A Framework for the Study of

Grid Inter-Operation Mechanisms

A Framework for the Study of **Grid Inter-Operation Mechanisms**

PROEFSCHRIFT

ter verkrijging van de graad van doctor
aan de Technische Universiteit Delft,
op gezag van de Rector Magnificus Prof. dr. ir. J.T. Fokkema,
voorzitter van het College voor Promoties,
in het openbaar te verdedigen
op dinsdag 20 januari 2009 om 15.00 uur

door

Alexandru IOSUP

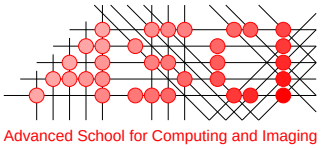
Master of Science in Computer Science, Universitatea Politehnicã Bucureşti, Roemenië
geboren te Boekarest, Roemenië.

Dit proefschrift is goedgekeurd door de promotor:

Prof. dr. ir. H.J. Sips

Samenstelling promotiecommissie:

Rector Magnificus	voorzitter
Prof.dr.ir. H.J. Sips	Technische Universiteit Delft, *promotor*
Dr.ir. D.H.J. Epema	Technische Universiteit Delft, *copromotor*
Prof.dr.ir. H.E. Bal	Vrije Universiteit Amsterdam, the Netherlands
Prof.dr. T. Fahringer	Universität Innsbruck, Austria
Prof.dr.ir. A.J.C. van Gemund	Technische Universiteit Delft
Prof.dr. M. Livny	University of Wisconsin-Madison, USA
Prof.dr.ing. N. Ţăpuş	Universitatea Politehnică Bucureşti, Romania
Prof.dr. A. van Deursen	Technische Universiteit Delft, *reservelid*



This work was carried out in the ASCI graduate school.
ASCI dissertation series number **169**.

This work was performed in the context of the Virtual Laboratory for e-Science project (`www.vl-e.nl`), which is supported by a BSIK grant from the Dutch Ministry of Education, Culture and Science (OC&W), and which is part of the ICT innovation program of the Dutch Ministry of Economic Affairs (EZ). The FP6 Network of Excellence CoreGRID funded by the European Commission (Contract IST-2002-004265, the CoreGRID European Virtual Institute on Grid Scheduling) sponsored several research visits. The PDS Group provided hosting and equipment.

ISBN/EAN: 978-90-9023297-3

To my family and friends, with love.

Acknowledgments

There are many people to whom I should thank for their influence on the outcome of my PhD work, and ultimately on this thesis. Despite space restrictions, I would like to mention briefly some names, more often than not in the chronological order of our encounters. If your name has been left out, please accept my non-nominative thanks.

I have finished my B.Sc. and my M.Sc. theses with the Politehnica University of Bucharest (UPB), Romania. During these "Romanian days", my academic and sometimes my personal steps have been influenced by long discussions with Nicolae Ţăpuş and the now late Irina Athanasiu, to whom I am in debt. I have done the projects for both the B.Sc. and the M.Sc. theses with Stephane Vialle, from Supelec, Metz, France. Steph has guided my first steps into the research world, and was co-author to my first refereed publication. It was not a good paper, and the venue was not top, but it gave me hope that I can start (and finish) a PhD track. Steph, thank you for all the help!

I have then joined the Parallel and Distributed Systems group led by Henk Sips in 2004. I would like to thank Henk for being always present when I needed advice. The person who has helped me the most during my PhD years (2004–2008) is my research supervisor, Dick Epema. My first meeting with Dick was during a long PhD interview. I managed to tell Dick then that grid computing was a research area good only for republishing old research results, and that scheduling has no future, research-wise. An interesting discussion ensued, and, as it turns out, my PhD thesis is mainly about scheduling in grid computing. Dick, I was wrong, though my harsh statements can be matched with many examples from the published grid computing research of the past decade. I have learnt a great deal about the professional side and even a little bit about the personal side of Dick Epema in these four years. He gave me enough space to enjoy doing my research, encouraged me whenever he felt I was feeling low, and prevented me as much as possible from over-reaching. He contributed to my working environment being pleasant yet challenging. Most importantly, he helped a great deal in making our joint work really successful. Thank you very much for all the help, Dick Epema!

In the summer of 2005, I started working on a grid performance evaluation tool, which later evolved into the GrenchMark framework (Chapter 5). In the early stages of this work, I have benefitted from many insightful discussion with Hashim Mohamed, my room-mate for four years, to whom I thank for both. I was also helped for this work by the Ibis grid programming toolkit creators, Jason Maassen and Rob van Nieuwpoort. Late in 2005, I made a GrenchMark-related research visit with the team of Uwe Schwiegelshohn at the University of Dortmund. The ensuing discussions with Ramin Yahyapour, Carsten Ernemann, Alexander Papspyrou, and the rest of the group, led to a better understanding of the current needs for a grid performance evaluation framework.

The year of 2006 was my year of scientific exploration, and I am deeply in debt to all my collaborators during this period, as they helped shaping my research skills. Work in computer resource management is always dependent on realistic or real workloads, and for grids, which are some of the largest and most complicated computing systems to date, even more so. During the early months of 2006, Catalin Dumitrescu and Hui Li joined me in what resulted in the creation of the Grid Workloads Archive (Chapter 3). At almost the same time, I have also started to work with Javier Bustos-Jimenez on understanding the characteristics of volatile grid environments (those environments set-up temporary, for a multi-institute project). Meanwhile, I have worked on Peer-to-Peer systems for much of the first half of 2006 with Paweł Garbacki and with Johan Pouwelse. While not documented in this thesis, the work was fruitful, and I would like to thank them both for their inspiring attitude towards hands-on research, and towards work that benefits millions of users. In the summer of 2006, I was a visiting researcher with Miron Livny's group at University of Madison-Wisconsin. I thank Miron for accepting to work with me in the first place; he's one of the busiest people I know. I also want to thank the Condor team members, and especially Zach Miller, Matt Farrellee, and Tim Cartwright, for welcoming and working with me without reserves. Besides work, Zach and Matt kindly showed me around Madison, and Tim taught me Go. Condor rocks, no kidding.

During the remaining two years of my PhD project duration I had the pleasure to collaborate with a wonderful group of people. In particular, I would like to thank Radu Prodan and Thomas Fahringer for their help, including facilitating my research visit to U. Innsbruck in the summer of 2008. I would also like to thank all my other collaborators for their help, not forgetting Hashim Mohamed, Ozan Sonmez, Mathieu Jan, and Shanny Anoep (all TU Delft), Vlad Nae, Simon Ostermann, Stefan Podlipnig, and Kassian Plankensteiner (all U. Innsbruck), Corina Stratan, Mugurel Andreica, and Nicolae Ţăpuş (UPB), Ioan Raicu (U. Chicago), Javier Bustos-Jimenez and Jose Miguel Piquer (U. Chile), Attila Kertesz (Sztaki), and Fan Ye, Norman Bobroff, and Liana Fong (IBM T.J. Watson).

This thesis would not have been finalized without the help of its evaluation committee: Henri Bal, Arie van Deursen, Dick Epema, Arjan van Gemund, Thomas Fahringer, Henk Sips, Miron Livny, and Nicolae Ţăpuş. Thank you very much for taking the time to evaluate this long PhD thesis.

I am in debt for the ever-encouraging (e-)presence of my friends and colleagues: Alex (Tzu), Alin, Anca, Cătălin, Claudia, Costin (Coco), Cristian, Daniel (Avocatul), Diodor (Biţi), Dragos (Junior), Erwin, Flavius, Hashim, Jan David, Jie, Johan, Laura, Laurenţiu, Matei, Mihai (Mo), Monica (C), Oana, Ozan, Paweł, Raluca (amândouă), Rameez, Răzvan (Bau), Remus, Sorana, Valentin (TTK), Vlad, Wouter, ... space prevents the due expression of gratitude. My basketball team-mates have also been there at all times, since I met them tri-weekly in training and in (low-level) Dutch league games, and oftentimes in international basketball tournaments. Fellow Punchers, and Alexandra, Dries, Henkie, Joao, Nick, Patrice, and Rita, I have enjoyed enormously all of these.

Last, but certainly not least, I would not be here to tell this long story without constant support from my family. I am grateful to my mother and to my father, also for giving me the freedom to fly on my own wings at an early age. I am happy to have a lively and "oh-so-active" sister, Andrea, from whom I had a lot to learn, though she is only two years older. Ana, my beloved wife, I have often been a burden, and I got in return only love and affection (and excellent food, mmmmmm). I love you very much, Ana, Ana-Maria, Andrea, Andrei, and Olga!

Alexandru Iosup

Contents

Chapter 1

Raising the Curtains

We use daily many wide-scale distributed services, perhaps unknowingly: we carry telephone discussions in the tram and on the street, we drink tap water at home and at the office, we use electricity in a conference room and (more recently) to recharge our electric cars. Given that our daily activity rely on these services, it is surprising how little attention we give to the service provider. As long as the service is cheap, reliable, and comfortable to use, we don't even know who the service provider is, and perhaps rightly so. In a competitive market, the service providers tend to integrate into alliances that provide better service at lower cost. For example, while hundreds of airlines or various sizes exist (e.g., Air France, KLM, US Airways, Tarom), three airline alliances integrating around ten percent of these airlines control two-thirds of the global air passenger traffic. By using the services of an airline alliance we are able to obtain in a matter of minutes and without the services of a travel operator a single ticket, even for a complex request involving multiple flights and several intermediate locations. Other day-to-day utilities, such as the telephone, water, and electricity, are similarly integrated and operated. However, the computer, which is a newer daily utility, still lacks such integration.

Computers are becoming more and more important for the well-being and the evolution of society. Over the past four decades, computers have permeated every aspect of our society, greatly contributing to the productivity growth[1] [Oli00; Jor02; Pil02; Gom06]. The impact of computer-based information technology (IT) is especially important in the services industry and in research [Jor02; Ber03a; Tri06; i2006], arguably the most important pillars of the current U.S. and European economies [i2006]. The inherent nature of the services market makes productivity improvements less likely than in the goods-producing sectors (the so-called "Baumol disease" [Tri06]): it is difficult to improve the economic output of the human part of the services market; computers are a way to automate the services market in the same way robots can automate the production of goods. To increase the innovation rate, research is evolving towards large-scale, grand-challenge science, carried out through distributed global collaborations enabled by the Internet, requiring access to very large-scale computing, network, and storage resources (the so-called "e-Science" [New03a; Hey05]). Thus, the integration of computers as an utility is a natural trend that benefits the whole society.

Coupled with the adoption of computers, the growth of the Internet over the last decade has enabled millions of users to access information anytime and anywhere, and has transformed information sharing into a utility like any other. However, an important category of users remained under-served: the users with large computational and storage requirements, e.g., the scientists, the companies that focus on data analysis, and the governmental departments that manage the interaction between the state and the population (such as census, tax, and public health). Thus, in the mid-nineties, the vision of the Grid as a universal computing utility was formulated [Fos98]. The main benefits promised by the Grid are similar to those of other integration efforts: extended and optimized service of the integrated network, and significant reductions of maintenance and operation costs through sharing and better

[1] Labor productivity (defined as output per hour) is one of the two factors in a product that characterizes the economic power of a country, the other being the size of the working population.

scheduling. While the universal Grid has yet to be developed, large-scale distributed computing infrastructures that provide their users with seamless and secured access to computing resources, individually called Grid parts or *grids*, have been built throughout the world, e.g., the DAS [Bal00] in the Netherlands, the e-Science Grid [Hey02] in the U.K., NorduGrid [Eer03] in the Nordic countries, OurGrid [And03] in Brazil, the Europe-based Enabling Grids for E-Science in Europe (EGEE) [Kra05], the U.S.-based Open Science Grid (OSG) [07a], Grid'5000 [Bol06] in France.

The subject of this thesis is the inter-operation of grids, a necessary step towards building *the* Grid. Grid inter-operation raises numerous challenges that are usually not addressed in the existing grids, e.g., efficiently managing workloads with volumes many times over the capacity of each individual grid, coordinated operation without the benefit of centralized control, and reliability at a truly large scale. We review these challenges as part of the problem of grid inter-operation in Section 1.1. New research challenges arise from the number and variety of existing grids, for example the lack of knowledge about the characteristics of grid workloads and resources, or the lack of tools for studying real and simulated grids. We present these challenges in Section 1.2. In Section 1.3 we introduce our approach for the problem of grid inter-operation: a framework for the study of grid inter-operation mechanisms. In Section 1.4 we summarize the contributions of this thesis to grid research, and present the outline of the thesis.

1.1 The Problem of Grid Inter-Operation

Since the formulation of the Grid vision [Fos98], hundreds of grids have been built in different countries, for different sciences, and both for production work and for proof-of-concept purposes (for computer science research). However, the vast majority of these grids work in isolation, running counter to the very nature of grids. Two research questions arise:

1. *How to inter-operate grids?* and
2. *What is the possible gain of inter-operating grids?*

Answering both questions is key to the vision of the Grid; we call these question *the problem of grid inter-operation*. Without answering the first question, the Grid cannot exist. Without answering the second, the main technological alternatives to grids, large clusters and supercomputers, will remain the choice of industrial parties.

We identify five main challenges for grid inter-operation (they are presented in more detail in Section 7.4.1, and addressed in Chapters 7 and 8):

Resource Selection The inter-operated grid must be able to efficiently select the resources for executing the joint workload of all the individual grids.

Resource Ownership The grids must be inter-operated without interfering with the ownership and the fair sharing of resources.

Scalability The inter-operated grid must be scalable with respect to the number of users, jobs, and resources.

Trust and Accounting The resource sharing in the inter-operated grid must be accountable and should involve only trusted parties.

Reliability The inter-operated grid must attempt to mask the failure of any of its components.

The answer to the first question is finding an architecture and its operation mechanism that can address these five challenges. Currently, there is no common solution to this problem. If there is no common resource management system, jobs must be specifically submitted to one of the grids, leading to inefficient resource use. A central meta-scheduler is a performance bottleneck and a single point of failure, and leads to administrative issues in selecting the entity that will physically manage the centralized scheduler. Hierarchical mechanisms (still centralized, but arguably with less demand per hierarchy node, thus more scalable) can be efficient and solve the trust and accounting issues, but still have single points of failure, and are administratively impractical in most situations. Completely decentralized systems can be scalable and fault-tolerant, but their efficiency has yet to be proven for computational workloads.

The second question can be answered by comparing the inter-operated grid with its alternatives with respect to each of the five challenges presented above. A qualitative comparison is possible between the centralized architecture, which is the most-used architecture for large-scale cluster and supercomputer systems, and the architectures used for building grid environments. However, a quantitative comparison raises additional difficulties: the lack of knowledge about the characteristics of grid workloads and resources, and the lack of tools for studying real and simulated grids. We discuss these difficulties in the next section.

1.2 How to Study Grid Inter-Operation?

In the previous section we have identified a lack of knowledge about the characteristics of grid workloads and resources, and the lack of tools for studying real and simulated grids; we call this by language abuse a "lack of research tools". This lack limits our ability to address three of the five main challenges that need to be addressed to answer the first research question (i.e., resource selection, scalability, and reliability), and denies us the possibility of answering the second research question. Thus, we formulate in this section a third research question:

3. *How to study grid inter-operation?*

We identify two main challenges in answering this question (they are addressed in Chapters 3 and 4, and in Chapters 5 and 6, respectively): lack of knowledge about real grids, and lack of test and performance tools.

Little is known about the behavior of grids, that is, we do not yet understand the characteristics of the grid resources and workloads. Mostly because of access permissions, no grid workload traces are available to the community that needs them. There are no workload models that include important grid application types, such as bag-of-tasks (BoTs) and workflows (WFs). Without the understanding and the modeling of real workloads, current research studies use synthetically generated workloads or workloads specific to other types of environments, thus being limited in scope and applicability. Similarly, we know little about the dynamic presence of grid resources: the resources may be added to or taken off from the grid environment at any time, resource failures will occur at the grid scale, etc.

From the more practical perspective, there exists no testing or performance evaluation tool that is generally accepted and used by the community; as a result, it is difficult to perform, and in particular to compare, realistic and repeatable experiments in real grid environments, and to draw strong conclusions from real experiments. Moreover, the simulation of grid environments is also hampered, as the several grid simulation packages that are available lack many of the needed features for large-scale simulations of inter-operated grids.

1.3 A Framework for the Study of Grid Inter-Operation Mechanisms

To address the problem of grid inter-operation we introduce in this work a framework for the study of grid inter-operation mechanisms. The framework comprises two main components: a toolbox for grid inter-operation research, and a method for the study of grid inter-operation mechanisms. We describe these two components in turn.

1.3.1 A Toolbox for Grid Inter-Operation Research

The toolbox for grid inter-operation research presented in this thesis contains four research tools: the Grid Workloads Archive (a collection of workload traces taken from real grid environments), a comprehensive model for grid resources and workloads (with information about workload specifics such as BoTs and WFs, and about resource availability), GrenchMark (a framework for testing real grid settings), and the Delft Grid Simulator (a framework for (multi-)grid simulation). We describe each of the four tools in turn.

Over the past two years, we have built the Grid Workloads Archive (GWA), which is at the same time a workload data exchange and a meeting point for the grid community. We have introduced a format for sharing grid workload information, and tools associated with this format. Using these tools, we have collected and analyzed data from nine well-known grid environments, with a total content of more than 2,000 users submitting more than 7 million jobs over a period of over 13 operational years, and with working environments spanning over 130 sites and comprising over 10,000 resources. The GWA (both content and tools) has already been used in grid research studies and in practical areas: in grid resource management [Li07c; Li07e; Li07f; Ios06a; Ios07d], in grid design [Ios07c], in grid operation [Ios06b], and in grid maintenance [Ios07e; Str07].

"What should be characterized and modeled in a system?" is a non-trivial question even for systems simpler than grids, e.g., parallel production environments [Dow99a]. We find that for grids, focus should be given to the resource change over time, and to workloads. We propose in our work a model for grid inter-operation that focuses on these two aspects. The resource change model characterizes both the short-term dynamics and the long-term evolution. The workload model takes into account individual jobs, job grouping (e.g., bags-of-tasks, workflows), and the user that submits the job. We find that the grid workloads are very different from the workloads of other widely used systems, such as parallel production environments and the Internet.

The GrenchMark framework for testing large-scale distributed computing environments focuses on realistic testing, and on obtaining comparable testing results across platforms. To achieve this goal, we give special attention to realistic grid workload modeling, and to generating workloads that can run in real environments. Our GrenchMark reference implementation addresses the practical problems of testing, and, in particular, of creating appropriate and repeatable experimental conditions in a large variety of environments. Over the past 18 months, we have used the reference implementation in over 25 testing scenarios in grids, in peer-to-peer systems, and in heterogeneous computing environments.

The current grid simulation environments still lack modeling features such as grid inter-operation, grid dynamics, and grid evolution, and research productivity features such as automated experiment setup and management. We address these issues through the design and a reference implementation of DGSim, a framework for simulating grid resource management architectures.

1.3.2 A Method for the Study of Grid Inter-operation Mechanisms

Our method for the study of grid inter-operation mechanisms is based on the belief that a grid inter-operation system must be either a direct application of the existing systems, or an extension thereof. Thus, we propose a three-step method:

1. Identify relevant aspects for the design of grid inter-operation systems, which have to be validated by classifying real grid systems according to the relevant aspects;
2. Assess qualitatively the grid inter-operation ability of the real grid systems according to the relevant aspects;
3. Design new grid inter-operation systems by combining and/or extending existing designs along the relevant aspects, and validate the new system through comparisons with existing systems.

In the first step we identify two main design aspects: the architecture of the inter-operated grid, and the mechanism for operating the architecture. We identify four main architectural and three operational classes; all the already existing grid resource management systems can be classified (and therefore studied more efficiently) using these classes.

In the second step, we perform a qualitative comparison of the existing architectures, and find that none is suitable for grid inter-operation.

Thus, in the third step we design a novel grid inter-operation system (architecture and mechanism), and we compare it with five alternatives. In our solution, the internal hierarchy of the grids is augmented with direct connections between nodes under the same administrative control, and the roots of the hierarchies are combined in a decentralized network. To operate this architecture, we employ the key concept of delegated matchmaking, in which resources that are unavailable locally are obtained through (delegated) matchmaking from remote sites, and added transparently and temporarily to the local environment. The comparison with five grid inter-operation alternatives uses extensively the toolbox for grid inter-operation research, and shows that our novel grid inter-operation system has good potential.

1.4 An Overview of the Thesis

We now present an overview of this thesis: a summary of the main contributions of this thesis, followed by an outline of the thesis.

1.4.1 Thesis Contributions

The major contribution of this thesis is three-fold:

1. *We formulate the problem of grid inter-operation.* We present a rationale and a formulation for this important grid research problem.
2. *We propose a framework for the study of grid inter-operation mechanisms.* Our framework comprises two main components: a toolbox for grid inter-operation research, and a method for the study of grid inter-operation mechanisms.
   (a) *We create a collection of grid workload traces.*

(b) *We create and validate a model for grid inter-operation.*

(c) *We design, implement, and use a tool for testing real large-scale distributed computing environments.*

(d) *We design, implement, and use a tool for the trace-based simulation of inter-operated grids.*

(e) *We propose a taxonomy of grid inter-operation systems.* The taxonomy focuses on two aspects: the architecture of the inter-operated grid, and the mechanism for operating the architecture. We show how this taxonomy can be used to compare existing systems, and as a guideline for the design of new systems.

3. *We design and validate a new grid inter-operation system.* Our solution, *Delegated MatchMaking*, includes a hybrid grid inter-operation architecture and a new grid inter-operation mechanism.

1.4.2 Thesis Outline

The thesis is split into four logical parts: the introduction, a toolbox for grid inter-operation research, a method for grid inter-operation, and the conclusion. The chapters composing each part and the logical links between them are depicted in Figure 1.1. The remainder of the thesis is structured as follows:

- Chapter 2 introduces a basic model for grid inter-operation. It defines the components of a grid system (i.e., processors, clusters, cluster and grid resource managers, gateways, and sites), the types of applications that can be found in a grid (e.g., unitary and composite, bags-of-tasks, workflows), the system users, and the job execution model. This chapter is based on material published in [Ios06b; Ios07c; Ios08e; Ios08g].

- Chapter 3 introduces the Grid Workloads Archive. The requirements of a grid workloads archive are analyzed. The design and the main features of the GWA, and its past, current, and potential use are presented. Finally, the workload archives used in computer science are surveyed and compared with the GWA. This chapter is based on material published in [Ios06a; Ios08d].

- Chapter 4 extends the basic model for grid environments into a comprehensive model for (multi-)grids. The chapter presents analysis results and models for grid resource dynamics and evolution, and for grid workloads focusing on parallel jobs and on bags-of-tasks. Finally, the analysis and modeling efforts for the resources and for the workloads of large-scale distributed (computing) systems are surveyed and compared with our modeling approach. This chapter is based on material published in [Ios06a; Ios07e; Ios07d; Ios08e].

- Chapter 5 introduces the GrenchMark testing framework. The design goals of a testing framework for large-scale distributed computing systems are discussed. The resulting design and the main features of the GrenchMark framework, its key implementation details, its validation, and its past, current, and potential use are presented. Finally, existing testing approaches for distributed computing systems are surveyed and compared with the GrenchMark framework. This chapter is based on material published in [Ios06b; Ios07a].

- Chapter 6 introduces the DGSim grid simulation framework. The requirements of a simulation framework for comparing grid resource management architectures are discussed. The design and the main features of the DGSim framework, its validation, and its past, current, and potential use are presented. Finally, existing simulation approaches for distributed computing systems

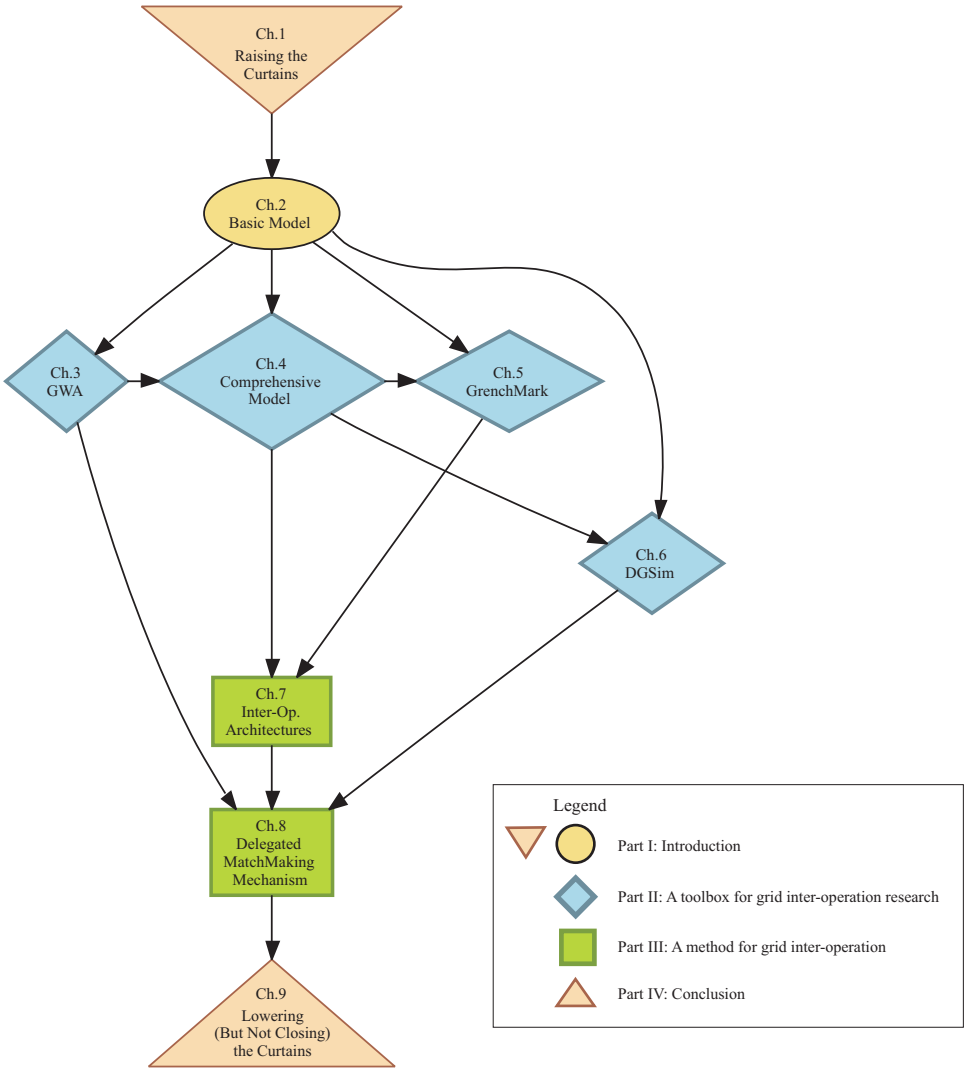


Figure 1.1: *The chapters of this dissertation and their inter-dependencies.*

are surveyed and compared with the DGSim framework. This chapter is based on material published in [Ios08f].

- Chapter 7 introduces a hybrid architecture for grid inter-operation. The relevant aspects for the design of grid inter-operation systems are identified and validated by classifying real grid systems according to the relevant aspects. The practical limitations of the centralized grid inter-operation approaches are evaluated in a real environment. The grid inter-operation ability of real grid systems according to the relevant aspects is assessed. A novel architecture for grid inter-operation is then introduced, which has the best potential of fulfilling the requirements of grid inter-operation. This chapter is based on material published in [Ios07c; Ios08b; Ios08g].

- Chapter 8 introduces a novel approach for grid inter-operation, Delegated MatchMaking. The approach, which couples the architecture introduced in the previous chapter with a novel inter-operation mechanism, is compared with five alternatives through trace-based simulations, and

is found to deliver the best performance. This chapter is based on material published in [Ios07c; Ios08b].

- Chapter 9 summarizes the thesis, presents our conclusions, and indicates several research directions stemming from this thesis.

Chapter 2

A Basic Grid Model

2.1 Overview

In this chapter* we present a basic Grid model.

2.1.1 Motivation and Problem Statement

Every research study of a system is based on a model of that system. Similarly, this thesis relies on the existence of a Grid system model. The problem addressed is finding a realistic model of the Grid environment that can be used in a variety of scenarios.

2.1.2 Key Ideas and Selected Results

The study of scheduling in distributed computing systems has a long history [Wan85], and many system models that have been proposed within this area were extended for various grid research scenarios [Buy02; Mar05]. In contrast with these approaches, our key idea in building a Grid model is the design of a modular model, so as to accommodate a wide variety of scenarios (instead of just one). We therefore split our Grid model into two parts: a basic part that deals with the common aspects of the Grid (such as the system, the jobs, the users, and the job execution), and an extension part that includes modules for various more realistic aspects (e.g., grid workloads). The basic part deals with an abstract model that can be used in a variety of scenarios, but which lacks the details needed for specific cases, such as performance evaluation for realistic system workloads. The extension part deals with the specific aspects that are relevant for our grid inter-operation research.

2.1.3 Organization of this chapter

We introduce in this chapter only the basic part of the Grid model used in this thesis, and the extension part in Chapter 4. The system, job, user, and job execution parts of the model are introduced in Sections 2.2, 2.3, 2.4, and 2.5, respectively. Last, in Section 2.6 we present the example of the DAS grid.

2.2 System Model

In the system model we assume that grid resources are clusters of computing resources (processors); thus, we model a multi-cluster grid. The clusters are provided and maintained by individuals or

*This chapter is based on previous work published in the IEEE/ACM Int'l. Symp. on Cluster Computing and the Grid (CCGrid'06) [Ios06b], the ACM/IEEE Conference on High Performance Networking and Computing (SC'07) [Ios07c], the IEEE Int'l. Symp. on High Performance Distributed Computing (HPDC'08) [Ios08e], and the IEEE/ACM Int'l. Conference on Grid Computing (Grid 2008) [Ios08g].

institutions (resource owners), and are not necessarily dedicated to grid usage. The processors may have different performance across clusters, but within the same cluster they are homogeneous. We employ the SPEC CPU benchmarks model for application execution time [SPE00; SPE08], that is, the time it takes to finish a task is inversely proportional to the performance of the processor it runs on.

Each cluster is managed by a Cluster Resource Manager (CRM), a middleware layer that provides an interface through which jobs are submitted for execution on the local resources. The CRM focuses on the management of a single set of resources, and is usually not capable of managing several distinct clusters. Some of the most-used CRMs today are Condor [Tha05a; Tha05b], Maui [Jac01], the Sun Grid Engine [Gen01], the Load Sharing Facility (LSF), and the Portable Batch System (PBS) [Hum06]. On top of the CRM level (i.e., on top of one or more CRMs) operates the Grid Resource Manager (GRM), a middleware layer that provides an interface through which jobs are submitted for execution on any of the CRMs' resources. Commonly-used GRMs are the Globus Toolkit [Cza98], Unicore [Erw02], Condor through the Condor-G [Fre01] interface, Nimrod-G [Abr02], etc.

A site is an administrative unit that combines all the elements with a common location, i.e., the physical resources grouped in a cluster, the CRM operating the cluster, and the GRM part that operates locally. We call *gateway* a machine used as an entry point to a site, to which jobs can be submitted and which serves as a file server (i.e., files can be transferred to and from the cluster through the gateway). On each site, there is only one *gateway*, although the machine can be a multi-processor system more powerful than any other node in the cluster. The computing power of the gateway cannot be used for executing user jobs. For administrative purposes, a site may exist even when it lacks physical resources.

To submit to a site jobs, the user employs a *user job manager* that has a server part operating on the site *gateway* and a client part operating on the user's machine (i.e., desktop or laptop). After submitting the jobs, the client may be disconnected from the server, and the server remains in the system to further represent the user and manage the received jobs for execution. For this reason, and by terms abuse, when using the terms "user job manager" we refer only to the server part of the user job manager.

Last, we model the Grid as a collection of sites coupled through an opaque inter-operation medium, that is, through an inter-operation medium with unspecified properties allowing the exchange of messages between the Grid sites. To emphasize that the properties of the inter-operation medium (such as the architecture or the message routing mechanism) are not specified, we call this model the generic Grid model, or the generic grid inter-operation model. Figure 2.1 depicts an example of our generic Grid model. In this specific example, from the sites that serve users directly, the ratio between the number of physical resources and the number of served users is the highest for Site-2 (100 users for 500 resources). In the figure, two sites do not have resources (i.e., Site-3 and Site-k), and three sites do not have any users (i.e., Site-1, Site-4, and Site-N). None of these sites would be able to function in the absence of the inter-operation medium. We detail the components of the inter-operation medium, i.e., the architecture and the operation mechanism, in Chapters 7 and 8, respectively.

2.3 Job Model

We now turn our attention to the job model.

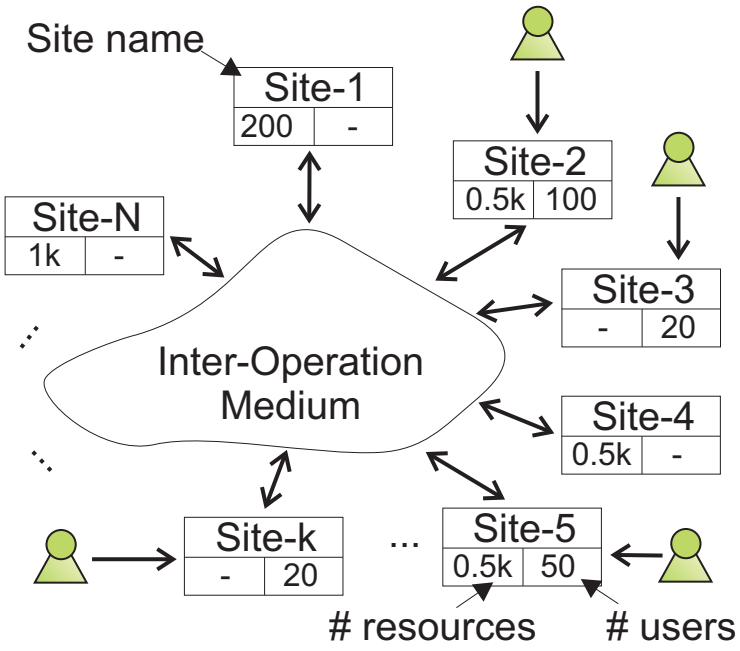


Figure 2.1: *The generic grid inter-operation model.*

2.3.1 Job Types

We first identify two types of jobs that are run in grids.

Unitary jobs This category includes jobs that can be submitted as a single, unitary, description to the typical scheduler, e.g., the CRM. Typical examples include sequential and parallel jobs, using as the messaging library an MPI [Kar03; Sur06] implementation or Ibis [Nie05; Wrz05]. For a unitary job, the job manager only has to deal with the job's programming model, including the assembly of the list of the machines the job is executed on for an MPI job, or adding the location of the nameserver to the parameters of an Ibis job.

Composite jobs This category includes jobs composed of several unitary jobs. To distinguish between the composite job and its components, we call the latter tasks. Figure 2.2 depicts several composite job structures: bags-of-tasks, chains of tasks, and workflows. A bag-of-tasks job (Figure 2.2 a) comprises several, possibly unrelated, tasks. A chain-of-tasks job (Figure 2.2 b) comprises several tasks linked through a dependency chain: each task depends on the previous task in the chain. The dependency between two tasks determines when the dependent task can start. Workflow jobs (Figure 2.2 c and d) are jobs with a directed acyclic graph structure. For a composite job, the job manager needs to take into account issues like task inter-dependencies, advanced reservation and extended fault-tolerance, besides the job tasks' programming model.

2.3.2 A Generic Grid Job Model

Based on the observations that grid workflows are the most generic composite job model, and that a unitary job can be seen as a composite job with only one task, we now present a generic grid job

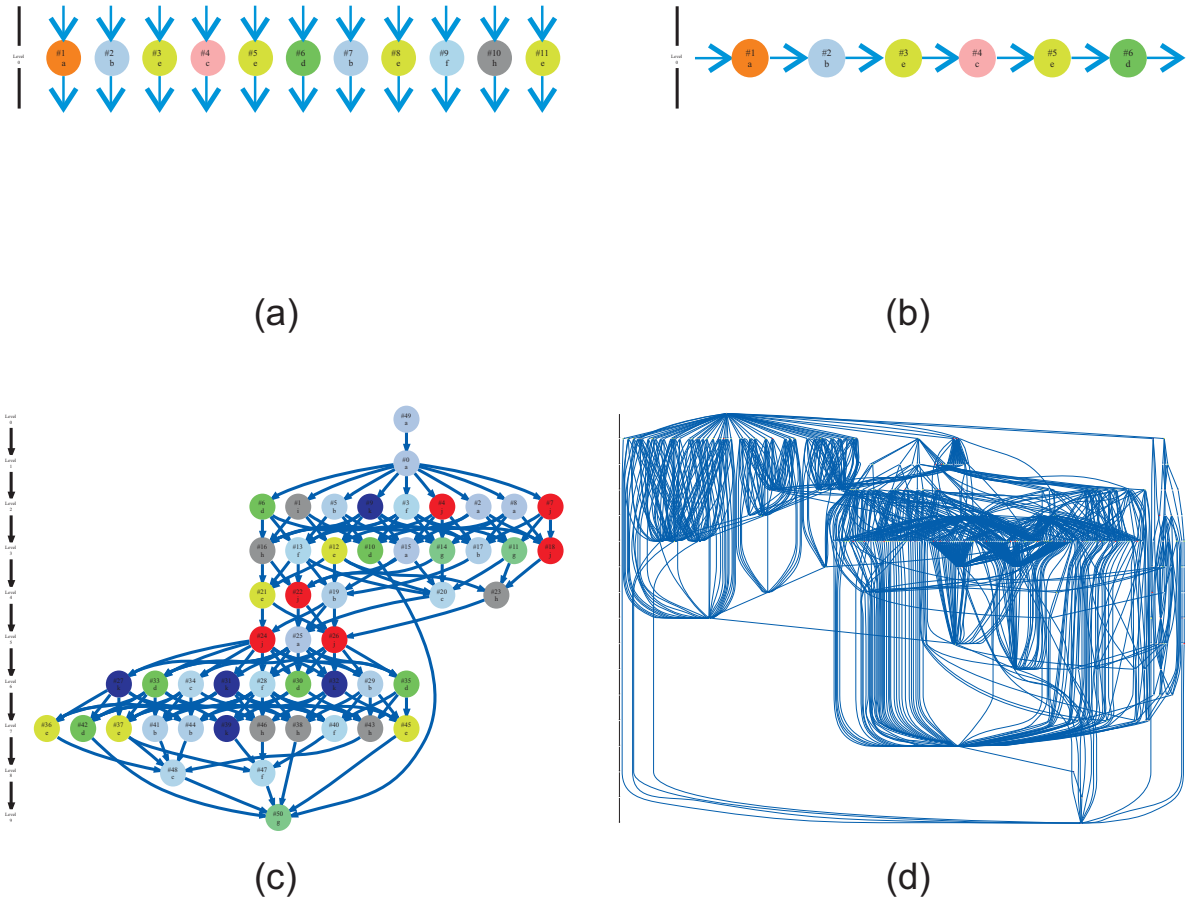


Figure 2.2: *Examples of composite applications: (a) a* bag-of-tasks*; (b) a* chain of tasks*; (c) a simple* workflow*; (d) a complex* workflow*.*

model. We use for this the model introduced by Coffman and Graham [Cof72]. A workflow of tasks is represented as a directed graph G in which the nodes are computational tasks, and the directed edges represent communication; we denote the number of nodes and the number of edges by N and E, respectively. For simplicity, we consider only directed acyclic graphs (*DAGs*). A computational task can start only when all its predecessors, both computation and communication tasks, are finished.

The following terminology and definitions are borrowed from [Cof72; Cor89]. We call *root* a node without predecessors and *leaf* a node without descendents; a DAG may have several roots and leaves. We further define a node's *level*, derived from breadth-first traversal of the task graph from its roots, as the minimal size of a path from a root to this node (in number of edges); the level of a root is 0. Finally, we call the maximum of the level of the leaves as the *graph level*, which we denote by H; we

define the *graph traversal height* as $L = H + 1$. A chain-of-tasks job may be seen as a degenerate workflow with L equal to the number of tasks. A master-worker job can be seen as a workflow with $L = 3$, which includes a task for the initial splitting and a task for the results gathering and post-processing; when these tasks are not considered the master-worker job becomes a bag-of-tasks and can be seen as a workflow with $L = 1$ and no task inter-dependencies. We further define the *branching factor* of a workflow as the ratio between its number of edges E and its number of nodes N. The branching factor determines the internal complexity of a composite job; we have shown in previous work that the value of the branching factor is directly proportional to the performance of the grid workflow engine that schedules it [Ost08].

2.4 User Model

In this section we present the user component of our basic Grid model.

2.4.1 Users

A user is in our model any source of jobs. Often, the grid resource owner is tightly connected (or even identical) to a community of users, whose jobs it must prioritize over the jobs of any other user. Thus, for each resource owner there are two classes of users: the users that are tightly connected with that resource owner (the *local* users) and the other users (the *remote* users).

There are many policies that can be employed by the resource owners to prioritize the jobs of the local users over the jobs of the remote users, for instance a cluster may dedicate 100% of its resources for running its local users' jobs when even one such job is submitted. At the other extreme, resource owners may choose to offer all their resources at any time to anybody belonging to the global grid community without discriminating between local and remote users.

Another common policy of the resource owners with respect to users is to limit the number of jobs they can simultaneously run on the resources of a single cluster, or even on the whole grid.

2.4.2 Virtual Organizations

A Virtual Organization (VO) is a group of individuals or institutions who share the computing resources of a Grid for a common goal [Fos98]. Often, a VO will submit all its jobs through the same job manager, e.g., because the VO has hired a single technician to handle all its dealing with the Grid or to simplify the accounting of the resource owners. In our model we assume that each VO can be mapped to a single user (the VO's representative user). Thus, the VOs share with their representative user the limitations imposed by the resource owners.

2.5 Job Execution Model

In this section we describe the job execution model.

After preparing the job input, the user submits the jobs to the user job manager. From the moment the user job manager acknowledges to the user that the jobs have been received, the jobs are considered to have been submitted to the Grid. First, the jobs are queued in the user job manager's queue, waiting to be submitted to the GRM. This allows the user job manager to shape the workload that effectively reaches the GRMs, for example by not allowing more than 100 jobs to be submitted to the GRM or in execution on grid resources concurrently. The job is then submitted by the user

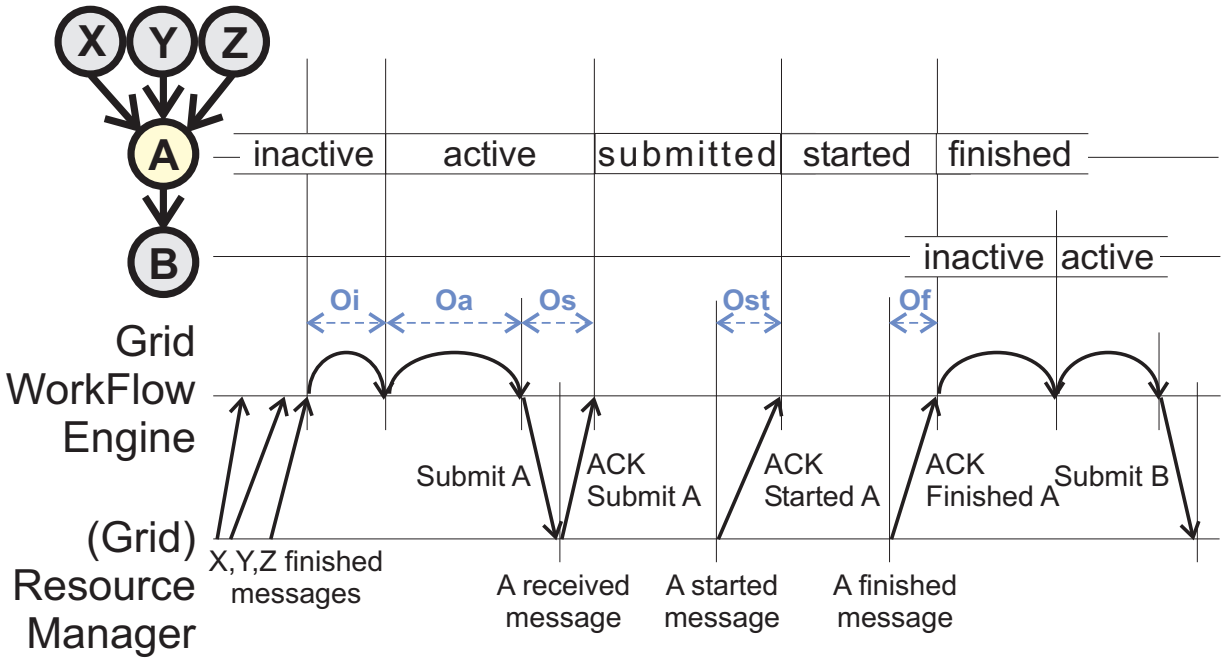


Figure 2.3: *The execution of a workflow task, A. Boxes represent states, and continuous arrows represent information flows. The overhead components, i.e., the O_*, are also depicted.*

job manager to a GRM. However, this does not mean that the job must be executed on the CRM operating directly under that GRM; when many GRMs operate in the same Grid the job may be routed (migrated before start) between them. Eventually, the jobs are submitted to the CRM of a selected cluster, where they are again queued, this time waiting for available resources on which to be executed, according to the resource owner's policies. Once started, tasks run to completion, so we do not consider task preemption or task migration during execution. Instead, tasks can be replicated and canceled, or migrated before they start.

