAM** Globus Grid Resource Allocation and Management.

**GRIS** Grid Resource Information Server. A server located on a Grid resource. It can be queried to obtain information about the resource's characteristics and its current state.

**GSI** Grid Security Infrastructure. A part of the Globus Toolkit that furnishes the basic infrastructure needed for reliable authentication of users and hosts and the secure communication between all entities participating in the Grid environment.

**HEP** High-Energy Physics.

**HLR** Home Location Register. A server that manages the accounts of a subset of the DataGrid users and resources (usually belonging to a specific Virtual Organization).

**HPC** High Performance Computing.

**HPM** Hybrid Pricing Model. An algorithm for resource pricing, see Section 7.2.2.
HTC High Throughput Computing.

INFN Italian National Institute of Nuclear Physics.

JDL Job Description Language (based on Condor ClassAds, see there).

LDAP Lightweight Directory Access Protocol (defined by RFC 1588 and 2251).

LHC Large Hadron Collider.

LHC-b One of the four major LHC experiments.

LRMS Local Resource Management System, e.g. BQS, LSF, PBS.

MDS Meta Directory Service. The MDS is identical to the GIS.

PA Price Authority. A DGAS component that sets resource prices. PA servers can be queried for prices and are responsible for a subset of the DataGrid resources (usually belonging to a specific Virtual Organization or a specific site).

QWT Queue Wait Time. The time a job is pending in a queue before starting execution.

RB Resource Broker (aka Matchmaker).

RFC Request For Comments. See http://www.ietf.org/rfc.html

RSD Relative Standard Deviation. The ratio of the standard deviation to the mean value.

SE Storage Element. A DataGrid resource that offers storage for data replicas.

SMP Symmetric Multi Processor

UCE Unit of Computational Energy. The product of a performance factor of a resource and the amount of resource usage.

UI User Interface.
APPENDIX D. GLOSSARY

UML Unified Modeling Language.

URI Uniform Resource Identifier.

VO Virtual Organization. An organization that administratively groups a set of users and/or resources.

WM Workload Manager.

WMS EDG Workload Management System.

WP EDG Work Package:

- Middleware
  WP1 Workload Management.
  WP2 Data Management.
  WP3 Grid Monitoring services.
  WP4 Fabric Management.
  WP5 Mass Storage Management.

- Grid Fabric and Testbed
  WP6 Integration Testbed.
  WP7 Network Services.

- Scientific Applications
  WP8 HEP Applications.
  WP9 Earth Observation Science Applications.
  WP10 Biology Applications.

- Management
  WP11 Dissemination.
  WP12 Project Management.

XML eXtensible Markup Language.
Bibliography


The Last Page

The last page is dedicated to my friends:

- denen, die ich in Aschaffenburg zurückgelassen habe: Achim, Detl, Falco, Norman, Oli (2x), Ralph, Ruben, Sandro, Steve, ...
- dem gesamten channel #midgard (bis zur nächsten Xmas-CP!)
- den echten, falschen, ehemaligen und gegenwärtigen Heidelberger: Ada, Alexander, Bibi, Bon, Christian, Elena, Georg, Hannes, Hans, Hartwig, Ivana, Kate, Katrin, Lale, Mirko, Noemi, Patricia, Ralf, Tobias, Tom, ...
- denen, die sonst wo im deutschsprachigen Raum leben: Andreas, Anja, Bettina, Birgit, Christoph, Giesbert, Karin, Mauro, Sev, Silke, Susi, ...
- to the “fruitcakes” of the Mannheim Open Circle: Aaron, Claudia, Dagi, Janet, Lisa, Rona, Tannya, ...
- a tutti coloro che sono ancora qua a Torino con me: Andrej, Carmen, Daniela, Edith, Eli, Felicia, Filipa, Francesco (tutti quanti :o), Iggy, Indrit, Karlene, Katia, Lale, Laura, Leslie, Nikola, Paola, Sebastiano, Vania, Zane, Zelka e le ragazze dell’isola sbagliata (Cri e Sara)
- a tutti coloro che sono almeno in Italia: Elena (anche lei dell’isola sbagliata), Fabio, Johanna, Marya (Goccia del Mare)
- a todos ellos que viven en España o America Latina: Ainhoa, Francesco, Gema, Teresa (miña Urania), Karen, María (todas las Mariás :o), Rocío, Sandra
- and to all of those who are somewhere out there in the rest of the world: Barbara, Beth (both of them :o), Dimitra, Fabrizio, Helga, Ji Hyun, John, Kate, Kitty, Lenka, Manuela, Mia, Olivier, Reda, Teija, ToZe
- ... and of course to my family:
- meinen Eltern, meiner Schwester Patrizia, Alex, Paul und dem Rest meiner Familie in Deutschland
- a nonna e tutti i miei cugini e zii in Italia
- y a Gretel e mi familia en España
- ... and finally to all those I forgot and all those I do not yet know.

Wind under your wings,
Rosario/Faust.