The step in the job execution model when the job moves to a GRM from the user job manager deserves more attention, as it involves much of the overhead associated with running jobs in grids (we show in Section 7.3.3 that this overhead reaches the order of minutes in normal conditions). Figure 2.3 depicts a workflow with five tasks (A, B, X, Y, and Z), with an emphasis on task A's lifetime under a hypothetical execution process. Initially, A is "inactive" (it cannot be submitted for execution due to dependencies on other tasks that have not yet finished). Then, its predecessor tasks (i.e., tasks X, Y, and Z) finish executing, and after some time needed for task post-processing and internal GRM state updates, the user job manager changes A's state to "active", and starts task A's submission process. The task remains "active" until the grid resource manager acknowledges the submission, after which A becomes "submitted". The task waits in the queue, then starts executing, and after some delay due to messaging, A's state becomes "started". Finally, the task finishes executing and after some more messaging delay, A's state becomes "finished". From that point, the user job manager can begin executing task B. We define five overhead components: O_i, O_a, O_s, O_{st}, and O_f, which express the delays of the inactive to active transition, of the time elapsed between a task becomes active and the moment it is submitted to the grid resource manager, of the messaging needed to obtain

Table 2.1: *Characteristics of the DAS system deployments.*

	System (multi-cluster grid)		
	DAS-1	DAS-2	DAS-3
Community			
Users	200+	300+	300+
Components			
Cities	4	5	4
Clusters	4	5	5
CRM	PBS	PBS, later SGE	SGE
GRM	-	Globus Toolkit	Globus Toolkit
Computation			
Number of cores	400	400	796
Processor type	200 MHz Pentium Pro	1 GHz Pentium 3	2.2+ GHz Opteron
Memory	256 MB	1 GB	4 GB
HDD, node	10 GB	40 GB	250-500 GB
HDD, gateway	-	-	2-10 TB
Operating system	BSD, later Linux	Red Hat Linux	Linux (several)
Network			
Type, inter-cluster	ATM (full-mesh)	SURFnet	Optical
Type, intra-cluster	Myrinet	Myrinet	Ethernet/Myrinet
Bandwidth, inter-cluster	6Mb/s	1 Gb/s	8 x 10 Gb/s

an acknowledged submission to a GRM, of the messaging needed to obtain a confirmation of the task's execution start, and of the messaging needed to perform the job clean-up and output transfer, respectively. The first four overheads manifest themselves when the system is overloaded and/or when the user job manager manages its state inefficiently. The fifth overhead, O_f, manifests itself when the tasks have significant amounts of data and/or when the user job manager's job clean-up and output transfer are inefficient.

2.6 An Example: The DAS grid

We conclude the basic model introduced in this chapter with an example: the Distributed ASCI Supercomputer (DAS) grid [Bal00]; we use the DAS system as part of a two-grid inter-operation in Chapter 8.

The DAS system was designed as a common computational machine used by the computer science and imaging groups of the Advanced School for Computing and Imaging (ASCI) research school [Cap07]. From the over 500 members of the ASCI school, around 350 of them (mostly computer scientists) are currently using the DAS. Three systems have been deployed so far, DAS-1 in 1997, DAS-2 in 2002, and DAS-3 in 2007 (built by ClusterVision). Table 2.1 summarizes their characteristics.

Viewed through our model, DAS-1 was a grid with four sites, each comprising exactly one cluster (there are no purely administrative sites). Each cluster uses the same CRM and GRM middleware. The DAS-1 system did not have any common-purpose grid middleware installed. Similarly, the user job manager is a simple set of command-line tools, and offers support only for a few unitary jobs, i.e., either single-processor or parallel (using MPI). Thus, the DAS-1 system was used mostly for research in programming models and systems software. The DAS-2 system was[1] a grid with five sites with a similar structure and purpose to the DAS-1 system. From 2006, DAS-2 added an administrative site that uses the KOALA [Moh05a; Moh05b] GRM to automatically balance load across clusters and to allow resource co-allocation, that is, the simultaneous allocation of resources located in different grid sites to unitary jobs that consist of multiple components [Buc03].

[1]The last cluster was removed from the DAS-2 grid in May 2008.

The DAS-3 system was built for a different purpose than the DAS-1 and the DAS-2 systems: to serve the research that will enable a computational infrastructure for the Dutch e-Science community (and its international collaborators) [Cap07]. The main difference between DAS-3 and the previous DAS systems is the amount of heterogeneity present in the system. For the first time, the DAS is heterogeneous in processor speed (2.2 GHz or more), processor structure (single- and multi-core), local disk space (250-500 GB), and local network (only Ethernet, or Ethernet and Myrinet). The DAS-3 system follows the site structure of the DAS-2 system: six sites (one purely administrative). The user job manager is still a set of command-line tools, but some composite jobs are now supported (e.g., bags-of-tasks).

2.7 Concluding Remarks

The study of any system requires an underlying system model. Our approach for modeling a complex system such as the Grid is to create a modular model with two main parts: a basic part that deals with the common aspects of the Grid, and an extension part that includes modules for various complex aspects. In this chapter we have presented the basic part of the Grid model used in this thesis, that is, the system, the jobs, the users, and the job execution.

Chapter 3

The Grid Workloads Archive

3.1 Overview

In this chapter* we present the Grid Workloads Archive (GWA), which is at the same time a grid workload data exchange and a meeting point for the grid community.

3.1.1 Motivation and Problem Statement

Very little is known about the real grid users' demand, in spite of the tools that monitor and log the state of these systems and traces of their workloads. Due to access permissions few grid workload traces are available to the community, which needs them. The lack of grid workload traces hampers both researchers and practitioners. Most research in grid resource management is based on unrealistic assumptions about the characteristics of the workloads. There are no grid workload models of important grid application types, such as bag-of-tasks (BoTs) and workflows (WFs). Thus, performance evaluation studies lack a realistic basis, and researchers often fail to focus on the specifics of grids. Most grid testing is in practice performed with unrealistic workloads, and as a result the middleware is optimized for the wrong use case, and often fails to deliver good service in real conditions [Ios06b; Kha06; Li06; Ios07b]. Thus, a fundamental questions remains unanswered: *How are real grids used?*

3.1.2 Key Ideas and Selected Results

To answer the question formulated in the previous section we have developed the Grid Workloads Archive (GWA). The goal of the GWA is to provide a virtual meeting place where the grid community can archive and exchange grid workload traces. Building a community fosters collaboration and quickens the permeation of ideas. Building a workloads archive makes the data available.

The GWA effort was initially motivated by the success of the Parallel Workloads Archive (PWA [Th07]), the current de-facto standard source of workload traces from parallel environments. We have also drawn inspiration from a number of archival approaches from other computer science disciplines, e.g., the Internet [Dan07; CAI07; Yeo06] and clusters-based systems [Sch07]. Extending established practice, we define the requirements for an archive of large-scale distributed system workload traces[1]. We design the GWA around building a grid workload data repository, and establishing a community center around the archived data. We further design a grid workload format for storing job-level information, and which allows extensions for higher-level information, e.g., composite jobs with bag-of-tasks structure (see Chapter 2). We develop a comprehensive set of tools for collecting, processing, and using

*This chapter is based on previous work published in the IEEE/ACM Int'l. Conference on Grid Computing (Grid 2006) [Ios06a], and in the Elsevier Future Generation Comp. Syst. [Ios08d].

[1]Throughout this thesis we use the terms "grid workload trace", "grid workload", and "grid trace" interchangeably.

grid workloads. We give special attention to non-expert users, and devise a mechanism for automated trace ranking and selection.

We have collected so far for the GWA traces from nine well-known grid environments, with a total content of more than 2000 users submitting more than 7 million jobs over a period of over 13 operational years, and with working environments spanning over 130 sites comprising 10,000 resources. Thus, we believe that the GWA already offers a good basis for the performance evaluation of grids and other large-scale distributed computing systems.

3.1.3 Organization of this chapter

The remainder of this chapter is structured as follows. In Section 3.2 we present the requirements of a grid workloads archive. Then, we describe the design and the main features of the Grid Workloads Archive in Section 3.3, and its past, current, and potential use in Section 3.4. Finally, in Section 3.5 we survey the workload archives used in computer science and compare our Grid Workloads Archive with the state-of-the-art.

3.2 Requirements of a Grid Workloads Archive

In this section we synthesize the requirements to build a grid workloads archive. Our motivation is twofold. First, grid workloads have specific archival requirements. Second, in spite of last decade's evolution of workload archives for scientific purposes (see Section 3.5), there is still place for improvement, especially with the recent evolution of collaborative environments such as Wikis.

We structure the requirements in two broad categories: requirements for building a grid workload data repository, and requirements for building a community center for scientists interested in the archived data.

Requirement 1: tools for collecting grid workloads. In many environments, obtaining workload data requires special acquisition techniques, i.e., reading hardware counters for computer traces, or capturing packets for Internet and other network traces. Obtaining grid workloads data is comparatively easy: most grid middleware log all job-related events. However, it is usually difficult to correlate information from several logs. This problem is starting to be solved by the use of unique job identifiers. Second, to keep the size of the logs small, fixed-size logs are used, and old data are archived or even removed. Third, due to political difficulties, parts of a data set may be obtained from several grid participants. Fourth, to provide uniformity, a workload archive provides a common format for data storage. The format must comprehensively cover current workload features, and also be extensible to accommodate future requirements. To conclude, there is a need for tools that can collect and combine data from multiple sources, and store it in a common grid workload format (*requirement 1*).

Requirement 2: tools for grid workload processing. Following the trend of Internet traces, sensitive information must not be disclosed. For grids, environment restrictions to data access are in place, so it is unlikely that truly sensitive data (e.g., application input) can be obtained or published. However, there still exists the need to anonymize any information that can lead to easily and uniquely identifying a machine, an application, or a user (*requirement 2a*).Time series analysis is the main technique to analyze workload data in computing environments. While many generic data analysis tools exist, they require specific configuration and policy selection, and input data selection and formatting. In addition, the data in the archive is often subjected to the same analysis: marginal distribution estimation and analysis, second and higher order moment analysis, statistical fitting, and

time-based load estimation. In addition, grids exhibit patterns of batch submission, and require that workload analysis is combined with monitoring information analysis. To assist in these operations, there is a need for grid-specific workload analysis tools (*requirement 2b*). The data donors and the non-expert users expect a user-friendly presentation of the workload analysis data. The GWA community needs tools that facilitate the addition of new grid workloads, including a web summary. Thus, there is a need for tools to create workload analysis reports (*requirement 2c*).

Requirement 3: tools for using grid workloads. The results of workload modeling research are often too complex for easy adoption. Even finding the parameter values for another data set may prove too much for the common user. By comparison to previous computing environments (e.g., clusters), grid models need to include additional (i.e., per-cluster, per-group) and more complex (e.g., batching) information. There is a need for tools to extract for a given data set the values of the parameters of common models (*requirement 3a*). The common user may also find difficult to generate traces based on a workload model. There is a need to generate synthetic workloads based on models representative for the data in the archive (*requirement 3b*). Since the grid workload format can become complex, there exists also a need for developer support (i.e., libraries for parsing and loading the data) (*requirement 3c*).

Requirement 4: tools for sharing grid workloads. Over time, the archive may grow to include tens to hundreds of traces. Even when few traces are present, the non-expert user faces the daunting task of trace selection for a specific purpose. There is a need for ranking and searching mechanisms of archived data (*requirement 4a*). There is a need to comment on the structure and contents of the archive, and to discuss on various topics, in short, to create a medium for workload data exchange (*requirement 4b*). One of the main reasons for establishing the grid workloads archive is the lack of data access permission for a large majority of the community members. We set as a requirement the public and free access to data (*requirement 4c*).

Requirement 5: community-building tools. There are several other community-building support requirements. There is a need for creating a bibliography on research on grid (and related) workloads (*requirement 5a*), a bibliography on research and practice using the data in the archive (*requirement 5b*), a list of tools that can use the data stored in the archive (*requirement 5c*), and a list of projects and people that use grid workloads (*requirement 5d*).

3.3 The Grid Workloads Archive

In this section we present the Grid Workloads Archive. We discuss its design, detail three distinguishing features, and summarize its current contents.

3.3.1 The Design

We envision five main roles for the GWA community member. The *contributor* is the legal owner of one or more grid workloads, and offers them to the GWA community. The *GWA team* helps the contributors to add data to the GWA archive. The *non-expert user* is the typical user of the archived data. This user type requires as much help as possible in selecting an appropriate trace. The *expert user* uses the archived data in an expert way. Mainly, this user type requires detailed analysis reports and not automatic ranking, consults the related work, and may develop new analysis and modelling tools that extend the GWA data loading and analysis libraries. The *GWA editor* contributes to the community by commenting on the contents of the archive, and by adding related work. One of the major design goals of the GWA is to facilitate the interaction between these five types of members.

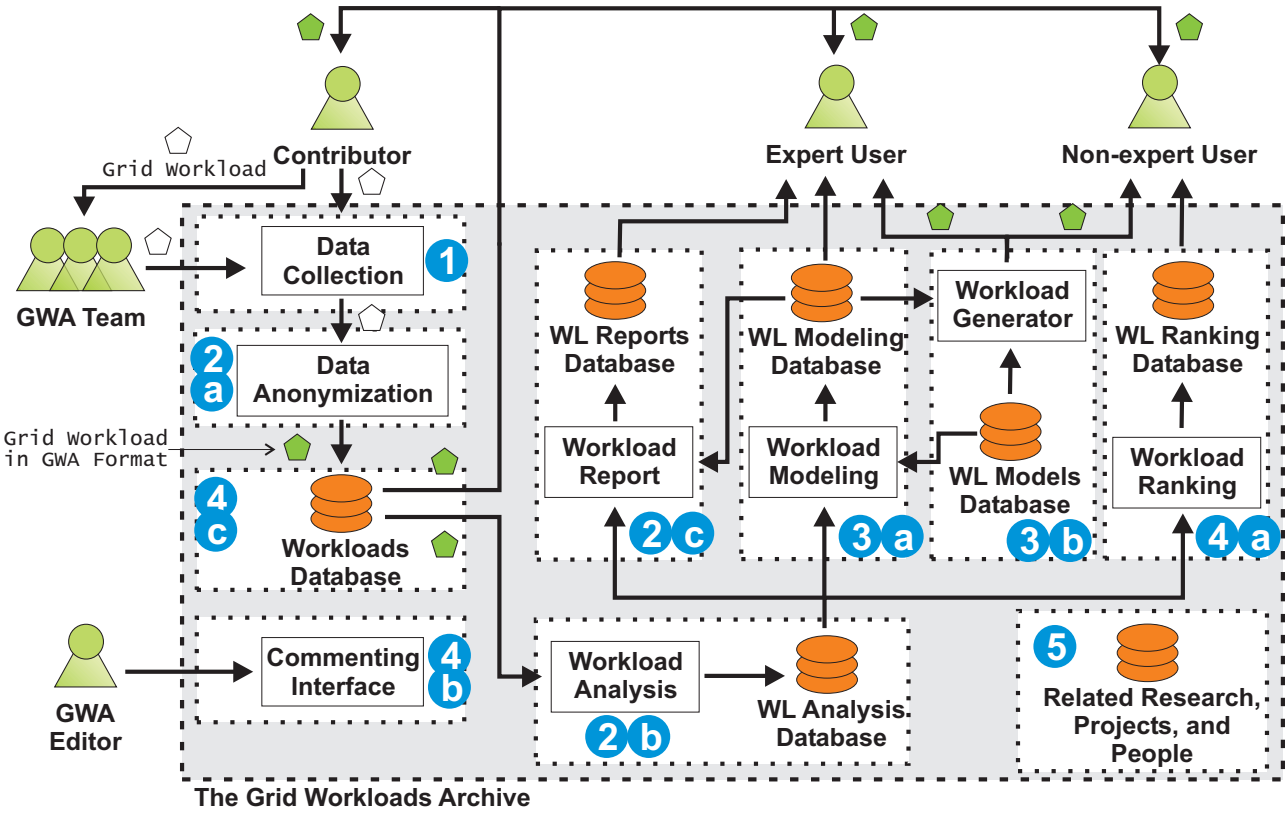


Figure 3.1: *An overview of the Grid Workloads Archive design. The design requirements (see Section 3.2) are marked on the figure.*

Figure 3.1 shows the design of the Grid Workloads Archive, with the requirements expressed in Section 3.2 marked on the figure. The arrows represent the direction of data flows.

There is one module for collecting grid workload data. The Data Collection module receives grid workloads from the contributor (or from the GWA Team, if the contributor delegates the task). There are several potential sources of data: grid resource managers (e.g., the logs of the Globus GRAM Gatekeeper), local resource managers (e.g., the job logs of SGE), web repositories (e.g., the GridPP Grid Operations Center), etc. The data are usually not in the GWA format, but in the format specific to the grid from which they were obtained. The main tasks of the Data Collection module are to ensure that the received data can be parsed, to eliminate wrongly formatted parts of the trace, and to format data provenance information.

There are three modules for processing the acquired data. The Data Anonymization module anonymizes the content received from the Data Collection module, and outputs in the Grid Workloads Archive format (see Section 3.3.2). If the Contributor allows it, a one-to-one map between the anonymized and the original information is also saved. This will allow future data to be added to the same trace, without losing identity correlations between added parts. The Workload Analysis module takes in the data from the Workloads Database, and outputs analysis data to the Workload Analysis Database. More details about this component are given in Section 3.3.5. The Workload Report module formats for the expert user the results of the workload analysis and sometimes of workload modeling.

There are three modules supporting the use of the archived data. The Workload Modeling module attempts to model the archived data, and outputs the results (i.e., parameter values) to the Workload Modeling Database. The input for this process is taken from the Workload Analysis Database (input data), and from the Workload Models Database (input models). Several workload models are supported, including the Lublin-Feitelson model [Lub03]. The Workload Generator module generates synthetic grid workloads based on the results of the Workload Analysis and Modeling results, or on direct user input. The third module (not shown in Figure 3.1) is a library for parsing the stored data.

The GWA contains three modules for data sharing. The Workload Ranking module classifies and ranks the stored traces, for the benefit of the non-expert user. This process is further detailed in Section 3.3.3. The GWA editor uses the Commenting Interface to comment on various aspects presented in the Grid Workload Archive's web site. The Workloads Database stores data in Grid Workloads Archive format. To enable quick processing, the data is stored as raw text and as a relational database. This module has a web interface to allow the public and free distribution of the data within.

The Grid Workload Archive contains various additional community-building support, e.g., bibliographies on previous and derivative related research, and links to related tools, projects, and people.

3.3.2 The Format for Sharing Grid Workload Information

One of the main design choices for the Grid Workloads Archive was to establish a common format for storing workload data. There are two design aspects to take into account. First, there are many aspects that may be recorded, e.g., job characteristics, job grouping and inter-job dependencies, co-allocation [Cza99], advance reservations [Smi00], etc. Second, grid workload data owners are reluctant to provide data for a format they have not yet approved. Thus, one must provide the simplest possible format for the common user, while designing the format to be extensible. We have designed a *standard Grid Workload data Format* (GWF) [Ios08c], which records detailed information about submitted jobs. To further ease the adoption of our format, and as a step towards compatibility with related archives, we base it on the PWA workload format (SWF, the de-facto standard format for the parallel production environments community) [Th07]. We add to this format several grid-specific aspects (e.g., job submission site, etc.) and extension capabilities. We specifically design the language for the following extensions, which we have identified in our previous work ([Ios06c]) as the most relevant for grid workload modeling: batches and workflows, co-allocation, malleability and flexibility, checkpointing, migration, reservations, failures, and economic aspects (e.g., user-specified utility). From a practical perspective, the format is implemented both as an SQL-compatible database (GWF-SQLite), which is useful for most common tasks, and as a text-based version, easy to parse for custom tasks.

Since grids are dynamic systems, using the workload data in lack of additional information (e.g., resource availability) may lead to results that cannot be explained. To address this issue, we have already designed and used a minimal format for resource availability and state [Ios07e].

3.3.3 The Trace Ranking and Selection Mechanism

The non-expert user faces a big challenge when faced with a large database of traces: *which trace to select?* We design a trace mechanisms that ranks traces and then selects the most suitable of them, based on the requirements of the experimental scenario.

We devise a classifier that evaluates a workload according to six categories: number of sites, number of virtual processors, number of users, number of jobs, average utilization, and number of reported

Table 3.1: *A model for automated trace ranking. The quality level is given by the number of stars ($\star$ sign).*

Category	Sample	$\star$	$\star\star$	$\star\star\star$	$\star\star\star\star$	$\star\star\star\star\star$
System Sites	-	1	2-5	6-10	11-20	>20
System Cores	0-100	101-1k	1k-5k	5k-10k	10k-25k	>25k
No. Users	0-50	51-100	101-200	201-500	0.5k-1k	>1k
No. Jobs	0-15k	15k-100k	100k-200k	200k-500k	500k-1M	>1M
Utilization	0-10%	11-20%	21-40%	41-60%	61-75%	>75%
Reported work	0	1	2-5	6-10	11-20	>20

publications using the traces. Note that the categories describe user-specific aspects, system-specific aspects, user-system interaction (utilization), and community relevance (reported work). For a given workload, the classifier assigns a number of stars for each category, from 0 to 5, higher values are better. The classifier is completely described by Table 3.1, which shows the mapping between value ranges and the number of stars for GWA's six categories. We denote by W_s a hypothetical workload that has sample-like characteristics, i.e., the values for its six categories are nearly 0.

We define the *workload signature* as the set of six values for workload's characteristics as output by the classifier. We denote by C_0 the set of all six categories. Consider an experimental scenario in which only some of the categories from C_0 are relevant for workload selection, e.g., the number of system cores and the average utilization for a resource provisioning scenario. Let C be the subset of C_0 that includes only the categories relevant for the scenario at hand; we call such C a scenario-dependent subset of C_0. We define the *partial workload signature for the characteristics in* C as the workload signature from which the characteristics not in C (with $C \subseteq C_0$) have been eliminated. Then, we define the *distance between two workloads* W_1 *and* W_2 as

$$D(C, W_1, W_2) = \frac{\sum_{k=1}^{|C|} (c_1^k - c_2^k)^2}{25 \times |C|}, \tag{3.1}$$

where C is the scenario-dependent set of characteristics, and $W_i = (c_i^1, c_i^2, ...)$ is the partial workload signature of workload i for the characteristics in C. The denominator in Equation 3.1 normalizes the values; the constant included in the divisor, 25, takes into account that the values are between 0 and 5 stars. In particular, we call $D(C_0, \cdot, \cdot)$ the *scenario-independent distance*, and $D(C_0, W_i, W_s)$ the *scenario-independent value* (short, *value*) *of workload* W_i.

The ranking and selection mechanisms use the distance between workloads. We present online the ranking table that uses the scenario-independent value of the traces present in the GWA. The GWA users can select traces using directly this table, or can obtain a different table by specifying a new C.

3.3.4 The Contents of the Workloads Database

Table 3.2 shows the nine workload traces currently included in the GWA. Note that several are under processing, or have pending publication rights. The data sources for these traces range from local resource managers (e.g., PBS) to grid resource managers (e.g., Globus GRAM) to user- and VO-level resource managers (e.g., Condor Schedd). In several cases, incomplete data is provided, e.g., for NorduGrid the trace does not include locally submitted (non-grid) jobs. The traces include grid applications from the following areas: physics, robotics, rendering and image processing (graphics), collaborative and virtual environments (v-environments), computer architectures simulations (CAS),

Table 3.2: *The GWA content (status as of August 2007). The ⋆ sign marks restrictions due to data scarcity (see text). The ◇ sign marks traces under processing. The ‡ sign marks traces with pending publication rights.*

ID	System	Period	Number of observed				
			Sites	CPUs	Jobs	Groups	Users
GWA-T-1	DAS-2	02/05-03/06	5	400	602K	12	332
GWA-T-2	Grid'5000	05/04-11/06	15	∼2500	951K	10	473
GWA-T-3◇	NorduGrid	05/04-02/06	∼75	∼2000	781K	106	387
GWA-T-4◇	AuverGrid	01/06-01/07	5	475	404K	9	405
GWA-T-5◇	NGS	02/03-02/07	4	∼400	632K	1	379
GWA-T-6◇	LCG	05/05-01/06	1⋆	880	1.1M	25	206
GWA-T-7‡	GLOW	09/06-01/07	1⋆	∼1400	216K	1⋆	18
GWA-T-8‡	Grid3	06/04-01/06	29	2208	1.3M	1⋆	19
GWA-T-9‡	TeraGrid	08/05-03/06	1⋆	96	1.1M	26	121
	Total	13.51 yrs	136	>10000	>7M	191	2340
	Average	1.35 yrs	15	1151	787K	21	260

artificial intelligence (AI), applied mathematics (math), chemistry, climate and weather forecasting (climate), medical and bioinformatics (biomed), astronomy, language, life sciences (life), financial instruments (finance), high-energy physics (HEP), aerospace design (aero), etc.; three of the nine GWA traces include only HEP applications. Note that due to trace anonymization it is not possible to map the jobs included in the GWA traces to specific applications or application areas.

The GWA-T-1 trace is extracted from DAS-2 [Bal00], a wide-area distributed system consisting of 400 CPUs located at five Dutch Universities. DAS-2 is a research testbed, with the workload composed of a large variety of applications, from simple single CPU jobs to complex co-allocated Grid MPI [Moh05a] or IBIS [Wrz05] jobs. Jobs can be submitted directly to the local resource managers (i.e., by *system users*), or to Grid gateways that interface with the local resource managers. To achieve low wait time for interactive jobs, the DAS system is intentionally left as free as possible by its users. The traces collected from the DAS include applications from the areas of physics, robotics, graphics, v-environments, CAS, AI, math, chemistry, climate, etc. In addition, the DAS traces include experimental applications for parallel and distributed systems research.

The GWA-T-2 trace is extracted from Grid'5000 [Bol06], an experimental grid platform consisting of 9 sites geographically distributed in France. Each site comprises one or several clusters, for a total of 15 clusters inside Grid'5000. The main objective of this reconfigurable, controlable, and monitorable experimental platform is to allow experiments in all the software layers between the network protocols up to the applications. We have obtained traces recorded by all batch schedulers handling Grid'5000 clusters (OAR [Cap05]), from the beginning of the Grid'5000 project up to November 2006. Note that most clusters of Grid'5000 were made available during the first half of 2005. The traces collected from Grid5000 include applications from the areas of physics, biomed, math, chemistry, climate, astronomy, language, life, finance, etc. In addition, the Grid5000 traces include experimental applications for parallel and distributed systems research.

The GWA-T-3 trace is extracted from NorduGrid [Eer03], a large scale production grid. In NorduGrid, non-dedicated resources are connected using the Advanced Resource Connector (ARC) as Grid middleware [Ell07]. Over 75 different clusters have been added over time to the infrastructure. We have obtained the ARC logs of NorduGrid for a period spanning from 2003 to 2006. In these logs, the information concerning the grid jobs is logged locally, then transferred to a central database voluntarily. The logging service can be considered fully operational only since mid-2004. The traces collected from NorduGrid include applications from the areas of CAS, chemistry, graphics, biomed, and HEP.

The GWA-T-4 trace is extracted from AuverGrid, a multi-site grid that is part of the EGEE project. This grid employs the LCG middleware as the grid's infrastructure (same as the GWA-T-6 trace). We have obtained traces recorded by the resource managers of the five clusters present in AuverGrid. The traces collected from AuverGrid include applications from biomed and HEP.

The GWA-T-5 trace is extracted from the UK's National Grid Service (NGS), a grid infrastructure for UK e-Science. We have obtained traces from the four dedicated computing clusters present in NGS. The traces collected from the NGS include applications from biomed, physics, astronomy, aero, and HEP.

The GWA-T-6 trace is extracted from the LHC Computing Grid (LCG) [07b]. LCG is a data storage and computing infrastructure for the high energy physics community that will use the Large Hadron Collider (LHC) at CERN. The LCG production Grid currently has approximately 180 active sites with around 30,000 CPUs and 3 petabytes storage, which is primarily used for high energy physics (HEP) data processing. There are also jobs from biomedical sciences running on this Grid. Almost all the jobs are independent computationally-intensive tasks, requiring one CPU to process a certain amount of data. The workloads are obtained via the LCG Real Time Monitor[2] (RTM). The RTM monitors jobs from all major Resource Brokers on the LCG Grid therefore the data it collects are representative at the Grid level. In particular, the GWA-T-6 is a long-term trace coming from one of the largest LCG sites, comprising 880 CPUs. The traces collected from the LCG include only HEP applications.

The GWA-T-7 trace is extracted from the Grid Laboratory of Wisconsin (GLOW), a campus-wide distributed computing environment that serves the computing needs of the University of Wisconsin-Madison's scientists. This Condor-based pool consists of over 1400 machines shared temporarily by their rightful owners [Tha05a]. We have obtained a trace comprising all the jobs submitted by one Virtual Organization (VO) in the Condor-based GLOW pool, in Madison, Wisconsin. The trace spans four months, from September 2006 to January 2007. The traces collected from GLOW include only HEP applications.

The GWA-T-8 trace is extracted from the Grid3, which represents a multi-virtual organization environment that sustains production level services required by various physics experiments. The infrastructure was composed of more than 30 sites and 4500 CPUs; the participating sites were the main resource providers under various conditions [Fos04]. We have obtained traces recorded by the Grid-level scheduler corresponding to one of the largest VOs: the Grid3/USATLAS; there are three major VOs in the system, the others being iVDgL and USCMS. These traces capture the execution of workloads of physics working groups: a single job can run for up to a few days, and the workloads can be characterized as directed acyclic graphs (DAGs) [Mam05]. The traces collected from Grid3 include only HEP applications.

The GWA-T-9 trace is extracted from the TeraGrid system, a system for e-Science, with more than 13.6 TeraFLOPS of computing power, and facilities capable of managing and storing more than 450 TeraBytes of data [06]. We have obtained traces recorded by the interface between the Grid level scheduler and the local resource manager of one of the TeraGrid sites: the UC/ANL. In the analyzed traces, workloads are composed of applications targeting high-resolution rendering and remote visualization; ParaView, a multi-platform application for visualizing large data sets [06], is the commonly used application.

[2]The Real Time Monitor is developed by Imperial College London http://gridportal.hep.ph.ic.ac.uk/rtm/.

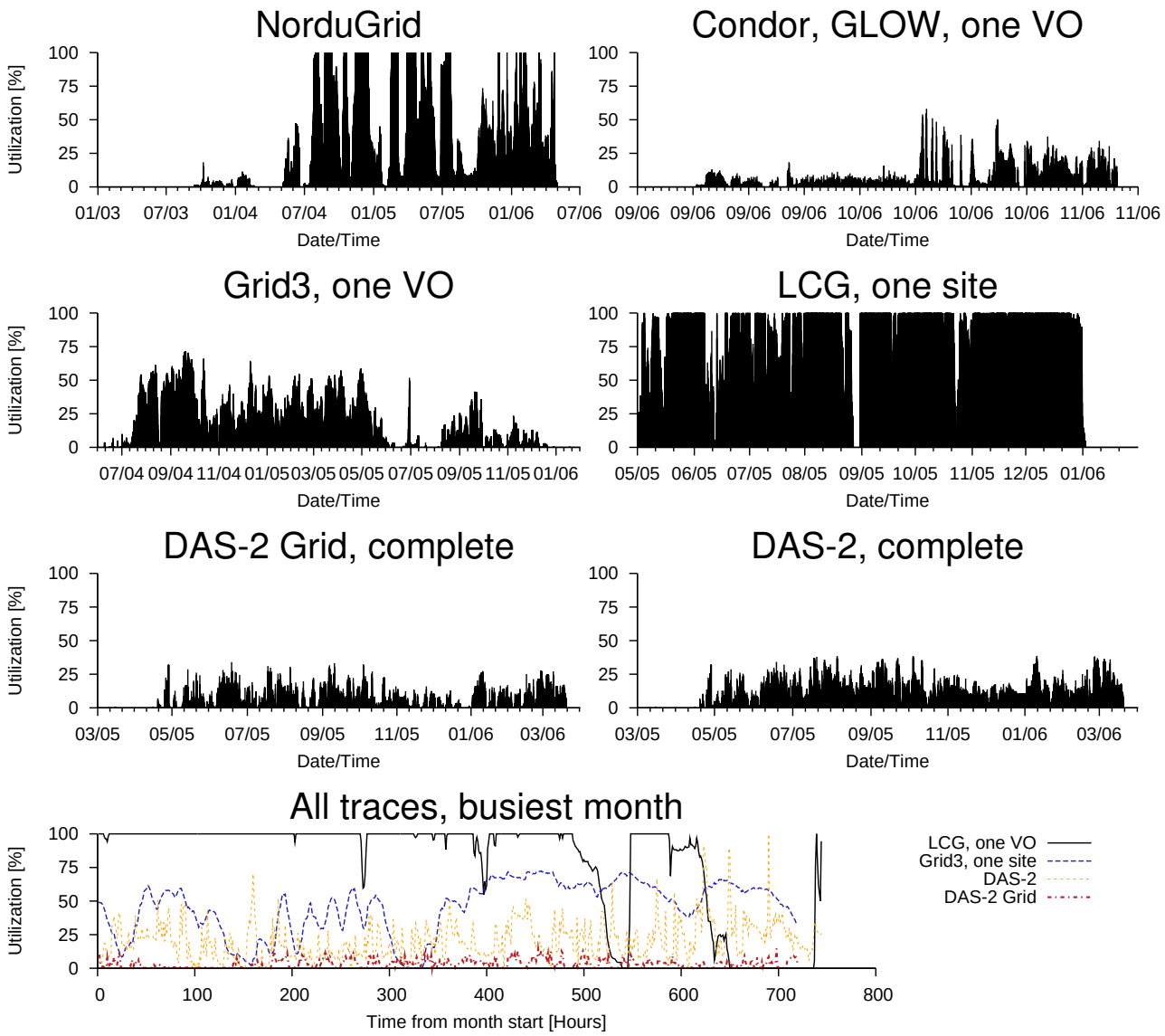


Figure 3.2: *System utilization over time for NorduGrid, Condor GLOW, Grid3, LCG, DAS-2, and DAS-2 Grid. The busiest month may be different for each system.*

3.3.5 The Toolbox for Workload Analysis, Reporting, and Modeling

The GWA provides a comprehensive toolbox for automatic trace analysis, reporting, and modeling. The toolbox provides the contributors and the expert users with information about the stored workloads, and can be used as a source for building additional workload-related tools.

The workload analysis focuses on three aspects: system-wide characteristics (e.g., system utilization, job arrival rate, job characteristics; comparison of sequential and parallel jobs' characteristics), user and group characteristics (i.e., similar to system-wide characteristics, but for all and top users), and performance analysis (e.g., resource consumption, waiting and running jobs, and throughput). The analysis enables a quick comparison of traces for the expert user, or a detailed view of one grid,

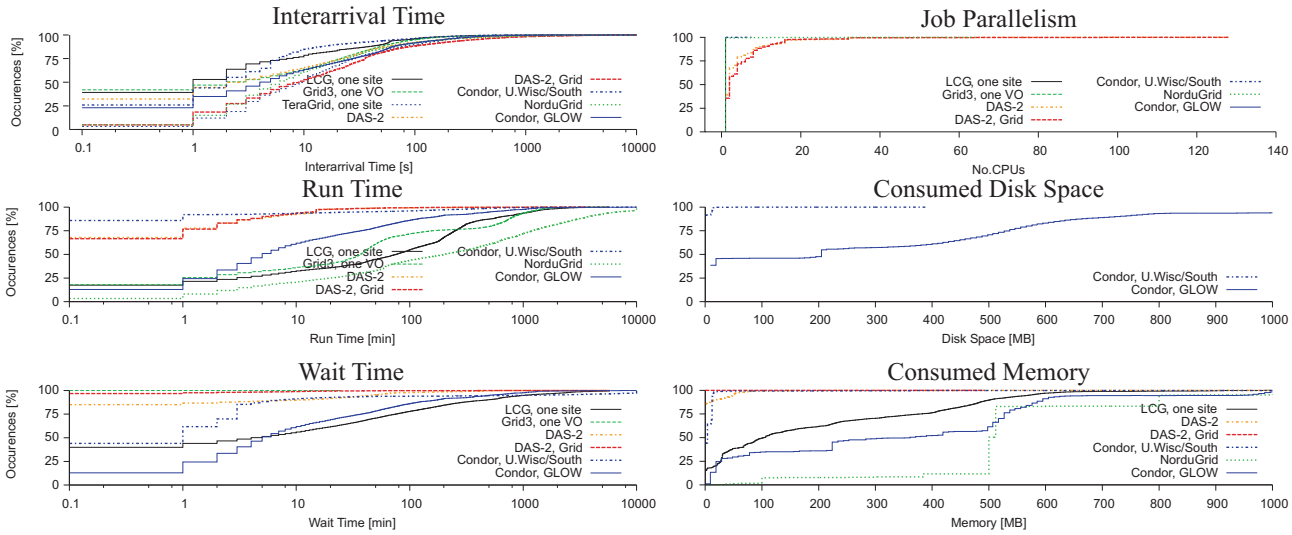


Figure 3.3: *CDFs of the most important job characteristics for NorduGrid, Condor GLOW, Condor UWisc-South, TeraGrid, Grid3, LCG, DAS-2, and DAS-2 Grid. Note the log scale for time-related characteristics.*

for the contributor (that is, the grid administrator); we detail in Section 3.4.1 several such uses. Figures 3.2, 3.3, and 3.4 show a sample of the workload analysis results.

Figure 3.2 shows that grid utilization ranges from very low (below 20% for DAS) to very high (above 85% for one cluster in LCG (trace `GWA-T-6`)). To ease the comparison, the bottom sub-graph depicts the hourly system utilization during the busiest month, for each environment. The average LCG system utilization is very high, at over 85% on average; this is caused by the nature of jobs run in LCG: exclusively single processor jobs, with the job runtime long relative to the environment overhead. The Grid3 utilization caused by one of the top VOs is also quite high, with an average around 20%. Speculating on the use incurred by the other two major and the several minor Grid3 VOs, this would lead to a system utilization of over 70%. The DAS system has a very low average utilization, below 10%. This is a consequence of the "15-minutes usage" rule of the DAS. The DAS Grid usage is roughly 50% of the overall DAS system utilization, for an average utilization below 5%. The system utilization is not stable over time: for all studied traces there are bursts of high utilization use between periods of low utilization.

Figure 3.3 depicts the cumulative distribution function (CDF) for the most important job characteristics for the GWA workloads: the inter-arrival time between consecutive jobs, the wait time, the runtime, the memory consumption, the consumed CPU time, and the job parallelism (number of CPUs per job). For all traces, in 90% of the cases a new job arrives at most 1 minute after the previous job. The runtime and wait time distributions confirm that the grids where the GWA traces were collected serve widely different categories of users, with application use from interactive (DAS) to computing-intensive/batch (NorduGrid). Similarly, the resource consumption characteristics are very different across traces. Finally, for six of the GWA traces, over 90% of the jobs are single-processor; for four of them the percentage is 100%. We believe that this corresponds to the real use of many grids for the following reasons. First, many grids offer facilities for sending bags of similar tasks (e.g., parameter sweeps) with a single command; the user's task of running large numbers of jobs is thus greatly simplified. Second, there are few parallel applications in the GWA traces relative to single-processor jobs. Even though three GWA traces include more than 10% parallel jobs (`GWA-T-1`,

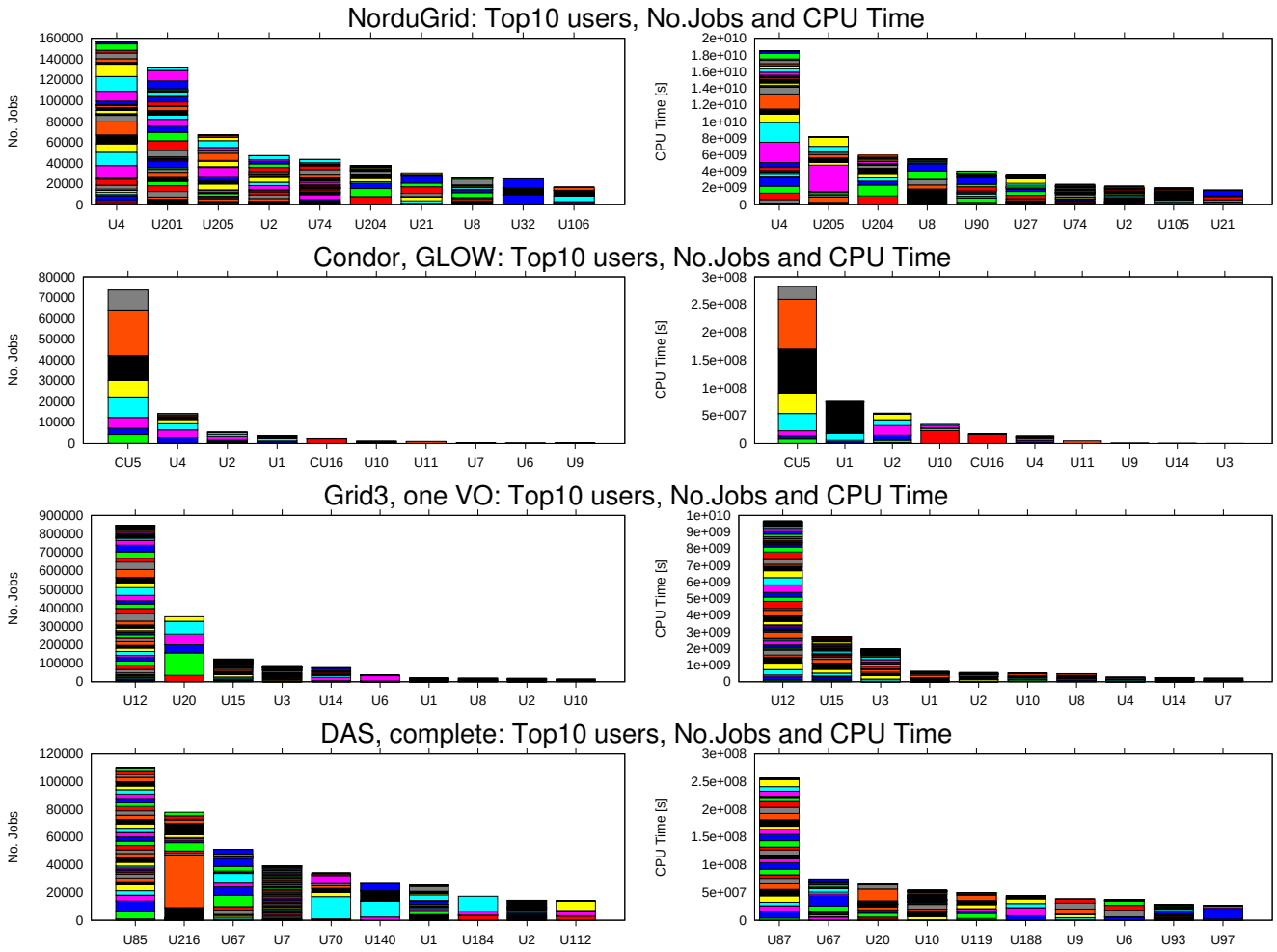


Figure 3.4: *The number of submitted jobs (left) and the consumed CPU time (right) by user per system: NorduGrid (top row), Condor GLOW (second row), Grid3 (third row), and DAS-2 (bottom). Only the top 10 users are displayed. The horizontal axis depicts the user's rank. The vertical axis shows the cumulated values, and the breakdown per week. For each system, users have the same identifiers (labels) in the left and right sub-graphs.*

`GWA-T-2`, and `GWA-T-9`), two of them (`GWA-T-1` and `GWA-T-2`) correspond to grids that run experimental applications for parallel and distributed systems research. For example, in DAS, three of the top five and six of the top ten users ranked by the number of submitted jobs are parallel and distributed systems researchers, which explains the high percentage of parallel jobs in the DAS traces. Third, for the periods covered by the traces, there is a lack of deployed mechanisms for parallel jobs, e.g., co-allocation and advance reservation. Co-allocation mechanisms were available only in the DAS, and, later, in Grid5000. The first co-allocation mechanisms that do not require advance reservation have been implemented in the DAS [Moh05a], which explains why 10% of the jobs present in the DAS traces are co-allocated jobs. Even with the introduction of co-allocation based on advance reservation in several of the other grids (i.e., Grid500), there is no evidence showing that co-allocation has become mainstream (the percentage of co-allocated jobs in the Grid5000 traces is below 1%). However,

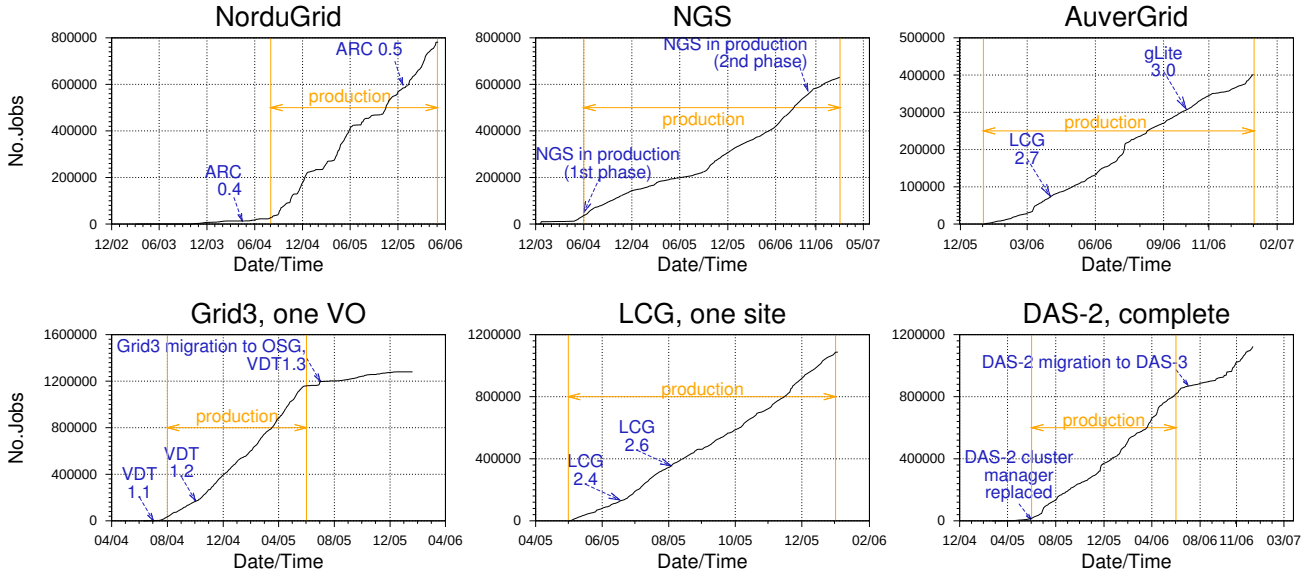


Figure 3.5: *The evolution of the cumulative number of jobs submitted over time for six grids. Special events such as middleware change are marked with dotted lines. The production period is also emphasized. ARC, LCG, gLite, and VDT are the key grid middleware packages used by the depicted grids.*

parallel jobs are still important in grids, e.g., for `GWA-T-5`, the parallel jobs account for 5% of the number of jobs, but 85% of the consumed CPU time.

Figure 3.4 shows that a small number of users (below 10) dominate the workloads in both number of submitted jobs and consumed CPU time, for all the analyzed traces. In DAS, the top user by the number of submitted jobs is an automated verification tool: jobs are constantly generated every two hours; Figure 3.4 depicts this situation with equally sized stripes.

The results depicted in Figures 3.2, 3.3, and 3.4 represent the outcome of an analysis on the complete traces. However, some of the systems were not in production from the beginning to the end of the period for which the traces were collected. Moreover, the middleware used by a grid may have been changed (e.g., upgraded or replaced) during the production period. To support the validity of our analysis results, we show below that the non-production periods captured in our traces include few jobs, and that the middleware changes do not significantly affect the properties of the submitted jobs.

Figure 3.5 shows the evolution of the cumulative number of submitted jobs over time. The period during which the grids are in production and the main events affecting them (e.g., middleware change, system evolution, system closure) are specially marked. We observe four main trends related to the rate of growth for the cumulative number of submitted jobs (the *input*). First, the production period (marked on the image with "production" for each depicted trace) has a homogeneous aspect, with the input much higher than for the non-production periods. Second, for the periods covered by our traces, the change of the middleware version does not have a significant impact on the input. Third, the period before entering production exhibits a low input (i.e., few job submissions) relative to the production period. Fourth, a system at the end of its production cycle loses its users in favor of the system that replaces it (a "migration" event occurs), starting about a month before the migration event; this situation is captured in the `GWA-T-1` (DAS-2) and the `GWA-T-8` (Grid3) traces. The first two trends, and the observation that most of the middleware changes are minor version increments

(with the notable exception of AuverGrid, which switched from 2.7 to 3.0–gLite is the successor of LCG), indicate that there is no significant change in the characteristics of the jobs that is due to the system change. The last two trends show that the characteristics of the jobs present in our traces are mostly influenced by the jobs submitted during the production period.

3.4 Using the Grid Workloads Archive

The GWA can also be beneficial in many theoretical and practical endeavors. In this section, we discuss the use of the GWA in three broad scenarios: research in grid resource management, for grid maintenance and operation, and for grid design, procurement, and performance evaluation.

3.4.1 Research in grid resource management

There are many ways in which the GWA can be used for research in grid resource management. We have already used the archived content to understand how real grids operate today, to build realistic grid workload models, and as real input for a variety of resource management theory (e.g., queueing theory).

The study in [Ios06a] shows how several real grids operate today. The authors analyze four grid traces from the GWA, with a focus on virtual organizations, on users, and on individual jobs characteristics. They further quantify the evolution and the performance of the Grid systems from which the traces originate. Their main finding is that the four real grid workloads differ significantly from those used in grid simulation research, and in particular that they comprise mostly single processor jobs. In another work, we have investigated the existence of batches of jobs in grids, and found that in several real grids batches are responsible for 85%–95% of the jobs, and for 30%–96% of the total consumed CPU [Ios07d]. The imbalance of job arrivals in multi-cluster grids has been assessed using traces from the GWA in another study [Ios07c].

Hui Li et al. conduct statistical analysis of cluster and grid level workload data from LCG, with emphasis on the correlation structures and the scaling behavior [Li07c; Li07e; Li07f]. This leads to the identification and modeling of several important workload patterns, including pseudo-periodicity, long range dependence, and "bag-of-tasks" behavior with strong temporal locality. The performance impact of the correlations between the workload characteristics is shown in simulations to influence significantly the system performance, both at the local and at the grid level [Li07a]. This gives evidence that realistic workload modeling is necessary to enable dependable grid scheduling studies.

The contents of the GWA has been used to characterize grids as queues, by assessing the jobs' wait and run time marginal distributions, by estimating the number of jobs arriving and exiting the system over time, and by computing the resource utilization rate [Ios06a]. Similarly, the traces have been used to characterize grids as service-oriented architectures, by assessing the jobs' goodput and throughput [Ios06a]. Finally, the traces have been used to show that grids can be treated as dynamic systems with quantifyable [Ios06d] or predictable behavior [Li04; Li07f]. These studies show evidence that grids are capable to become a predictable, high-throughput computation utility.

The contents of the GWA has also been used to evaluate the performance of various scheduling policies, both in real [Ios06b] and simulated [Ios06d; Ios07c; Li07b] environments. Finally, the tools in the GWA have been used to provide an analysis back-end to a grid simulation environment [Ios07c].

3.4.2 Grid maintenance and operation

The content of GWA can be used for grid maintenance and operation in many ways, from comparing real systems with established practice (represented in the archive), to testing real systems with realistic workloads. We detail below two such cases.

A system administrator can compare the performance of a working grid system with that of similar systems by comparing performance data extracted from their traces. Additionally, the performance comparison over time (e.g., for each week represented in the trace) may help understanding when the operated system has started to behave outside the target performance level. Since grids represent new technology for most of their users, a lower performance in the beginning may represent just a learning period; to distinguish between this situation and system misconfiguration, the beginning of the traces of other starting systems (e.g., `GWA-T-1`) can be compared with the system under inquiry.

In large grids, realistic functionality checks must occur daily or even hourly, to prevent that jobs are assigned to failing resources. Our results using data from the GWA show that the performance of a grid system can rise when availability is taken into consideration, and that human administration of availability change information may result in 10-15 times more job failures than for an automated solution, even for a lowly utilized system [Ios07e]. To perform realistic functionality testing, tools like the GrenchMark can "replay" selected parts of the traces from the the GWA in the real environments [Ios06b]. Similarly, functionality and stress testing are required for long-term maintenance. Again, tools like GrenchMark make use of the data stored in the GWA to run realistic tests.

Grid monitoring systems like Caltech/CERN's MonALISA [New03b] and GridLab's Mercury [Bal01] are now common tools that assist in the grid maintenance and operation. Based on a trace and tools from the GWA, we have assessed the trade-off between the quality of information and the monitoring overhead; our simulation results show that a reduction of 90% in monitoring overhead can be achieved with a loss in accuracy of at most 10% [Str07]. Similarly, a system manager may choose to setup the monitoring system based on a similar analysis, using the same tools and his system's data.

3.4.3 Grid design, procurement, and performance evaluation

The GWA has already been used in many grid design, procurement, and performance evaluation scenarios.

The grid designer needs to select from a multitude of middleware packages, e.g., resource managers. Oftentimes, the designer uses "what-if" scenarios to answer questions such as *What if the current users would submit 10 times more jobs in the same amount of time? Or 50 times, or 100 times...*, or *If the users of another environment could submit their workload to our environment, what would be the success rate of the jobs submitted by these combined communities?* Using workloads from the GWA, and a workload submission tool such as GrenchMark, the designer can answer these questions for a variety of potential user workloads. We have shown a similar use of the GWA's content for the DAS environment in our previous work [Ios06b].

During the procurement phase, a prospective grid user may select between several infrastructure alternatives: to rent compute time on an on-demand platform, to rent or to build a parallel production environment (e.g., a large cluster), or to join a grid as a resource user. The reports published by the GWA show evidence that grids already offer similar or better throughputs and can handle much higher surges in the job arrival rate, when compared with large-scale parallel production environments (see Table 3.3 and Figure 3.6 for a summary of these reports).

Similarly to system design and procurement, performance evaluation can use content from the GWA in a variety of scenarios, e.g., to assess the ability of a system to execute a particular type of

Table 3.3: *Grid vs. Parallel Production Environments: processing time consumed by users, and highest number of jobs running in the system during a day. The "Type" column shows the environment type: PProd for parallel production, or Grid for grid computing.*

Environment Name	Type	Data Source /Analysis	System Processors	Goodput [CPUYr/Yr]	Spike [Jobs/Day]
NASA iPSC	PProd	[Th07; Fei07]	128	92.03	876
LANL CM5	PProd	[Th07; Fei97]	1,024	808.40	5358
SDSC Par95	PProd	[Th07; Win96]	400	292.06	3407
SDSC Par96	PProd	[Th07; Win96]	400	208.96	826
CTC SP2	PProd	[Th07; Hot96]	430	294.98	1648
LLNL T3D	PProd	[Th07]	256	202.95	445
KTH SP2	PProd	[Th07; Fei98a]	100	71.68	302
SDSC SP2	PProd	[Th07; Mu'01]	128	109.15	2181
LANL O2K	PProd	[Th07]	2,048	1,212.33	2458
OSC Cluster	PProd	[Th07]	57	93.53	2554
SDSC BLUE	PProd	[Th07]	1,152	876.77	1310
LCG, 1 Cluster	Grid	[Ios06a]	880	750.50	22550
Grid3, 1 VO	Grid	[Ios06a]	2208	360.75	15853
DAS-2	Grid	[Ios06a]	400	30.34	19550
NorduGrid	Grid	[Ios08d]	∼3,100	770.20	7953
TeraGrid, 1 Site	Grid	[Ios06a]	∼200	n/a	7561
Condor, GLOW, 1 VO	Grid	[Ios08d]	∼1,400	104.73	6590

workload [Ios06b; Ios07b], to find the throughput of the system for the common usage patterns, or to measure the power consumption and failure rate under different workload patterns. Note that the same approach may be used during procurement to compare systems using trace-based grid benchmarking.

3.4.4 Education

The reports, the tools, and the data included in the GWA can greatly help the educators. We target courses that teach the use of grids, large-scale distributed computer systems simulation, and computer data analysis. The reports included in the GWA may be used to better illustrate concepts related to grid resource management, such as resource utilization, job wait time and slowdown, etc. The tools may be used to build new analysis and simulation tools. The data included in the archive may be used as input for demonstrative tools, or as material for student assignments.

3.5 Related Work

In this section we survey several archival approaches in computer science areas, e.g., Internet, clusters, grids. We assess the relative merits of the surveyed approaches according to the requirements described in Section 3.2; Table 3.4 summarizes our survey. In comparison with the Internet community efforts, the GWA contains tools to generate and use synthetic grid workloads, besides the raw grid workload data. In comparison with the other efforts, the GWA offers more tools for processing, using, and sharing the stored data.

The research community has started to understand the importance of computer systems' performance evaluation based on real(istic) traces at the beginning of the '70s [Fer72]. By the beginning of 1990s, this shift in practice had become commonplace [Jai91; Cal93]. In beginning of the 1990s the invention of the world-wide web [BL92], and the gradual lowering of the bandwidth and disk storage costs, paved the way for the first workload archives.

In 1995, the Internet community assembled the first publicly and freely available workload archive: the Internet Traffic Archive (ITA). ITA has undergone several updates over the years, which show

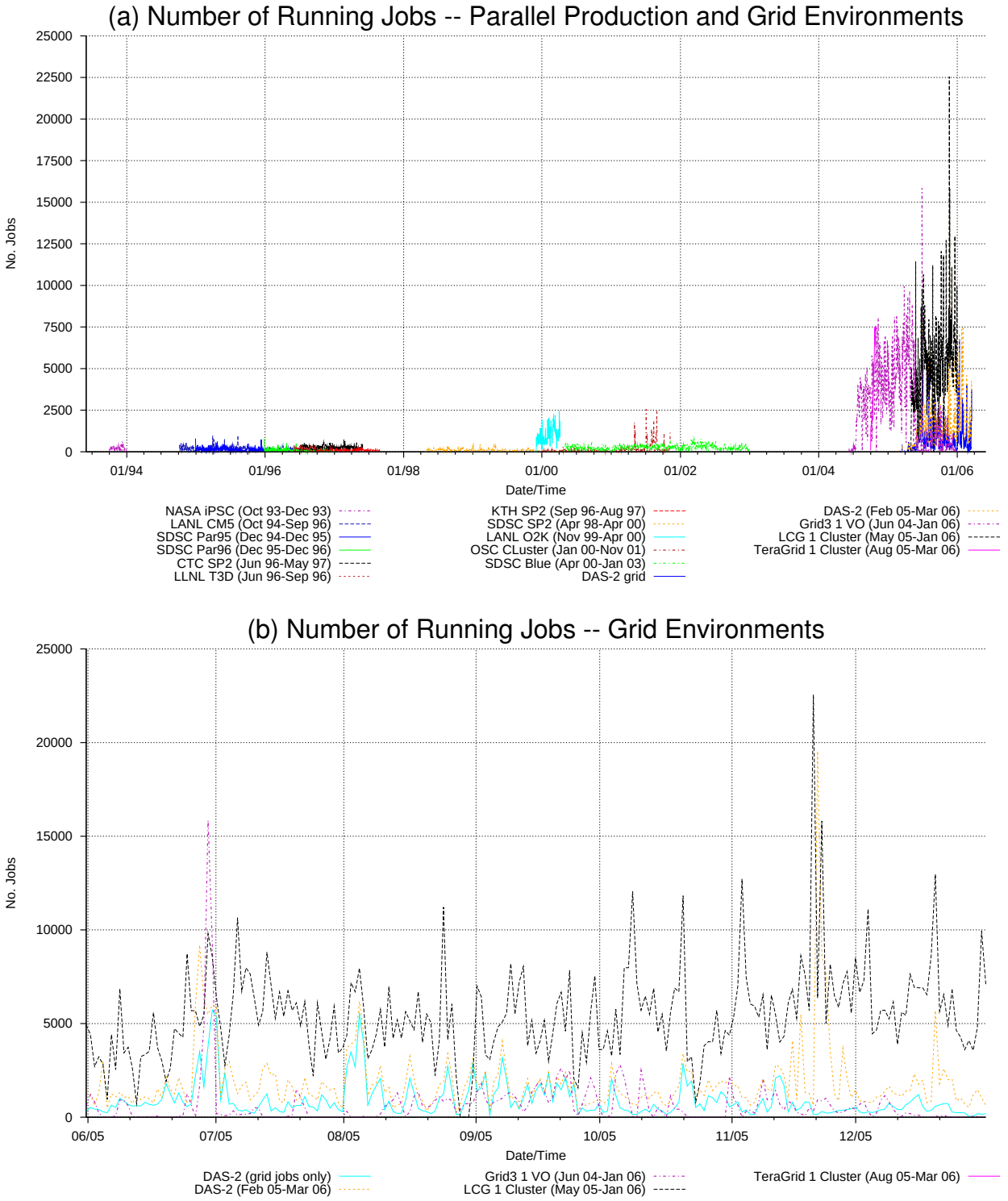


Figure 3.6: *Running jobs during daily intervals for grid and parallel environments: (a) comparative display over all data for grid and parallel environments; (b) comparative display for data between June 2005 and January 2006, for grid environments only.*

Table 3.4: *A summary of workload archives in Computer Science. The +, ∼, and - signs denote a feature that is present, for which an insufficient approach has been taken, or which lacks, respectively.*

Workload Archive (Since)	1 collect	2 process			3 use			4 share			5 build community			
		a	b	c	a	b	c	a	b	c	a	b	c	d
The Internet														
ITA [Dan07] (1995)	-	+	+	+	-	-	-	-	+	+	-	+	+	-
WIDE [Cho00] (1999)	-	+	+	+	-	-	-	-	+	+	-	+	+	-
CAIDA (2002)	+	+	+	+	-	-	-	+	+	+	+	+	+	+
CRAWDAD [Yeo06] (2005)	-	+	+	+	-	-	-	+	+	∼	+	+	+	+
NTTT [NTT07] (2006)	-	-	+	+	-	+	-	-	-	+	∼	∼	-	-
Single-computer Systems														
BYU [Per07b] (1999)	+	-	+	-	-	-	+	-	+	+	-	+	+	-
NMSU [Per07a] (2002)	-	-	-	-	-	-	-	-	-	∼	+	-	+	-
CADRE [Pab07] (2003)	+	-	+	+	-	-	-	+	-	+	∼	∼	+	-
Cluster-based Systems														
PWA [Th07] (1999)	+	+	-	-	-	+	+	∼	-	+	+	+	-	+
MAUI HPC [Cen07] (2001)	-	-	-	-	-	-	-	-	-	+	-	∼	-	-
CFDR [Sch07] (2007)	-	-	-	-	-	-	-	-	-	∼	-	-	-	-
Grids														
DGT [Kon07c; Kon07b] (2005)	-	-	-	-	-	-	-	-	-	+	-	∼	+	-
Other Archives of Interest														
RAT [God07] (2007)	-	-	-	-	-	-	-	-	-	+	-	+	-	+
GWA (2006)	+	+	+	+	+	+	+	+	+	+	+	+	+	+

why it still is surprisingly modern: it has tools for collecting and processing data, mailing lists for commenting, its data is publicly and freely available, etc. The Internet community has since created several other archives, i.e., WIDE, CAIDA's archives, CRAWDAD, and NTTT. The CAIDA archives [CAI07; Dat07; MOA07] are combined the largest source of Internet traces. CRAWDAD is the first archive dedicated to the wireless networks community. These archives have gradually evolved towards covering most of the requirements expressed in Section 3.2 for the Internet community. Notably, with the exception of NTTT, they do not offer tools for using the offered data; NTTT offers tools for generating synthetic workloads.

Contrary to the Internet community, the computer systems communities are still far from addressing Section 3.2's requirements. The computer architectures community started its first and most successful database (BYU) at the end of the 1990s. Since then, several other archives have started, e.g., NMSU and CADRE, but have yet to improve on the results of the BYU archive. For the cluster-based communities, the Parallel Workloads Archive (PWA) covers many of the requirements, and has become the de-facto standard for the parallel production environments community. The MAUI HPC archive started separately, but has since been included in the PWA. The DGT, CFDR, and RAT archives only resource availability and failure traces.

The PWA is the closest project to our GWA both in target (parallel computing environments) and realization. Recently, the PWA has added several grid traces to its content. However, the PWA workload format [Cha99] was not designed for grid workloads, and loses information on many grid-specific aspects, including the number of used nodes (which may be different from the number of used processors, for multi-processor nodes), co-allocation, submission site (which may be different from the execution site), and job exit status. The Grid Workloads Archive is the first archive to accommodate the requirements of grid workloads, and thus complements the PWA and other approaches. To the best of our knowledge, the GWA is also the first to satisfy all the requirements of a workloads archive.

3.6 Concluding remarks

While many grids are currently serving as e-Science infrastructure, very little is known about the real users' demand. The lack of grid workloads hampers the research on grid resource management, and the practice in grids design, management, and operation. To collect grid workloads and to make them available to this diverse community, we have designed and developed the Grid Workloads Archive. The design focuses on two broad requirements: building a grid workload data repository, and building a community center around the archived data. For the former, we provide tools for collecting, processing, and using the data. For the latter, we provide mechanisms for sharing the data and other community-building support. We have collected so far traces from nine well-known grid environments.

For the future, we plan to bring the community of resource management in large-scale distributed computing systems closer to the Grid Workloads Archive. We believe that our archive will be useful for many scientific directions including, but not limited to, scheduling in and performance evaluation of such systems.

Chapter 4

A Comprehensive Model for Multi-Cluster Grids

4.1 Overview

In this chapter* we extend the basic model for multi-cluster grids presented in Chapter 2 towards a comprehensive model for multi-cluster grids.

4.1.1 Motivation and Problem Statement

Many of today's grids, e.g., the DAS, the e-Science Grid, NorduGrid, OurGrid, the EGEE, the OSG, and Grid'5000, are so-called multi-cluster grids, as they comprise computing resources grouped in clusters. In Chapter 2 we have introduced a basic model for multi-cluster grids, covering the resource types, the job types, the user types, and the job execution model. However, two time-varying aspects of multi-clusters grid were not covered by the basic model: the resource availability, and the system workload.

At the grid scale, at any time a significant part of the system resources may be out of the users' reach due to distributed resource ownership, scheduled maintenance, or unpredicted failures. In a multi-cluster grid, all the resources of a cluster may be shared with the grid only for limited periods of time, e.g., during local night-time. Often, many of a grid's resources are removed by their owners from the system, either individually or as complete clusters, to serve other tasks and projects. Furthermore, grids experience the problems of any large-scale computing environment, and in addition are operated with relatively immature middleware, which increases further the resource unavailability rate. Grid resources are dynamic in both number and performance. We identify two types of change: over the short term (e.g., machines rebooting, processors failing and being replaced), and over the long term (e.g., the addition of a new cluster, clusters going out of service, new nodes being added to a cluster). We call the former type of change *grid dynamics*, and the latter *grid evolution*. Disregarding grid dynamics during grid design may lead to a solution with low reliability. Disregarding the grid evolution may lead to a solution that does not match the systems of the future. While many studies cover the resource availability in computing systems that are related to multi-cluster grids (i.e., supercomputers and single clusters [Sah04; Sch06], and desktop grids [Kon07b; Kon07a]), there is currently no study of the resource availability in multi-cluster grids. The previous studies may be unsuitable for characterizing resource availability in multi-cluster grids: a badly administrated cluster may lead to much higher unavailability for all the resources of that cluster than for the resources of other clusters, the failure of a cluster gateway compromises user access to any of that cluster's resources but not to the resources of other clusters, etc. Thus, an important question arises: *What are the characteristics of the resource (un)availability in multi-cluster grids?*

*This chapter is based on previous work published in the IEEE/ACM International Conference on Grid Computing (Grid 2006) [Ios06a], the IEEE/ACM Int'l. Conference on Grid Computing (Grid 2007) [Ios07e], Euro-Par 2007 [Ios07d], and the IEEE Int'l. Symp. on High Performance Distributed Computing (HPDC'08) [Ios08e].

While many grids exist, little is known about the their workload, with many guesses being treated as hard facts. At one extreme, grid researchers have argued that grids will be the natural replacement of tightly coupled high-performance computing systems, and therefore will take over their highly parallel workloads [Ern02; Ern04; Ios06c; Ran08a]. At the other extreme stand arguments that grids are mostly useful for running conveniently parallel applications, that is, large bags of identical instances of the same single-node application [Tha05a]. The reasons for the existence of bags-of-tasks are the relatively high network latencies, the complexities of parallel programming models, and the nature of the scientific computational work (e.g., repeated simulations, parameter sweeps). The lack of information about the characteristics of grid workloads hampers the testing and the tuning of existing grids, and the study and evolution of new grid resource management solutions, e.g., new scheduling algorithms. Without proper testing workloads, grids may fail when facing high loads or border cases of workload characteristics. Without detailed workload knowledge, tuning lacks focus and leads to under-performing solutions. Without an understanding of real workloads, current research studies use synthetically generated workflows that are unrealistic, and are therefore limited in scope and applicability. Thus, an important question arises: *What are the characteristics of grid workloads?*

4.1.2 Key Ideas and Selected Results

The answer we give to each of the two research questions formulated in the previous section comprises four components: model design, data analysis, modeling, and generative process. The model design component presents the elements of the model. The data analysis component determines for each model element the statistical properties of the real data. The modeling component leads to the selection of parameter values for the model elements. We conclude each answer with the description of a generative process that uses the model to generate synthetic data for simulation and testing purposes.

The key idea of our approach to answer the first question expressed in the previous section is to analyze and model both the grid dynamics and the grid evolution. In Section 4.3.1, we present an analysis of the grid dynamics. Several other studies characterize or model the availability of environments such as super- and multi-computers [Tan93], clusters of computers [Ach97; Zha04; Fu07], meta-computers (computers connected by a wide-area network, e.g., the Internet, also called desktop grids) [Kon07b; Nur05], and even peer-to-peer (file-sharing) systems [Bha03; Gum03; Pou05]. Our analysis is the first based on long-term availability traces from a multi-cluster grid environment. We analyze and further model four aspects of grid resource availability: the time when resource failures occur, the duration of failures, the number of nodes affected by a failure, and the distribution of failures per cluster. The modeling results show that grid computational resources become unavailable at a high rate, and that nodes have rapidly increasing chances of failing with their uptime, negatively affecting the ability of grids to execute jobs with long duration, even if they are single-processor jobs.

Due the collection in the Grid Workloads Archive (see Chapter 3) of workload traces from production grids, it is now possible to study the characteristics of grid workloads, and answer the second research question from the previous section. The analysis tools of the Grid Workloads Archive have revealed that in contrast to the workloads of tightly coupled high-performance computing systems, a large part of the workloads submitted to grids consists of large numbers of single-processor tasks (possibly inter-related as parts of bags-of-tasks). At first glance, this seems to favor the "conveniently parallel applications" extreme; however, in some grid environments the tightly coupled parallel applications still are important. Thus, the key idea of our attempt is to focus on bags-of-tasks, but to also model the parallel jobs that exist in grid workloads. To this end, we assume that all jobs arrive

grouped as bags-of-tasks; we introduce in Section 4.4.2 a workload model for bags-of-tasks in grids. For the case when the group size is one (that is, the jobs arrive independently), we adapt to grids the Lublin-Feitelson workload model [Lub03], which is the de-facto standard workload model for parallel production environments (Section 4.4.1).

The seminal work in grid workload modeling by Hui Li has demonstrated that long-range dependence and self-similarity appear in grid workloads [Li07f]. We have also shown in our previous work that job arrivals are often bursty [Ios06a] (see also Figure 3.2, Table 3.3, and Figure 3.6). However, similarly to the situation in telecommunication traffic models before the discovery of self-similarity in workloads [Wil00], most workload models used by grid researchers are built around the Poisson arrival process [Li05; Med05; Son05; Li07e; Li07d][1]. While being the most common models employed in computer science, and in particular in queuing theory and its numerous applications [Kle75; Kle76], the Poisson models cannot model self-similar arrival processes [Lel94; Pax95; Err02]. The impact of self-similarity has been shown to be that the queuing backlogs are in general worse with self-similar than with Poisson traffic [Nor94]. The main advantage of our workload model over Poisson models is its potential ability to generate the self-similarity observed in grid environments (i.e., when the workload characteristics such as inter-arrival and job service time are modeled with one of the heavy-tailed distributions described in Section 4.2.3).

4.1.3 Organization of this chapter

The remainder of this chapter is structured as follows. We start with an introduction to the modeling process followed in this chapter in Section 4.2. Then, in Sections 4.3 and 4.4 we present analysis and modeling results for grid resource dynamics and evolution, and for grid workloads, respectively. Finally, in Section 4.5 we survey the analysis and modeling efforts for the resources and for the workloads of large-scale distributed (computing) systems.

4.2 How to Model?

This section presents the method we follow to answer the research questions formulated in Section 4.1.1.

4.2.1 How are models used?

The goal of modeling is to create a representation of the real data that is as close as possible to the original. For each characteristic of the modeled system, the modeling goal is to find a well-known statistical distribution that can be sampled to generate synthetic yet realistic characteristic values. These values are later used in real-world experiments or in simulations. The use of a well-known distribution facilitates in addition mathematical analysis, thus enabling the comparison of real or simulated experiments with theoretical results.

An important quality of a model is its ease-of-use, that is, the complexity of the model should be such as to allow anybody to use and apply the model. To achieve this quality, a model has first to focus on what the typical user knows, e.g., in experimental research on grid scheduling it is unlikely that a user would know much about the third and higher order statistics of the job arrivals (see Section 4.2.2). Second, the modeler must ensure that the values for the model parameters can be easily extracted

[1] We include here the Markov-modulated Poisson processes that can model a self-similar process only when having an infinite number of states. Although they can approach this goal by increasing the number of states, for tractability the number of states must remain in the range of two-four, which leads to the generation of non-self-similar traffic.

from what the typical user has at his disposal, e.g., from directly measurable parts of a real system such as the number of jobs in the system, and from the public-access Grid Workloads Archive data.

4.2.2 Common probability and statistics notions

We assume the reader is familiar with with the notions of probability function, discrete and continuous random variable, cumulative distribution function (CDF), and probability density function (PDF); for exact definitions we refer to the textbook by Soong [Soo04].

First Order Statistics

While the CDF or the PDF completely characterize a random variable X, it is often of interest to find simpler descriptions of the dominant characteristics of X.

Definition 4.1. Define the u *percentile* of a random variable X as the smallest x_u s.t. $u = P(X \leq x_u) = F_X(x_u)$; thus, x_u is the inverse of the function $u = F_X(x)$, with domain $[0, 1]$ and range all the possible x values. Further, define the n *quartile* of a random variable X, with $n \in \{0, 1, 2, 3, 4\}$, as the values that divide the range of the random variable into four equally-sized parts.

The quartiles are usually referred to as Q_n, where Q_1 is also called the *lower quartile* and is equal to the 25^{th} percentile, and Q_3 is also called the *upper quartile* and is equal to the 75^{th} percentile. The special values Q_0, Q_2, and Q_4 are called the *minimum*, the *median*, and the *maximum* values, respectively; the median effectively divides the CDF into two equal parts.

Definition 4.2. Let $g(X)$ be a real-valued function of the random variable X. Then, the *expectation* of $g(X)$ is $E\{g(X)\} = \sum_i g(x_i) f_X(x_i)$ if X is a discrete random variable, and $E\{g(X)\} = \int_{-\infty}^{\infty} g(x) f_X(x) dx$ if X is a continuous random variable. Let $g(X) = X^n$; the expectation $E\{X^n\}$, when it exists, is called the nth *moment* of X and is denoted by α_n.

The first moment, α_1, may be regarded as the center of the mass of the distribution, and can be also called the *average value*, the *mean*, or the expectation of X ($\mu = E\{X\}$). Note that the mean of a distribution may not exist, but the median always exists. The other important moment is the second order (*variance*, $\sigma^2 = E\{(X - \mu)^2\}$).

The *central moments* of a random variable X are its moments with respect to its mean.

Definition 4.3. (i) The nth central moment is defined as $\mu_n = E\{(X - \mu)^n\}$; $\mu_n = \sum_i (x_i - \mu)^n f_X(x_i)$ if X discrete, and $\mu_n = \int_{-\infty}^{\infty} (x - \mu)^n f_X(x) dx$ if X continuous. (ii) The *variance* of X is defined as the second central moment, μ_2 (or σ_X^2, or $var(X)$), and characterizes the dispersion of the random variable from the center of the mass. (iii) The positive square root of μ_2 is called the *standard deviation* of X; in contrast to variance, the standard deviation has the same unit as the mean. (iv) A dimensionless number that characterizes the dispersion relative to the mean is the *coefficient of variation* (CoV), defined as $v = \frac{\sigma}{\mu}$.

4.2.3 Common statistical distributions for computer science

The data that needs to be modeled comes from a real computer system process observed. Thus, it is useful to model it such that the model reminds of the real process. we assume the reader is familiar with the Poisson process [Kle75].

Table 4.1: *Summary of the properties of statistical distributions. The acronym "nscf" is used to denote that no simple closed form is known.*

Distribution	Parameters	PDF (f) and CDF (F)	Mean	Median	Variance
Exponential	λ - inverse scale (or rate)	$f(x;\lambda) = \lambda e^{-\lambda x}$ $F(x;\lambda) = 1 - e^{-\lambda x}$	λ^{-1}	$ln2/\lambda$	λ^{-2}
Erlang	$k > 0$ - shape $\lambda > 0$ - rate k integer	$f(x;k,\lambda) = \frac{\lambda^k x^{k-1} e^{-\lambda k}}{(k-1)!}$ $F(x;k,\lambda) = \frac{\gamma(k,\lambda x)}{(k-1)!} =$ $= 1 - \sum_{n=0}^{k-1} e^{-\lambda x}(\lambda x)^n/n!$	k/λ	nscf	k/λ^2
Hyper-Exponential	n - stages λ_i - rates p_i - probabilities	$f(x) = \sum_{i=1}^{n} p_i f_{X_i}(x)$	$\sum_{i=1}^{n} \frac{p_i}{\lambda_i}$	$\sum_{i=1}^{n} \frac{ln2}{\lambda_i}$	$\sum_{i=1}^{n} \frac{p_i}{\lambda_i{}^2}$
Gamma	$k > 0$ - shape $\theta > 0$ - scale	$f(x;k,\theta) = x^{k-1}\frac{e^{-x/\theta}}{\Gamma(k)\theta^k}$ $F(x;k,\theta) = \frac{\gamma(k,x/\theta)}{\Gamma(k)}$ $\Gamma(z) = \int_0^\infty t^{z-1}e^{-t}dt$ or $\Gamma(n) = (n-1)!$ - gamma function, $\gamma(a,x) = \int_0^x t^{a-1}e^{-t}dt$ - incomplete gamma function	$k\theta$	nscf	$k\theta^2$
Normal	μ - location σ - scale	$f(x;\mu,\sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$ $f(x;\mu,\sigma)$ in integral form	μ	μ	σ^2
Log-Normal	μ - location $\sigma \geq 0$ - scale	$f(x;\mu,\sigma) = \frac{1}{x\sigma\sqrt{2\pi}} e^{-\frac{(ln(x)-\mu)^2}{2\sigma^2}}$	$e^{(\mu+(\sigma^2/2))}$	e^{μ}	$(e^{\sigma^2}-1)e^{2\mu+\sigma^2}$
Pareto	$k > 0$ - shape x_m - scale	$f(x;k,x_m) = \frac{k x_m^k}{x^{k+1}}$ $F(x;k,x_m) = 1 - (\frac{x_m}{x})^k$	$\frac{kx_m}{k-1}, k > 1$	$x_m \sqrt[k]{2}$	$\frac{kx_m^2}{(k-1)^2(k-2)}, k > 2$
Weibull	$\alpha > 0$ - shape $\beta > 0$ - scale	$f(x;\alpha,\beta) = \frac{\alpha}{\beta}(\frac{x}{\beta})^{\alpha-1}e^{-(x/\beta)^\alpha}, x \geq 0$ $F(x;\alpha,\beta) = 1 - e^{-(x/\beta)^\alpha}$	$\beta\Gamma(1+\frac{1}{\alpha})$	$\beta ln(2)^{1/\alpha}$	$\beta^2\Gamma(1+2/\alpha) -$ $\beta^2\Gamma^2(1+1/\alpha)$
A more general 3-parameter form of the Weibull distribution includes an additional waiting time parameter ξ. The formulas for the 3-parameters Weibull can be obtained by replacing x with $x-\xi$ in the above formulas.					

Phase Type vs. Heavy-Tailed distributions

Two classes of distributions occur often in the modeling of computer systems: the phase type distributions and the heavy-tailed distributions. *Phase type distributions* characterize well a system of one or more inter-related Poisson processes occurring in sequence (phases). Examples of such distributions that are commonly used are the exponential, the Erlang, the hyper-exponential, and the Coxian distributions. *Heavy-tailed distributions* have tails that are not exponentially bounded, that is, the probability of the occurrence of events with values far to the "right" of the mean and median is higher than for the exponential distribution, and the variance is infinite. The commonly used heavy-tailed distributions are the Lognormal, the Power, the Pareto, the Zipf, and Weibull.

Commonly used distributions

In the following we present the common distributions used in computer science. Table 4.1 summarizes the properties of these distributions. For detailed descriptions regarding each of these distributions, and on derriving the mathematical formulas presented here, we refer to the textbook of Evans et al. [Eva00].

The *exponential distribution* is used to model many computer-related processes, from the event inter-arrival time in a Poisson process to the service time required by jobs [Kle76]. The main characteristic of this distribution is that it is "memoryless" (e.g., no matter how much time elapsed since the last event arrival, the expected remaining time until the next event remains the same). The main advantage in using this distribution is that a system with independent jobs with exponential service time that arrive with exponential inter-arrival time can be modeled as a basic Markov chain, and the main performance characteristics of the system can be easily extracted.

The *Erlang distribution* was first used to examine the number of telephone calls handled simultaneously at switching stations (a more modern destination would be today's call centers). The Erlang

distribution matches well events in a system where jobs pass through n phases before completion, each phase comprising one exponential distribution (an Erlang distribution with $n = 1$ is an exponential distribution). The Erlang distribution is less variable than an exponential distribution with the same mean.

The *hyper-exponential distribution* is a compound distribution comprising n exponential distributions, each with its own rate. This can be seen as a system with several queues, where request arrival in each queue follows an exponential distribution. The hyper-exponential distributions are more variable than an exponential distribution with the same mean.

The *Coxian distribution* combines the properties of both Erlang and hyper-exponential distributions into a distribution family that can exhibit both low and high variability. While this makes them attractive in a variety of settings, they are also more difficult to use than hyper-exponentials, and have more (free) parameters than the Erlang distribution.

The *gamma distribution* is a good approximation for the time required to observe exactly k (the shape parameter) arrivals of Poisson events. Thus, when $k = 1$ the gamma function characterizes the time to observe one arrival, that is, it is an exponential function. Gamma is a versatile distribution that can attain a wide variety of shapes and scales. Its variance can be much higher than for an exponential function of similar mean when the scale parameter is set accordingly.

The *normal distribution* models well additive processes, that is, a system in which many statistically identical but independent users make requests.

The *log-normal distribution* models random variables with the logarithm normally distributed; it can be thought as the result of a multiplicative process over time, e.g., the long-term return of a savings account. The log-normal distribution has higher variability than the normal distribution for the same mean.

The *Power, Pareto, and Zipf distributions* model well the "Pareto principle", according to which "80% of the population owns 20% of the wealth". This principle, deriving from the power-law phenomenon [New05], occurs in computer system in many situations, including the user interest in a specific file [Bar98; Men03], the UNIX process lifetimes [HB97], web file sizes [Cro97], etc. However, the power-law has also been (wrongly) attributed to any distribution with high right-skew and wide range of values; Newman [New05] argues that the log-normal distribution is often an alternative to empirical distributions believed to be Pareto.

The *Weibull distribution* is a versatile fit for data that are distributed normally, exponentially, or heavy-tailed. This distribution has been used often to model the failures of computer systems [Cas82; Tan93; Sch06; Ios07e], user session inter-arrival time [Bar98; Lel94], data center I/O patterns [Pet96], etc. The Weibull distribution with $\alpha = 1$ is an exponential distribution with rate $\lambda = 1/\beta$. The variability of the Weibull distribution depends on α: when $\alpha > 1$ the variability is below 1, and when $\alpha < 1$ the variability increases quickly as α decreases.

Other Hyper- and Mixed distributions

A hyper-distribution defines a compound distribution that includes several identical distributions (their number is called the number of steps of the hyper-distribution) and a way to discriminate between them. The main advantage of a hyper-distribution is its mathematical tractability. The Hyper-Erlang distribution with two steps was used to model the request inter-arrival times in supercomputers [Jan97].

Other mixtures of distributions are possible, but their use is rare, as they raise the complexity of the mathematical analysis. A Lognormal-Pareto mix was used for characterizing streaming media [Jin01], a Gamma-Pareto mix was used as heavy-tailed function for characterizing multiplayer online games

traffic [HS03], an an Erlang-Coxian mix was used for analyzing cycle stealing systems [Oso06].

4.2.4 Our Modeling Process

Throughout this chapter we follow the modeling process that is depicted in Figure 4.1. This process involves three main steps: design, analysis, and modeling.

In the design step, first the model components are designed. This part involves re-using model components from previous models and also adding the model components that are specific to the system at hand. Second, the important model components are selected from the set of all components. The main focus in this sub-step is the ease-of-use of the model (see Section 4.2.1). Finally, the important model components are simplified, to avoid over-fitting and again with the ease-of-use in mind.

In the analysis step, first the characteristics of each model component are evaluated, with the results being the common probability and statistics summary presented in Section 4.2.2. Second, the evolution of the characteristics over time is observed; large variations over time may lead to a refinement of the model design to introduce more time-variant elements (e.g., special components for daily cycles). Third, the correlations between various characteristics are evaluated; the pair-wise correlations are the most commonly studied. The existence of a provable strong correlation necessarily leads to extending the model with aspects that describe the correlation.

In the modeling step, first the candidate distributions are selected based on the results of the analysis (especially the mean, the median, and the variance). The candidate distributions are then fit against the real data. There are two main methods for fitting: moment matching algorithms, and the Maximum Likelihood Estimation (MLE) method [Ald97]. Moment matching algorithms attempt to perform the fitting based only on the analysis results and on the parameters of the candidate distributions (e.g., a fit to the normal distribution would assign the location parameter μ the mean of the real data). While many moment matching algorithms that match closely the first three moments of

1. Design
   (a) Design model components.
   (b) Select important model components.
   (c) Simplify model components (eliminate over-fitting).
2. Analysis
   (a) Determine the characteristics of the model components from real data.
   (b) Determine the evolution of the characteristics over time.
   (c) Determine the correlation between various characteristics.
3. Modeling
   (a) Select candidate distributions based on analysis results.
   (b) Fit candidate distributions against real data.
   (c) Evaluate the goodness-of-fit and eliminate bad candidate distributions.
   (d) If needed, select the "average system" model when multiple candidate distributions remain.

Figure 4.1: *An overview of the modeling process.*

any phase-type distribution exist [Alt85; Joh89; Joh90; Tel03; Bob05; Oso06], the problem of fitting the first three moments of heavy-tailed distributions remains open. Thus, we use in this work the second approach, the MLE method, which delivers good accuracy for the large data samples specific to workload traces. After finding the best fits for each candidate distribution, goodness-of-fit tests are used to assess the quality of the fitting for each distribution, and to establish the best fit. For each candidate distribution with the parameters found during the fitting process, we formulate the hypothesis that the empirical data are derived from it (*the null-hypothesis* of the goodness-of-fit test). We use the Kolmogorov-Smirnov test (KS-test) [Lil69] for testing the null-hypothesis. The KS-test statistic D estimates the maximal distance between the CDF of the empirical distribution of the input data and that of the fitted distribution. The null-hypothesis is rejected if D is greater than the critical value obtained from the KS-test table. The KS-test is robust in outcome (i.e., the value of the D statistic is not affected by scale changes, like using logarithmic values). The KS-test has the advantage over other traditional goodness-of-fit tests, like the t-test or the chi-square test, of making no assumption about the distribution of the data (note: Pearson's chi-square test is applied to binned data (e.g., a data histogram), but the value of the test depends on the how the data is binned). The KS-test can disprove the null-hypothesis, but *cannot* prove it. However, a lower value of D indicates better similarity between the input data and data sampled from the theoretical distributions. We use this latter property to select the best fits.

As an alternative to the MLE/KS-test approach, we perform a standard regression analysis to fit the real (trace) data for each characteristic to a linear or to an exponential function [Sch86]. We then compute the Pearson coefficient of determination, R^2, which indicates the proportion of the variability in the data set that is explained by the fitted model. The range of values for R^2 is [0,1]; a low value of R^2 (e.g., below 0.6) means that the model is not a good fit; conversely, R^2 higher than 0.9 gives significant confidence in the fit between the model and the real data. Similarly to the KS-test, R^2 cannot demonstrate the correlation between the model and the data. The use of R^2 is limited in practice by its strong requirements: the number of model parameters needs to be small (e.g., one-two), the sensitivity to data outliers is high, etc.

When multiple real system data traces are available for fitting, it is possible that there is no single candidate distribution that provides the best fit. In this case, the modeling step ends with the selection of the best candidate distribution as the "average system" model, based on the goodness-of-fit values. For each model characteristic, a candidate distribution that has the lowest average D value over all traces considered is selected as the average system fit. When for two or more candidate distributions the difference of their D values is below 0.01, the selected distribution is that which fits best the most empirical traces from which the average system model is derrived. The data for each trace are then fit independently to the candidate distribution, resulting in a set of best fit parameters. The average of this set are taken as the parameters of the average system.

4.3 Resource Dynamics and Evolution

In this section we present our analysis and modeling results for grid resource dynamics and evolution.

4.3.1 A Model for Resource Dynamics

We being with the study of the characteristics of resource dynamics in multi-cluster grids. We assume that in the short-term there are no changes in the performance of individual resources. Thus, for the remainder of this section we use the terms resource dynamics and resource availability interchangeably.

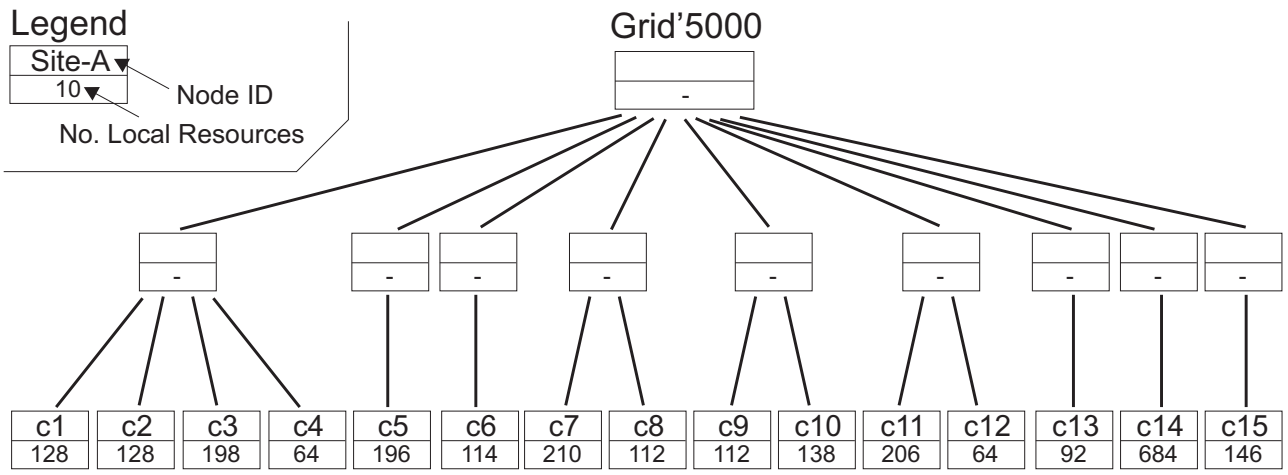


Figure 4.2: *The structure of Grid'5000: the sites, the clusters, and the number of processors per cluster.*

Throughout this section we use the following terminology. We define resource availability as the state of a resource that is able to execute user jobs. Conversely, we define a failure as the state of a resource in which the resource cannot execute the user jobs; to mark the antagonism between the availability and failure states we also call the latter unavailability. We define the failure event (the unavailability event) as the beginning of a transition of the resource state between available and unavailable, and the repair event (the availability event) as the end of the transition of the resource state between unavailable and available. We define the failure inter-arrival time (IAT) as the time elapsed between two consecutive failure events, when the failure events are ordered by the time of their occurrence. The mean time between failures (MTBF) is a typical characteristic of resource availability [Ebe97]. We define the duration of a failure as the time elapsed between the occurrence of the failure, and the recovery of the resource affected by the failure. Our definition is similar to that of recovery time used in [Ebe97] or of time-to-repair used in [Tan93; Zha04]. We define in a similar way the availability duration. We also investigate the notion of groups of unavailabilities, which we call *correlated failures*. Let $TS(\cdot)$ be the function that returns the time stamp of an event, either failure or repair. Then we define a correlated failure with time parameter Δ as the maximal set of failures (ordered according to increasing event time) in which for any two successive failures E and F, $TS(F) \leq TS(E) + \Delta$. This definition of correlated failures allows us to investigate how a single failure (either a node or a set of nodes) can affect other nodes by triggering other failures. Last, we define the size of a failure as the number of failures of the correlated failure to which the failure belongs.

Model Overview

Our model of resource availability in multi-cluster grids considers four aspects: 1) the times when resource failures occur, 2) the duration of failures, 3) the number of nodes affected by a failure, and 4) the distribution of the number of failures per cluster. Compared to traditional resource availability models [Tan93; Gra90; Zha04], ours adds the necessary link between the failures and the clusters where they occur.

We study availability data taken from a large-scale multi-cluster grid: Grid'5000 [Bol06]. Grid'5000 is an experimental grid platform consisting of 9 sites geographically distributed in France. Each site

Table 4.2: *Summary of the Grid'5000 availability and workload traces.*

		Observed				
Period	Clusters	No. Nodes	Avail. Events	No. Users	User Jobs	Work [CPUy]
05/'05-11/'06	15	1296	600k	445	750K	585

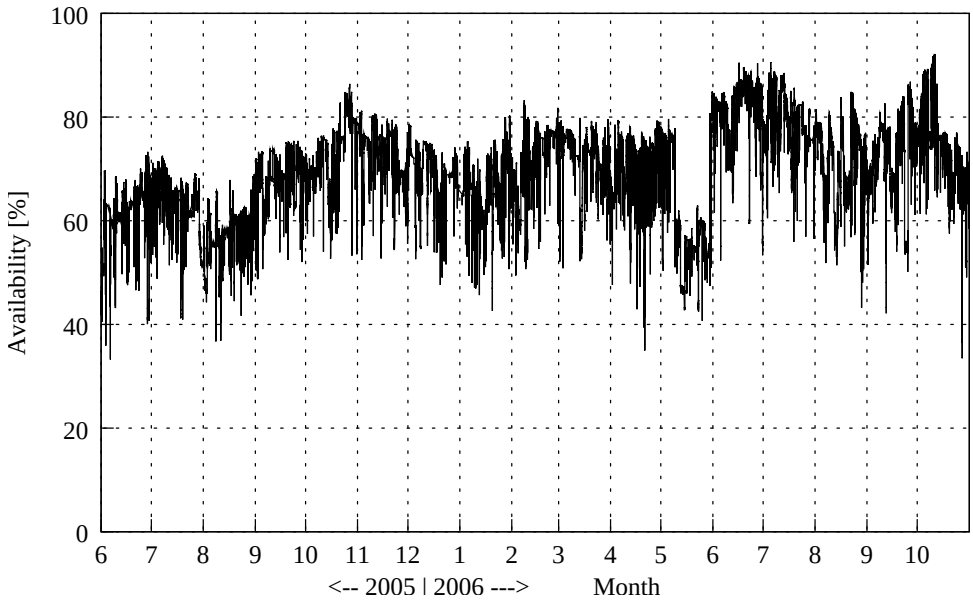


Figure 4.3: *Availability of resources in Grid'5000 at the grid level over the time.*

comprises one or several clusters, for a total number of 15 clusters and over 2,500 processors. Each cluster comprises a set of dual-processor nodes; we use in this section the terms node and resource interchangeably. Figure 4.2 shows the structure of Grid'5000. The number of processors per cluster is valid for 12 December, 2006. We have obtained availability traces recorded by all batch schedulers handling Grid'5000 clusters (OAR [Cap05]), from mid-May 2005 to mid-November 2006 (over one year of data). The traces record availability events with the following data: the node whose availability state changes (i.e., a node becomes either available or unavailable (a failure occurs)) and the moment of the event. Together, these traces comprise more than half a million individual availability events. Between mid-2005 and mid-2006 the number of processors in Grid'5000 has steadily grown from about a thousand to over two thousand. Table 4.2 summarizes the content of the Grid'5000 availability traces, and of the corresponding workload trace for this period. We refer the reader to the Grid Workloads Archive [Ios08d] for more details about the workload trace of Grid'5000.

We follow the steps of the modeling process described in Section 4.2.4, in turn model design, analysis, and modeling. For the analysis part, we consider three levels of data aggregation: the grid, the cluster, and the node level. The grid level aggregates (un)availability events for all the nodes in the grid. Similarly, the cluster level aggregates (un)availability events for all the nodes in the cluster.

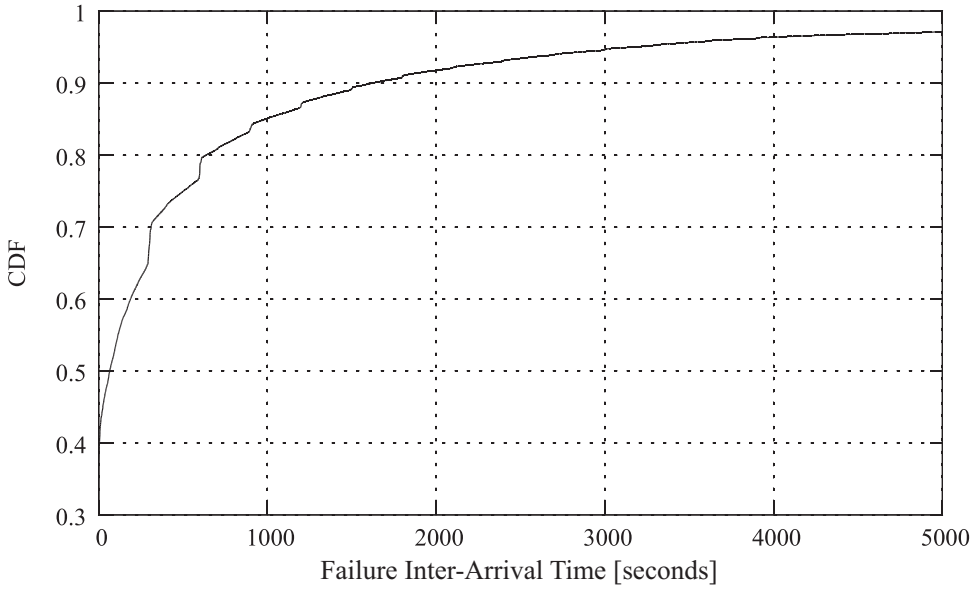


Figure 4.4: *Cumulative distribution function (CDF) of the failure inter-arrival time of Grid'5000.*

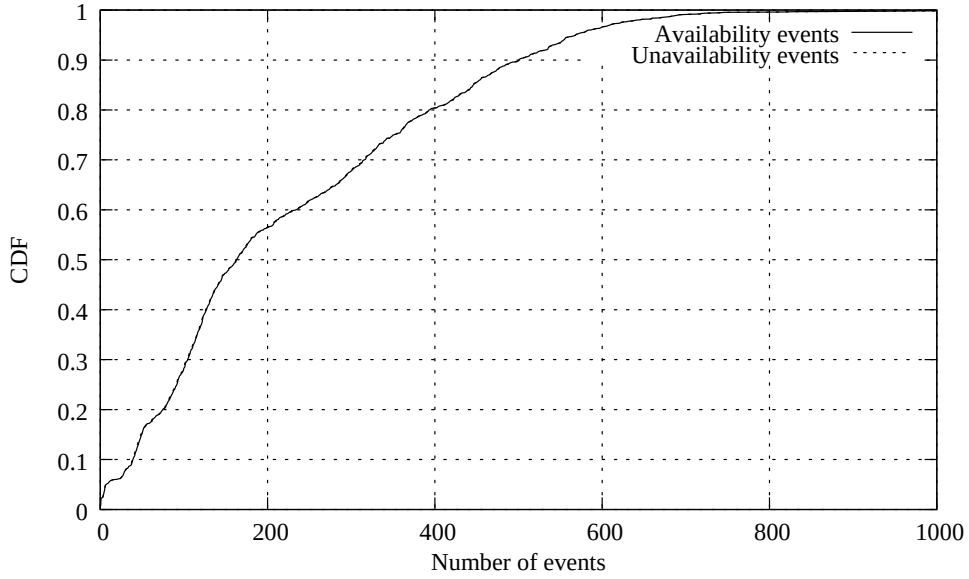


Figure 4.5: *Cumulative distribution function (CDF) of the number of availability state change events per node for all the nodes of Grid'5000 during a period of 540 days. The two curves are indistinguishable: failures are always followed by repairs.*

Analysis Results

Figure 4.3 shows the availability of resources in Grid'5000 at the grid level[2]: on average, 69% (±11.4[3]), with a maximum of 92% and a minimum of 35%. The MTBF at the grid level is 744±2631 seconds

[2]May 2005 is not shown as the availability information of clusters started being recorded at different dates for each cluster.

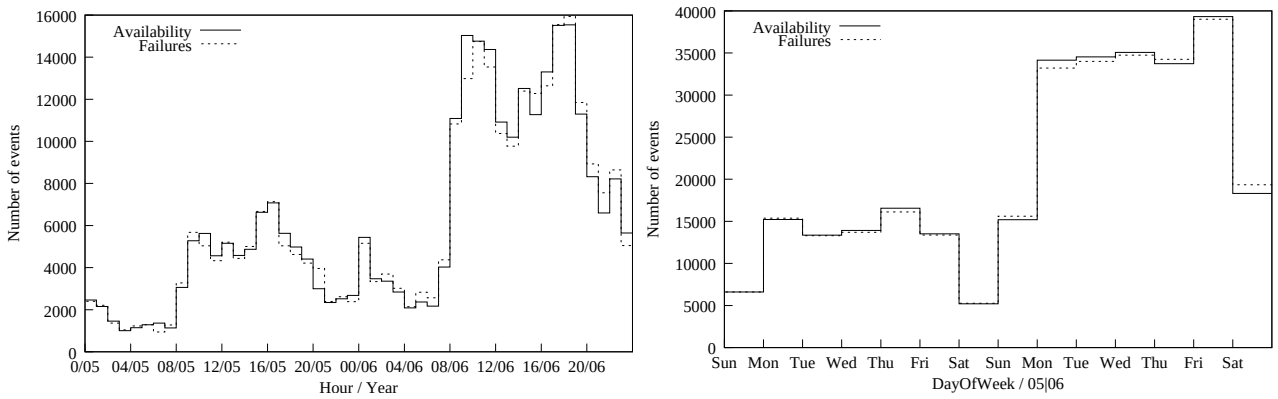


Figure 4.6: *Daily (top) and weekly (bottom) patterns of the number of availability and failure events. For both graphs, the left parts depict data for 2005, and the right parts depict data for 2006.*

(the average is around 12 minutes). Figure 4.4 shows the CDF of the failure inter-arrival times at the grid level. Over 85% of the failures occur within 1200 seconds (20 minutes) of the previous failure. Thus, the ability of the grid to run grid-wide parallel jobs with a runtime above 20 minutes is questionable. The resource availability at the cluster level varies from 39% to 98%. We are also interested in the average cluster MTBF (the average of the MTBF value for each cluster). This value cannot be computed by straightforwardly multiplying the MTBF at the grid level with the number of clusters (15 in Grid'5000), as each cluster has a different size. The values of the MTBFs of all clusters are 18404±13848 seconds (the average is around 5 hours). As expected, this value is much higher than the MTBF at the grid level.

At the node level, we find that, on average, a node fails 228 times for a trace that spans over 548 days. However, some nodes fail only once or even never during this period. Figure 4.5 shows the CDF of the number of state change events per node, for all the nodes of Grid'5000. Half of the nodes exhibit 200 failure events or less. Conversely, a tenth of the nodes exhibit 500 failure events or more, and less than one percent of the nodes exhibit more than 1000 failure events.

The average availability duration of a node is 161,315±113,678 seconds (the average is around 45 hours), whereas the average failure duration is of 51,533±48,267 seconds (the average is around 14 hours). In both cases, the value of the coefficient of variation (the ratio between the standard deviation and the average) is quite high (almost 1). Note that for the failure duration, values may include night hours during which administrators of sites of a grid are not available. Furthermore, some node failures may require, for instance, a processor or a memory slot to be replaced, which adds to the failure duration.

In addition to the analysis of the basic statistics at the node level, we also look to establish whether there are time patterns in the occurrence of (un)availability events. Figure 4.6 shows the daily and weekly patterns in the number of occurrences for these events in the years 2005 and 2006. The daily values indicate that the availability state change events follow a similar pattern to that of the execution of jobs: a peak during work hours (more events from 8am to 8pm) and a weekly pattern (fewer events occurring during weekends). The scale difference for the number of failures between the years 2005 and 2006 is correlated with the increase in size of Grid'5000; given the peak values, the number of failures per processor has grown slightly from 2005 to 2006.

[3] ± stands for standard deviation.

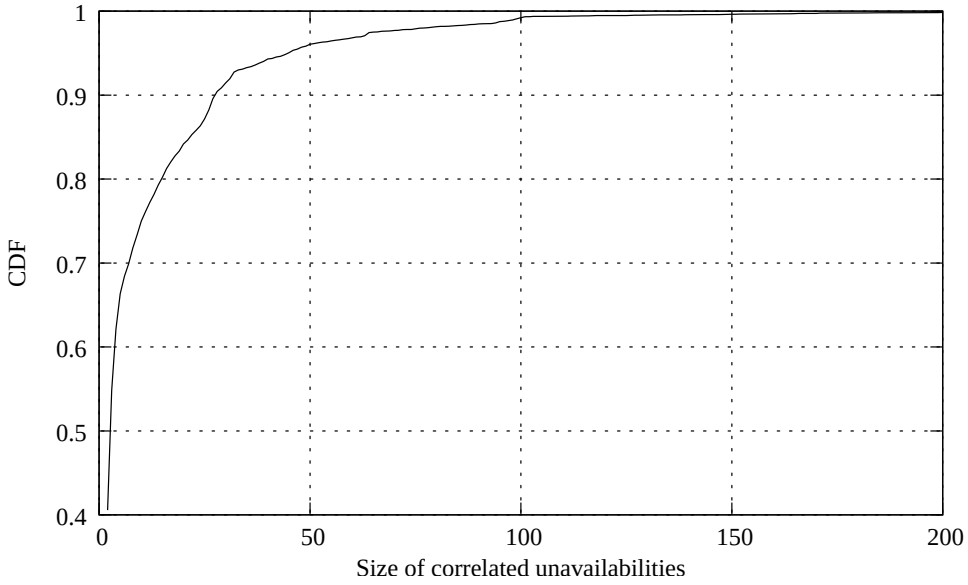


Figure 4.7: *CDF of the size of correlated failures for* $\Delta = 60s$.

We now analyze the correlated failures at the grid level, that is, when constructing the correlated failure we take into account all the failure events, regardless of their origin (cluster where they occur). In our analysis, we vary the parameter Δ defining correlated failures from 1 to 3600 seconds. However, we present only selected results for $\Delta = 60s$ as: 1) results for lower values of Δ (i.e., 1, 10 and 30 seconds) are similar, and 2) this value is twice a commonly used value for timeout/delays in network operations (30 seconds). The number of correlated failures with size higher than one is 7,500, for a total of around 85,000 failure events. Therefore, the correlated failures represent around 30% (**!**) of the total number of failures events in the trace. Figure 4.7 shows the CDF of the size of the correlated failures for $\Delta = 60s$. Our analysis shows that, on average, the size of a correlated failure is 11.0±21.0, with a maximum of 339. This latter value is little less than the size of the largest cluster, which consists of 342 nodes. In addition, we have analyzed the number of sites involved in a correlated failure: its range is 1.06±0, with a maximum of 3 (for $\Delta = 60s$), which means that correlated failures generally do not extend beyond a single site.

We conclude the analysis by summarizing the basic statistical properties illustrated during the analysis. Table 4.3 shows the minimum, maximum, mean, and median values, and the 1^{st} and the 3^{rd} quartiles of the Grid'5000 resource availability trace. The results for the failure inter-arrival time and for the failure size (rows A and C, respectively), show that the ratio between the mean and the median is relatively homogeneous across clusters. This indicates that a single distribution can be used across all clusters to model the real failure inter-arrival time data and the failure size data, respectively. Depending on the ratio between the mean and the median of the duration of failures, there are two main classes of clusters: class 1 with a ratio of about 1:1 (clusters c1 and c8), and class 2 with a ratio of about 1:6-9 (the remaining clusters). This may indicate the need for separate distributions for each of the classes, or for a distribution with more degrees of freedom, e.g., hyper-exponential or hyper-gamma. However, class 1 contains only clusters where few jobs have been submitted over the duration considered for this study. Therefore, we choose to disregard this class, and use a single

Table 4.3: *Summary of the basic statistical properties of the Grid'5000 availability data. Values higher than 10000 have been reported as ">10k". The omitted Max rows (for A and B data) contain only ">10k" values.*

	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C15
A. Failure inter-arrival time [s]															
Min.	45	19	3	48	36	23	21	150	54	9	36	84	109	9	4
Q1	3513	1165	1150	604	1357	1500	1640	3002	1709	1005	1509	533.5	601	2356	901
Median	7258	3388	1794	1207	3903	4764	4584	5213	4311	4225	4012	1883.5	1202	4660	1517
Mean	6527	4608	2841	2817	4927	5399	5202	5734	5239	4640	4864	3492.3	3178	5369	3041
Q3	>10k	8702	3475	3600	>10k	9824	9570	9783	9703	8156	8142	6432.8	4397	8582	3495
B. Failure duration [s]															
Min.	9	0	0	0	0	4	2	2	1	7	0	2	0	0	0
Q1	971	122	195	151	97	159	126	358	247	116	152	86	125	106	175
Median	4475	264	225	303	100	163	157	1432	259	121	163	110	181	124	201
Mean	5022	1907	1047	2025	1655	554	772	1301	1103	418	561	1272	421	561	995
Q3	9631	1833	1080	2445	679	169	479	1763	294	134	179	1157	339	162	1304
C. Failure size [number of processors]															
Min.	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Q1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Median	1	1	2	1	1	1	1	1	1	1	1	2	1	1	1
Mean	9.4	7.4	7.9	3.2	4.9	5.9	6.1	5.0	5.8	5.8	5.4	6.2	3.4	2.6	6.4
Q3	4	7	10	1	2	2	2	2	1	2	2	6	1	2	4
Max.	97	200	165	228	319	267	185	336	200	98	100	43	295	339	67

distribution to model the failure duration.

Modeling Results

We now present the modeling results obtained for each of the four elements of our model for resource availability in multi-cluster grids: failure inter-arrival time, failure duration, failure size, and failure location.

We first perform a graphical analysis of fitting the availability data. This allows us to eliminate from the modeling process the distributions which clearly do not fit to the data. Figure 4.8 shows the graphical analysis of the fit between the CDF of the logarithm of inter-arrival time of failures, for one cluster. Clearly, the exponential distribution is not a good fit. The other distributions yield reasonably close results, with the Weibull distribution looking especially promising. Figure 4.9 shows the graphical analysis of the fit between the CDF of the logarithm of failure duration, for one cluster. The Weibull, log-normal and even normal distributions look promising. Figure 4.10 leads to similar conclusions. We therefore select for detailed model fitting the normal, the lognormal, the Weibull, and the gamma distributions.

We now attempt to fit statistical distributions to our availability data. We fit the above mentioned distributions using the Maximum Likelihood Estimation (MLE) method. Then, we perform goodness-of-fit tests to assess the quality of the fitting for each distribution, and to establish a best fit for each of the model parameters. For each distribution, we formulate the hypothesis that the Grid'5000 availability data comes from that distribution, whose parameters are found during the fitting process (*the null-hypothesis* of the goodness-of-fit test). We use the Kolmogorov-Smirnov test for testing the null-hypothesis. We find that the best fits for the failure inter-arrival time, the failure duration, and the failure size are the Weibull, the lognormal, and the Weibull distributions, respectively. Table 4.4 shows the parameter values of the best fit of the best model for the Grid'5000 availability data per cluster and for the overall system, respectively. The results for the failure inter-arrival time are alarming: the shape parameter α of the Weibull distribution is larger than 1, which indicates an increasing hazard rate function (the frequency with which a system or component fails, provided that it has survived so far [Ebe97]). This indicates that the longer a computing node stays in the system, the higher the

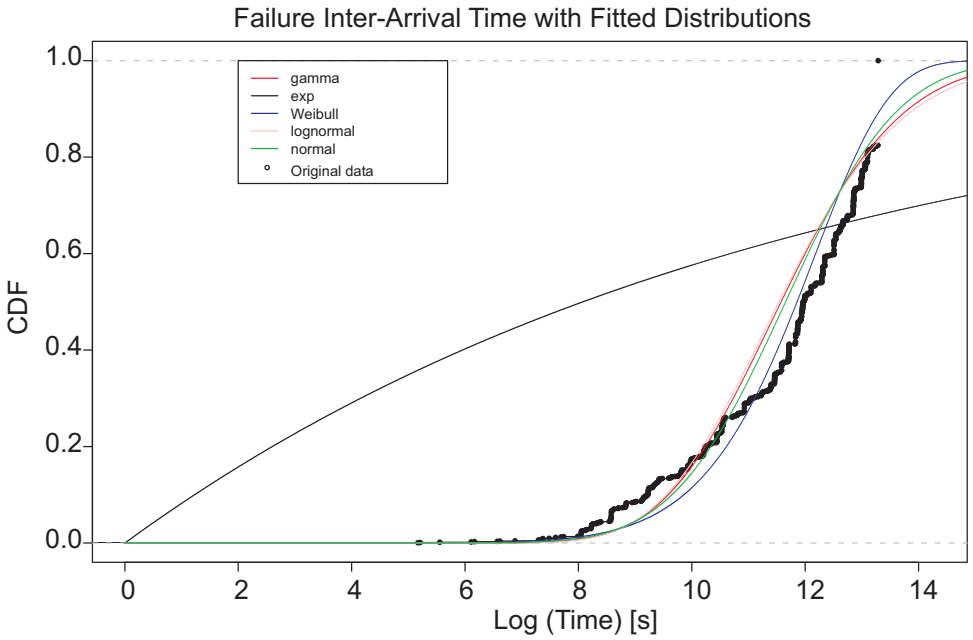


Figure 4.8: *Sample fit between the CDF of the logarithm with base two of the empirical failure inter-arrival time and various statistical distributions.*

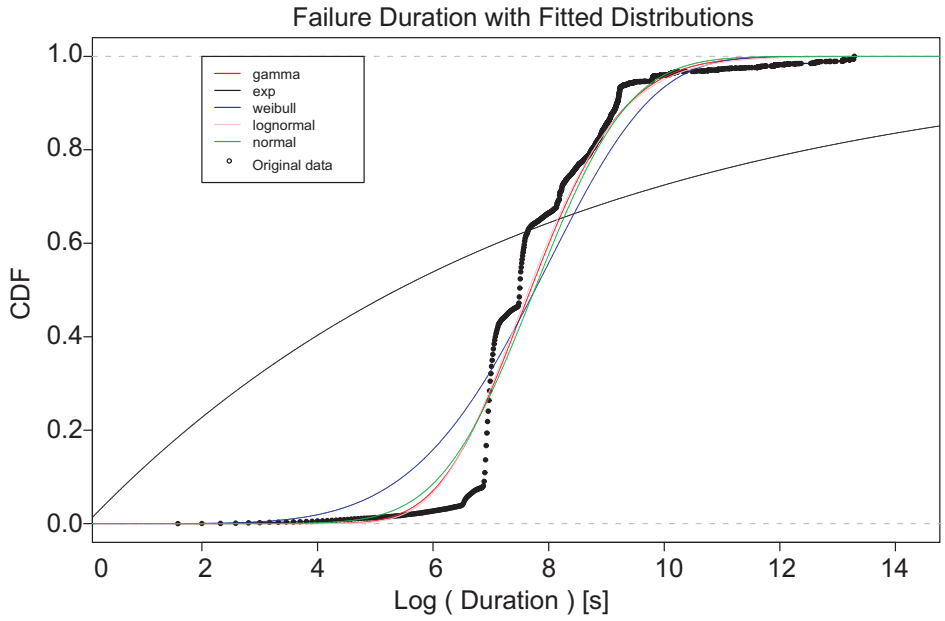


Figure 4.9: *Sample fit between the CDF of the logarithm with base two of the empirical failure duration and various statistical distributions.*

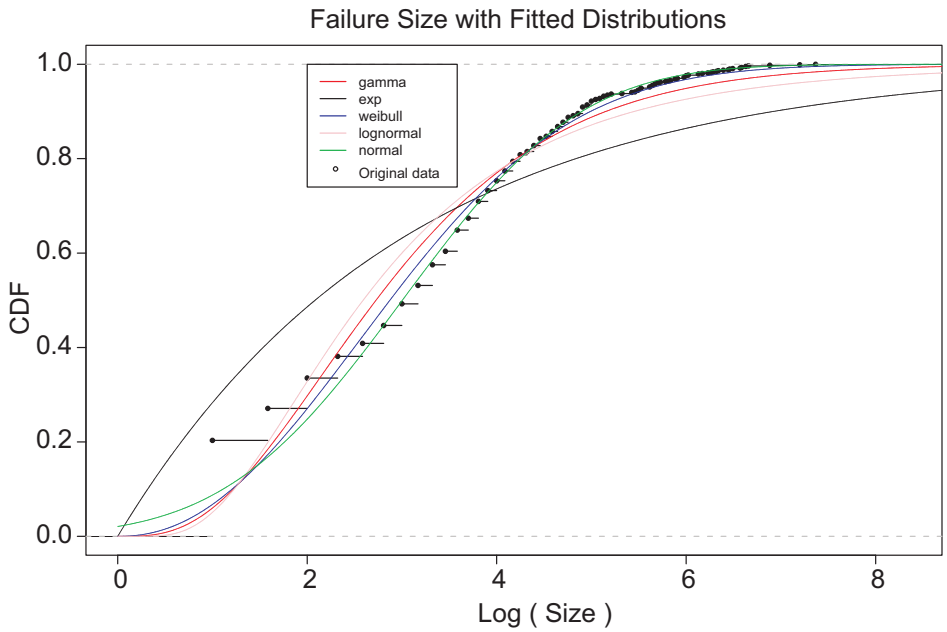


Figure 4.10: *Sample fit between the CDF of the logarithm with base two of the empirical failure size and various statistical distributions.*

probability of the node's failure, preventing long jobs from finishing.

To model the location of the occurrence of failure events we first obtain the empirical distribution characterizing the fraction f_c of failures occurring at cluster c, from the total number of failures occurring in the system. We prefer to investigate the number of failures per cluster, and not the number of failures per cluster normalized by the cluster size, because the latter assumes that the failures are due to the larger cluster sizes, which is not supported by our empirical findings. Table 4.5 shows for each cluster the numbers of failures and the values of f_c[4]. This model assumes that there exists no correlation between the occurrence of failures at different clusters. Since the clusters are located separately, and the network between them has numerous redundant paths, we see no evidence to consider otherwise.

Then, we try to fit a well-known distribution to the empirical distribution. To this end, we order the clusters by their number of failures in decreasing order, and try to find the distribution that matches the empirical distribution. Let $g(\cdot)$ be the function describing the empirical distribution of these ranked data, so $g(r) = n$ for the cluster of rank r with n failures. We find that the exponential distribution $n = \alpha \times e^{\beta \times r}$ fits well the empirical distribution for $\alpha = 79156$ and $\beta = -0.2626$. Similarly, we find that a power distribution $n = \alpha \times r^{\beta}$ fits well the empirical distribution for $\alpha = 149185$ and $\beta = -1.4703$. However, the exponential distribution underestimates the probability of the first ranked cluster by around 50%, while the power distribution overestimates the same value 20%.

The model of the distribution of failures per site is constructed similarly to the model of the distribution of failures per cluster, taking into consideration which cluster belongs to which site. By adding the values corresponding to all the clusters belong to the same site, we obtain the empirical distribution characterizing the fraction f_s of failures occurring at site s, from the total number of

[4]Note that these numbers depend on the date when clusters were made available to users.

Table 4.4: *Fitted model parameters for the Grid'5000 availability data per cluster and for the whole system.*

A. Failure inter-arrival time [s]		
	Weibull	
Cluster	α	β
C1	13.32417	12.76841
C2	8.82691	12.14668
C3	9.49738	11.50066
C4	7.19553	11.29538
C5	7.88283	12.18533
C6	9.64997	12.41524
C7	9.96752	12.37562
C8	12.94117	12.58577
C9	10.71829	12.41374
C10	7.79759	12.09732
C11	10.21044	12.29004
C12	6.60194	11.54027
C13	7.29009	11.47595
C14	11.88544	12.47498
C15	8.11074	11.49423
Grid'5000	9.66772	12.23796
B. Failure duration [s]		
	Lognormal	
Cluster	μ	σ
C1	2.40913	0.22558
C2	2.14771	0.27527
C3	2.15353	0.19601
C4	2.14365	0.34139
C5	2.03851	0.29664
C6	2.03350	0.13773
C7	2.07713	0.19244
C8	2.26752	0.17555
C9	2.13891	0.18302
C10	1.97464	0.14232
C11	2.03296	0.14878
C12	2.07510	0.28043
C13	2.03505	0.16220
C14	1.99036	0.17595
C15	2.13636	0.22205
Grid'5000	2.33916	0.26363
C. Failure size [number of processors]		
	Weibull	
Cluster	α	β
C1	1.71898	3.60045
C2	2.01629	3.40298
C3	2.17394	3.40015
C4	1.45947	2.45525
C5	1.75449	2.92801
C6	1.69720	3.08240
C7	1.58754	3.07541
C8	1.63994	2.87220
C9	1.59014	3.31293
C10	1.78423	3.38625
C11	1.63853	3.22535
C12	2.42596	3.28213
C13	1.72418	3.17537
C14	1.63197	1.80350
C15	1.81717	3.19713
Grid'5000	1.58526	2.61400

failures occurring in the system. A similar process can be followed using the exponential and the power distributions that fit the empirical distribution at the cluster.

Using the Model

We now present the use of our resource availability model to generate synthetic resource failure data for simulation and testing purposes. Figure 4.11 shows an overview of this process. For each new failure, the cluster where the failure occurs, the moment of the failure event, the failure duration, and the failure size, are generated using the distributions and the parameter values from Tables 4.5 and 4.4,

Table 4.5: *Numbers and fractions of failures per cluster and per site. Additional horizontal separators group clusters belonging to the same site. See text for distributions fitting this data.*

Cluster	No. Failures	f_c	Site	No. Failures	f_s
C1	13003	0.0442	S1	69076	0.2348
C2	25432	0.0865			
C3	25892	0.0880			
C4	4749	0.0161			
C5	8222	0.0279	S2	8222	0.0279
C6	5750	0.0195	S3	8222	0.0279
C7	34546	0.1175	S4	37422	0.1272
C8	2876	0.0097			
C9	5349	0.0181	S5	7510	0.0254
C10	2161	0.0073			
C11	12901	0.0438	S6	14024	0.0476
C12	1123	0.0038			
C13	21131	0.0718	S7	21131	0.0718
C14	123353	0.4197	S8	123353	0.4197
C15	7417	0.0326	S9	7417	0.0326

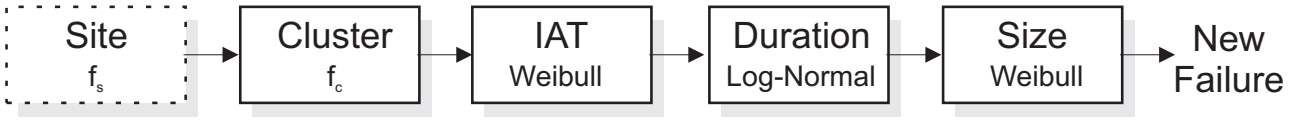


Figure 4.11: *Generating resource failure data using the proposed availability model.*

respectively. The site where the failure occurs can be generated using the values of the parameter f_s from Table 4.5.

4.3.2 A Model for Resource Evolution

We have introduced in Section 4.3.1 a model for grid resource dynamics, which characterizes the short-term changes in resource availability status. We now turn our attention to the long-term grid resource evolution.

Model Overview

Grid evolution refers to the evolution of the physical resources (i.e., number, performance), and of the logical resources (i.e., the middleware). The evolution of processor performance has been investigated by Kee et al. [Kee04]. We have given evidence that major system middleware changes occur every six to twelve months, and that the systems have a production cycle of two-three years in Figure 3.5 (Section 3.3.5). We consider in this section two new aspects of the evolution of the system resources: the number of clusters in the system and the number of processors per cluster.

As data sources we consider historical data from the WLCG grid[5] and from the Top500 Supercomputers List[6]. The WLCG grid comprises mostly clusters of common off-the-shelf computers shared by universities and by research laboratories. The Top500 Supercomputers List comprises the top 500 machines (by computing power) whose data can be made public. The WLCG data contains daily information about the state of the WLCG clusters, including the (maximum) number of processors for each

[5]For all the WLCG-related information used in this section, the data sources are http://goc.grid.sinica.edu.tw/gstat/ and http://pcalimonitor.cern.ch/reports/.

[6]For all the Top500 Supercomputers List-related information used in this section, the data source is http://www.top500.org.

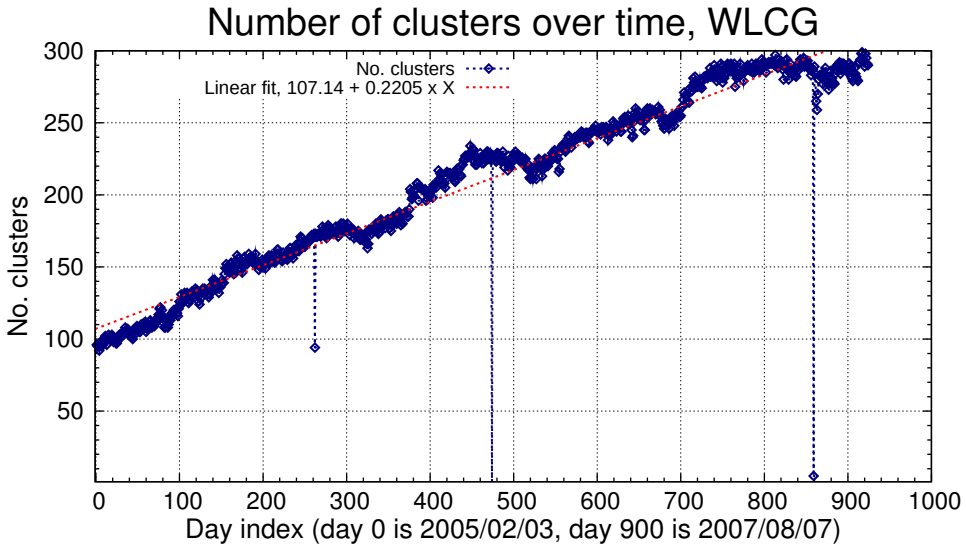


Figure 4.12: *The evolution of the number of clusters over time for clusters in WLCG.*

of the clusters that have been active within the WLCG for the past day. The Top500 Supercomputers List is published twice per year, and its data include the number of processors for each machine listed. For the evolution of the number of clusters we use the public WLCG historic data. For the study of the evolution of the cluster size we use the public historic data for the Top500 Supercomputers List (only the clusters) and of the WLCG.

We follow the modeling process described in Section 4.2.4. We fit two types of curves to the empirical data: linear and exponential. We use Pearson's coefficient of determination, R^2, to assess the goodness-of-fit. We find that for the WLCG data the best fit for both the number of clusters and for the median cluster size is linear; for the Top500 Supercomputers List the best fit for the median cluster size is exponential.

Analysis and Modeling Results

We first analyze the evolution of the number of clusters. Figure 4.12 shows the evolution of the number of clusters over time, for clusters in WLCG. In less than three years, the WLCG system has grown from 100 to 300 clusters; a linear fit $y = y_0 + m \times x$ gives a Pearson's coefficient of determination $R^2 = 0.93$ for $y_0 = 107.14$ and $m = 0.2205$, where $y(x)$ is the number of clusters in the system when x days have elapsed since the moment when the number of clusters reached a value of around 100 clusters (the intercept y_0). Similarly, the evolution of the number of clusters in the Top500 Supercomputers List, depicted in Figure 4.13 has grown from 28 to 373 between the days 1000 and 3450, starting with 1997/11/01; a linear fit $y = y_0 + m \times x$ gives a Pearson's coefficient of determination $R^2 = 0.95$ for $y_0 = -125.60$ and $m = 0.1518$.

To conclude, we find that linear growth with a rate of one-two clusters added every week models well the evolution of the number of clusters.

The evolution of the cluster size follows the patterns depicted in Figures 4.14 and 4.15. For WLCG, a system built mostly with common off-the-shelf computers, the system growth is determined by the growth of the largest system size. The mean is much higher than the median and falls outside the

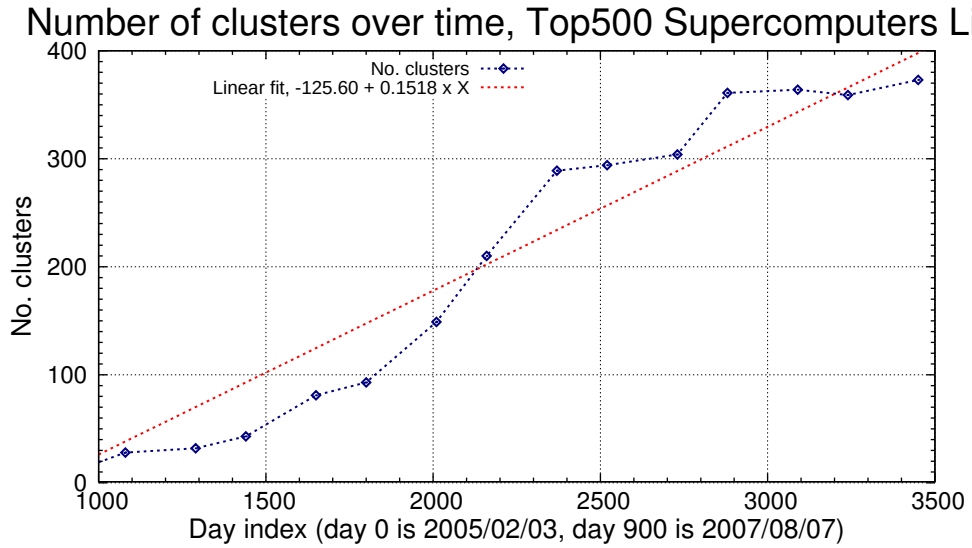


Figure 4.13: *The evolution of the number of clusters over time for cluster systems in the Top500 Supercomputers List.*

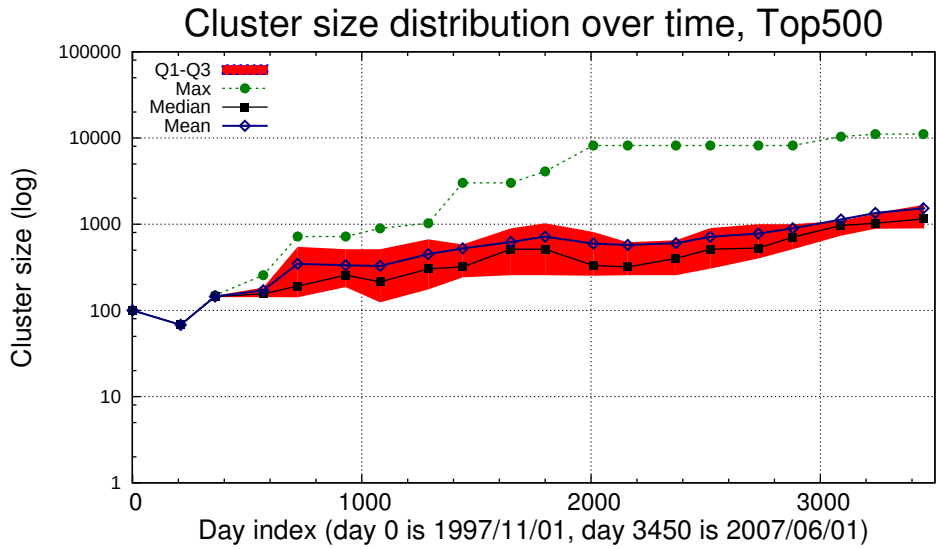


Figure 4.14: *The evolution of cluster sizes over time for clusters in WLCG. (Q1 and Q3 denote the first and the third quartile, respectively).*

inter-quartile range (Q1-Q3); this further confirms that the largest system size is dominant for the WLCG size evolution. The median system size growth is fit the line $y = y_0 + m \times x$, with $y_0 = 21.3$ and $m = 0.0096$, where $y(x)$ is the median system size x days after 2005/02/03; given the low value of m, the median system size can be considered constant for periods of three months or less.

For the high-end systems represented in the Top500 Supercomputers List the trend, depicted in Figure 4.15, is reverse: the median system size is the fastest growing from the basic statistics. Its growth is fit by an exponential curve $y = \alpha \times e^{\beta \times x}$, where $y(x)$ is the median system size at x days

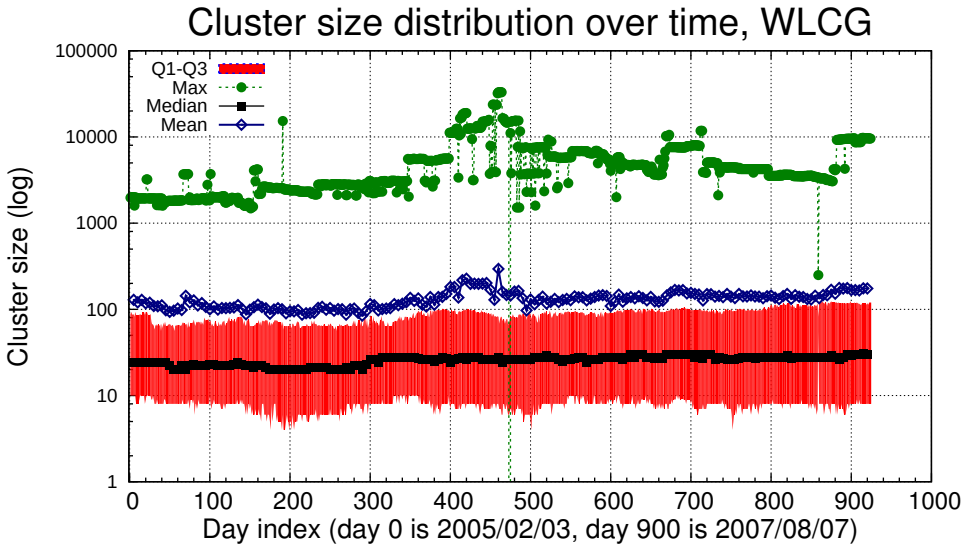


Figure 4.15: *The evolution of cluster sizes over time for cluster systems in the Top500 Supercomputers List. (Q1 and Q3 denote the first and the third quartile, respectively).*

after 1997/11/01; $\alpha = 93.431$ and $\beta = 0.1225$ give a Pearson's coefficient of determination $R^2 = 0.90$.

To conclude, we find that the evolution of the cluster size depends on the type of clusters that have been used to build the grid. For grids built with common off-the-shelf computers, this evolution element is well modeled by linear growth of the median system with the rate of one node every 100 days (**!**). For grids built with high-end systems, this evolution element is well modeled by an exponential curve; the median cluster size grows on average with two-three nodes per week.

4.4 Workload

In this section we present analysis and modeling results for selected grid workload aspects. We have argued in Section 4.1.1 that grid workloads are diverse, and some may include both parallel and single-processor jobs, while others are entirely made up of the latter. We have also indicated in Section 4.1.2 that our solution to this problem is to model both cases with a single model. The key idea for this model is that jobs arrive mostly in groups (as bags-of-tasks); this situation is modeled in Section 4.4.2. When the jobs arrive independently, their characteristics are well modeled by the Lublin-Feitelson supercomputer workload model, but the parameters of this model need to be adapted for the grid (see Section 4.4.1).

4.4.1 Adapting the Lublin-Feitelson Workload Model to Grids

In this section we summarize the Lublin-Feitelson model for independent jobs, and adapt it for grids.

Model Overview

We now present the Lublin-Feitelson model for supercomputer workloads. This is arguably the most-used workload model for supercomputers and for parallel production environments. The model focuses on three aspects: the arrival process, the job size (parallelism), and the job runtime. Figure 4.16 details

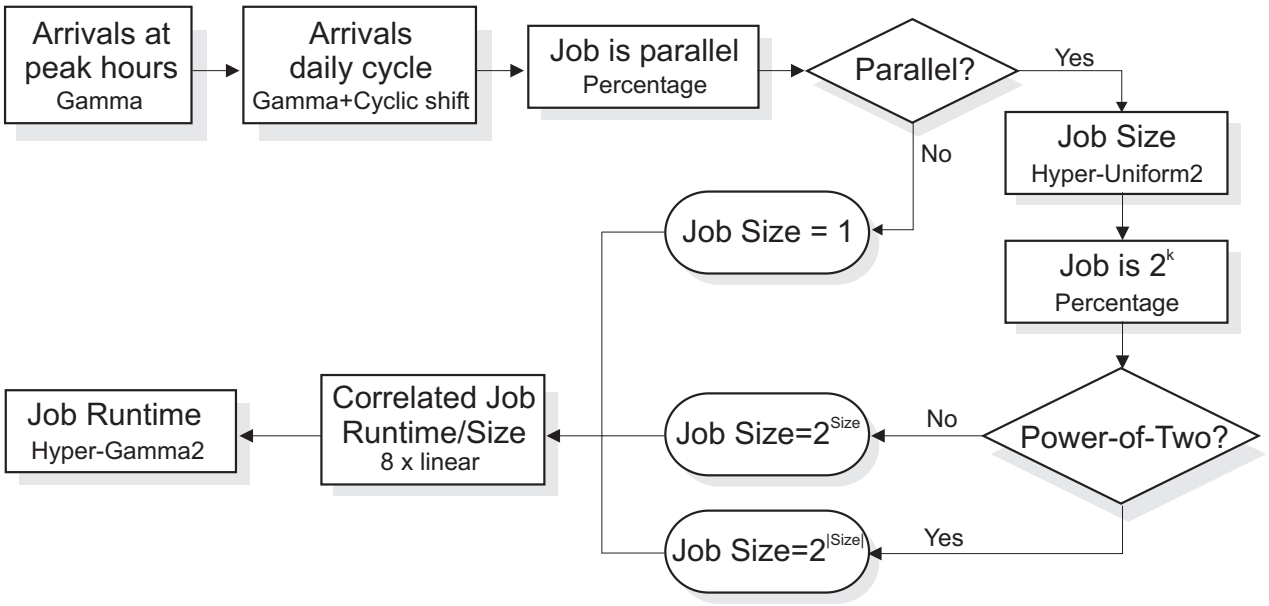


Figure 4.16: *An overview of the Lublin-Feitelson workload model [Lub03].*

the components of the Lublin-Feitelson model. The arrival process has two components: the arrivals during peak hours (when the arrival process is considered stable), and the daily cycle of the number of job arrivals. Each of these two components is modeled with a gamma distribution; the gamma distribution for the daily cycle is made cyclic through a shifting procedure with a peak period of around five hours. The job size has a complex model: first a parameters is used to characterize whether the the job is parallel (otherwise its size is one); for parallel jobs another parameter is used to characterize whether the job size is a power of two, and then a two-stage hyper-uniform distribution and its associated parameters are used for determining the job size. Finally, the model of the job runtime considers two components: the actual runtime modeled as a two-stage hyper-gamma distribution, and a driver for the actual runtime that quantifies as eight independent linear equations the correlation observed in the trace data between the job size and the job runtime.

The parameters for each distribution have been obtained following a process similar to that described in Section 4.2.4: the data are fit to candidate distributions using the MLE approach, and the KS-test is applied to assess the goodness-of-fit. In contrast to our process, the Lublin-Feitelson model is hand-tuned, i.e., the selection of the best fits is done manually. The Lublin-Feitelson model has been validated using four traces from the Parallel Workloads Archive by the model authors [Lub03] and by Talby, Feitelson, and Raveh [Tal07]; the traces are representative for supercomputers and for parallel production environments.

Adaptations for Grid Workloads

We now present the three changes we have made to the Lublin-Feitelson model in order to adapt it to grid workloads.

First, we have replaced the original arrival process with the arrival process described in Section 4.4.2. Thus, we have a single arrival process for both independent jobs and bags-of-tasks. We

Table 4.6: *Characteristics of the seven grid traces used to validate the BoT workload model.*

Trace ID	System Name	System Size [CPUs]	Trace Duration [Years]	Trace Size [tasks]
T1	DAS-2	400	1.5	1.1M
T2	Grid'5000	∼2500	2.5	1.0M
T3	NGS	378	3	0.6M
T4	AuverGrid	475	1	0.4M
T5	SHARCNET	6,828	1	1.1M
T6	LCG	24,515	0.03	0.2M
T7	NorduGrid	∼2000	2	0.8M

also found that the best fit for the grid data is not Gamma but the Weibull distribution.

Second, we have observed that in grids the percentage of parallel jobs is very small. Thus, the percentage of single-processor jobs is 95% in grids as opposed to below 25% in supercomputers.

Third, and as a consequence of the second change, we are able to drop the complex modeling of the correlation between the job size and the job runtime. Otherwise, since the number of parallel jobs is very small, we run the risk of over-fitting to the three-eight classes of job sizes required by the original Lublin-Feitelson model.

4.4.2 A Model for Bags-of-Tasks in Grid Workloads

In this section we present a workload model for large-scale distributed computing systems that focuses on Bag-of-Tasks applications. While four decades of research have led to many valuable models for computing system workloads [Ros65; Lub03; Li07e], these studies focus on modeling single (sequential or parallel) tasks. In comparison, our study is the first to model explicitly jobs with Bag-of-Tasks structure (BoTs, see Chapter 2) in the context of multi-cluster grids.

Model Overview

The model for BoTs proposed in this paper focuses on four aspects: the submitting user, the BoT arrival patterns, the BoT size, and the intra-BoT (individual task) characteristics.

Seven grid workload traces from the Grid Workloads Archive (GWA) [Ios08d] are used to validate the BoT model. Table 4.6 summarizes the characteristics of the seven considered systems and of their traces. Six of the traces presented in this work have been collected for periods of over one year, five traces collect data for over half a million jobs, and four traces have been collected from systems with over a thousand processors. We conclude that the traces used in this work are representative for the workloads of large-scale distributed computing systems; for a complete description of the traces we refer to the GWA web site and to [Ios08d]. The seven traces do not necessarily include information on BoTs. We show in Section 4.4.2 how to extract such information from logs that contain only information on independent tasks.

We follow the modeling process described in Section 4.2.4: for each characteristic the real data are fitted to one of the candidate distributions using the MLE method, then the results of the fit are estimated using the KS-test. For each model characteristic, the candidate distribution with the lowest D value is selected for each workload trace; the parameters of the best fit distributions are recorded as an instance of the model of the respective trace.

We define a theoretical "average system" as the system that has the average properties of the seven systems considered in this work. Using the average system properties we can generate synthetic

yet realistic traces, without using a single real system as a reference. The distributions that fit best the average system model and their parameter values are extracted using Step 3.d of the modeling process described in Section 4.2.4. Similarly to the average workload models built for other types of systems [Lub03], we cannot claim that the model of our average system workload, including the parameter values, represents the user behavior of an actual system. Instead, the main strength of this model is that it represents a common basis for the traces from which it has been extracted. By varying the parameters of the model characteristics, traces that are statistically similar to the GWA traces can be obtained for any system. By using the selected model parameters, results obtained by different researchers can be compared.

Analysis Results

We have shown in Section 3.3.5 several important workload analysis results: the utilization ranges widely across systems and the utilization pattern can be bursty (Figure 3.2, Table 3.3, and Figure 3.6), the workloads are dominated by few users (Figure 3.4), and a large percentage of the jobs (sometimes even all) are single-processor jobs (Figure 3.3).

Besides the job size, five other job characteristics are depicted in Figure 3.3: the inter-arrival time, the wait time, the runtime, the memory consumption, and the consumed CPU time. We present their characteristics in turn.

With a probability above 33% there is at least one job arriving at most 10 seconds after the previous one, and with a probability above 90% a new job will come at most 20 minutes after the previous one, for all systems. The LCG site behaves as a typical batch processing environment: over 50% of the jobs must wait 10 minutes and more before starting.

As expected, the waiting time for the DAS-2 and DAS-2 Grid environments is very small: the systems can be considered as interactive. Surprisingly, we observe this feature for the Grid3 environment as well. We explain this by the fact that the value was reported by the last middleware tool, which is in this case an on-node job wrapper instead of the off-node global or even local scheduler. Also expected, the jobs run in production environments are much longer (100 times) than their research environment's counterparts.

The data shows evidence that only single-processor jobs are run in the LCG and Grid3 production environments; arguably, this situation will change with the penetration of new distributed programming models (e.g., Ibis [Wrz05]), and with the maturity of the systems. The DAS-2 and DAS-2 Grid environments host mostly parallel applications, with almost 50% of the job requests being for power-of-two sizes. But the tendency of the predominance of single-processor is starting to show even here: the number of single-processor jobs in the DAS grew from 2.5% in 2003 [Li05], to about 30% in our trace. The average size of jobs in DAS-2 and DAS-2 Grid is 4.1 and 4.6, respectively; the grid users from the DAS-2 environment submit more parallel jobs than the local DAS-2 users.

The memory consumption is much higher (factor 30-1000) for production jobs, when compared to research jobs. The average job memory consumption for LCG, DAS-2, and DAS-2 Grid is around 200MB, 6.5MB, and 100KB, respectively. Similar to Li et al. [Li05], we find that the CDF of the memory consumption shows the existence of specific values around which most applications are clustered. The two environments for which we have analyzed the disk space used by the job's I/O have widely different characteristics, with the Condor GLOW having large input (over 200MB in over 50% of the cases, similar to the memory and I/O requirements of high-energy physics applications [Iam06]) in comparison with the Condor/South group (always below 10MB, most sizes below 100KB).

The CPU time consumption also shows the difference between the production and the research

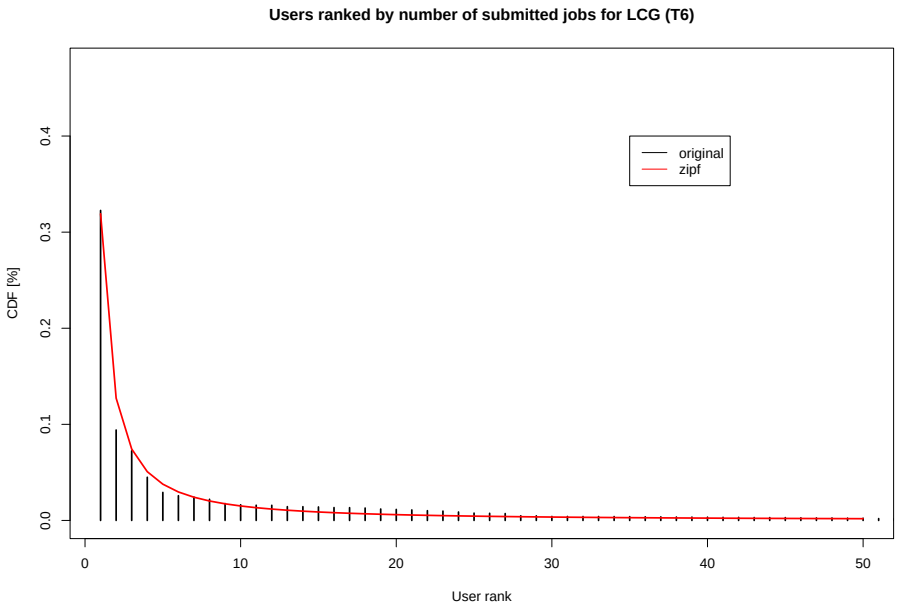


Figure 4.17: *Sample fit between the PDF of the users sorted by the number of submitted jobs and the Zipf distribution for the LCG trace (T6 in Table 4.6).*

environments: production jobs consume much bigger amounts than research jobs, with a factor of about 20; the ratio between this factor and the factor for the job runtime shows that the jobs from the production environments are smaller than the jobs in the research environments (by an average factor of 5).

Modeling Results

We now present the modeling results obtained for our model for BoTs in grid workloads.

We first perform a graphical analysis of fitting the availability data. Figure 4.17 allows us to graphically analyze the fit between the PDF of the users sorted by the number of submitted jobs and the Zipf distribution for one sample system (trace T6): the fit seems very good for the whole value range. Figure 4.18 shows the CDFs of the BoT inter-arrival time and of various distributions for one sample system (trace T1). The Weibull and the normal distributions give very good matches. The lognormal and the gamma distributions give visually worse fits for the horizontal axis range between 5 and 14. Figure 4.19 shows the CDFs of the BoT size and of various distributions for one sample system (trace T6). The Weibull and the gamma distributions give very good matches. The normal distribution does not fit well the values for the horizontal axis range between 4.2 and 6.8. The lognormal is a bad fit for the horizontal axis values above 3.0 (thus, for most values).

We now attempt to fit statistical distributions to each of the seven traces. We first use the MLE method, then perform goodness-of-fit tests to assess the quality of the fitting for each distribution, and to establish a best fit for each of the model parameters. However, a fit for every statistical distribution from the list of candidate distributions introduced in Section 4.2.3 to each of the seven traces would quickly become intractable; instead, we rely on the results of the graphical analysis. To confirm the graphical analysis results, we perform a full fit for the BoT size. The model parameters obtained from

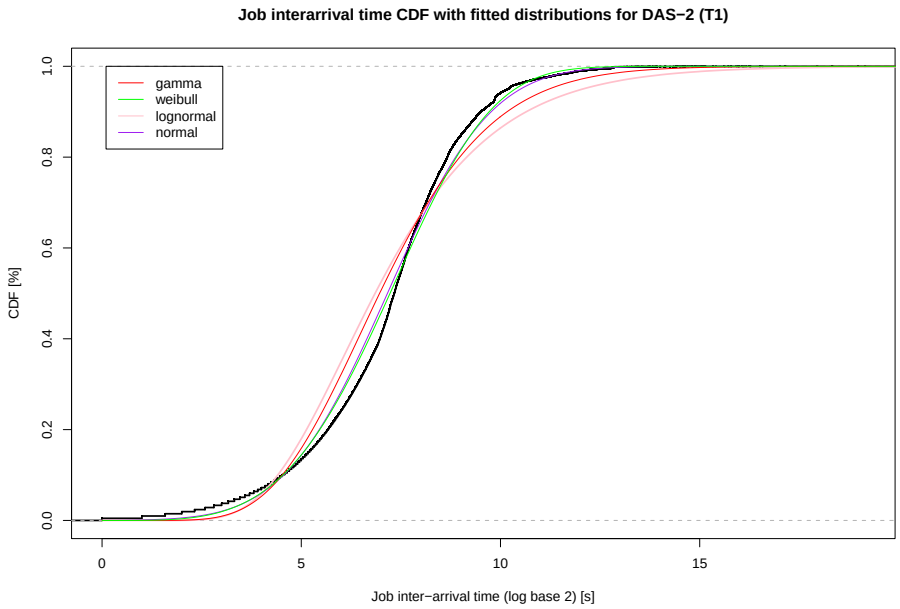


Figure 4.18: *Sample fit between the CDFs of the BoT inter-arrival time and of various distributions for the DAS-2 trace (T1 in Table 4.6).*

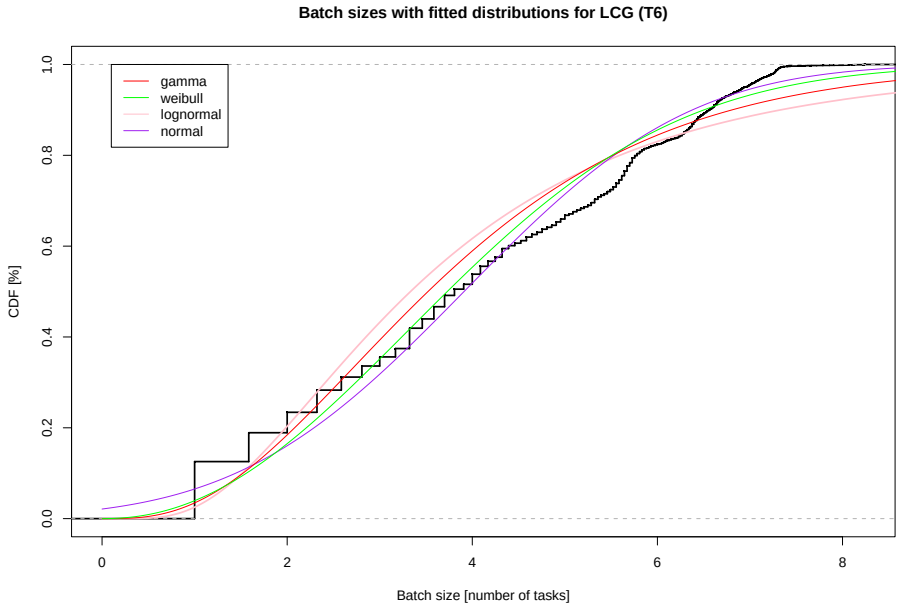


Figure 4.19: *Sample fit between the CDFs of the BoT size and of various distributions for the LCG trace (T6 in Table 4.6).*

the fitting process and the goodness-of-fit results are shown in Tables 4.7 and 4.8, respectively. The results confirm the visual cues of Figure 4.18: the Weibull distribution is the best fit for the DAS-2

Table 4.7: *Parameters for the fitting of various distributions to the BoT inter-arrival time for the seven traces.*

	lognormal		exp	normal		weibull		hyperexp			gamma	
system	mean	sd	rate	mean	sd	shape	scale	p	rate1	rate2	shape	rate
T1	1.92	0.34	0.14	2.03	7.16	4.06	7.91	0.34	0.14	0.14	10.38	1.44
T2	2.00	0.31	0.13	1.97	7.69	4.27	8.42	1.00	0.11	0.14	12.95	1.68
T3	2.02	0.28	0.13	1.80	7.79	4.94	8.48	3.72	0.12	0.15	15.28	1.95
T4	1.83	0.38	0.15	2.07	6.57	3.87	7.33	1.22	0.13	0.21	8.60	1.29
T5	1.90	0.33	0.14	1.91	6.94	4.17	7.65	1.91	0.14	0.16	10.98	1.57
T6	1.72	0.37	0.17	1.80	5.82	4.05	6.48	3.97	0.15	0.24	9.03	1.54
T7	2.05	0.29	0.13	1.97	8.00	4.37	8.74	1.00	0.11	0.13	14.36	1.79

Table 4.8: *Results for the KS goodness-of-fit test for the BoT inter-arrival time for the seven traces. The best fit is depicted in boldface font.*

system	lognormal	exp	normal	weibull	hyperexp	gamma
T1	0.13	0.37	0.07	**0.06**	0.37	0.11
T2	0.15	0.40	0.09	**0.09**	0.42	0.13
T3	0.16	0.42	0.10	**0.09**	0.39	0.14
T4	0.12	0.34	0.07	**0.05**	0.30	0.10
T5	0.13	0.38	0.08	**0.08**	0.37	0.11
T6	0.11	0.35	0.06	**0.04**	0.31	0.09
T7	0.14	0.41	**0.08**	0.09	0.43	0.11
Avg.	0.14	0.38	0.08	**0.07**	0.37	0.11

Table 4.9: *The parameter values for the best fits of the statistical distributions to the BoT model for the seven studied traces, and the model parameters for the "average" system (see text). N, LN, W, and G stand for the normal, lognormal, Weibull, and gamma distributions, respectively. Z stands for the Zipf distribution (the second parameter is the number of unique users).*

Trace	User	Bag-Of-Tasks			Task	
ID	Ranking	IAT	Daily Cycle	Size	ART	RTV
T1	Z(1.25,333)	W(4.06,7.91)	G(2.62,0.13)	W(1.75,2.91)	N(1.78,3.87)	W(1.64,10.21)
T2	Z(1.39,481)	W(4.27,8.42)	W(1.57,20.54)	G(2.47,1.64)	N(2.31,4.97)	N(5.54,9.56)
T3	Z(1.30,379)	W(4.94,8.48)	W(1.64,25.42)	W(2.02,1.58)	N(3.50,3.51)	G(1.79,0.29)
T4	- (see text)	W(3.87,7.33)	W(1.72,25.49)	W(1.78,1.51)	G(3.55,0.47)	N(6.41,11.73)
T5	Z(1.25,412)	W(4.17,7.65)	W(2.44,28.99)	W(1.37,1.89)	N(3.06,7.45)	W(1.93,13.65)
T6	Z(1.32,216)	W(4.05,6.48)	W(1.71,23.86)	N(1.33,2.71)	LN(1.82,0.34)	W(2.85,14.21)
T7	Z(1.36,387)	N(1.97,8.00)	W(1.62,22.18)	W(1.80,2.17)	N(2.76,9.04)	W(2.86,16.58)
Avg	Z(1.31,368)	W(4.25,7.86)	W(1.79,24.16)	W(1.76,2.11)	N(2.73,6.1)	W(2.05,12.25)

data, with the normal distribution a close second; the lognormal and the the gamma distributions are worse fits than Weibull, while the exponential and the hyper-exponential distributions do not fit the data.

For each distribution d and for each trace, we formulate the hypothesis that the trace data comes from the distribution d, whose parameters are found during the fitting process (*the null-hypothesis* of the goodness-of-fit test). We use the Kolmogorov-Smirnov test to test the null-hypothesis and to select the best fit. The selected distributions for each trace and their parameters are depicted in Table 4.9. Trace T4 does not include user information. The 'Avg' row in Table 4.9 presents the parameters of the average system. For the BoT inter-arrival time data presented in Tables 4.7 and 4.8, the best overall fit is the Weibull distribution (see text below). In the following we discuss the results for each of the four modeled aspects: the submitting user, the BoT arrival patterns, the BoT size, and the intra-BoT (individual task) characteristics. We also study an important potential correlation between these aspects.

Submitting User In many computing environments, a small number of users dominate the workload [Dow99b; Ios06a]. The Zipf distribution is used in many fields for characterizing "rank data", where the relative frequency of a given rank is determined by the Zipf distribution function [New05]. The Zipf distribution was fitted to the user ranking based on their relative job submission frequencies as follows. First, the users are ranked based on number of submitted jobs in descending order (the lowest rank is equal to the number of unique users in the trace). The probability associated with each rank is calculated by dividing the number of submitted jobs for each user by the total number of jobs in the trace.

BoT Arrival Patterns The BoT arrival patterns are modeled in two steps: first the inter-arrival time (IAT) between consecutive BoT arrivals during peak hours, and then the IAT variations caused by the daily submission cycle.

Similar to the results in [Ios06a], the hours between 8AM and 5PM are found to be "peak hours", with significantly more arrivals than during the rest of the day. Only the data for BoTs arriving during the peak hours are considered when modeling the IAT. According to established modeling practice [Lub03], a logarithmic transformation with base 2 is applied to these data to reduce the range and the effect of extreme values; this does not affect the quality of the data fitting. The Weibull distribution is selected as the average system fit and as the best fit for six of the seven traces.

The daily cycle is modeled similarly to [Lub03]. First, the day is split into 48 slots of 30 minutes each, then the number of BoT arrivals during each of the 48 slots is counted, and finally this data set is fitted against the candidate distributions. The Weibull distribution is again selected as the average system fit; it is also the best fit for six of the seven traces.

BoT Size Similarly to IAT modeling, a base-two logarithmic transformation is applied before fitting to the batch size (i.e., the number of tasks in a BoT). The Weibull and the normal distributions are tied, followed closely by gamma; the Weibull distribution is selected as the average system fit due to the higher number of best fits for individual traces: five against one.

The average BoT size per trace is between 5 and 50, while the maximum BoT size can be on the order of thousands. Depending on size, we define nine classes of BoTs: of size 2-4, of size 5-9, of size 10-19, of size 20-49, of size 50-99, of size 100-199, of size 200-499, of size 500-999, and of size 1000 and over. We also define the Small and the Medium classes encompassing the BoTs that fall in the classes 2-4 and 5-9, and 10-19 and 20-49, respectively. We expect the size-based classes to have different performance characteristics under different scheduling configurations.

Intra-BoT Characteristics The modeled intra-BoT characteristics are the average task runtime (ART) and the task runtime variability (RTV).

Similarly to IAT modeling, a base-two logarithmic transformation is applied before fitting to the task runtime. The normal distribution is the average system fit and the best fit for five of the seven traces.

The intra-BoT task runtime variability is defined as the variance of runtimes of the tasks belonging to the same BoT. The Weibull distribution is the average system fit and the best fit for four of the seven traces.

Correlation Between BoT Size and Intra-BoT Characteristics We now look at the correlation between the BoT size and the intra-BoT characteristics. We evaluate for each class of BoTs the task runtime variability and the task runtime. Figure 4.20 depicts the variability of the task runtimes for each size-based class of BoTs between 5 and 1000, relative to the average variability of the aggregated Small and Medium BoT classes (which form the majority of the BoTs); we have obtained similar results for the BoTs sized 5-9 and 2-4 (the latter not depicted in Figure 4.20). There is no significant difference between the various BoT classes. We have obtained similar results for the task

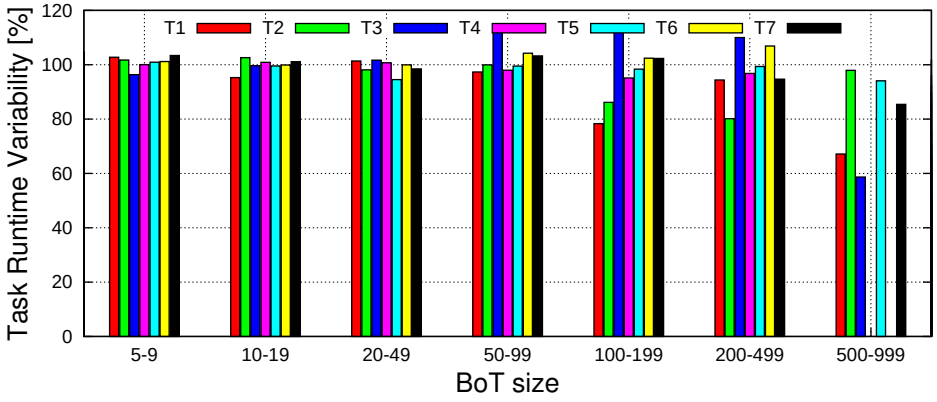


Figure 4.20: *The variability of the task runtimes for each size-based class of BoTs between 5 and 1000, relative to the average variability of the aggregated Small and Medium BoT classes. For a trace and BoT size range, values closer to 100% denote variability closer to the median BoT variability of the trace.*

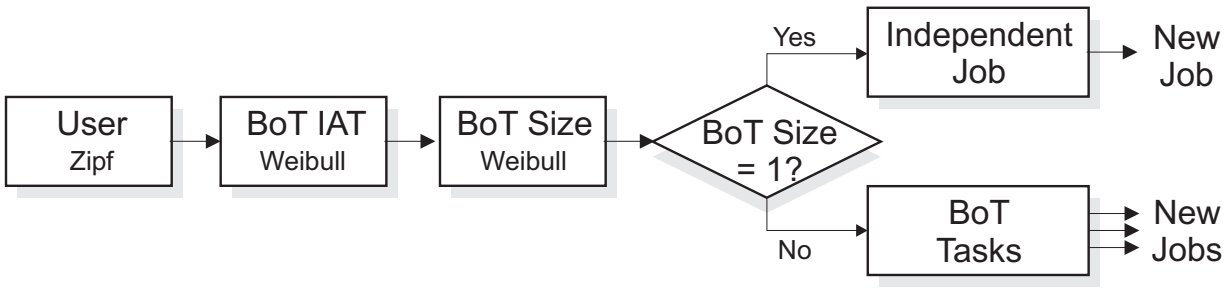


Figure 4.21: *An overview of our BoT-oriented grid workload model.*

runtime case (not depicted here). We conclude that the BoT size and the intra-BoT characteristics considered in our model are not strongly correlated.

Using the Model

We now present the use of our grid workload model to generate synthetic job data for simulation and testing purposes; Figure 4.21 shows an overview of the process. For each new job, the user who submits the jobs is generated first, using the corresponding distribution and parameter values for the average system (the cell in row 'Avg' and column 'User Ranking' from Table 4.9). The inter-arrival time between the last job and the job being generated is obtained by sampling the inter-arrival time distribution (column 'IAT'), and by scaling this value by the weight of the corresponding hour as given by the daily cycle (column 'Daily Cycle'). Then, the job size is obtained by sampling the job size distribution (column 'Size'). If the generated job size is above one, the job is considered to have a bag-of-tasks structure, with tasks of size one and runtime obtained from sampling the normal space with average given by the distribution in column 'ART', and standard deviation given by the distribution in column 'RTV'. If the generated job size is one, the job is considered to be modeled by

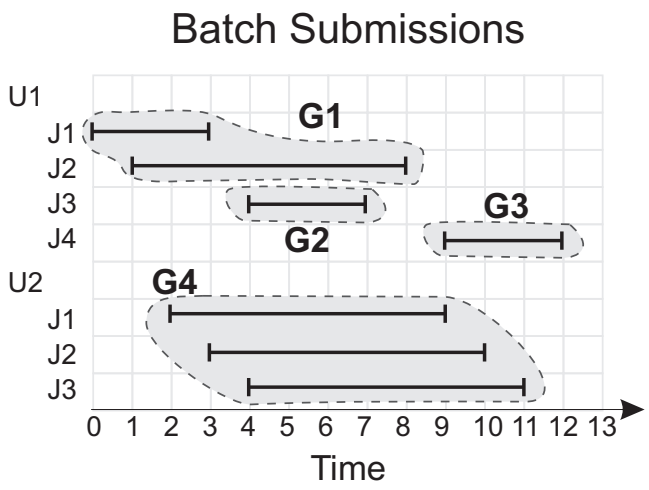


Figure 4.22: *The batch submissions of a hypothetical grid workload. The horizontal axis depicts the discrete time units. The vertical axis depicts the users and their jobs; the batch submissions are emphasized.*

the Lublin-Feitelson model adapted for grids (see Section 4.4.1).

Extracting Bag-of-Tasks Data From Any Workload

Due to the design of the grid middleware, not all the grid workload traces contain explicit information about the Bags-of-Tasks to which their jobs belong. In this section we propose a method for extracting this information automatically.

We first introduce a formal definition of bags-of-tasks. Let W be the workload of a grid, which we consider as a set $\{J_i | i = 1, ..., |W|\}$ of jobs ordered according to increasing submission time, so $ST(J_i) < ST(J_j)$ if $i < j$, where $ST(\cdot)$ returns the submission time of a job. A *group of jobs* G is a subset of W, which we assume again to be ordered according to submission time. The *seed* J_G of a group G of jobs is the job in G with the earliest submission time. We define the *arrival time of a group* G of jobs as the submission time of its seed, $ST(J_G)$, and we define the *inter-arrival time between two groups* G, G' as the difference between their arrival times $IAT(G, G') = ST(J_{G'}) - ST(J_G)$. We also define the *duration of a group* G as the difference between $ST(J_G)$ and $LFT(G)$, where $LFT(\cdot)$ returns the finish time of the last job of a group of jobs.

If the function $user(J)$ returns the identify of the user who has submitted job J, then we denote by $W_u = \{J_i | user(J_i) = u\}$ the *group of jobs submitted by the same user* u. A *batch submission with time parameter* Δ *of user* u is a maximal contiguous subsequence G of W_u such that for any two successive jobs J, J' in G, $ST(J') \leq ST(J) + \Delta$. We further define a *Parameter Sweep Application* (PSA) as the *batch submission with time parameter* Δ *of user* u with the additional constraint that all jobs execute the same application. We denote by $\mathbf{G}_u^B(\Delta)$ the *ordered set of batch submissions with time parameter* Δ *of user* u containing all the batch submissions with time parameter Δ of user u, ordered according to their arrival times. Note that for any two successive G, G' in $\mathbf{G}_u^B(\Delta)$, $ST(J_{G'}) > LFT(G) + \Delta$. From hereon, we call collectively the elements of $\mathbf{G}_u^B(\Delta)$ of any user u in the workload as batch submissions, or *bags-of-tasks*. Conversely, we define the *non-batch submissions* as the individual jobs from W that do not belong to any group of jobs in $\mathbf{G}_u^B(\Delta)$. Figure 4.22 illustrates the bag-of-tasks definition using the jobs submitted by two users, $U1$ and $U2$, over some period of time. The user $U1$ submits three

Table 4.10: *Summary of the content of the traces used to validate the automatic extraction of bags-of-tasks information. The ⋆ sign marks restrictions due to data scarcity (see text). UJM and LRM/GRM are acronyms for User Job Manager and Local/Grid Resource Manager jobs log, respectively. GRP and USR represent the number of unique groups/VOs and of users in the workload, respectively.*

Trace ID	System	Source	Period	Number of observed Sites	CPUs	Jobs	GRP	USR	Consumed CPUTime
T2	Grid'5000	LRM	2.5 yrs.	15	∼2500	1.0M	10	473	651y
T7	NorduGrid	GRM	2 yrs.	∼75	∼2000	0.8M	106	387	2443y
X1	GLOW	UJM	0.3 yrs.	1⋆	∼1400	0.2M	1⋆	18	55y

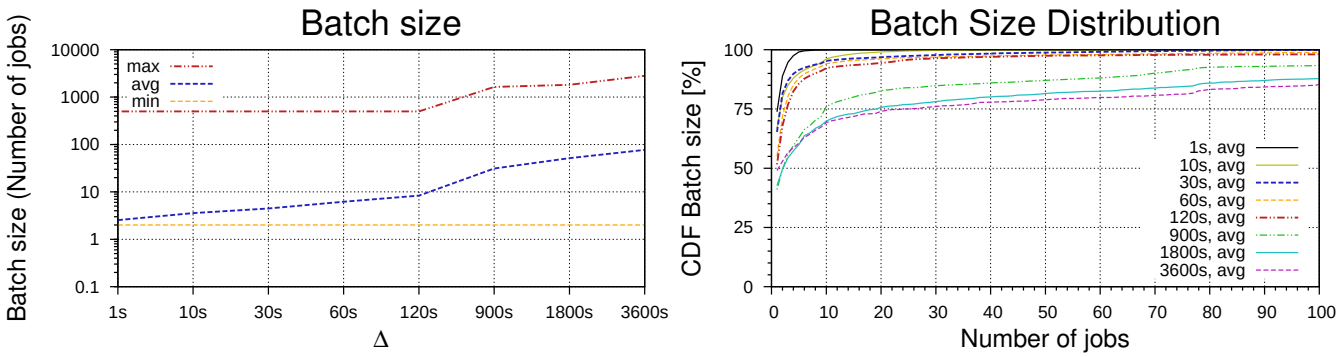


Figure 4.23: *The impact of parameter Δ on the bag-of-tasks size, for trace T-3.*

batches ($G1$, $G2$ and $G3$), whereas user $U2$ submits only one batch ($G4$).

By using the formal model of bags-of-tasks, the missing mapping between jobs and bags-of-tasks can be obtained automatically from the traces. To validate this approach, we use three long-term grid traces from the GWA. Each trace records the jobs that have been submitted to the grid, and includes for each job at least the identity of the user submitting it, and the job's time information (the time of the job submission, job start, and job end). In Table 4.10, we summarize the content of the traces considered in this work: Grid'5000, NorduGrid, and GLOW. Our workload analysis covers almost two million jobs submitted over a period of three years by more than 800 users. Note that the data sources have been selected to represent different types of grids: research for Grid'5000 and production for NorduGrid and GLOW, with GLOW a much more dynamic system than NorduGrid.

Figure 4.23 shows the impact of Δ on the batch size, for trace X1. The curve displaying the maximum batch size has a breaking point around $\Delta = 120s$. This could indicate that due to the grid middleware overhead, which is commonly in the order of tens of seconds, this is the lowest value for Δ for which meaningful results are obtained. Furthermore, there are three groups of values for the CDF of the batch sizes obtained for different values of Δ: one group for $\Delta = 1s$, one group for $\Delta \in 10s, 30s, 60s, 120s$ and one group for $\Delta \in 900s, 1800s, 3600s$. Given that the recording accuracy for our grid traces is $1s$, we disregard the first group. Similarly, taking into account the breaking point for the maximum batch size, we disregard the third group. From the second group, we select $\Delta = 120s$ as the basis for the reported results in this section; this gives the largest sizes of bags-of-tasks. We have obtained similar results for the other traces.

4.5 Related Work

In this section we survey the analysis and modeling efforts for the resources and the workloads of large-scale distributed (computing) systems.

4.5.1 Resource Dynamics and Evolution

We survey several approaches for modeling resource availability in four areas of computing systems: mainframes and supercomputers, networks of workstations and clusters, the Internet and other large-scale networks, and grids. Table 4.11 summarizes our survey; our work is the first in the area of multi-cluster grids.

The work of Siewiorek [Cas82; Lin90], Iyer [Iye82; Tan93], and Gray [Gra86; Gra90] has revealed the most important characteristics of computer resource dynamics: more failures during the system's infancy, the Weibull distribution is a good model of the failure inter-arrival time, failures often occur in bursts or are "chained", the system load and the failure rate are strongly correlated, finite state machines and multiple cause analysis are useful when building a model that is tightly coupled to a particular system.

The majority of the later efforts, ours included, have departed from the seminal work of Siewiorek, Iyer, and Gray in one or more of the following three directions: 1) better models for the failure arrivals and duration [Hea02; Zha04; Sch06; Ios07e], 2) more modeled aspects, e.g., arrival in bursts at the node [Zha04] and cluster [Ios07e] levels, the failure location [Zha04; Ios07e], the number of states between system availability and unavailability and 3) analyzing and modeling the resource dynamics for other systems, e.g., supercomputers [Sch06], networks and workstations and clusters in enterprise and academic environments (category B in Table 4.11), Internet-related environments (category C in Table 4.11), and grids [Ios07e]. Looking at Table 4.11 for research gaps (the 'no' cells), this research body could be extended towards a study of individual failure types and of correlations between failures of different types in the context of multi-cluster grids.

Three factors also affect the quality of these studies: the size of the systems for which the data was gathered, the data source, and the length of the period over which the data was gathered (duration). The size of the systems ranges between 4 and over 330,000(!) processors, with the values over 10,000 typical more to volunteer computing and to grids than to single data centers. The data source can be the reports of the system customers or system administrators, an automated logging system, or a one-time measurement process. The reports usually include a degree of human error or omission, and studies based on them are usually biased (sometimes the bias is quantifiable). Due to privacy issues, measurements are the only choice for large-scale Internet-based systems, e.g., peer-to-peer systems and volunteer computing; however, such measurements are difficult for scalability and representativeness reasons. The duration of the studies ranges between weeks and years, with the longest covered period being 9 years [Sch06]. In some systems at least one resource fails every few minutes [Zha04; Ios07e]; in others a failure is an event that only occurs after days or months of operation [Gra86; Lin90; Tan93]. Siewiorek argues that when the study does not distinguish between the various causes and types of resource dynamics, the duration needed to obtain meaningful results grows [Cas82]. Thus, a duration of at least a few months is required for a meaningful study; however, this is difficult for system sizes over 100,000 nodes.

While the concept of resource evolution is new, previous work on grid resource characterization offers the necessary information on the current structure of grids. This chapter has characterized the network topology, the cluster size, and the individual node processors; there is currently no work that

Table 4.11: *A summary of the resource availability modeling efforts for computing systems.*

Study	System (No./Size)	Data Source (Length)	Bursts of Failures			Individual Failures				
			Type	Arrival	Size	Type	Arrival	Duration	Location	States
A. Mainframes and Supercomputers										
[Cas82]	PDP,CM* (4 sys./15)	Sys.logs (1 year)	no	no	no	no	Model	no	no	2
[Iye82]	VAXcluster (4 sys./4)	Reports (1 year)	no	no	no	Cause (2 classes)	MTTF	no	no	2
[Gra90] ([Gra86])	Tandem	Reports (3 years)	chain	no	CDF	Cause (7 classes)	MTBF	MTTR	no	2
[Tan93]	VAXcluster (1 sys./7)	Sys.logs (10 months)	Correlated	Markov chain	from arrival	Cause (9 classes)	MTBF+ CDF	MTTR+ CDF	no	multiple
[Sch06]	at LANL (22 sys./24k)	Sys.logs (9 years)	no	no	no	Cause (6 classes)	CDF+ Model	CDF+ Model	no	2
B. Networks of Workstations and Clusters										
[AD95]	at UCB (1 sys./53)	Msmts. (2 months)	no	no	no	no	Availability over time	Idle Time CDF	no	2
[Ach97]	academic (3 sys./400)	Sys.logs (2 weeks)	no	no	no	no	Availability CDF	Stability CDF	no	2
[Bol00]	Microsoft (1 sys./51k)	Msmts. (2 months)	Correlated	no	Pair-Wise PDF	no	Uptime CDF	Downtime CDF	no	2
[Hea02]	academic (3 sys./126)	Sys.logs (5-18 mo.)	no	no	no	no	Model	no	no	2
[Zha04] ([Sah04])	IBM TJWRC (1 sys./395)	Sys.logs (1 year)	Burst at node	Model	Constant	no	From burst (Rate)	Constant (Increment)	Per node Model	2
[Kon07b]	Entropia (1 sys./100s)	Msmts. (1 month)	no	no	no	no	Availability CDF	no	no	2
[Roo07]	UND (1 sys./500)	Sys.logs (4 months)	no	no	no	no	MTBF+ CDF	MTTR+ CDF	no	5
C. Internet, Peer-to-Peer Systems, and Volunteer Computing										
[Lin90]	File servers (13 sys./52)	Sys.logs (22 months)	no	no	no	no	Model	no	no	3
[Lon95]	Internet (1 sys./1.2k)	Msmts. (3 months)	no	no	no	no	MTBF+ CDF	MTBR+ CDF	no	2
[Kal99]	Mail servers (1 sys./500)	Sys.logs (6 months)	no	no	no	Cause (7 classes)	Uptime PDF	Downtime Basic stats	no	multiple
[Opp03]	Web servers (3 sys./3k)	Sys.logs (5-7 mo.)	no	no	no	Cause (6 classes)	Rate Basic stats	MTTR	no	multiple
[Bha03]	P2P (1 sys./1.5k)	Msmts. (2 weeks)	Correlated	no	Pair-Wise PDF	no	Availability CDF	no	no	2
[And06]	BOINC (1 sys./330k)	Sys.logs (1.5 years)	no	no	no	no	Rate over time	Lifetime CDF	Country+ Time Zone	2
[Kon07a]	XtremWeb (1 sys./4.4k)	Msmts. (2 months)	no	no	no	no	TBF CDF	no	no	2
D. Multi-cluster Grids										
Section 4.3	Grid5k (15 cl./2.5k)	Sys.logs (1.5 years)	Burst at cluster	Model	Model	no	From burst	Model	Per cluster Model/Table	2

characterizes the logical topology of grids. The network topology model is often borrowed from the more mature Internet models [Wax88; Doa96; Cal97; Med01]. Lu and Dinda assume that individual nodes are supercomputers and characterize them in terms of architecture type, processor type and speed, number of processors, memory size, disk size, hardware vendor, operating system, and available software [Lu03]. In contrast, Kee et al. [Kee04] propose a cluster-oriented characterization of resources; they model the number of nodes per cluster, the number of processors in a node, and the memory size per node. They also present an evolution of the processor types over time; this can be seen as an instance of the processor-level evolution, which contrasts to our system- and cluster-level evolution.

4.5.2 Workloads

In this section we survey several workload modeling approaches in the area of computing systems, e.g., networked machines, clusters, grids. Table 4.12 summarizes our survey; our work is the first to propose a grid workload model in which BoTs and parallel jobs coexist.

The importance of the workload for evaluating the system performance has been emphasized many times [Jai91; Lo98; Cha99; Mu'01; Fei02]. However, in contrast to the evolution of the models for resource reliability, where the initial work established much of the general model characteristics, the workload modeling research evolved at much slower pace. Four decades ago, modeling the workload involved only finding the instruction mix of the applications of one computer [Ros65; Arb66]. Today, it involves correlating information about several characteristics and validating the resulting model using real traces from a variety of systems [Jai91; Fei02]. Over these four decades, the workload modeling follows three main directions.

The first direction is to identify the characteristics that needed to be modeled. Rosin [Ros65] was the first to show that a computer system workload contains many short- and just a few long-running jobs. Leland and Ott [Lel86] were the first to model the job runtime. The memory/disk requirements were modeled independently by Calzarossa and Serazzi [Cal94], and by Harchol-Balter and Downey [HB97]. The job sizes are modeled by Feitelson et al. [Fei07; Fei96; Lub03] for supercomputers, and by Iosup et al. [Ios06a; Ios08e] for grids. The phenomenon of grouped jobs submission has been largely ignored or treated implicitly; the repeated execution of jobs by a single user and the bags-of-tasks are explicitly modeled by Feitelson et al. [Fei07; Fei96; Lub03] and by Iosup et al. [Ios06a; Ios08e], respectively. The arrival process has received much attention; following the discovery of self-similarity in Internet traffic [Lel94], Li et al. [Li07f] have shown evidence of the existence of the same phenomenon in grid traces, and Iosup et al. [Ios08e] have proposed a multi-user model that can be used to generate self-similar workloads.

The second direction is from incomplete to complete models. In incomplete modeling [Lel86; HB97; Fei96], traces taken from real systems complement the results of the modeling, e.g., in the Leland and Ott model [Lel86] the arrival of the jobs is taken from the real traces while the job runtime is modeled. In contrast, a complete model uses only synthesized parameters; with the availability of more traces, it became possible to validate complete models. The first complete model was introduced by Jann et al. [Jan97], with two others being the Lublin-Feitelson model [Lub03] and the models proposed by Iosup et al. ([Ios08e] and Section 4.4).

The third direction is from over-fitted to tractable models. An over-fitted model is a model that uses tens to hundreds of parameters to fit the original data. For example, it is possible to match many empirical distributions when using hyper-exponential distributions; doing so may requires many steps and leads to (too) many parameters [Fei07; Fei96; Jan97; Li05]. While large numbers of parameters allow better fitting to the original data, showing the generality of the model or making it accessible

Table 4.12: *A summary of the workload modeling efforts for computing systems.*

	Sys.Type (Sys.No./Size) (Data Len.)	Number of Params.	Modeled Aspects								
			VOs/ Users	Job Grouping			Individual jobs				
Study				Type	Arrivals	Size	Arrivals	Size	Runtime	Mem/Disk	Correlations
[Lel86]	UNIX (6 sys./6) (4 months)	2	-	-	-	-	trace	fixed = 1	Polynomial (2 params)	-	-
[Cal94]	UNIX (1 sys.) (4 hours)	n x r x r + m x p + 1	n users r commands/ user (n x r^2 params)	-	-	-	Markov chain (per user)	fixed = 1	k partitions m classes/ partition (m x p params)	Yes	I/O∼RT (1 param)
[HB97]	UNIX (5 sys.) (1 year)	2 + trace	-	-	-	-	trace	fixed = 1	ext. [Lel86] (1 param)	as runtime (1 param)	-
[Fei96] ([Fei07])	SC (6 sys./1.5k) (1-12 months)	43 + trace	-	repeated execution	Poisson (1 param)	Zipf (2 params)	from Job Grouping	Harmonic 10 classes (10 param) + trace	H-Exp2 10 classes (30 params)	-	size∼RT 10 classes
[Jan97]	SC (1 sys./322) (3 months)	90	-	-	-	-	H-Erlang2 10 classes (40 params)	10 classes	H-Erlang2 10 classes (40 params)	-	size∼IAT size∼RT
[Lub03]	SC (3 sys./1.5k) (9-24 mo.)	23	-	-	-	-	peak hours daily cycle 2 x Gamma (4 params)	parallel power-of-two H-Uniform2 (6 params)	H-Gamma2 (5 params)	-	size∼mem Linear (2 params)
[Li05]	grid (5 cl./400) (1 year)	60	15 classes	-	-	-	H-Exp2 (4 params) Gamma/Weibull (2 params)	H-LogUniform2 (4 params)	15 classes Weibull / Lognormal (60 params)	3 classes (3 params)	size∼mem
[Med05]	grid (1 cl./140) (1.5 years)	n x 2 x (3 + 4 or 3 + t)	n groups Lognormal (2n params)	-	-	-	log in-out (2 params) rate (1 par.) t groups/ Two Exp. (t or 4 params)	-	-	-	-
[Son05]	grid (5 sys./2.2k) (3-36 mo.)	k x (2+s+r) or 2k + trace	k user groups (k params)	-	-	-	-	power-of-two per user (1 + k x s)	per user (k x r params)	-	implicit
[Li07d]	grid (100s cl./24k) (11 days)	6 (VO) 9 (User)	VO, Exp (2 params) User, H-Exp2 (5 params)	-	-	-	MMPP2 (4 params)	-	-	-	-
[Ios08e]	grid (7 sys./10ks) (1-30 mo.)	12	Zipf (2 params)	BoTs	peak daily cycle 2 x Weibull (4 params)	Weibull (2 params)	from Job Grouping	fixed = 1	Normal + Weibull (4 params)	-	implicit
Sec. 4.4	grid (7 sys./10ks) (1-30 mo.)	12 + 15 (par.)	like [Ios08e] (2 params)	BoTs	like [Ios08e] (4 params)	like [Ios08e] (2 params)	from Job Grouping	like [Lub03]	like [Lub03] like [Ios08e]	Basic statistics+ CDF	implicit

to the common user are difficult. Some of the over-fitted models were obtained automatically, which allows the user to tune the trade-off between the number of parameters and accuracy [Cal94; Son05; Med05]. Important tractable models were provided by Harchol-Balter and Downey [HB97], by Lublin and Feitelson [Lub03], by Li et al. [Li07d], and by Iosup et al. ([Ios08e] and Section 4.4).

Looking at Table 4.12 for research gaps (the '-' cells), the research body reviewed in this section could benefit from additional studies regarding other job groupings (e.g., workflows), and from modeling the memory/disk requirements of grid jobs.

4.6 Concluding Remarks

While many grids now exist, little is known about the characteristics of their resources and of their workloads. Analyzing these characteristics is important for various resource management tasks, including performance evaluation, resource procurement and capacity planning, and for guaranteed quality-of-service. Even more important, building models for the grid resources and for the grid workloads enables focus for and the ability to compare the work of various researchers. In this chapter we have presented analysis and modeling results for both grid resources and grid workloads. For the former, we have focused on the distinction between the short term grid dynamics and the long term grid evolution. We have also introduced a new model for grid workloads that focuses on bags-of-tasks. The models proposed in this chapter have been validated with long-term traces taken from real grid systems.

However comprehensive the grid model presented in this chapter, there is still much to be done. The resource dynamics and evolution require more in-detail modeling for specific systems and failure causes. Given the amount of data now available in the Grid Workloads Archive (see Chapter 3), grid workload modeling can go in many directions, from new or better models of basic workload characteristics (e.g., file transfer patterns, cluster locality of job submissions, detailed user modeling) to new models of complex workload aspects (e.g., workflows and other e-Science experiments that comprise groups of jobs). Finally, as grids are more and more employed in production, or evolve into very different systems, new trends will emerge and require the attention of the modeling community.

Chapter 5

The GrenchMark Testing Framework

5.1 Overview

In this chapter* we present the GrenchMark framework for analyzing, testing, and comparing grid settings.

5.1.1 Motivation and Problem Statement

From the family of large-scale distributing computing environments, grids arguably raise now the most difficult problems. Early testing in real grids reveals much lower performance than expected from analysis and simulations, e.g., much higher job wait time [Ios06a]. Failure rates range from 10% to 45% [Ios06b; Kha06; Li06], which means that the failure rate in today's grids is much higher than that of contemporary parallel production installations [Sch06]. Testing the functionality of the most widely installed grid middleware with batteries of simple probes reveals around one problem in every three tests [Ios07b]. Testing in grids raises all the problems of testing in large-scale distributed computing systems, and also many additional issues, e.g., there exists a thick and complex middleware layer with almost no standardization, there exist new types of applications (e.g., large parameter sweeps, or large workflows), the environment availability is dynamic, the tester shares the environment with common users, the grid workloads differ significantly from those of clusters and supercomputers [Lub03; Ios06a] but no common grid workload model exists, etc. Repeated and realistic testing is a proven industrial way to alleviate some of these problems, but such testing is difficult in grids. A common platform for testing in grids can be built within an adequate testing framework, but *How to build an adequate testing framework for grids?*

5.1.2 Key Ideas and Selected Results

To answer the question formulated in the previous section we have designed the GrenchMark testing framework. The GrenchMark framework automates the testing process, from workload specification to obtaining results. To support realistic testing, the framework specifies mechanisms to generate new and reutilize existing workloads. To guarantee that the generated workloads can actually be submitted to the tested system, the workload generator uses real and synthetic applications, and deals with the job submission specifics. To facilitate repeatable testing, the framework stores test provenance and annotation data (e.g., monitoring information), and is capable to replay tests. The data is organized hierarchically, so that large testing projects can be managed, and collaborative testing may be performed. Finally, to enable the comparison of test results across systems, the framework defines

*This chapter is based on previous work published in the IEEE/ACM Intl. Symp. on Cluster Computing and the Grid (CCGrid'06) [Ios06b] and on work under review at the Elsevier Performance Evaluation [Ios07a].

a testing process, and test metrics that aggregate information at different levels, and that take into account the environment specifics.

We have adapted the GrenchMark framework to the problems of grids, and developed a reference implementation for testing in grids. The reference implementation provides a portable, extensible, and high-performance grid testing tool. We have tested with this tool environments of hundreds to thousands of resources managed by Condor or Globus, two of the most used middleware for managing large-scale distributed computing environments. In the past two years, we have run more than 250,000 test jobs, in over 25 fully-automated testing scenarios from three broad test areas: performance evaluation [Moh05b; Son06], reliability testing [Ios06b; Moh05b], and functionality testing [Ios06b; Lam05]. After being tested with GrenchMark, the Koala grid scheduler [Moh05b] has been successfully released to operate on the DAS [Bal00], a grid that serves over 300 Dutch scientists.

5.1.3 Organization of this chapter

The remainder of this chapter is structured as follows. In Section 5.2 we present the design goals of a testing framework for large-scale distributed computing systems. Then, we describe the resulting design and the main features of the GrenchMark framework in Section 5.3, its key implementation details is Section 5.4, its validation in Section 5.5, and its past, current, and potential use in Section 5.6. Finally, in Section 5.8 we survey existing testing approaches for distributed computing systems and compare the GrenchMark framework with the state-of-the-art.

5.2 Design Goals for LSDCS Testing Frameworks

In this section we synthesize the design goals for testing large-scale distributed computing systems. Our motivation is twofold. First, large-scale distributed computing systems (LSDCSs) have specific testing requirements, e.g., unique workload characteristics, the problem of scale. Second, in spite of last decade's evolution of tools for various testing purposes (see Section 5.8), there is still place for improvement, especially in managing tests for the testing community, e.g., sharing best-practices, and test and provenance data storage. Below we present seven design goals (in four categories) specific to frameworks for testing LSDCSs. In Section 5.8 we evaluate the related work according to these design goals.

1. **Realism** To perform realistic tests, there is a need for realistic workload generation. However, the realistic workloads are not a goal *per-se*; a workload may be realistic for one scenario, and unrealistic for another.

2. **Reproducibility** Compared with existing computing systems, LSDCSs have dynamic availability and varying load. However, compared with the Internet, in many cases the full system state can be recorded. Thus:

   (a) **Same Environment** There is a need to ensure that tests are (re-)performed in experimental conditions that match the test design (e.g., test workload, background load).

   (b) **Data Provenance** There must exist support for provenance – where a piece of data comes from, and through which process [Bun01; Sim05] – and result annotations. In some cases the experimental conditions must be recorded as a complement to the test results.

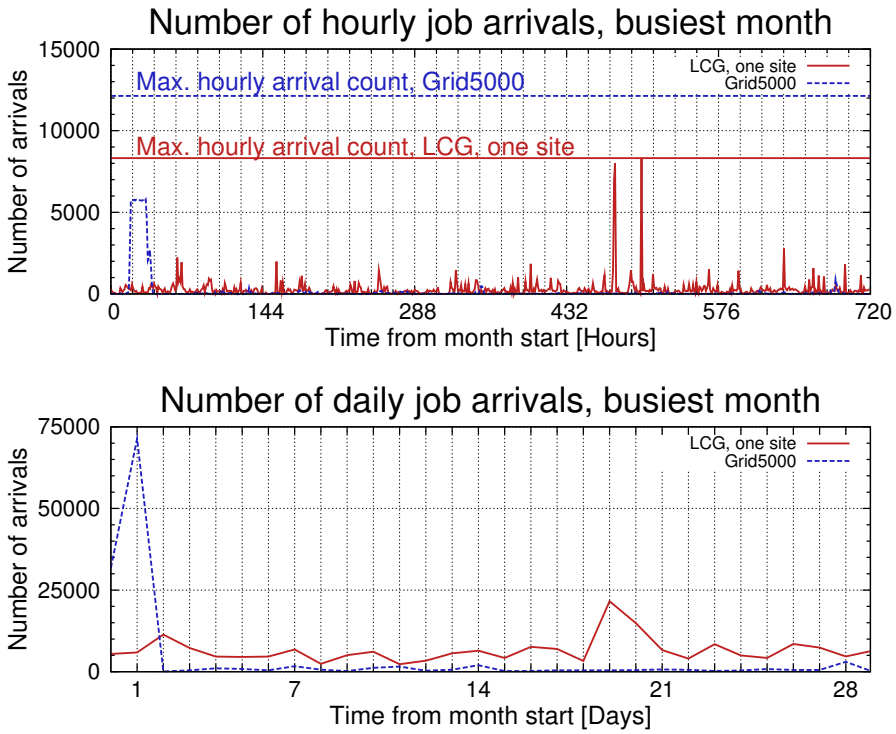


Figure 5.1: *Hourly and monthly job arrival count for two grid systems. Only the month with the highest number of job arrivals (the busiest month) is depicted.*

3. **High-Performance** The testing framework must cope with the scale of the tested system. In particular:

   (a) **Workload Generation and Submission** The workload generation and submission are the only part of a testing framework whose performance is related to that of the tested system's. For example, in load-testing a system, the workload submitter must ensure a submission rate higher than that of any existing user. Figure 5.1 shows the hourly and daily job arrival count for two grid systems, during their busiest month of operation. The maximum hourly (daily) count for Grid5000 [Bol06] is 12,131 (71,393) jobs; rates of $1-9\times10^3$ ($1-5\times10^5$) job arrivals/hour (day) have been reported for other grids [Ios06a]. Thus, rates of 2×10^4 submissions/hour and 10^6 submissions/day must be supported for testing grid computing systems. The generation process must support similar rates if the testing framework generates the workload on-line.

   (b) **Test Results Management** Large amounts of data are produced during testing in an LSDCS. As a first step towards Online Analytical Processing (OLAP) capabilities, there must exist a hierarchy for results storage and aggregation, i.e., by project, by scenario, by test run [Bow07]. There may also be a need for coordination of test tasks and subtasks, where the results of one test (or their aggregates) are used as input to subsequent runs.

4. **Extensibility** There are many types of LSDCSs, e.g., grid computing, volunteer computing, peer-to-peer computing. As a result:

   (a) **New Environments** The design of a testing tool should be extensible, so that new environments can be supported, i.e., through plug-ins or configurable policies.

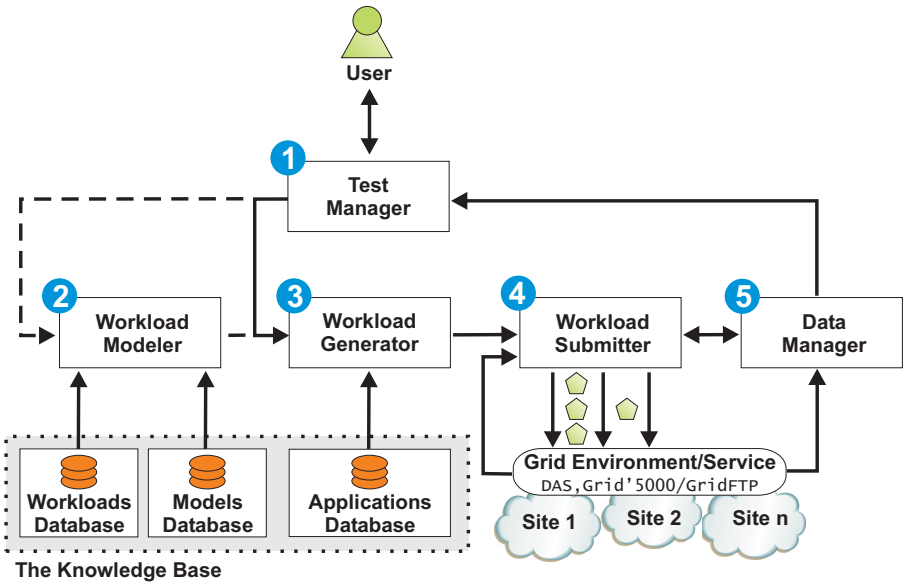


Figure 5.2: *An overview of the GrenchMark framework design.*

(b) **Test Templates** With so many alternatives, there must exist a set of templates (best practices) for commonly used scenarios. The scenarios are configured by the user at test time.

5.3 The GrenchMark Framework Design

In this section we present an overview of the GrenchMark framework, and describe some of its distinguished features: workload analysis and modeling, generation of complex workloads, results analysis, and storage of provenance and annotation data.

5.3.1 Overview

Figure 5.2 shows an overview of the GrenchMark framework design. The design specifies only the minimal requirements of LSDCS testing. By design, all the GrenchMark framework components can be extended to accommodate more testing requirements.

The Test Manager receives from the user a scenario specification and provides to the user an analysis of the test results. The scenario may include anything from testing using one synthetic application that arrives at 10am every day to a test for which the Test Manager decides on the experiment design. The Test Manager coordinates the pre-test (deployment), test (execution), and post-test (data archiving and analysis) processes in a way that ensures test repeatability.

The Workload Modeler fits existing grid workload data to a grid workload model. The workload data and the model are given as input by the user, or taken from the Workloads and Models Databases.

The Workload Generator is responsible for realistic workload generation. It receives as input either a workload specification from the Test Manager, or a model and its parameters from the Workload Modeler, or both. This component can truncate existing workload traces, and can mix workloads to create complex scenarios: bursty arrivals, workloads that evolve over time, or workloads with different

arrival times. Using the Applications Database, the Workload Generator also deals with application-specifics, e.g., generating appropriate input files and command-line parameters. Finally, the Workload Generator enables dynamic workloads, if the testing scenario requires workloads that have different workload characteristics depending on the system's response. In Section 5.6.2 we use this latter feature to assess the success rate when running jobs with inter-dependencies.

The Workload Submitter coordinates a pool of job submitters that operate either locally or remotely to ensure that the test workload jobs are submitted on-time. Through the job submitters, the Workload Submitter records information about the submission, the start, and the end time of each job, with accuracy of at least 1 second. The interval between the actual job submission time and the submission time specified in the generated workload must not be higher than one tenth of the recording accuracy, e.g., 0.1s. To ensure this quality of service, the pool of job submitters may be implemented to execute on the same machine as the Workload Generator, or it can be distributed using a static or a work stealing mechanism [Wrz05]. This component may also configure the system that monitors the tested system, i.e., to increase the sampling rate or the verbosity of the tools output for the duration of the test.

The Workload Submitter is also responsible for executing dynamic workloads, e.g., for workflows the start of a job may depend on the completion of another. To this end, the tested environment may return results or status information to the Workload Submitter. This feedback can be used in various dynamic scenarios, e.g., to submit jobs that have as input the output of other jobs (support for testing workflows), or to ensure that at most some fixed number of jobs are submitted at the same time (support for service level agreements/QoS testing).

The Data Manager is responsible for data warehousing, and for data analysis. The stored data are the testing results, the associated provenance data, and sometimes the annotation data (e.g., monitoring information during the testing period). Data are at least indexed by project, by scenario, and by test run. The results of the analysis (arrow from the Data Manager to the Test Manager in Figure 5.2) can be used by the Test Manager to dynamically generate new test configurations, e.g., adapting model parameters, selecting different testing techniques.

5.3.2 Workload Analysis and Modeling Features

The Workload Modeler can automatically analyze an input workload, e.g., to extract the empirical distribution of the workload jobs' runtime. Depending on the testing scenario, the analysis extracts data for traditional job characteristics [Fei98b], or for environment-specific characteristics. Some of the grid-specific characteristics are [Ios06c]: co-allocation, grouped (e.g., batch, workflow) submission of jobs, economic requirements (i.e., utility and cost functions per job), and user-, group- and virtual organization-level information. Figure 5.3 shows sample analysis results for the workload traces available in the Grid Workloads Archive [Ios08d].

The results of the analysis can be used to generate realistic workloads, e.g., by sampling from each of the empirical distributions (note that this workload generation uses an implicit workload model in which there exists no correlation between the characteristics of a job). Alternatively, the Workload Modeler can fit the data to traditional statistical distributions, as specified by a workload model; the purpose of this operation is to automatically obtain realistic values for the model parameters.

5.3.3 Results Analysis

We consider in the GrenchMark framework job-, operational-, application-, and service-level metrics.

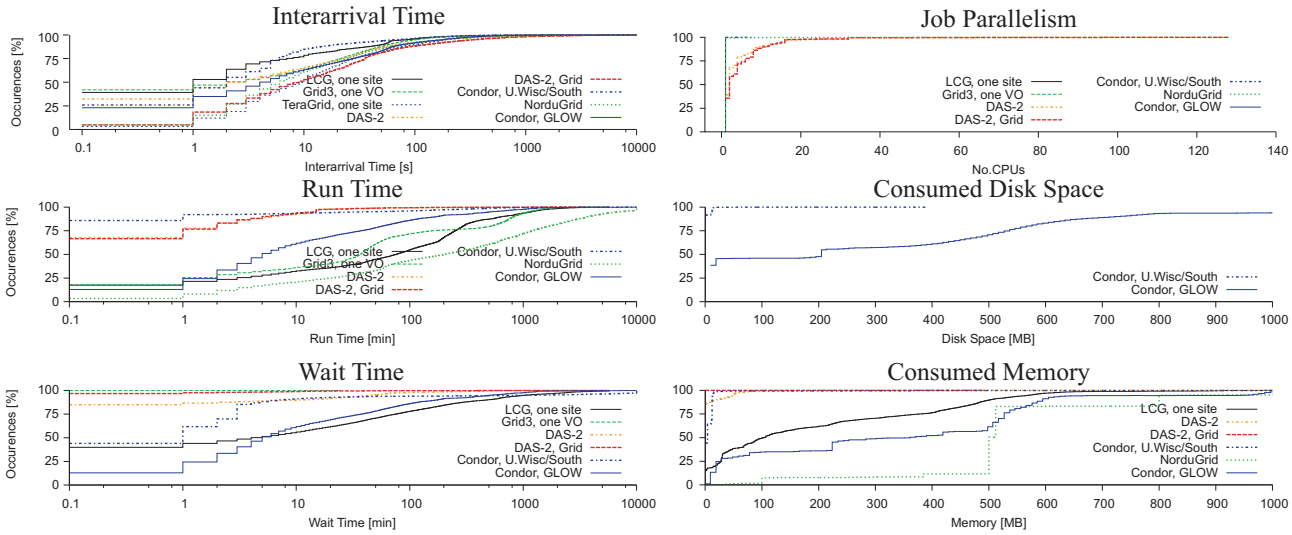


Figure 5.3: *CDFs of the most important job characteristics for NorduGrid, Condor GLOW, Condor UWisc-South, TeraGrid, Grid3, LCG, DAS-2, and DAS-2 Grid. Note the log-like scale for time-related characteristics.*

The *job-level metrics* (e.g., arrival rate, resource consumption, etc.) describe the execution of individual jobs [Fei98b].

The *operational-level metrics* are used to describe jobs submitted as an aggregate, e.g., batches of jobs, workflows. For a comprehensive list of workflow-related metrics we refer the reader to [Tru07].

The *application-level metrics* and the *service-level* metrics are specific to the applications that provide a specific service. At the application-level the analysis follows the evolution of internal application counters (e.g., number of states explored) over time. At the service-level the analysis follows the finding of solutions for a user's problem.

Depending on the test, the results analysis may assess metrics at one or more levels, from a simple count of jobs successfully completed to an in-depth statistical analysis of metrics at all levels and with detailed graphing and reporting.

5.3.4 Provenance and Annotation data

Provenance describes the source data and the derivation process that leads to the results set [Bun01; Sim05]. Provenance data can be used to increase the trust in the results (audit and data quality), to simplify searching and indexing test results (test management), and to facilitate the exchange of results (benchmarking). The following information is automatically recorded: the unique test identifier, the test description, and all the data that are produced by or depends on factors from outside the testing framework (e.g., are produced by the tested environment). In this way, we hope to support as many future uses as possible.

The main problem in storing provenance data is the size of the stored data, and the way the information can be published (or searched). We argue that the provenance data size is negligible when compared to the size of the testing input and results.

Testing data is immutable; additional tests may add data, but cannot modify existing data. GrenchMark also allows immutable data annotations, e.g., immutable data associated with the testing data. Typically, the user will annotate test data with monitoring information, e.g., the number of

available machines over the testing period. Note that, depending on the test requirements, the size of the monitoring information may become larger than the testing data.

In GrenchMark, provenance data, except for the test identifier, is decoupled from the main test data, and stored in a separate relational database. This raises issues of data integrity, that is, the provenance information may be lost when copying just the test results data. However, this greatly simplifies the reverse operations of searching, of indexing, and of publishing the provenance data.

5.4 The GRENCHMARK Reference Implementation

In this section we present the GRENCHMARK reference implementation (GMark-RI)[1]. We begin with an overview of our approach, then present three of its distinctive features.

5.4.1 Overview

We have developed GMark-RI using Python[2]. While Python is portable, its performance is similar to that of other byte-code compiled languages (i.e., may be lower than the performance of a low-level compiled language such as `C`); we show in Section 5.5 that the performance of GMArk-RI is suitable for testing LSDCSs.

The GMark-RI Workload Modeler is an extension of the analysis tools that we have developed for the analysis of four grid traces [Ios06a]. The main additions are the automation of the analysis and modeling process, and a mechanism for representative area selection, described in Section 5.4.4.

The GMark-RI can replay traces from various production environments. However, since a real trace contains tens of thousands to millions of jobs and span periods of months to years, a truncation process that selects a part of the input trace (a *trace area*) is required. We have implemented two alternative mechanisms for job selection. First, we have developed a mechanism for (representative) area selection (see Section 5.4.4), and implemented it in the Workload Modeler. Second, the Workload Generator can filter out jobs and select the *first-N* jobs according to various criteria, and scale various aspects of the workload, e.g., the requested resources, or the job runtimes. New filters may be added as plug-ins. The configuration of the filter may be generated by the Workload Modeler's representative area selection algorithm, or specified by the user. The Workload Modeler can currently use data from two Workloads Databases: for grid environments the Grid Workloads Archive (GWA) [Ios08d], and for parallel production environments the Parallel Workloads Archive (PWA) [Th07]. Both these databases are maintained by and considered representative for their respective communities.

The GMark-RI Workload Generator is based on the concepts of *unit generators* and of job description file (JDF) *printers*. The *unit generators* produce detailed submission information for each application in the workload. The appropriate unit generator is selected and instantiated by the Workload Generator from the Applications Database. New generators can be added in the Applications Database. The printers take the workload unit information and create job description files suitable for grid submission. Currently, GMark-RI includes printers for the Condor ClassAds and the Globus RSL formats. Thus, GMark-RI can submit jobs to the following resource managers: KOALA [Moh05b], Globus GRAM [Cza98], and Condor [Tha05b]. New printers can be added though a plug-in system.

We have implemented three synthetic test applications: `sserio`, a sequential application with parameterizable computation, memory and I/O requirements, `smpi`, an MPI application with pa-

[1]GMark-RI is available at `http://grenchmark.st.ewi.tudelft.nl/`.
[2]Python is available at `http://www.python.org/`.

```
# File-type: text/wl-spec
#Jobs Type SiteType Total SiteInfo ArrivalTimeDistr OtherInfo
25   sser   single    1 *:? Poisson(120s)  StartAt=0s
25   sserio single    1 *:? Poisson(120s)  StartAt=60s
25   smpi1  single    1 *:? Poisson(120s)  StartAt=30s,ExternalFile=smpi1.xin
25   smpi1  single    1 *:? Poisson(120s)  StartAt=90s,ExternalFile=smpi2.xin
```

Figure 5.4: *A GMark-RI workload description example.*

rameterizable computation, communication, memory, and I/O requirements, and `swf`, an application implementing the job model used in the PWA and in the GWA. We also support other synthetic and real applications; we have performed tests with over 40 real C, Java, and MPI applications [Moh05b; Ios06b; Son06].

For (pseudo-)random number generation, the Workload Generator offers support for many widely used statistical distributions. GMark-RI uses the MT19937 variant of the Mersenne Twister [Mat98] to generate random numbers from the uniform distribution.

The GMark-RI Workload Submitter coordinates a multi-threaded pool of job submitters. During the submission of the jobs, it reports all job submission commands, the turnaround time of each job, including the grid overhead, the total turnaround time of the workload, and various statistical information.

The GMark-RI Data Manager archives and analyzes the test results, and produces an SQL relational database stores: project, sub-project, and test description and summaries, process information (e.g., time-stamped information about the process events), job execution information, other test environment-specific information (e.g., data obtained from Condor logs). The analyzer processes the database and produces detailed reports, some of which we present in Section 5.6.

5.4.2 Workload Description Languge

With the automation of every step in the testing process, the most difficult step in using the GMark-RI is describing the test workload. To ease this task, we have designed an extensible workload description language. Figure 5.4 shows a sample workload description file. The workload comprises 100 jobs of four different types (25 jobs of each type). Lines 3-4 are used to generate sequential jobs of types `sser` and `sserio`, with default parameters. Internally, the `sser` and the `sserio` unit generators generate the individual jobs data. Based on this information, the printer for the tested system, e.g., the Condor printer for a Condor-based system, is used to create the workload's job description files. Lines 5-6 are used to generate MPI jobs of type `smpi1`, with parameters specified in external files `smpi1.xin` and `smpi2.xin`. All four job types follow a Poisson arrival process, with an average arrival rate of 1 job every 120 seconds. The first job of each type starts at the time specified in the workload description with the `StartAt` tag. For MPI jobs, the specified external file (tag `ExternalFile`) contains other application-specific parameters (see also Section 5.4.3).

5.4.3 Extending the GrenchMark Reference Implementation

GMark-RI has been designed with an incremental approach in mind, and facilitates future extensions. The framework design can be easily extended, for instance by adding various workload generation *notions* (e.g., users, virtual organizations). GMark-RI also offers a *plug-in system*, which can be used to add unit generators (new application types) and printers (support for other grid resource managers).

```
SiteTypesWithWeights=nonfixed/30;fixed/50
SitesWithWeights=fs1;fs2;fs3;fs4
NComponentsWithWeights=1/50.0;2/10.0;3/16.2;4/10.0;5/5.0;8;10;15
TotalCPUsWithWeights=2/20.0;4/30.0;5;8;10;16;20;32
...
```

Figure 5.5: *A GMark-RI workload specification language extension.*

To extend the workload generation process, the user has first to extend the workload specification language, and then to write a plug-in that uses the language extension; the only requirement is that the extension language is based on *Key=Value* statements. A file written in an extension language is automatically parsed, and the data is available to the user when the plug-in is invoked. For the language extension, GMark-RI offers library functions to parse simple values (e.g., `boolean`s, `string`s, `integer`s, and `float`s), and more complicated constructs (e.g., lists, and lists of values with associated weights). The plug-in developer can decide to use these library functions, or to parse the extension language statements independently.

Figure 5.5 shows an example of a file written in an extended language, for generating a workload with co-allocated jobs. Lines 1-4 define the empirical distributions from which the job characteristics `SiteType`, `Site`, `NComponents`, and `TotalCPUs` are respectively sampled. A default weight of 1.0 is automatically assigned to the elements of the `Site` empirical distribution, for which the user has not specified the weight. Jobs with 1 component (weight=50.0) and 4 total CPUs (weight=30.0) are more likely to be generated (see dotted markings in Figure 5.5).

5.4.4 Representative Area Selection

It is common for a testing procedure to require as input a small subset of a workload trace, i.e., the hour during which the number of arrivals is the highest. We call the *representative area* from a trace T, with length L and with target characteristics C, the subset of length L from trace T whose characteristics are the closest to C from all possible subsets of length L from trace T. C can be either completely specified by the user, or be specified by a user-given percentile with the actual values computed from the input trace.

We have developed a window-based mechanism that selects from an input trace and a set of characteristics the representative area of length equal to the window size W, and which starts only at discrete multiples of the window size.

Consider that a user needs to perform a test that requires four hours of realistic workload input; the user has a one-year trace that is representative for the test. The GMark-RI selection mechanism is used to extract the representative area of size $W = 4h$. In a testing scenario for which a fixed job arrival rate of 100 jobs/hour is desired, the user may need to select the representative area with desired characteristics "number of job arrivals equal to 400". In a high-load testing scenario, the user may need to select the representative area with desired characteristics "number of job arrivals equal to the 90% percentile of the trace". The 90% percentile value is automatically computed from the set of trace subsets of width $W = 4h$ that start with the beginning of the trace, cover the whole trace, and do not overlap.

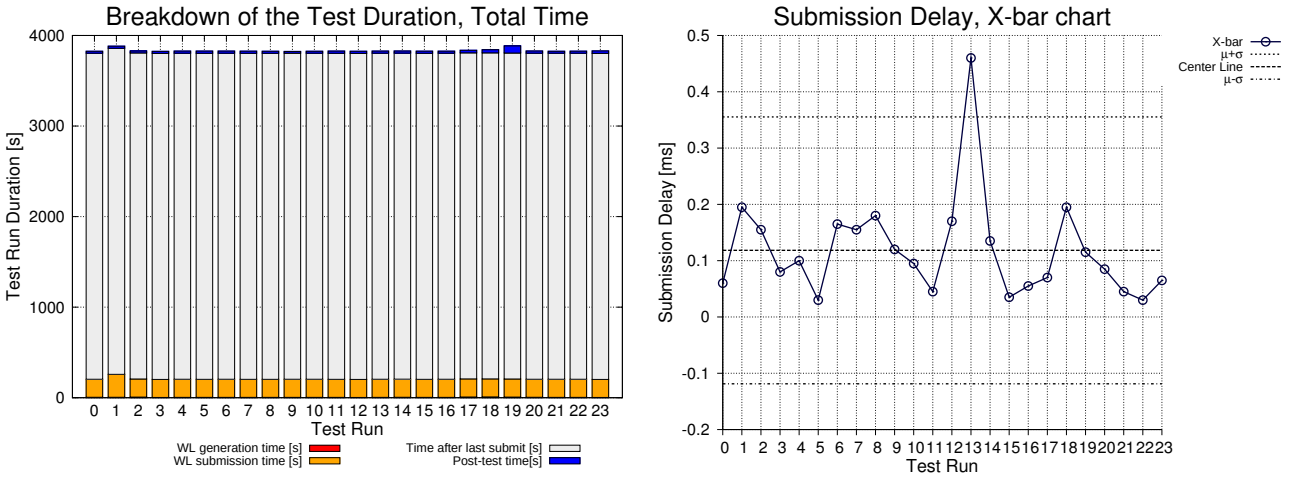


Figure 5.6: *Validation of the GMark-RI test process. (*left*) The test duration per process component shows negligible overhead for 24 consecutive test runs. (*right*) The average job submission delay (X-bar) for 24 consecutive test runs shows that the submission delay is under control.*

5.5 Testing the Reference Implementation

In this section we present the validation and the performance evaluation of the reference implementation. We show that GMark-RI fulfills the design goals 2a (the same experimental conditions for the same test configuration) and 3a (high performance testing).

5.5.1 Validation

We define the *submission delay* of a job as the difference between the actual time of submission and the desired time of submission, as specified in the test configuration.

We show in this section that GMark-RI operates with low overhead and submission delays, ensuring that the tests are are re-performed in very similar experimental conditions (design goal 2a). To this end, we use the same test configuration to generate 24 test runs of 1000 jobs, that is, 200 batches of 5 jobs each (WL generation component in Figure 5.6 (left)). The jobs in one run are submitted with a 1-second inter-batch delay (WL submission component in Figure 5.6 (left)). The test continues after the last submission for one hour (WL execution component in Figure 5.6 (left)). After one hour, all jobs still running or waiting in the system queue are stopped, and the results of the test are processed (post-processing component in Figure 5.6 (left)). All the other configuration settings are set to their default values.

Figure 5.6 (left) shows the test duration per component, for 24 consecutive test runs. The WL execution component accounts for over 90% of the execution time. Figure 5.6 (right) shows the average job submission delay for each of the 24 consecutive test runs. The average of all the average submission delays (the centerline) is $\mu = 0.118ms$; the standard deviation is $\sigma = 0.237$. The individual averages are equally spread around the centerline. There is no point falling outside the range $(\mu - 3\sigma, \mu + 3\sigma)$; there are no trends to indicate that the submission process is out of control [Nel84]. We conclude that the submission process is controlled [She80], ensuring that the same test can be re-performed in practically identical experimental conditions.

Table 5.1: *The characteristics of the experimental platform used for evaluating the performance of GMark-RI.*

	A. Workload Generator	*B. Workload Submitter*
CPU and Memory		
CPU Vendor/Model	Intel/Pentium 4	4 × Intel/Xeon
CPU Frequency	3.2GHz CPU	2.4GHz
RAM Memory Size	1GB	2GB
Hard-Disk		
Vendor/Model	Western Digital/WD1200JB	Western Digital/WD2000JB
Performance Specifications (declared)		
Rotational Speed (nominal)	7,200 RPM	7,200 RPM
Buffer Size	8 MB	8 MB
Average Latency (nominal)	4.20 ms	4.20 ms
Write Seek Time (average)	10.9 ms	10.9 ms
Transfer Rates		
Buffer-to-Disk (Max)	602 Mbits/s	602 Mbits/s
Buffer to Host (EIDE)	100 MB/s	100 MB/s
Software		
Operating System	WindowsXP/SP2	Linux 2.6.17-1.2142 FC4smp
Python	2.4.4	2.4.4

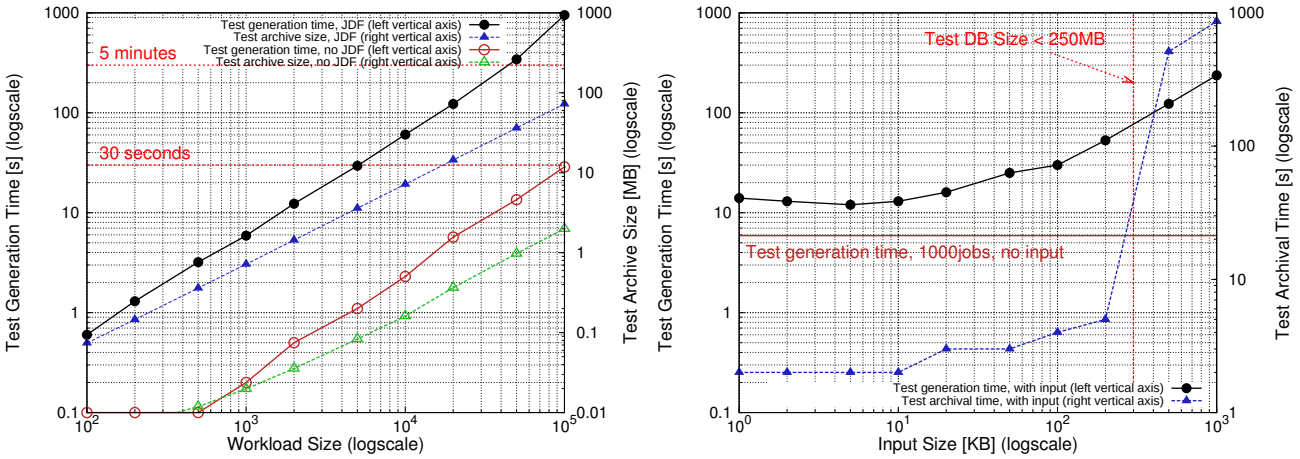


Figure 5.7: *Performance evaluation of the GMark-RI generator and submitter components. (*left*) The speed and the output size of the GMark-RI workload generator for various workload sizes, with and without per-job JDFs, and without job input data. (*right*) The duration of workload generation and archival for various sizes of per-job input data, with per-job JDFs and a workload of 1000 jobs.*

5.5.2 Performance Evaluation

We show in this section that GMark-RI is a high performance testing tool, that is, it supports generation and submission rates of 2×10^4 submissions/hour, or 6 submissions/second (design goal 3a).

We test the performance of the GMark-RI Workload Generator for workload sizes of up to 100,000 jobs on the experimental platform described in Table 5.1. Figure 5.7 (left) shows the speed and the output size of the GMark-RI workload generator, with and without per-job job description files (JDFs, see Section 5.4.1), and with no job input data. With per-job JDF, the workload generation duration is dominated by the I/O time (creating the job directory and the JDF), but GMark-RI can still generate around 100 jobs/second. Without per-job JDFs, the generator's speed is much higher: around 3000 jobs/second. Figure 5.7 (right) shows the duration of workload generation and archival for various sizes of per-job input data, with per-job JDFs and a workload of 1000 jobs. We generate only all-zeroes input data. At only 1KB/job, the input generation doubles the workload generation

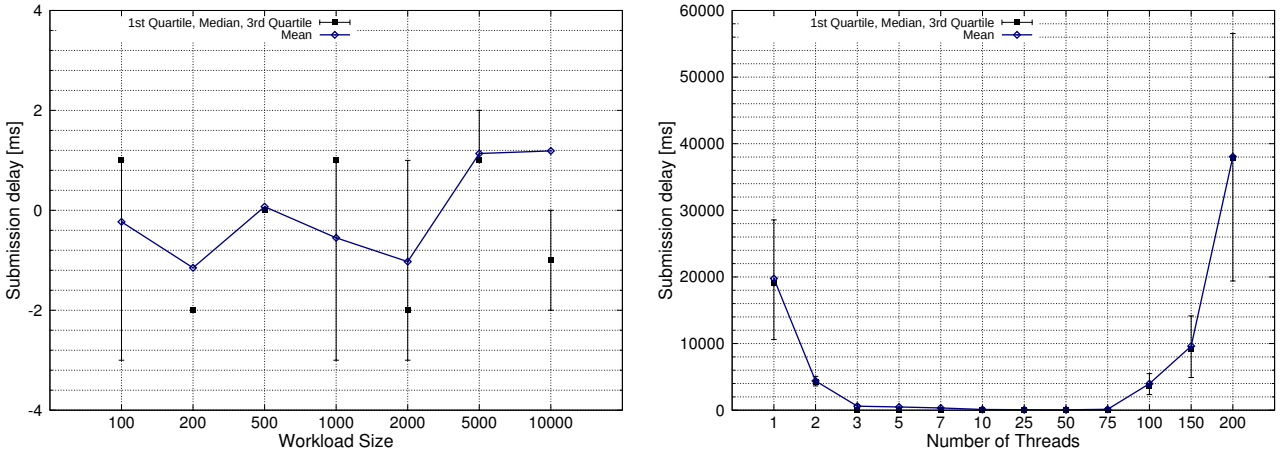


Figure 5.8: *Performance evaluation of the GMark-RI generator and submitter components. (*left*) The speed and the output size of the GMark-RI workload generator for various workload sizes, with an arrival rate of 10 jobs/second and with 5 submission threads. (*right*) The influence of the number of submission threads on the submission delta, for workloads of 10000 jobs, with an arrival rate of 100 jobs/second.*

time. After 10KB/job, and up to 1000KB/job, the input workload generation time depends linearly on the input size, with a onefold increase every 26KB/job. We conclude that the GMark-RI Workload Generator accomplishes the design goal 3a for input sizes less than 500KB/job; after that size, the GMark-RI needs to generate the workload before the submission begins.

We now test the performance of the GMark-RI Workload Submitter on the experimental platform described in Table 5.1. The main difference from the Workload Generator tests is the use of a multi-processor machine for submission. We let the submitter complete the submission of the workload, then estimate the average submission delay (the accuracy). Figure 5.8 (left) depicts the accuracy of the GMark-RI workload submitter, for various workload sizes, and with a job arrival rate of 10 jobs/second. Using 5 submission threads (the default GMark-RI setting), the submission delay remains on average below 2 ms. The 1st and the 3rd quartiles of the submission delay remain below 4 ms, which is well below the target of 0.1 s (see Section 5.3.1). There is no correlation between the submission delay and the workload size. Given that the measurement precision required for grid computing systems is 1 second, we conclude that the job submission delay of the GMark-RI workload submitter is negligible. Figure 5.8 (right) shows the influence of the number of submission threads on the submission delay, when jobs arrive at a much higher rate (100 jobs/second), for workloads of 10000 jobs. With only 1-3 threads, the workload submitter cannot cope with the high arrival rate. Adding more than 75 submission threads decreases the accuracy of the submitter. A setup with 10-75 submission threads (3-19 per processor) is optimal for our experimental platform. We conclude that the GMark-RI Workload Submitter accomplishes the design goal 3a.

5.6 Experiments using GrenchMark

We have used GMark-RI in a variety of testing scenarios in grids [Moh05b; Ios06b; Son06], in peer-to-peer file-sharing [Roo06], and in management of heterogeneous resources (i.e, using Condor [Tha05b]). Table 5.2 shows a summary of the scenarios for which the reference implementation was used, proving that GMark-RI accomplishes the design goals 1 and 4 (see Section 5.2). Overall, we have run more than

Table 5.2: *A summary of the scenarios for which the reference implementation was used.* *Acronyms: WT - wait time, AWT - avg. waiting time, ART - avg. run time, RespT - response time, ASD - avg. slowdown, FR - failure rate, U - utilization, O - overhead, G'put - goodput, T'put - throughput, NN - number of nodes, OC - communication overhead, CI - content information, Rec - recommendations, NC - node coverage, DTA - detected pollution attempts, CTP - connections to polluters. The ⋆ sign marks unpublished results.*

Scenario	Metrics	Graphs	Ref.
A. Grid performance and reliability evaluation			
1. Single- and mixed-application workloads			
a. Generic test	AWT,ART,ASD		[Ios06b; Moh05b]
b. System overhead	O,RespT	RespT breakdown	⋆
c. System fairness	G'put,T'put	System-level G'put/T'put	⋆
	WT,RT	Scatterplot WT/RT	⋆
	U	Pareto chart jobs/node	⋆
2. "What-if" analysis			
a. System change	AWT,ART		[Ios06b]
b. System inter-operation	AWT,ART		[Ios06b]
c. Bursty arrivals	U	U over time	[Ios06b; Moh05b]
3. Grid settings comparison			
a. Single- vs. co-allocated jobs	AWT,ART,FR		[Ios06b]
b. Unitary vs. composite jobs	AWT,ART,FR		[Ios06b]
c. Scheduling policies	AWT,ART,FR		[Moh05b; Son06]
4. Application- and Service-level performance			
a. Generic test	G'put,T'put	G'Put,T'put over time	⋆
b. Service selection	G'put	Bi-histogram G'Put	⋆
B. Grid functionality testing			
1. Ability to execute diverse workloads			
a. Single-application workloads	FR		[Ios06b; Moh05b]
b. Mixed-applications workloads	FR		[Ios06b; Moh05b]
c. Reliability under stress-load	FR		[Ios06b]
2. Stability, quality control, other functionality over time			
a. System stability analysis	WT	Box-and-whiskers WT	⋆
b. Quality control	WT	Xbar chart WT	⋆
c. Periodic functionality testing	FR		[Ios06b]
C. Peer-to-Peer functionality and reliability evaluation			
1. Functionality testing			
a. Trace replay	NN,U	NN/U over time	[Roo06]
b. Protocol functionality	OC,CI,NC,CTP	OC/CI/NC/CTP over time	[Roo06]
c. Protocol correctness	CI,Rec	Rec vs. CI	[Roo06]
2. Reliability testing			
a. Protocol resilience to churn	NC,Rec,DTA	NC/Rec vs. DTA over time	[Roo06]
b. Protocol resilience to pollution	NC,Rec,DTA,CTP	Rec vs. DTA, CTP vs. DTA	[Roo06]

250,000 test jobs in the last 18 months, in over 25 fully-automated testing scenarios. We summarize 13 of these scenarios in this section; 7 in Section 5.6.1 and 6 in Section 5.6.2. We do not report here results of testing peer-to-peer systems; for more details we refer the reader to [Roo06].

Unless otherwise noted, each of the experiments in this section demonstrates that GMark-RI meets the design goals 1, 2a, 3a, and 4b. We demonstrate meeting the design goals 2b and 3b in Sections 5.6.1 and 5.6.1, respectively. With the whole Section 5.6.1 we demonstrate that GMark-RI can assess metrics from the four performance levels described in Section 5.3.3. By showing that GMark-RI can be used for testing a heterogeneous resource pool (Section 5.6.1) and a homogeneous multi-cluster grid (Section 5.6.2), we prove meeting the design goal 4a.

5.6.1 Testing a Heterogeneous Resource Pool

In this section we present a sample of the results of the experiments performed during two weeks on the computer pool at U.Wisc.-Madison. The 600 processors used in these experiments are managed with Condor. A recent study shows that Condor is one of the most used resource management systems: in

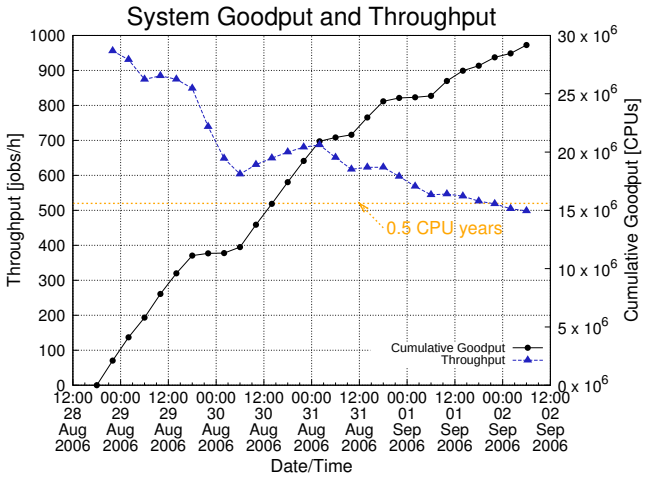


Figure 5.9: *The evolution of system-level goodput and throughput over time.*

2006, Condor was used to manage at least 60,000 resources grouped in over 1750 pools, with the pool size ranging from below 10 to over 5000 processors [Tha06].

System Performance

The experiments in this section prove the ability of GMark-RI to manage test results (design goal 3b), both by aggregating results for the overall set of tests (Figure 5.9), and by presenting detailed and aggregated results for individual test runs (Figure 5.10). Note that GMark-RI is used here to compute both job-level and operational-level metrics.

From the user's point of view, there are two important metrics of a resource manager's performance: goodput, and job wait time.

Figure 5.9 depicts the throughput and the goodput of the system for 100 consecutive runs of 1000 jobs each. The user obtains a high rate of goodput even in a production environment: over 0.5 CPUyears of goodput in two days. Condor is fair with respect to resource consumption, and the throughput and goodput rates are halved after the user exceeds his quota (at the beginning of 31 Aug 2006).

Figure 5.10 (left) shows the job wait time properties of 24 consecutive runs of 1000 jobs each. It has been argued that to succeed in the industry, grids have to be predictable [Ken02], or dependable [Fos98], or both [Ken03]; the ability of GMark-RI to assess performance metrics over time is critical in assessing the dependability and the predictability of the system. The mean and median job wait time are similar across different runs. Within each run, the range of the job wait times between the first and the third quartile is from 1281s to 1647s, or from 60% (Run 0) to 120% (Run 19) of the median time. Similarly, the full range of the job wait time values is from 150% to 240% of the median time, with an average of 190%. This shows that some jobs get better treatment in Condor than others. Figure 5.10 (right) shows which jobs are thus favored by Condor. The jobs arriving first exhibit high wait time variability. Overall, there is a trend for jobs arriving later to wait more than the jobs arriving earlier. There appears to be no correlation between the average wait time and the run index, i.e., later runs do not necessarily have a higher average waiting time.

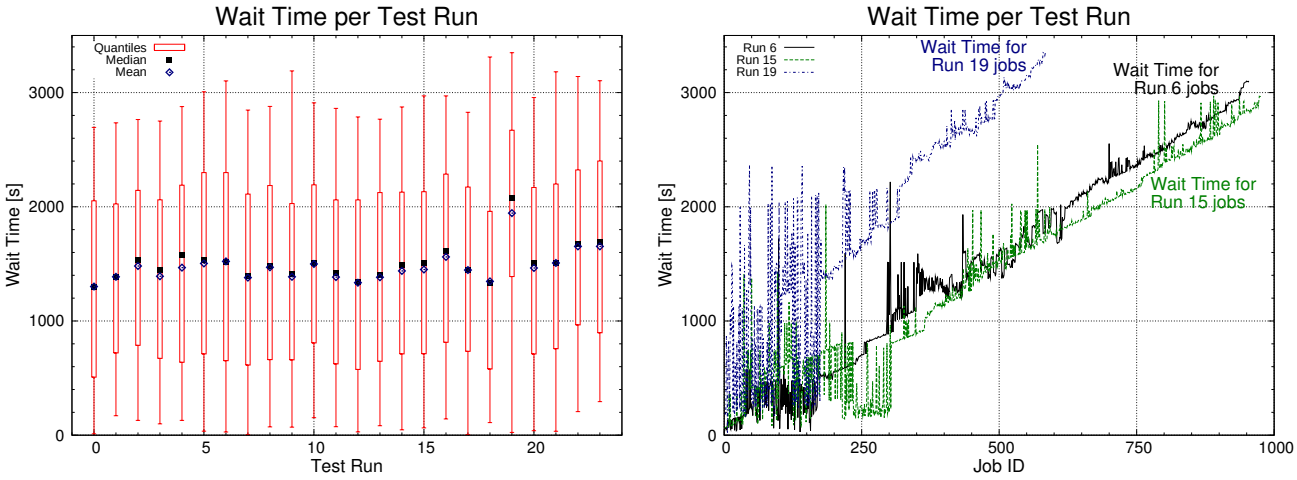


Figure 5.10: *Characterization of the job wait time. (*left*) The job wait time distribution for 24 consecutive test runs, depicted as* box-and-whiskers *plots with an additional point for the mean. (*right*) The job wait time per job, for 3 test runs, selected for low (Run 15), average (Run 6), and high (Run 19) wait time.*

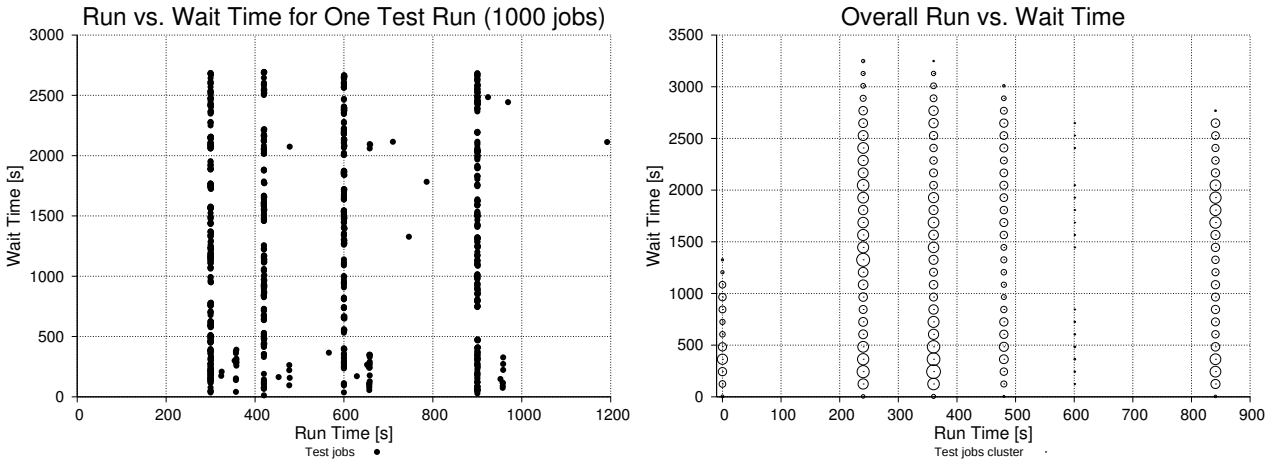


Figure 5.11: *Scatterplot of jobs run time vs. wait time. (*left*) Scatterplot for all jobs in one test run. (*right*) Scatterplot for all jobs in all tests. Each circle depicts a cluster of jobs with run time and wait time around the circle's center. Larger circles represent clusters with more jobs. Only clusters of at least 100 jobs are depicted.*

System Fairness

We investigate the slowdown fairness with which Condor dispatches jobs, where the slowdown of a job is expressed as the ratio between its run time and its response time (wait time plus run time). Ideally, the slowdown is identical for small and long jobs. Figure 5.11 presents scatterplots of jobs run time vs. wait time. Figure 5.11 (left) shows that there is no relationship between a job's run and wait time for jobs belonging to the same test run. Similarly, Figure 5.11 (right) shows that there is no relationship between a job's run and wait time for all the jobs in all tests, i.e., around 200,000 jobs. Furthermore, the variation of wait time does not change with the variation of run time. Thus, the Condor job dispatching is overall not slowdown fair.

We further investigate Condor's load balancing fairness. Ideally, Condor will assign an equal number of jobs per managed machine. Figure 5.12 (left) shows the Pareto chart of the distribution

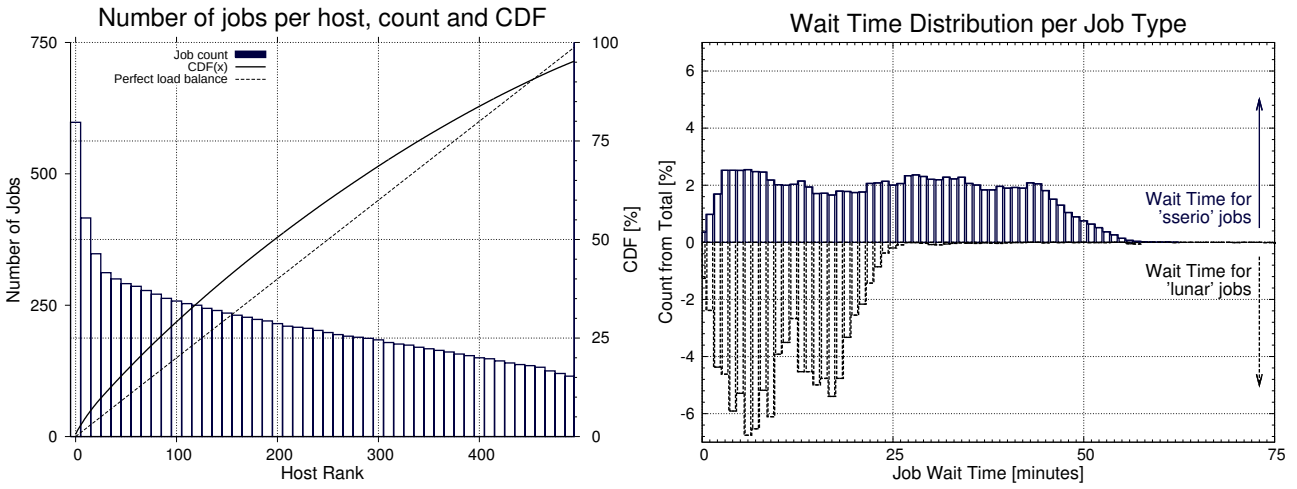


Figure 5.12: *(*left*) Condor load balancing fairness. Pareto chart of the distribution of jobs per host. Only ranks of the form* $10k, k = 1, 2, ...$ *are depicted with bars. (*right*) Condor batch size fairness. Bi-histogram of the job wait time, for two job types: 'sserio' (depicted upwards) and 'lunar' (depicted downwards).*

of jobs per host. The real and the ideal job distributions have similar CDFs. The plot confirms that Condor is load balancing fair. To obtain the data needed for this plot, GMark-RI combines test data with information obtained from the resource manager's logs, which has been added to the results database as annotation data (design goal 2b).

Finally, we look into the fairness in scheduling when jobs are submitted in batches of different sizes. Figure 5.12 (right) is a bi-histogram of the job wait time, for two job types: 'sserio' and 'lunar'. The 'sserio' jobs are submitted in batches of various sizes, from 5 to 200. The 'lunar' jobs are submitted only as batches of 5 jobs. The 'sserio' jobs have a wait time centered around 25 minutes while the 'lunar' jobs are centered around a wait time value of 12 minutes. With the exception of a few outliers, the spread of 'sserio' jobs' wait time is higher than that of 'lunar' jobs. With respect to distributional shape, the 'sserio' histogram is almost symmetric while the 'lunar' histogram is skewed right. In addition, the 'sserio' distribution shows no clear mode while the 'lunar' distribution has two: around 6 and around 16 minutes. It is therefore better to send jobs in small batches.

Application- and Service-level Performance

Figure 5.13 (left) shows the evolution of application- and of service-level throughput over time for 10 consecutive runs of 1000 jobs each. Each of the jobs explores states from a large search space with S dimensions, where S is specified as a parameter to the job. The jobs output the interesting states found in their exploration; an interesting state is a state whose properties satisfy application-specific constraints. We define the application-level (service-level) throughput as the number of (interesting) states explored (found) over the time unit, set here to one second.

The tested system achieves an overall application-level throughput of over 15 million states explored per second, and an overall service-level throughput of over 0.5 interesting states discovered per second. The performance decreases with time due to Condor's fair resource consumption (see Section 5.6.1). The decrease becomes predictable after 6 hours. This allows a service provider to reliably guarantee service levels, even with the Condor fair use policy in place. These experiments demonstrate the ability of GMark-RI to compute application- and service-level metrics, that is, metrics that depend on results reported by the user application. This is done through the Data Manager plug-in system.

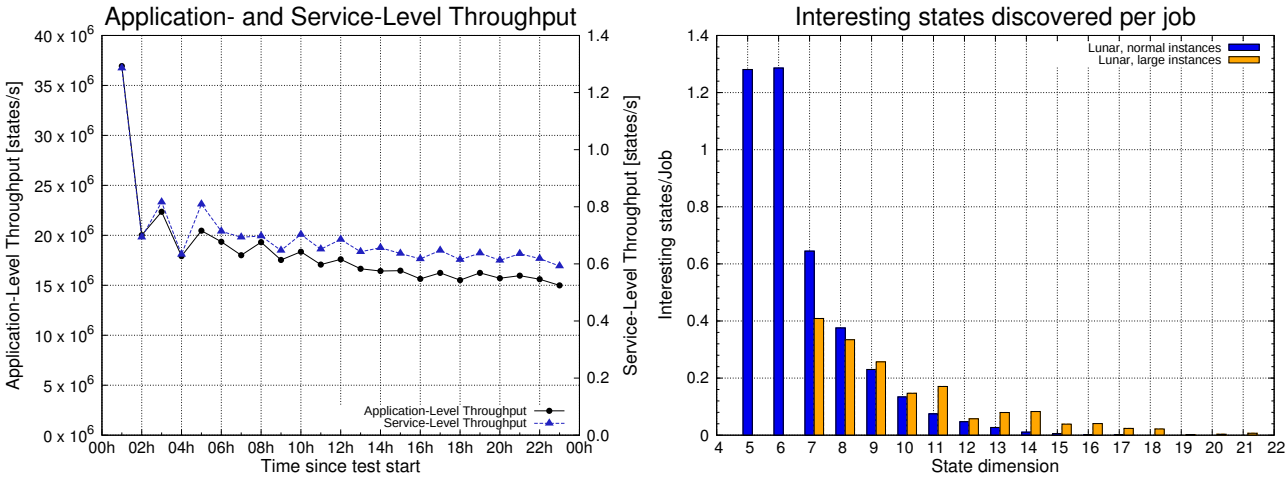


Figure 5.13: *(*left*) The evolution of application- and of service-level throughput over time. (*right*) Service-level goodput per state dimension for the 'lunar' application.*

Two types of test jobs are considered during the experiments: jobs that explore a normal-sized space (*normal instances*), and jobs that explore a large-sized space (*large instances*). We define the service-level goodput per state dimension S for the test jobs as the number of interesting states found in the space of dimension S. Figure 5.13 (right) depicts the service-level goodput for various state dimensions for both job flavors. The normal jobs have a much better service-level goodput for dimensions up to 7. For dimensions of 8 through 12, both the normal and the large jobs behave similarly. For dimensions of 13 and higher, the large jobs become the best choice. These experiments demonstrate the ability of GMark-RI to compute service-level metrics that depend on the job parameters. This enables a more fine-grained analysis and partitioning of results (i.e., it enables sub-...-sub-projects, based on job input parameters).

5.6.2 Testing a Multi-Cluster Grid

We describe in this section several testing scenarios in a multi-cluster grid. The main difference from the tests presented in Section 5.6.1 is that the resources used in these tests are assumed to be static, that is, the resources should exhibit a high availability rate. We find that for today's grids the ability to reliably run jobs is even more important than computational performance. We define a *successful job* to be a job that acquires its requested resources, runs, finishes, and returns all results within the time allowed for it. Unless otherwise stated, we use the success rate of the jobs as the performance metric for the following experiments. Another goal for these experiments is demonstrating that GrenchMark (and GMark-RI) can be used in a variety of testing scenarios.

For all the following tests we have used the DAS system[3] as an experimental environment. The DAS system comprises 5 clusters, with 32 up to 72 identical dual-processor SMP nodes each. The grid middleware is Globus, arguably the most used grid middleware; the Globus project counts over 5,000 file transfer servers and about 100,000 service gateways (a service gateway can manage anything from one cluster to one (shared) resource) [The07]. Koala, the DAS grid scheduler [Moh07], operates a job queue on top of all five clusters. Grid jobs enter the Koala queue, and are redirected to the local cluster queues. Additionally, jobs may be sent directly to the local cluster queues, by-passing Koala.

[3]Distributed ASCI Supercomputer, `http://www.cs.vu.nl/das2/`

Table 5.3: *The results for the first case of* what if *analysis.*

Workload	Submit rate scaling	No. jobs	No. CPUs	Success rate
DAS2-FS3-1x	1x	14	1-50	100%
DAS2-FS3-10x	10x	71	1-50	100%
DAS2-FS3-25x	25x	102	1-50	96%
DAS2-FS3-50x	50x	426	1-50	76%
DAS2-FS3-100x	100x	739	1-50	59%

Table 5.4: *The results for the second case of* what if *analysis.*

Workload	No.jobs	Job size	Success rate
DAS2-FS3 + OSC	1103	1-32	76%
DAS2-FS3 + CTC	863	1-16	70%
DAS2-FS3 + SDSC'96	873	1-32	77%

GMark-RI was used to generate and submit the test workloads to the grid queue. We used both modeled and real (trace-based) workloads as test workloads. Some jobs could not finish in the time for which they requested resources, and were stopped automatically by the KOALA scheduler. This situation corresponds to users under-estimating applications' runtimes, and illustrates the ability of GMark-RI to ensure realistic testing conditions. Each test is stopped 20 minutes after the submission of the last job. Thus, some jobs did not run, as not enough free resources were available during the time between their submission and the end of the workload run.

What-if analysis: system reliability under special conditions

We consider three use cases for illustrating GRENCHMARK's capabilities for *what-if* analysis: system change, grid inter-operability, and special situations. In the first two cases the environments under study have been thoroughly studied, and their workloads modeled [Win96; Hot96; Li05].

First, we consider the case where a testing environment would be placed under (much) more strain. The resource management middleware of the clusters in our testing environment has been replaced, due to the problems when faced with an increasing number of job submissions. As a result, the work of the whole DAS community was negatively affected for a period of two weeks. Such a change could have been prevented if the question *What if the current users would submit 10 times more jobs in the same amount of time? Or 50 times, or 100 times...* would have been answered during a quiet period. We take a thoroughly studied trace [Li05] of the DAS system and re-run it in the new environment. The trace was recorded when the previous resource manager was still in place, and contains over 425,000 jobs run throughout 2003. To limit the testing period, we consider only the first 1000 jobs submitted to one of the five DAS clusters over a period of maximum three hours. We also scale the jobs submission times to make them 10, 25, 50, and 100 times smaller than the original submission times. Table 5.3 details the filtered and scaled traces and shows the success rate, from the total number of submitted jobs. We conclude that the new resource management system (including KOALA) can handle without a decrease of reliability an up to 10 times increase in the job arrival rate, for the user base characterized by the input traces.

Second, we ask the question: *If the users of another environment could submit their workload to our testing environment, what would be the success rate of the jobs submitted by these combined communities?* This situation occurs when from two existing production environments one environment is put temporarily or permanently out of production, and only one environment remains to run the submissions of jobs from both user communities. We took the same trace as in the first *what-if* analysis

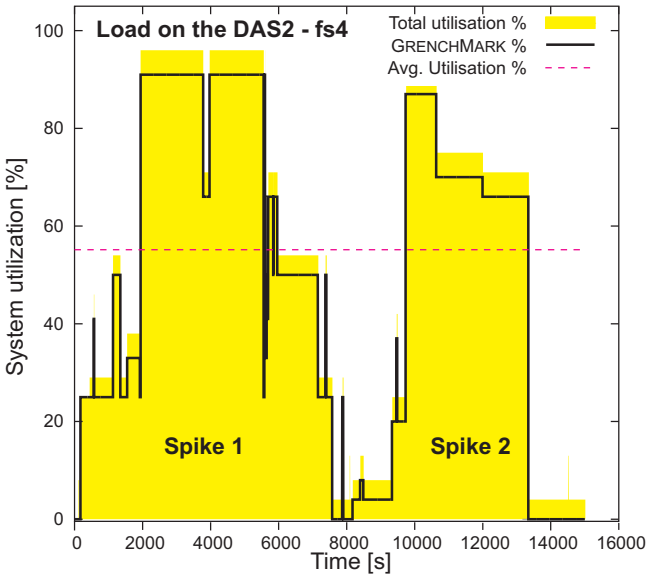


Figure 5.14: *The utilization of the DAS fs4 cluster for* bursty *submissions.*

case, and combined it with either of the OSC, the CTC, or the SDSC'96 traces from the PWA. We selected just the jobs with runtimes below 900 seconds. For each trace, we select the representative areas with $W = 3h$ and the desired characteristic "number of job arrivals equal to the 90% percentile of the trace"(see Section 5.4.4). Table 5.4 shows the detailed structure of the generated workloads, and the success rate for running them. The observed success rate was above 70% for all tests. We conclude that the DAS (including KOALA) can handle the proposed combinations of communities, with a reasonable success rate for submitted jobs.

Third, we consider the question *What if the system is suddenly subjected to a large number of submitted jobs (bursts)?*. Using workloads based on synthetic applications, we generated two bursts, and for each a background load, filling 25% and 5% of the system's capacity, respectively. During the tests, the background load generated by other users was between 0% and 5%. We restrict our tests to one of the DAS clusters (fs4). Figure 5.14 shows the system utilization graph during the period when the two workloads are executed. Two spikes can be clearly identified on the utilization graph. As expected, running the first burst workload (spike 1) takes longer, due to the higher background utilization of the system. The system performs correctly under strain: the maximum system utilization is 95%.

Comparing grid settings

This section presents two situations in which GRENCHMARK can be used for comparing grid settings. We consider two important features of grids: co-allocation (i.e., the simultaneous allocation of resources from several locations (e.g., clusters) for a single application [Cza99]), and workflow execution (i.e., the ability to execute a set of jobs, where a job's start may depend on the successful finish of other jobs [Yu05]). We call *composite jobs* the workflows that can be represented as directed acyclic graphs (DAGs) where the nodes represent jobs and directed edges represent job dependencies. We further call *single-site jobs* the jobs that are not co-allocated, and *unitary jobs* the jobs that have no dependencies, and of which no other job depends.

Table 5.5: *Results of the comparison of success rates for single-site vs. co-allocated jobs.*

Workload	# of CPUs	Component no.	Component size	Success Rate
`single-site`	2-32	1	2-32	98%
`coallocated`	2-32	1-8	1-16	77%
`coallocated+`	2-128	1-15	1-16	71%

Table 5.6: *Comparison of success rates for unitary vs. composite jobs with or without execution fault tolerance.*

Workload	# of Jobs/ Sub-Jobs	Success rate when fault tolerance is OFF	Success rate when fault tolerance is ON
`unitary`	100/100	90%	100%
`composite`	10/108	70-90%	100%

We consider a comparison between the success rates of single-site and co-allocated jobs in a grid environment without reservation capabilities. Executing single-site jobs should be simpler than executing co-allocated jobs. We use three workloads of 100 jobs each, one with single-site jobs, one with the same jobs co-allocated at various sites, and one with larger jobs co-allocated at various sites. Table 5.5 shows the detailed structure and the success rate for the three workloads. As expected, the single-site jobs have a higher success rate than the co-allocated jobs, in this case by over 20%. This is due to the atomic reservation problem of co-allocation: when the resources cannot be reserved, local users can acquire resources selected for co-allocation just before they are claimed by the co-allocating scheduler. With the average DAS system load below 25% [Li05; Ios06a], small and large co-allocated jobs have almost the same success rate.

We consider a *comparison between the success rate of unitary and of composite applications in a grid environment without reservation capabilities, and with or without fault tolerance.* We call *sub-jobs* the jobs that are part of a composite job; we consider that all sub-jobs of a composite job are submitted at the same time. We consider only composite applications for which the sub-jobs are unitary applications. We define a job that *can run* as a job for which all its dependencies have been met. Since KOALA does not support the execution of composite jobs, we have built a simple execution tool, which executes jobs that can run as soon as they become available. The execution tool also allows for flexible fault tolerance schemes; we use here a simple *retry failed* mechanism with sub-job granularity. Table 5.6 shows the success rate for unitary and composite jobs, with or without fault tolerance. Without fault tolerance, the success rate of composite jobs drops quickly if the first sub-jobs in the DAG's topological sort fail. With fault tolerance, for a system with a high success rate for jobs, e.g., the DAS, the simple retry mechanism is enough to ensure optimal reliability.

5.6.3 Discussion

The tests presented in this section have followed the uniform GRENCHMARK testing process. In addition, GRENCHMARK has automatically provided answers for the most common test questions. Table 5.2 lists in the columns "Metrics" and "Graphs" some of these answers. The results are always presented in the same way, and may take into account the system's characteristics (e.g., size), to facilitate the comparison across systems. Thus, using GRENCHMARK enables sharing the results across a larger community of practitioners, and increases the confidence in individual results versus the case when the results are obtained using distinct testing processes[4]. Using our reference implementation,

[4]We paraphrase here Metcalfe's law, which states that the value of a telecommunications network is proportional to the square of the number of users of the system.

GMark-RI, has two additional benefits.

First, GMark-RI has allowed performing a large variety of test scenarios that can be broadly grouped into three areas: grids, large-scale heterogeneous computing, and peer-to-peer systems. For peer-to-peer systems (see Table 5.2, group C), GMark-RI was instrumental in trace-based testing the functionality of a peer-to-peer protocol, and in assessing the reliability of a peer-to-peer system against resource failure (churn) and malicious user behavior (pollution) [Roo06].

Second, GMark-RI has greatly reduced the time needed to perform the tests. The selection of the appropriate test scenario depends very much on the (human) test designed, and may take anywhere from a few hours to a few days, regardless of the testing tool. However, creating the testing scenario after making the selection only takes a few minutes. Similarly, changing the parameters of the scenario is also a matter of minutes. This greatly improves the currently prevailing practice in the LSDCS community of writing a new testing tool for every new occasion. Finally, extracting metrics that have not been defined in the default Data Manager's analysis tools means in GrenchMark (and GMark-RI) writing appropriate SQL queries, which is much faster than writing custom data processing tools.

5.7 Research Directions

In this section we present two directions for future research related to the GrenchMark framework, towards becoming the basis of a common performance evaluation framework for large-scale computing environments such as grids.

Enabling Testing Capabilities on Large-Scale Experimental Platforms Large-scale experimental computer science platforms have been and are currently being built with the purpose of assessing the characteristics of computer systems in realistic conditions. Several such platforms already exist, e.g., PlanetLab [Chu03], Grid'5000 [Bol06], etc. While the middleware pre-installed on these testbeds includes deployment and coordination tools (e.g., the CoDeeN[5]), it does not include support for testing: it lacks test management, workload generators, etc. Thus, GrenchMark can provide for these testbeds support for real-life testing. However, two research questions need to be further investigated: (a) How to provide testing as a reliable service to the user in an unreliable, large-scale, and distributed system?, and (b) How to ensure the reproducibility of the experiments in such systems? With GrenchMark, we have taken first steps in both these directions, by saving provenance data, generated workloads, and even temporary results to the (remote) Data Manager. More work is required in distributing the testing tools (especially the workload submitter) for large-scale fault-tolerant testing, in allowing the workload generator to respond to changes in the tested system's state, and in creating realistic background testing conditions (load and failures).

LSDCS Performance Database Enabling testing on large-scale experimental platforms is only the beginning in providing a basis for LSDCS performance evaluation studies. Towards this end, we plan to cover two orthogonal issues: creating a grid workload model, and creating an LSDCS performance database. The main feature of GrenchMark, the ability to deal with grid workload traces, has been hampered until recently by the lack of traces. Indeed, many organizations view this data as revenue-generating (in the industry), or critical for obtaining grants (academia), and are reluctant to make the data public. To address this issue, we have created in Nov 2006 the Grid Workload Archive [Ios08d]. We have gathered until Aug 2007 traces from 9 grid environments,

[5]CoDeeN is a Content Distribution Network for PlanetLab. It includes tools for web-based distribution (CoBlitz), synchronized deployment (CoDeploy), monitoring (CoTop/Mon/Viz), etc. [Online] http://codeen.cs.princeton.edu/. Jul 2007.

Table 5.7: *A summary of testing projects in parallel and distributed computing. The +, ∼, and - signs denote a feature that is present, for which an insufficient approach has been taken, or which lacks, respectively.* *Acronyms: Cl - cluster, G - grid, I - Internet, P2P - peer-to-peer (area), C - computing, D - data transfer, S - data storage (sub-area), P - probe, μK - micro-kernel, AB - application benchmark, SWL - simple workload, CWL - complex workload (realistic), AD - ADAGE, BT - BitTorrent, DN - DummyNet, G5K - Grid5000, I - Internet, PL - PlanetLab (Peer-to-Peer).*

Project	Area	Sub-Area	Design Goals (see Section 5.2) 1 realistic	2 reproducible a	2 reproducible b	3 high-perf. a	3 high-perf. b	4 extensible a	4 extensible b
GrenchMark	G,P2P	all	all	+	+	+	+	+	+
Grid/Cluster benchmarking									
HPCC [Lus06]	Cl	all	μK,AB	+	-	-	+	-	-
NAS NGB [Van02]	G	C,D	μK	+	-	-	-	-	-
GridBench [Tso05]	G	C,D	μK,AB	+	-	-	+	+	+
Grid performance evaluation									
DiPerF [Rai06]	G	C	SWL	+	-	+	-	+	+
Quake [Hoa07a]	G	C	SWL	+	-	-	-	-	+
Grid functionality testing									
INCA [Sma04]	G	C	P	+	+	-	+	-	-
GRADS [Chu04]	G	C,D	AB	+	-	-	-	-	-
NMI Test [Pav06]	G	C	AB	+	-	-	+	+	+
Internet testing									
NIMI [Pax98]	I	D	CWL	∼	+	+	-	+	-
WAWM [Bar99]	I	D	CWL	∼	+	+	∼	+	+
NETI@Home [Sim04]	I	D	P	-	∼	-	-	-	-
Custom peer-to-peer testing (Target/Environment/Additional tools)									
DHT/PL [Dab04]	P2P	D	SWL	+	-	-	-	-	-
several/G5K/DN [Bus05; Nus07]	P2P	D	SWL	+	-	-	-	-	-
2Fast/DAS+I [Gar06]	P2P	D	SWL	∼	-	-	-	-	-
JXTA/several [Hal05; Ant07]	P2P	D	SWL	+	-	+	-	-	-

with a total content of more than 2000 users submitting more than 7 million jobs over a period of over 13 operational years, and with working environments spanning over 130 sites comprising 10000 resources. We have shown in our previous work [Ios06a; Ios07d] that grid workloads can have very different properties from those in other computing environments, and in particular from existing parallel production and supercomputing environments [Jan97; Lub03]. We now intend to research aspects in LSDCS grid workload modeling. The second direction of work is the creation of an on-line, freely-accessible LSDCS performance database. We plan to use GMark-RI and run benchmark-like grid workloads in LSDCS environments, and to report the results in a standard form. Similarly to the Performance Database Server [Ber94], the CADRE [Pab07], the Prophesy [Wu01], and the TOP500 Supercomputer Sites [Don95; Don07] projects, which offer free access to benchmarks and performance data for high-performance computing systems, the LSDCS performance database could be used for several purposes. First and foremost, various statistical techniques could be applied to the data within to gain insights into the causes of poor performance. In particular, this approach would enable thorough work on performance models for grids and other LSDCSs. Second, data mining can help reveal classes of equivalence for LSDCS systems. This would facilitate comparing grids and other LSDCSs with other types of systems, e.g., clusters and supercomputers, for procurement purposes. This would also facilitate automated resource selection, a current major problem in LSDCSs.

5.8 Related Work

We survey several studies related to testing in real environments, from testing grids to testing large-scale distributed systems. Table 5.7 shows a summary of these studies; more details about a few selected projects are given below. Compared to previous grid testing tools, GrenchMark focuses more on the testing process, with additional types of workload data sources, richer workload generation, and more detailed results analysis. Compared to Internet testing tools, GrenchMark takes specific steps to ensure repeatable testing (rare in the Internet), and adapts the realistic workload-based testing paradigm to the specifics of grids. Finally, compared to other peer-to-peer testing approaches, GrenchMark is the first to propose a rigorous testing procedure.

Grid/Cluster benchmarking Following results from the parallel systems community, e.g., the NASA Parallel Benchmarks, several grid performance evaluation and benchmarking approaches focus on tests using micro-kernels, and application benchmarks [Lus06; Van02; Tso05]. These approaches have a limited applicability in grids, due to the heterogeneity and the dynamic state of grids.

Grid performance evaluation Few performance evaluation tools have been proposed in the context of grids [Rai06; Hoa07a]. Even for these few, the research effort has been directed to providing distributed deployment and testing solutions. Thus, relatively little focus has been given to realistic workload generation, and to detailed results analysis. The DiPerF [Rai06] is a distributed performance evaluation system that can be used to study the performance properties of Grid and Web services. The QUAKE tool [Hoa07a] is a distributed benchmarking tool that can be used to study the dependability properties of Grid and Web services. The QUAKE architecture is similar to that of DiPerF, except that through an external failure injection tool, i.e., the FAIL-FCI [Hoa07b], QUAKE can be used for dependability testing.

Grid functionality testing Several testing suites are available for functionality testing, based on the submission of single jobs or of simple workloads to the tested grid [Pav06; Chu04; Sma04; De 07; Ric04].

Internet testing The research road for testing the Internet and the World Wide Web has started with simple observations (and characterization) from a single point in the network, and is still evolving towards completely decentralized testing using realistic workloads. We survey here only three recent advances. For more of the same, we refer the reader to [And02]. The National Internet Measurement Infrastructure (NIMI) project [Pax98] deals with problems related to scale: decentralized control of measurements, authentication and security, and deployment. The Wide Area Web Measurement (WAWM) project [Bar99] uses nodes distributed across the Internet to study Web performance, and the SURGE [Bar98] Web traffic generator to create representative workloads on tested servers. The NETI@Home project [Sim04] collects network performance statistics from end-systems at a central server, and analyzes them off-line. Recently, the problem of submitting workloads in real environments has been re-considered for the case of high-performance networks, and several tools for traffic generation have been proposed and evaluated [Ava05; Hor07].

Peer-to-peeer testing The performance of peer-to-peer systems is mostly evaluated through analysis and simulation. However, with large-scale file-sharing networks, e.g., BitTorrent [Coh03], KaZaA, Gnutella, or e-Donkey, currently used by millions of users, testing in real environments is becoming increasingly important [Dab04; Bus05; Nus07; Gar06; Ios06e]. Relative to other peer-to-peer systems, the performance of JXTA, a Java-based peer-to-peer communication library, has been the subject of numerous studies [JXT07; Hal05; Ant07]. The JXTA community initiated the Bench project [JXT07], to collect performance and scalability measurements from various environments. The work in [Hal05; Ant07] evaluates the performance of the primitive JXTA operations in LAN and

WAN settings, respectively. A DHT-based P2P is evaluated on a custom testing environment using several hundred peers in PlanetLab [Dab04]. Various P2P systems have been evaluated on custom environments built on top of Grid5000 [Bus05; Nus07]. 2Fast, a BitTorrent-based file-sharing protocol, is also tested in a custom testing environment built on top of the DAS and Internet [Gar06].

Acknowledgements

We would like to thank the Politehnica University of Bucharest team (in particular to Dr. Nicolae Tapus, to Corina Stratan, and to Mugurel Andreica) for their help with extending the GrenchMark framework towards distributed testing and towards testing complete grid middleware stacks. We are also grateful to the DAS-2 team (in particular to K. Verstoep and P. Anita) and to the Condor UWisc team (in particular to Dr. M. Livny) who kindly provided access to their computational environments for our tests.

5.9 Concluding remarks

Today's integration of tens of thousands of resources into single large-scale distributed computing systems brings with it performance, reliability, and functionality problems, which may cancel out the benefit of integration. Understanding the causes and the extent of these problems is an important research topic. To address this problem we have presented in this chapter the GrenchMark framework for testing large-scale distributed computing systems. The framework focuses on realistic and repeatable testing, and on enabling test results comparison across systems. GrenchMark is the first approach to meet all the design goals specific to frameworks for testing LSDCSs: realism, reproducibility, high performance, and extensibility.

We have evaluated the performance and the robustness of the GrenchMark reference implementation, GMark-RI, and have shown that they are appropriate for testing LSDCSs. Then, we have used GMark-RI in 25 fully-automated testing scenarios in the areas of performance evaluation, reliability testing, and functionality testing. Notably, we have performed hundreds of tests comprising hundreds of thousands of test jobs in large Condor-, Globus-, and BitTorrent-based environments; the results presented in this chapter have been obtained in real production systems.

Chapter 6

The Delft Grid Simulation Framework

6.1 Overview

In this chapter* we present the Delft Grid Simulation (DGSim) framework for simulating grid settings.

6.1.1 Motivation and Problem Statement

Many of today's grid resource management (GRM) components such as job scheduling architectures, data management algorithms, and inter-operation mechanisms, need to be improved or even (re-)designed from scratch. In this process, simulation is critical to assess large numbers of experimental settings in a reasonable amount of time, and to design and evaluate the systems of the future. Despite steady progress, today's grid simulators [Buy02; Ran02; Jon04; Cam04; Dum05; Kur07; Car07] still lack important features in system modeling, experiment setup, and experiment management. To address these issues, in this chapter we answer an important research question: *How to build an adequate simulation framework for grids?*

6.1.2 Key Ideas and Selected Results

To answer the question formulated in the previous section we have designed the DGSim simulation framework. Our belief is that three decades of evolution in general computer simulation techniques are sufficient to enable the development and optimization of grid resource management simulators (GRMS) by most interested scientists. In particular, many books [Fuj99; Law00; Ban01] and survey articles [Fis95; Per06] attest to the progress made in parallel and distributed discrete event simulation. Therefore, we focus with DGSim less on generic simulation techniques, and more on the specific requirements of the GRM simulation process, for instance, by automatically generating realistic grid systems and workloads, by supporting typical GRM components, and by automating and speeding up the typical experiment (see Section 6.2). Our contribution is twofold:

- We design the DGSim framework for simulating GRM architectures, focusing on workload and system modeling, and on automatic experiment setup and management ("from configuration file to results");
- We make methodological improvements to the area of GRMS, by introducing the concepts of grid evolution and job selection policy, and by extending the state-of-the-art for grid inter-operation, grid dynamics, and workload modeling.

*This chapter is based on previous work published in Euro-Par 2008 [Ios08f].

6.1.3 Organization of this chapter

The remainder of this chapter is structured as follows. In Section 6.2 we present the requirements of a simulation framework for grid resource management architectures. Then, we describe the resulting design and the main features of the DGSim framework in Section 6.3, its validation in Section 6.4, and its past, current, and potential use in Section 6.5. Finally, in Section 6.6 we survey existing simulation approaches for distributed computing systems, and compare the DGSim framework with the state-of-the-art.

6.2 Requirements for Simulating Grid Resource Management Architectures

In this section we synthesize the design goals for completing the path from hypothesis to simulation results in grid resource management research. Our motivation is twofold. First, this type of research has specific requirements, e.g., unique experimental setup characteristics. Note that these requirements do not only fit grids, but also large-scale distributed computing systems in general. Second, in spite of the evolution of GRMSs over the last decade (see Section 6.6), there is still place for improvement, especially in automating the experimental setup and in supporting large-scale experiments. Below we present seven requirements, divided into three categories, specific to simulation frameworks for GRM research; in Section 6.6 we use them to evaluate the related work.

1. **Experimental Setup** A major challenge in today's grid simulation landscape is designing a realistic experimental setup. Specifically:
   (a) **Grid System** There is a need to generate models of grids that include the topology, resource, and inter- and intra-operation (e.g., service level agreements) of real grids. The model may also include varying resource availability.
   (b) **Grid Dynamics and Evolution** It is important to evaluate resource management solutions for future system configurations. With grid systems being highly dynamic in the number and the size of their resources on both short and long term (see also Section 6.3.3), there is need to generate grid dynamics and evolution events.
   (c) **Grid Workload** To ensure that tests are performed under conditions that match the test design, there is a need to synthesize realistic grid workloads.
2. **Experiment Support for GRMs** Perhaps the most limiting factor in the adoption of a simulation framework is the degree of automation.
   (a) **Performing Grouped Simulations** There is a need to automate the execution of groups of individual simulations, e.g., for a specific scenario. The simulation frameworks should be able to run in an unreliable environment, i.e., simulating grids *on* grids.
   (b) **Performing Batches of Simulations** There is a need to automate the execution of batches of simulations with different parameter settings. The simulation frameworks must generate the corresponding configuration files, ensure that the simulator is executed, and collect the results for on-line and off-line analysis.
   (c) **Storing and Analyzing Simulation Results** Large amounts of data are produced when simulating grid resource management solutions. As a first step towards Online Analytical Processing (OLAP) capabilities, there must exist a hierarchy for results storage and aggregation, i.e., by experiment, by scenario, by run.Second, to promote the comparability of

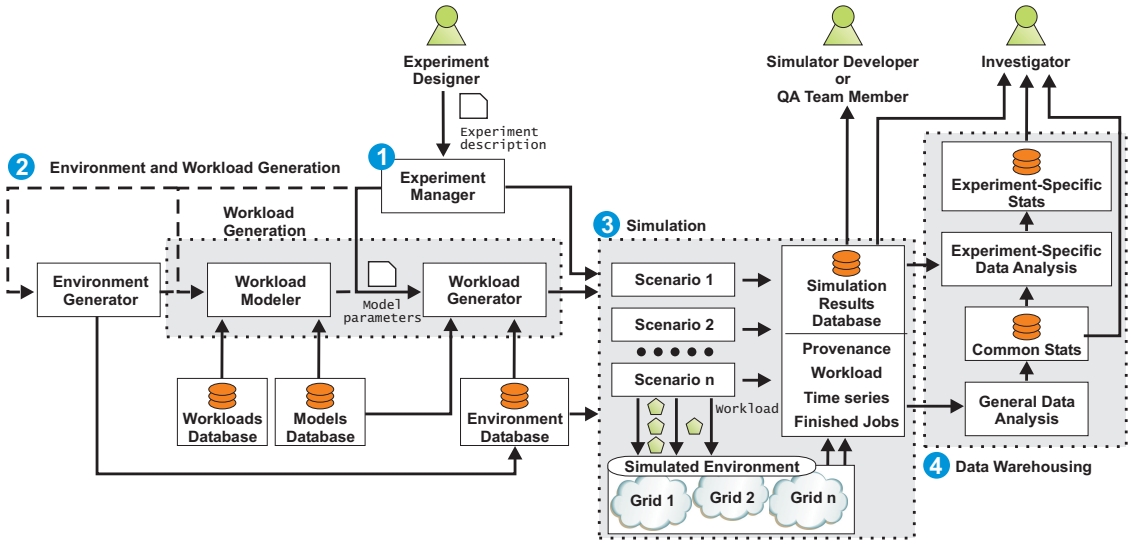


Figure 6.1: *An overview of the design of the* DGSIM *Framework.*

results across different research groups, a standard procedure for analysis and presentation of results must be followed. In particular, the most common grid resource management metrics should be automatically extracted from the simulation results.

3. **Performance** Given the sizes of today's clusters, and even those of real experimental grids, there is a need for simulated environments of at least several thousands of processors. At the same time, given the high percentage of single processor jobs present in grid workloads [Ios06a], there is a need for a simulator to deal with workloads of tens of thousands to even millions of jobs. We therefore describe the performance of a simulation framework with five metrics: (M1) experimental setup time, (M2) the number of scenarios, (M3) the number of individual simulations, (M4) the size of the simulated systems, and (M5) the workload to which the simulated systems are subjected. The individual simulation time is not among these metrics, as it depends heavily on what is actually simulated, e.g., the execution of a simulator of an NP-complete scheduling algorithm may not finish regardless of the actual capability of the simulation engine. Similar to the discussion about workload realism, good performance depends on the specifics of the experiment.

6.3 The DGSIM Framework: Design and Implementation

In this section we discuss the design and the implementation of the DGSIM framework.

6.3.1 Overview

We envision four main roles for the DGSIM user. The *Investigator* formulates the problem and is interested in the results. The *Experiment Designer* is responsible for designing and performing a set of experiments that give answers to the problems raised by the Investigator. The *Simulator Developer* helps the Experiment Designer perform the experiments by implementing novel components

for the simulator, e.g., a new scheduling algorithm, or a new analysis tool. The *QA Team Member* is responsible for validating the experiments, e.g., by duplicating simulation results.

Figure 6.1 shows an overview of the DGSim design. The arrows represent the direction of data flows from components. The main components of DGSim are:

1. The Experiment Manager coordinates the flow of an experiment. If the Experiment Designer did not fix the experimental environment, it calls the Environment Generator. Similarly, if the workload model parameters are not fixed, it calls the Workload Modeler. After generating the environment and the workload, the Experiment Manager calls the Simulation component.

2. The Environment and Workload Generation component automates the experimental setup. The Environment Generator will generate for each environment the starting topology, which includes the resource specification and the logical inter-connection between resources, and the system dynamics and evolution (see Section 6.3.3). Similarly, the Workload Generation ensures that realistic workloads are generated or simply extracted from existing grid workload traces. This component can also automatically extract parameters for a given model from an input grid workload trace.

3. The Simulation component executes large numbers of individual simulations grouped by scenario and run. The computing power may be provided by one machine (e.g., the scientists' computer) or by a grid. The Simulation component also collects all the results and stores them into the Simulation Results Database for future analysis. The stored data are the simulation results and provenance data (simulator parameters, version, and events).

4. The Data Warehousing component is responsible for organizing data, and for data analysis. Data are indexed by experiment, by scenario, and by run. This component analyzes for all simulations the time series of job arrivals, starts, and completions, and produces statistical results for the common performance metrics, e.g., time-related, related to resource consumption, and related to resource waste. It also analyzes the messages exchanged by the simulated entities. Additional analyzers may be used to process the data.

6.3.2 A Model for Inter-Operated Cluster-Based Grids

In this section we present the system model used in the DGSim framework. Our model extends previous grid simulation approaches (see Section 6.6) in several ways. In previous work, all the jobs in a queue are considered as the input set of the scheduler; our job scheduling model also allows the selection of only some of the queued jobs, through a selection policy. In current grid simulators, the resource management architecture is either purely centralized or purely decentralized. In comparison, the inter-operation model of DGSim also considers the hierarchical and the (more realistic) hybrid architectures. Finally, existing simulators have used an information model in which only one of the job runtime, the cluster dynamic status, and the resource speed is not (accurately) known. In contrast, the information model of DGSim allows all the pieces of information used in the scheduling process to be inaccurate or even missing.

The resource model (already introduced in Chapter 2) assumes that grids are groups of clusters of homogeneous resources, e.g., processors, network, storage. There is no logical connection between the use of various resource types hard-coded in the simulator. The clusters may have logical links with other clusters (e.g., see Section 6.3.2); the clusters may be virtual, that is, contain no resources. We use the Worldwide LHC Computing Grid (WLCG) [07b] as a running example. The computational

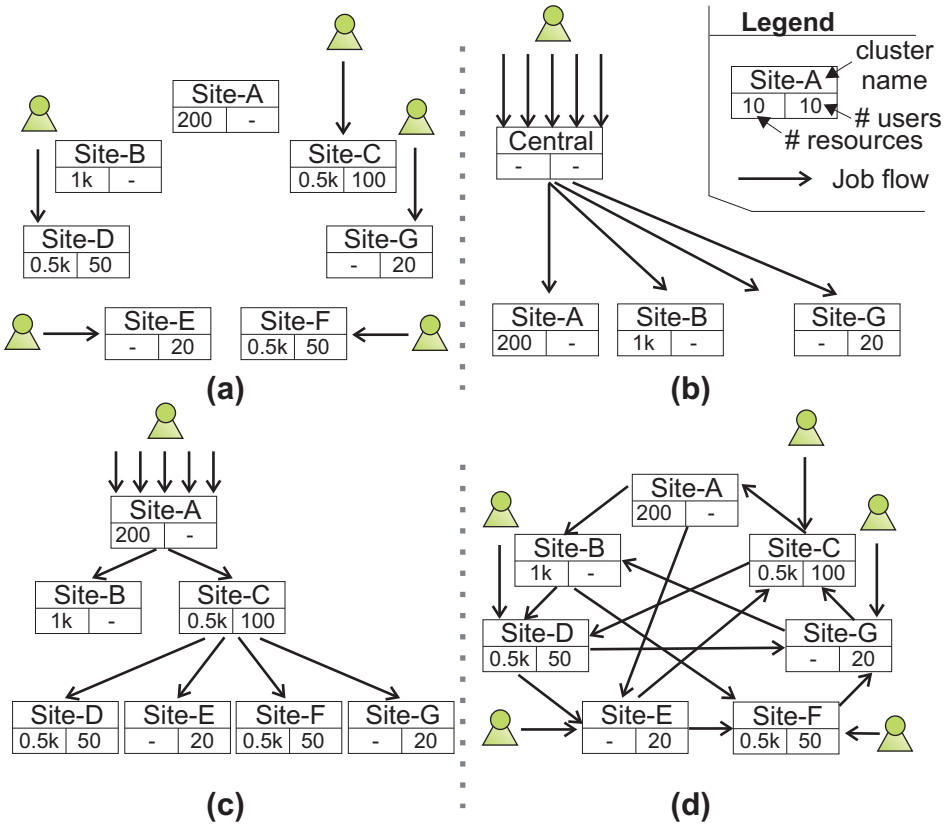


Figure 6.2: *Grid inter-operation: (a) independent clusters; (b) centralized meta-scheduler, (c) hierarchical meta-scheduler; (d) distributed meta-scheduler.*

speed of a resource is modeled similarly to WLCG's[1], that is, in SPECInt2K[2]; the values for WLCG's processors speeds range from 0.1 to 6.0, with an average of 1.184.

The workload model (also introduced in Chapter 2) takes into account individual jobs, job grouping (e.g., batches, workflows), and the user/VO to whom the job belongs. We also introduce the notions of a user's local cluster, where all the jobs of a user are submitted, and of job's arrival cluster. We assume the execution time of a job to be proportional to the speed of the processor it runs on.

The inter-operation model While many of today's grids still operate in isolation, an integration trend has recently emerged, similar to what happened in the past decades with the growth of the Internet from isolated networks. We therefore integrate in our simulation framework the concept of grid inter-operation. For this purpose, we have implemented six architectural alternatives, of which four are depicted in Figure 6.2; Chapter 8 presents each of these architectures in detail. The independent (or isolated) clusters architecture assumes there is no load exchange between the clusters. The centralized meta-scheduler assumes the load exchange is coordinated by a central node. With hierarchical meta-scheduler, the load exchange is coordinated by a hierarchy of schedulers. The distributed meta-scheduler assumes there is no central control for load sharing and each cluster can exchange load with any other cluster. For the multi-level architectures, DGSim already provides several load distribution

[1]For all the WLCG-related information used in this chapter, the data sources are http://goc.grid.sinica.edu.tw/gstat/ and http://pcalimonitor.cern.ch/reports/.

[2]SPEC's Int2K, [Online] Available: http://www.spec.org/cpu2000/.

policies, e.g., Worst- and Best-Fit, Round-Robin, and load-dependent.

The job scheduling model Each cluster contains one or more scheduling queues. From each queue, the queue's *job selection policy* extracts an eligible set of jobs to be scheduled; the eligible set can be ordered by tagging the extracted jobs. The eligible set is scheduled on the local resources using the queue's *job scheduling policy*. The events which trigger the job selection and the job scheduling policies are dependent on the simulated setup, that is, they can be set by the user. A DGSim user can easily modify or re-implement the selection and the scheduling policies at various levels of the GRM architecture, using the Python scripting language. DGSim already provides seven selection policies, and over ten scheduling policies, including FCFS, two FCFS/backfilling variants, and eight scheduling policies for bags-of-tasks [Cas00]. While we have not experimented with several queues at the same cluster, e.g., to simulate a cluster with per-VO queues, the DGSim user can implement such a setup and use the workload model's user/VO characteristics to set the arrival queue of a job, and prioritizing the treatment of queues.

The information model deals with the accuracy and the timeliness of the information in an inter-operated (distributed) grid. A scheduling policy that deals with a job needs pieces of information that may be unavailable, inaccurate, or unknown in a realistic setting: the job's amount of work to be computed, a remote cluster's dynamic status, and even the remote cluster's static resource characteristics. Information inaccuracy can be the result of predictions; the DGSim implementation currently provides over ten predictors widely used in large-scale distributed systems [Sah03].

6.3.3 Grid Dynamics and Grid Evolution

We have already argued that grids change over time. We identify two types of change: over the short term, determined by dynamic resource availability (e.g., processors failing), and over the long term, determined by static resource availability (e.g., the addition of a new cluster to a grid, or of new processors to its clusters). We call the former *grid dynamics*, and the latter *grid evolution*. Only one of the surveyed grid simulators considers grid dynamics, i.e., GSSIM [Kur07]; none of them considers grid evolution. Moreover, in comparison with GSSIM, our grid dynamics model also considers the concept of correlated failures, that is, of failures that affect several resources at the same time; this phenomenon is common in and important for the performance of both clusters and cluster-based grids [Zha04; Ios07e].

The grid dynamics model considered by DGSim describes the changes in resource availability status, and includes the following aspects: the clusters where a change occurs, the number of resources involved in the change, the time when the change occurs, and the time until the next status change for the same resource(s). DGSim currently implements the realistic grid dynamics model proposed in [Ios07e] and detailed in Section 4.3.1.

The grid evolution model (introduced in Section 4.3.2) The grid evolution phenomenon is depicted in Figure 4.12: in less than three years, the WLCG has grown from 100 to 300 clusters; during the same period, the median cluster size has increased by just 20% (not shown). To account for grid evolution, we have designed a generator that for a grid system outputs the topology and resources for a time frame that spans from "present" to "far-future". A researcher can opt to perform simulations only for the present grid system, or for several sizes up to the far-future.

6.3.4 Grid Workload Generator

One of the most important problems when performing simulations of computer systems is setting the input workload. The experiment results may not be meaningful when using unrealistic workload models. To assess the behavior of a resource management component, workloads that incur the same average system load but have very different characteristics need to be generated. Moreover, it is sometimes useful to have some overlap of the characteristics of the generated input, to put in evidence the impact of the non-overlapping characteristics; then, only the values of some of the model parameters can be changed.

Realistic workload models DGSim currently supports two realistic workload models: the Lublin-Feitelson (LF) model for jobs in parallel supercomputers and clusters [Lub03] and a new model for bags-of-tasks in grids; a detailed description of the latter is presented in Chapter 4. The LF model considers for a job its parallelism, its runtime, and their correlation, and for the whole workload the arrival patterns (i.e., daily cycle, peak hours); this model has been validated using four long-term traces from parallel production environments and used by numerous grid researchers [Cas06; Ios07c]. The model for bags-of-tasks in grids considers, in addition to the LF model, the individual users and the grouping of jobs in bags-of-tasks; this model has been validated using seven long-term grid traces, including five from the Grid Workloads Archive (GWA) [Th08].

Iterative workload generation The realistic grid workload models are complex, that is, they have many parameters and correlations between parameters. Generating a workload with a desired load from a complex model is difficult, especially when only some of the model parameters can be changed. To this end, we have developed an Iterative grid Workload Generation algorithm (IWG). The IWG algorithm takes as input a description of the target system (S), the target load (L), the maximum deviation from the target load ϵ, the workload model, and the fixed parameters of the model. It outputs a workload generated with the model such that the workload incurs on the target system the target load (taking into account ϵ). To achieve this, the algorithm tries to iteratively change (increase or decrease) the value of a non-fixed parameter, e.g., the job inter-arrival rate. The algorithm quits after a fixed number of iterations.

6.4 Testing the DGSim Reference Implementation

In this section we present validation and performance evaluation tests performed on our DGSim reference implementation (DGSim-RI). The performance results, coupled with the sizes of the environments presented in our previous work [Ios07c; Ios07e], demonstrate that DGSim-RI fulfills the performance-related requirements of GRMs.

6.4.1 Validation

Arguably the crux of developing a simulator is its validation. According to Sargent, "a model is considered valid for a set of experimental conditions if the model's accuracy is within its acceptable range, which is the amount of accuracy required for the model's intended purpose" [Sar05]. We perform three sets of validations. First, we check that the simulation results in Section 6.5.1 are very similar to real system behavior, and that the results in Section 6.5.2 lead to a reasonable distinction between architectures belonging to three classes (i.e., independent clusters, centralized grid schedulers, and decentralized grid schedulers).

As a second validation test, we use for a selected scenario the functional validation of the simulator

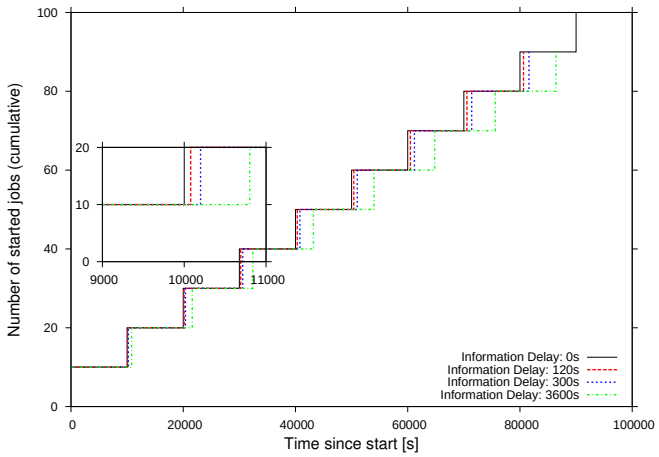


Figure 6.3: *The functional validation of the* DGSim *reference implementation using operational graphics.*

Table 6.1: *The comparison of the raw performance metrics of the simulated and of the real environment.*

Metric	Environment is Simulated	Real	Results Difference *Simulated − Real*
Job Completion Ratio	100%	90%	+10%
Avg. Wait Time	6s	56s	-50s
Avg. Execution Time	205s	196s	+9s
Avg. Response Time	211s	252s	-41s

through operational graphics [Sar05], that is, through displaying various performance measures over time to ensure that the internals of the simulation are correct. Figure 6.3 depicts the number of started jobs over time for a system with one cluster on top of which operates a centralized grid scheduler; both operate a FCFS queue. The cluster has 10 resources of identical speed of 1 work unit/second. A workload of 100 jobs of constant size 10000 work units arrives at time 0 at the grid level. The grid scheduler sends jobs to the cluster as soon as it notices cluster's idle resources. The grid scheduler's information service has an information delay δ that is constant throughout the simulation. We set δ to 0, 120, 300, and 3600 seconds. As expected, the curve depicting the number of started jobs is delayed equally to the value of δ.

As a third validation test, we compare the execution of a workload comprising co-allocated jobs [Cza99; Buc03; Moh05b] in the real DAS-3 environment to the execution of the same workload in a simulated DAS-3 environment. The considered DAS-3 environment has four clusters that are heterogeneous in both the number of processors and their performance; the environment comprises 386 processors. We use for our validation purposes a synthetic parallel application that is communication-intensive; the application is written in MPI and executes "all gather" operations at each step. By using the co-allocation capabilities of the DAS, this application can run in this DAS-3 environment on resources coming from one to four clusters; the exact execution configuration is in these experiments decided transparently to the user by the co-allocating grid scheduler Koala [Moh05b]. The validation experiments are done in two steps. First, we must obtain real data about the execution time of the application when executed on resources coming from various multi-cluster configurations. The data is then used as input for the simulator. This is the most time-consuming part of the validation experiment. For the second step we submit through the system gateway the same workload to both the simulated and the real environments, and compare the performance metrics for executing the

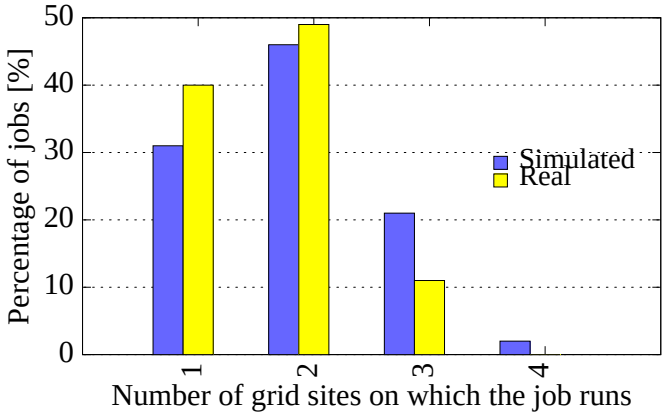


Figure 6.4: *The validation of the* DGSim *reference implementation. Comparison of a simulated and the real DAS environments for a workload of co-allocated jobs.*

workload. The workload consists of 300 different instances of the same application, submitted over a period of six hours. To address the variability of the real system, we perform five identical runs of the input workload, and average the results. To assess the performance of the system when the workload is stabilized, we extract from the six hours of the experiment the performance metrics for the middle period of five hours. Table 6.1 summarizes the results of this validation experiment. The average job wait time is much higher for the real system (56 vs. 6 seconds). The job wait time varies greatly during the experiment and is dependent on the system state, e.g., a more loaded gateway causes higher wait time. The average job execution time is higher in the simulator (by 5%). We attribute this difference to two factors. The first factor is the inability of the real system to complete all jobs during the interval considered for reporting, as a direct consequence of the higher wait time in the real system. The second factor is the slightly different scheduling decisions taken by the real and by the simulated scheduler. Figure 6.4 depicts for both the simulated and the real environments the percentage of jobs that are executed on resources (co-)allocated from one, two, three, and four DAS clusters. The simulator is the only one that is able to execute jobs on resources c-allocated from four clusters, and has a higher percentage of jobs that execute on resources co-allocated from three clusters. Due to the difference between the intra- and the inter-cluster network capabilities, the jobs executed on three and on four clusters take longer than those executed on one and on two clusters. The difference in the average job response time between the simulated and the real environments is the sum of the differences for the average job wait time and the average job execution time. We conclude that, apart from the sub-estimation of the overheads of the real environment, the simulated environment closely reflects the behavior of the real environment.

6.4.2 Performance Evaluation

We have identified in Section 6.2 that the experimental setup time is an important performance metric of a simulator (see Table 6.2 for the other performance metrics, i.e., M2-M5, of DGSim and of other simulators). The major component of the experimental setup time is the time spent in

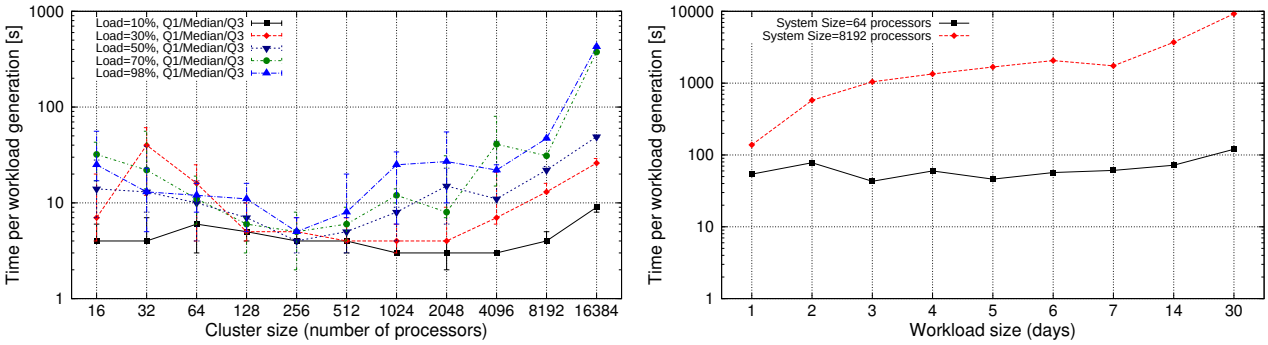


Figure 6.5: *The performance of the* DGSim *reference implementation. (*left*) Generation time for one-day workloads, for various system sizes and loads. (*right*) Total generation time for sets of workloads with load from 10% to 95%, for various workload durations and system sizes.*

generating workloads using the IWG algorithm (see Section 6.3.4). Figure 6.5 shows the generation time for various values of the parameters system size, load, and workload duration using IWG on the Lublin-Feitelson model [Lub03]. As expected, generating workloads for higher load (larger system size) takes on average more time for the same system size (load). However, Figure 6.5 (left) shows that the generation time for an arbitrary load can vary greatly for short workloads (e.g., one day). We attribute this to the low number of jobs in a short workload, which reduces the chance that a randomly generated workload incurs a certain load; when it does not, a new IWG iteration needs to start, taking more time. Figure 6.5 (right) shows that the generation time for the same load and system size increases linearly with the workload duration. The actual time values (obtained on a commodity desktop PC with 2.2GHz CPU and 1GB RAM) show that even for the largest clusters the setup time is below one hour. Taking into account also the ability to execute in parallel (through the grid interface) the conveniently parallel tasks of workload generation and of simulation, we conclude that the system has high performance for the experimental setup and for the simulations.

6.5 Experiments using DGSim

We have already used DGSim in a variety of experiments. In this section we present a summary of these experiments, with the purpose of demonstrating the usefulness of our framework. For each experiment we describe the problem and the setup; for more details we refer to [Ios07e; Ios07c].

6.5.1 Performance Evaluation Using Real Workload Traces

Summary Using DGSim, we have assessed the behavior of five grid resource management architectures under real load [Ios07c].

Setup For this experiment, we have simulated the inter-operation of the DAS-2 and Grid'5000 grids, so that together they constitute a system with 20 clusters and over 3000 resources. The input workload was a contiguous subset of 12 months from the real grid traces taken from DAS-2 and Grid'5000, starting from 01/11/2005 [Th08]. Figures 6.6 and 6.7 show the configuration files for the environment and for the simulator setup, respectively. Figure 6.8 depicts a comparative display of the number of jobs started by each of the five architectures over time. At the middle of days 26 and 27 the number of jobs submitted to one of the clusters surges, and `DMM` and `fcondor` start migrating

```
ID  Cluster       NProcs
c01 DAS/fs0       144
...
c06 G5K/site1/c1 128
c07 G5K/site1/c2 128
...
```

Figure 6.6: *Sample configuration file for the environment setup.*

```
Sim      PlugIn
cern     sim_cern.py
condor   sim_condor.py
DMM      sim_dmm.py
fcondor  sim_fcondor.py
koala    sim_koala.py
```

Figure 6.7: *Sample configuration file for the simulator setup.*

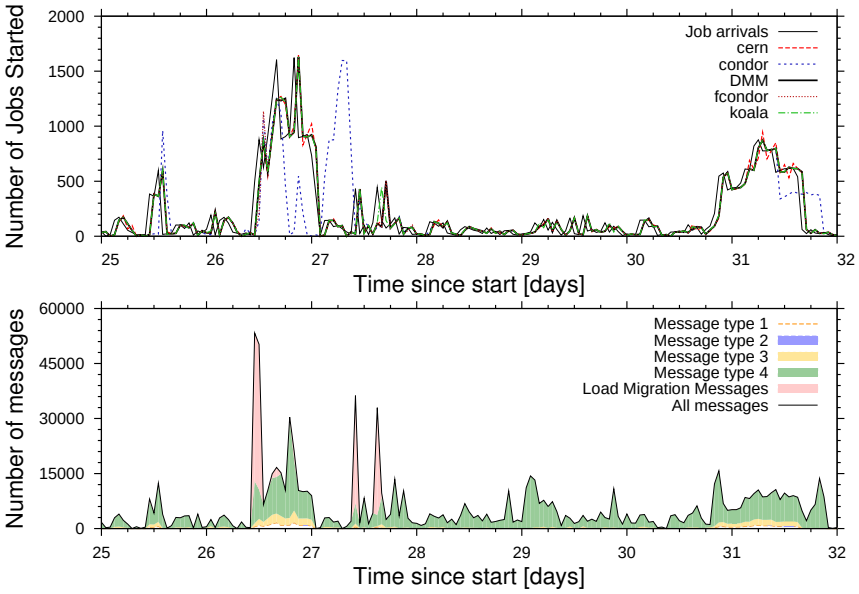


Figure 6.8: *System operation over time: (*top*) comparison of the number of jobs started by five grid resource management architectures. (*bottom*) the number of messages for* `DMM`*, per message type.*

```
experiment.ID = E2-1D
scenario.SimTypes = DMM,sep-c,cern,condor,fcondor,koala
scenario.Loads = 10,30,50,60,70,80,90,95,98
scenario.TracesBaseDir = K:\EuroPar08\traces\10x\
scenario.BaseOutputDir = K:\EuroPar08\results\
...
```

Figure 6.9: *The configuration file for the second experiment.*

load across clusters (the bottom row depicts the messages of `DMM`). This demonstrates how using DGSim's automated analysis features, including graphing, facilitates understanding the causes of the performance variations across various grid configurations.

6.5.2 Performance Evaluation Using Realistic Workload Traces

Summary Using DGSim, we have assessed the performance of six grid resource management architectures under realistic load from 10% to 98% [Ios07c].

Setup Similarly to the previous experiment, we have simulated the inter-operation of the DAS-2 and Grid'5000 grids. First, the workloads are generated using the LF model with the parameter values extracted automatically from the grid traces present in the GWA. Then, the experiment is run automatically; in particular, all the configuration files (540, that is, 6 architectures $\times$ 9 loads $\times$ 10 repetitions of each simulation for statistical soundness) are automatically generated. Finally, the results (over 20GB of data) are automatically analyzed to produce more than 20 common performance metrics. Throughout the process the tools use the same experiment configuration file, depicted in Figure 6.9. This configuration file specifies, among others, the experiment unique identifier (tag `experiment.ID`), the six simulated architectures (tag `scenario.SimTypes`), the nine loads under which the simulated system will operate (tag `scenario.Loads`), and the input and output base directories (tags `scenario.TracesBaseDir` and `scenario.BaseOutputDir`, respectively; the

Table 6.2: *A summary of simulators in grid computing research. The metrics M1-5 are evaluated for the largest reported set of experiments. Acronyms: T – from trace, M – from model, D – dynamic, G – generate until reaching the user-specified goal.*

		Experimental Setup			Experiment Support			Performance					
	Distinctive							M2	M3	M4		M5	
Simulator	Feature	a	b	c	a	b	c	scen.	sim.	proc.	sites	jobs	load
BeoSim	co-allocation	-	-	M	-	+	+	350	20k	100	5	4M	-
ChicSim	data replication	-	-	M	-	-	-	12	72	-	30	6k	-
CSim/Grid	co-allocation	-	-	M	-	+	-	100	4k	200	4	-	70%
GangSim	usage SLAs	-	-	T	-	+	+	10	100	300	20	1k	-
GridSim	economic SLAs	-	-	TM	-	-	-	100	100	10k	15	5k	-
GSSIM	automation	TMD	M	M	-	-	-	1	1	4	1	10	-
MONARC	CERN	-	-	M	-	-	+	100	100	2k	5	5k	-
OptorSim	data replication	-	-	-	-	-	+	50	0	-	20	10k	-
SimGrid	batch&DAG jobs	TMD	-	TM	-	+	-	100	10k	300	-	4k	-
DGSim	see Section 6.3	TMD	TM	TMG	+	+	+	100	6k	3k	20	1.4M	300%

last part of the `scenario.TracesBaseDir` tag (`10x`) shows that there are 10 sets of traces in the input directory, one for each experiment repetition).

6.6 Related Work

In this section we survey nine approaches to the simulation of grid computing systems: BeoSim [Jon04], ChicSim [Ran02], CSim/Grid [Buc03], GangSim [Dum05], GridSim [Buy02; Ran05; Sul07], GSSIM [Kur07], MONARC [Leg00], OptorSim [Cam04], and SimGrid [Cas00; Leg03; Car07]. We assess the relative merits of the surveyed approaches according to the requirements described in Section 6.2. Table 6.2 summarizes our survey; other simulators that we have reviewed [Tak99; Pha03; He05; Cas06; Ram07] may improve marginally on this body of knowledge. All the surveyed simulators have a discrete model representation, that is, the simulation model changes state only at discrete points in simulation time[3]. With the exception of GangSim and MONARC, which are discrete time-stepped simulators, all the surveyed simulators are discrete event simulators. To speed-up the simulation, DGSim can run on clusters and grids to execute multiple individual simulations in parallel, while MONARC and SimGrid use multi-threading to speed-up individual simulations. Compared to the previous simulation tools, DGSim focuses more on the simulation process, with richer grid system and workload generation, and support for large numbers of individual and grouped simulations. This is particularly visible in the size of the reported experiments: DGSim can use workloads orders of magnitude larger than the other simulators (with the exception of BeoSim), on similarly sized simulated environments; BeoSim can work with a workload of similar size, but on a system that is an order of magnitude smaller.

6.7 Concluding remarks

The evolution of today's grids into the technological solution of choice for sharing computing resources depends on the ability of its developers to improve significantly the current grid resource management systems. To accomplish this difficult task, a toolbox in which simulators play a critical role is needed. In this chapter we have introduced DGSim, a framework for simulating grid resource management architectures, which focuses on automating and optimizing the overall simulation process, from hypothesis to obtaining simulation results. From the methodological side, DGSim introduces or extends the concepts of grid inter-operation, grid resource dynamics, grid evolution, and grid workload model. We have presented in this chapter the design, the reference implementation, and two real cases where DGSim was used.

[3]For all the terms related to simulation we refer to the textbook of Fujimoto [Fuj99].

For the near future, we plan to continue the development of DGSim by extending it with libraries of algorithms and mechanisms for job scheduling, data management, and grid inter-operation. This extension will allow scientists to setup simulations in which these algorithms can be readily compared with alternatives.

Chapter 7

Alternatives for Grid Inter-Operation

7.1 Overview

In this chapter* we present the alternatives for grid inter-operation.

7.1.1 Motivation and Problem Statement

The decision to inter-operate grids leads to non-trivial design choices with respect to resource selection, gateway ownership (see Chapter 2), scalability, trust and accounting, and reliability. If there is no common resource management system, jobs must be specifically submitted to one of the grids, which may lead to poor load balancing. If a central meta-scheduler is installed, it can quickly become a bottleneck leading to unnecessarily low system utilization, it will be a single point of failure leading to break-downs of the combined system, and it is unclear who will physically manage the centralized scheduler. Other traditional solutions can also be impractical. Hierarchical mechanisms (still centralized, but arguably with less demand per hierarchy node) can be efficient and scalable, but still have single points of failure and are administratively impractical (i.e., who administers the root of the hierarchy?). Completely decentralized systems can be scalable and fault-tolerant, but they can be much less able for accounting than their hierarchical alternatives. While many solutions to building the Grid have already been proposed [And03; Cap05; Clu07; Moh05b; Bru99; Sai03; Sch99], none has so far managed to achieve acceptance in the grid world. Thus, a question must be answered: *What are the alternatives for grid inter-operation?* Answering this question gives system owners the tools to select the best alternative for inter-operating their systems.

Switching to a new system architecture is not easy for an already deployed system; regardless of the answer to the previous research question, the system owners would be reluctant to change if the current approach is suitable for their needs. Many of today's grids are based on centralized architectures, e.g., the DAS, Grid'5000. Other grids still force their users to select the resources from clusters that are not otherwise connected, e.g., the DEISA [Cle07]. In both cases, but especially for the centralized architectures, the gateways may become the system bottlenecks. An important question arises: *Under which practical circumstances do the centralized architectures become the system bottleneck?* Answering this question can help the owners of already deployed systems decide to change to a better approach, and gives new owners the chance to understand and follow the best practice.

*This chapter is based on previous work published in the ACM/IEEE Conference on High Performance Networking and Computing (SC'07) [Ios07c] and the IEEE/ACM Int'l. Conference on Grid Computing (Grid 2008) [Ios08g]. Part of this work was invited to and appeared in the special edition of the Journal of Scientific Programming, "Best Papers from SuperComputing 2007" [Ios08b].

7.1.2 Key Ideas and Selected Results

Two aspects are critical for the design of grid inter-operation approaches: the architecture and the operation mechanism. The architecture defines the logical structure of the (multi-)grid system. The operation mechanism defines the way the user jobs are managed (e.g., executed) by the architecture. We survey the existing approaches for (multi-)grid resource management with a focus on these two aspects.

The survey reveals that many real systems employ centralized architectures, or architectures where the clusters do not exchange load with each other. To understand the suitability of these approaches, we assess the overhead and the scalability of these approaches in practice. Key to these experiments using real systems is our belief (confirmed by the obtained results) that the interactions between the various layers of the grid middleware stack (from cluster resource manager to the user job submitter) are too complex to be ignored. Our results give evidence that the centralized approaches cannot deal with the load bursts that occur in grids.

We also do a qualitative analysis of the suitability of the other existing approaches for inter-operating grids, and show that none of the existing architectures possesses the necessary characteristics. Thus, we design a novel architecture that is a hybrid between the hierarchical and the decentralized approaches. This architecture leverages a *hierarchical* architecture in which nodes represent computing sites, and in which the nodes at the same hierarchical level and operating under the same authority (parent) are allowed to form a *completely decentralized* network. In this way, we attempt to combine the efficiency and the control of traditional hierarchical architectures with the scalability and the reliability of completely decentralized approaches. We show in Chapter 8 how this architecture can be efficiently operated.

7.1.3 Organization of this chapter

The remainder of this chapter is structured as follows. In Section 7.2 we review the grid inter-operation alternatives. In Section 7.3 we evaluate the practical limitations of the centralized grid inter-operation approaches. We then make a qualitative comparison of the alternatives for grid inter-operation in Section 7.4. Finally, in Section 7.5 we introduce a novel architecture for grid inter-operation.

7.2 A Review of Grid Inter-Operation Architectures

In this section we review approaches for grid inter-operation, from an architectural and from an operational point of view, and for each approach we give a concise description and one or more references to real systems that use it.

7.2.1 Architectural Spectrum

Below we present our taxonomy of architectures that can be used as (multi-)grid resource management systems (GRMS). We illustrate this taxonomy in Figure 7.1. We discuss the capability of each architecture to inter-operate grids in Section 7.4.

Independent clusters: each cluster has its local resource management system (LRMS), i.e., there is no meta-scheduler. Users have accounts on each of the clusters they want to submit jobs to. For each job, users are faced with the task of selecting the destination cluster, typically a cumbersome and error-prone process.

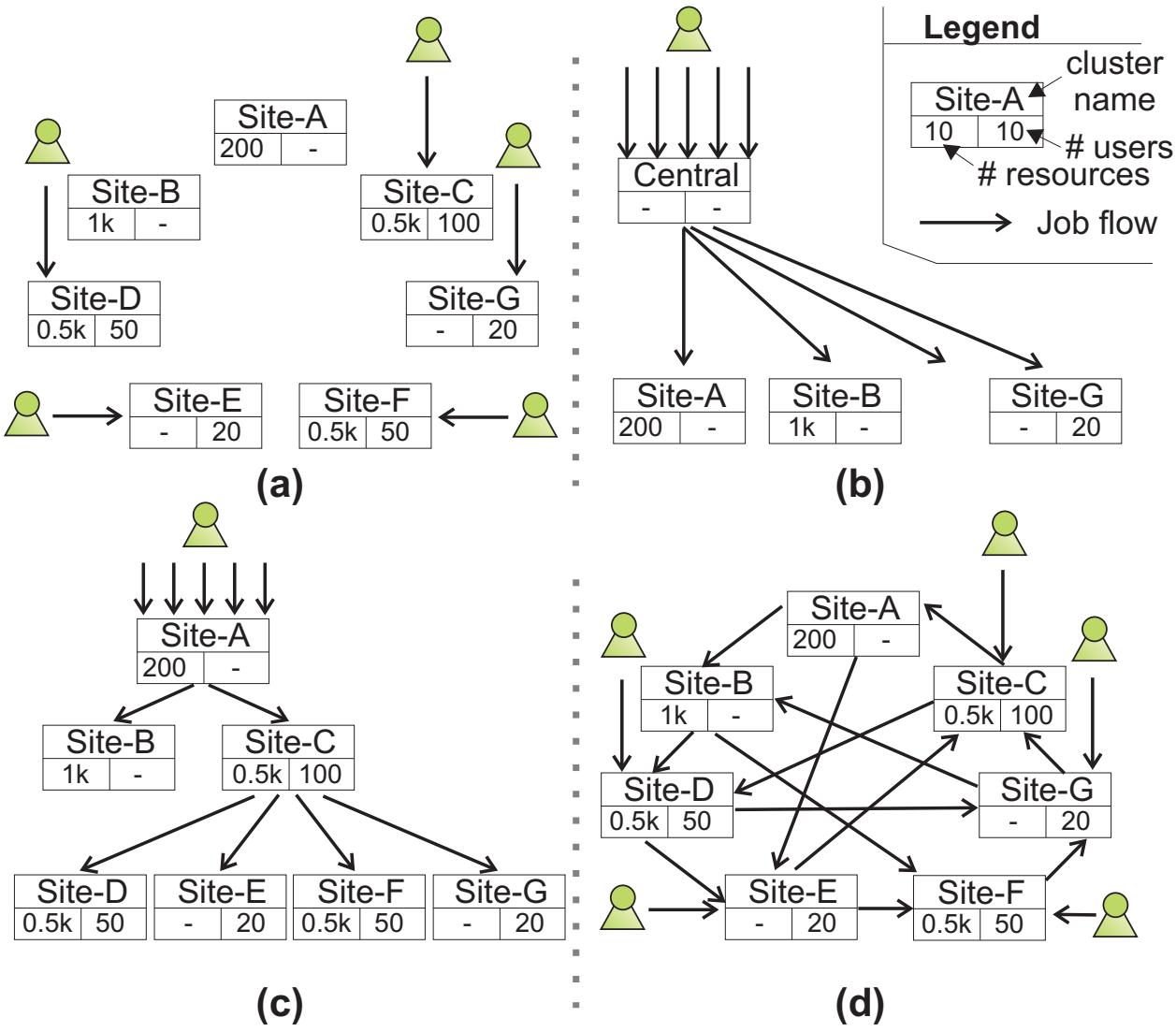


Figure 7.1: *The meta-scheduling architectures: (a) independent clusters; (b) centralized meta-scheduler, (c) hierarchical K-level meta-scheduler; (d) distributed meta-scheduler with static links. The grid participants Site-A and Site-B do not have any users, only resources. The grid participants Site-E and Site-G do not have any resources, only users.*

Centralized meta-scheduler: there exists one (central) system queue, where all grid jobs arrive. From the central queue, jobs are routed towards the clusters where they are dispatched. The clusters may optionally employ an LRMS, in which case jobs may also arrive locally. It may be possible for the manager of the central queue to migrate load from one LRMS to another.

Hierarchical K-Level meta-scheduler: there exists a hierarchy of schedulers. Typically, grid jobs arrive either at the root of the hierarchy, or at the clusters' LRMSs. In both cases, jobs are routed (migrated) towards the clusters' LRMSs. The Hierarchical 2-Level metascheduler is the most encountered variant [Bru99; Clu07] .

System	Architecture	Operation
Condor [Lit88; Tha05a]	Independent	Matchmaking
Globus GRAM [Cza98]	Independent	Job routing
Unicore [Erw02]	Independent	Job routing
Condor-G [Fre01]	Centralized	Matchmaking
Nimrod-G [Abr02]	Centralized	Job routing
AppLeS [Ber03b]	Centralized	Job routing
Alien [Sai03]	Centralized	Job pull
JoSH [Caw04]	Centralized	Job routing
Koala [Moh05b]	Centralized	Job routing
OAR(Grid) [Cap05]	Centralized	Job routing
CCS [Bru99]	Hierarchical 2-Level	Job routing
PUNCH [Kap99]	Hierarchical	Job routing
Moab/Torque [Clu07]	Hierarchical 2/3-Level	Job routing
OAR(Grid) v.2 [Cap07]	Hierarchical	Job routing
NorduGrid ARC [Ell07]	Indep./Federated	Job routing
NWIRE [Sch99]	Federated	Job routing
GridWay [Hue04; RM07]	Federated	Job routing
Condor flocking [Epe96; Tha05a]	Federated	Matchmaking
OurGrid [And03]	Distributed, dynamic	Job routing
Askalon [Sid06]	Distributed, dynamic	Job routing

Table 7.1: *Currently deployed resource management systems for (multi-)grids.*

Distributed meta-scheduler: similarly to the independent clusters architecture, each cluster has its LRMS, and jobs arrive at the individual clusters' LRMSs. In addition, cluster schedulers can share jobs between each other. This forms in effect a distributed meta-scheduler. We distinguish between two ways of establishing links between clusters (sharing): static (fixed by administrator, e.g., at system start-up), and dynamic (automatically selected). We also call a distributed meta-scheduler architecture with static link establishment a *federated clusters* architecture.

7.2.2 Operational Spectrum

We define an operational model as the mechanism that ensures that jobs entering the system arrive at the place where they can be run. We identify below three operational models employed by today's resource management systems.

Job routing: jobs are routed by the schedulers from the arrival point to the resources where they can run through a push operation (scheduler-initiated routing).

Job pulling: jobs are acquired by (unoccupied) resources from a higher-level scheduler through a pull operation (resource-initiated routing).

MatchMaking: jobs and resources are connected to each other by the resource manager, which thus acts as a broker responding to requests from both sides (job- and resource-initiated routing).

7.2.3 Real Systems

There exist many resource management systems for (multi-)grids. Below we present a selection, which we summarize in Table 7.1, including references.

Condor-based systems

Condor is a cluster management system. As such, it can straightforwardly be used in an independent clusters environment. The main focus of Condor is high throughput computing, that is, ensuring the highest average utilization of the system over a long period of time (the average system capacity). The Condor system is widely used in academic and industrial environments [Tha06] (see also Section 5.6.1).

The Condor system contains five main components: the Negotiator, the Collector, the Accountant, the user job manager, and the resource manager. The user job manager and the resource manager advertise they requests and their offer, respectively, using the classified advertisements (ClassAds) semi-structured data language [Ram98]. The Collector obtains and stores the resource offers and other state information. The Negotiator matches periodically the requests with the offers (matchmaking), and informs both parties in case of a match. The matching process uses a variant of weighted fair-scheduling, the parameters of which (e.g., weights, usage quotas) can be customized. The Accountant records the resource usage by communicating with all the other components; accounting issues are resolved in the favor of the user. The user job manager launches a process for each job that needs to be executed; this process effectively controls the execution of the job on the remote resource(s). The Condor system also offers numerous practical facilities: preserving the local environment characteristics while computing on remote machines, checkpointing and migration, application restart in case of failures, etc.

The Condor flocking mechanism extends Condor to allow the execution of jobs on resources coming from any of the clusters of a multi-cluster grid. Two independent flocking mechanisms have been developed for Condor: gateway flocking [Epe96] and direct flocking [Tha05a]. In gateway flocking each two grid clusters can be linked through the use of gateway nodes, which pass information about the free resources in both clusters. The gateway node presents itself to the local Negotiator as a resource manager (one resource at a time); when a job is matched to the advertised resource, the gateway node selects a resource from the free resources it represents to actually execute the job onto. In direct flocking the user job manager uses the services of multiple Negotiators, in round-robin fashion. The order in which Negotiators are contacted is static and manually configured. In practice, the configuration contains a list of all the Negotiators to which a user has organizational contact. Both flocking approaches effectively decentralize the Condor system, but have drawbacks in their ability to select resources. The gateway used in gateway flocking applies a resource selection policy independent of the local Negotiator; thus, it can make in good faith a poor resource selection, or can maliciously advertise resources with very good characteristics even when none are in fact available (and this cannot be proven). The direct flocking requires the user to select the Negotiator, which can result in poor performance due to the manual configuration; also, the different users that use direct flocking compete for the remote resources outside the control of the Negotiators. The two flocking forms can be applied at the same time in the same system. In the remainder of this chapter and in Chapter 8 we call direct flocking simply flocking. The Condor flocking system allows the inter-operation of federated clusters.

The Condor-G system implements a centralized architecture and operates through matchmaking on top of Globus grids; it also borrows the resource discovery, job submission, job management, and error recovery from Condor. Condor-G is one of the first systems to prevent the lengthy grid queueing times by obtaining resources from multiple clusters, installing resource managers on them, and then sending jobs directly to these resource managers, effectively by-passing the grid queues. Condor-G also extends the Condor fault-tolerance mechanisms for multi-cluster grids: it maintains a permanent job state database which can be used to track all the jobs that are managed at the individual grid clusters. Other important features include support for single sign-on security, and a basic mechanism

for executing jobs on remote machines where required files are not available and the local policy does not permit access to the local file systems.

Globus-based systems

The Globus Resource Allocation Manager (GRAM) is the resource management system of the Globus Metacomputing Toolkit, allowing the use of many cluster managers through a standard interface. Similarly to the Condor system, it can straightforwardly be used in an independent clusters environment. The Globus system is widely used in academic and industrial environments [The07] (see also Section 5.6.2).

The GRAM system contains two main components: the Gatekeeper and the Job Manager. The Gatekeeper manages the local resources and handles resource requests. In comparison with Condor, all resources are assumed to be dedicated to the grid (though tolerance to resource failures exists). The Gatekeeper launches a Job Manager for each job that needs to be executed; the jobs are routed to the resources where they execute. The user requests are expressed in Resource Specification Language (RSL), a structured language.

The Globus system is a layered architecture built around the concept of services; higher level services can be developed on top of one or more services [Cza98]. GRAM is the lowest level of Globus resource management architecture. An example of a higher-level service built on top of GRAM is the GateWay meta-scheduler [Hue04; RM07]. GridWay operates on top of grids that use Globus or other Distributed Resource Management Application API (DRMAA) enabled resource managers. Similarly to the Condor flocking (and with the same problem of resource access competition that is not controlled by the system), in GridWay the user job managers can obtain resources by directly contacting any cluster scheduler; as such, GridWay effectively implements a federated architecture with job routing.

The Koala co-allocating grid scheduler

Koala implements a centralized architecture and operates through job routing on top of grids that use Globus or DRMAA-enabled resource managers [Trö07], e.g., PBS/Torque, the Sun Grid Engine, GridWay, Condor, etc. Koala is one of the first real tools to implement co-allocation, that is, it can simultaneously allocate resources located in different clusters for the execution of the same job. Koala supports both best-effort and reservation-based co-allocation of resources.

Koala has two main components: the Co-Allocator and the Job Dispatcher. The Co-Allocator receives jobs from the users, and determines for each job the data transfers and the cluster where the job should be executed. Then, the Co-Allocator forwards the job to the Job Dispatcher, which operates on top of the cluster management systems. The Job Dispatcher obtains resources from the cluster management systems and starts the jobs on the remote resources; at the same time, it attempts to minimize the time during which the obtained resources wait for the data transfers (processor and data co-allocation).

Other real systems

The Nimrod-G, AppLeS, Alien, JoSH, and OARGrid all implement centralized architectures. The Nimrod-G architecture is the first to employ market-based scheduling in large-scale distributed computing systems: it considers the trade-off between cost and execution time in a system which charges for resource consumption. The AppLeS set of tools are some of the first to provide application-specific

scheduling in large-scale distributed computing systems: they optimize the execution of various bags-of-tasks-based applications. Alien is used in (a part of) CERN's production grid. JoSH operates on top of the Sun Grid Engine. OARGrid is used in research grids. OARGrid operates through job routing on top of the OAR cluster manager. Starting with version 2 (released late 2007-early 2008), OAR also supports resource aggregation, which can be used to build hierarchies of resources. OAR is one of the first real tools that can co-allocate jobs. OAR only supports reservation-based co-allocation.

PUNCH, CCS, and Moab/Torque are both hierarchical meta-schedulers. PUNCH and CCS are two of the first tools for creating large-scale distributed computing systems with a hierarchical architecture. CCS is also one of the first tools that can operate clusters and super-computers together; it was used mainly in research environments. The commercial package Moab/Torque is currently one of the most used resource management systems.

The NorduGrid ARC implements an independent clusters architecture operated through job routing. However, the job submission process contacts cluster information systems from a fixed list, and routes jobs to the site where they could be started the fastest. This effectively makes NorduGrid a federated clusters architecture.

NWIRE, OurGrid, and Askalon are all distributed clusters architectures operated through job routing. NWIRE and OurGrid implement a federated clusters architecture. NWIRE is the first such architecture to explore economic, negotiation-based interaction between clusters. OurGrid is the first to use a "tit-for-tat" job migration protocol, in which a destination site prioritizes migrated load by the number of jobs that it has migrated in the reverse direction in the past. Finally, Askalon is the first to build a negotiation-based distributed clusters architecture with dynamic link establishment.

7.3 Practical Limits for Centralized Architectures

A fundamental premise for resisting architecture change is that the existing solution is already suitable for the task. Many of today's grids are based on centralized architectures, e.g., the DAS, Grid'5000. In this section we assess the practical limits for the centralized architectures.

7.3.1 Approach

To assess the practical limits for the centralized architectures we stress-test several complete grid middleware stacks (from the cluster resource manager to the user job submitter) in a real environment, and observe the overhead and the scalability of these stacks. This approach relies on four key ideas: stress-testing to assess the overhead and the scalability of the system, testing complete grid middleware stacks, testing with various workloads, and testing in real environments. We now discuss these four ideas in turn.

In contrast to evaluating the raw performance through load-testing, we stress-test the systems to assess their overhead and scalability. This choice is motivated by the goal of the study: we do not want to find the best system for a specific task, but rather assess the ability of the tested systems to handle the many types of bursty job arrivals that occur often in grids [Ios06a; Li07f]. In other words, we want to understand the circumstances under which these systems become the system bottleneck. We have depicted the components of the job execution overhead in Figure 2.3; we further assess the scalability of the gateway under load by following its resource consumption.

In contrast to evaluating only the grid resource manager, our tests involve complete grid middleware stacks. The interactions between the middleware layers have grown too complex to be ignored, e.g., one layer acting alone can exhibit negligible overhead, but manifest as a serious bottleneck when

interacting with the other middleware stack layers. A complete grid middleware stack comprises the local cluster resource manager, the grid resource manager, and the user job manager. In Section 7.3.2 we discuss the middleware stacks used in these experiments.

The current grid workloads already comprise a wide variety of applications. Thus, we test with workloads comprising bag-of-tasks, chain-of-tasks, and other workflow jobs. While bag-of-tasks jobs are a dominant feature of today's grids, it can be expected that as the grid use becomes increasingly automated, jobs with a more complex structure, e.g., workflows, will become prevalent. To maximize the system stress, the tasks of these jobs are single-processor; for many grid middleware stacks, e.g., Condor-based or Globus-based, this maximizes the number of started job management processes, which in turn maximizes the resource consumption and limits the system scalability.

Our choice for testing in real environments, while time-consuming and more difficult to implement, is the only way to characterize system characteristics as overhead and scalability. Techniques such as mathematical analysis and simulation fall short in several respects, including the underestimation of the overheads [Ios06a]. The only alternative to testing in real environments is analyzing existing system logs or traces. However, these logs and traces are scarce, only cover a limited amount of scenarios, and can at most hint towards the conditions that pose problems. The authors' own experience with log and trace analysis [Ios06a; Ios07e; Ost08] is that the logged information is usually incomplete and sometimes even inconsistent with the information logged by the other layers of the middleware stack.

7.3.2 The Experimental Setup

In this section we present the experimental setup used for testing centralized grid middleware stacks.

The Test Environment

For the experiments we used and extended GrenchMark (see Chapter 5). We have extended GrenchMark with modules that generate multiple types of workflow-based workloads (including bag-of-tasks and chain-of-tasks workflows), and provide overhead and resource consumption statistics.

We used for testing a cluster of 16 common-of-the-shelf PCs located at the Politehnica University of Bucharest. Each PC has a dual-processor Pentium IV at 3.2GHz and 2GB RAM; the computers are interconnected by a 1 Gigabit Ethernet network. One of the nodes is used as the cluster gateway, and runs the cluster and the grid management systems, and the user job manager. The environment is monitored by the MonALISA monitoring system [New03b] which has enabled us to gather resource consumption data from the workstations.

The reason for selecting this test system size is twofold: it allows the tests to be performed in a reasonable amount of time (about two weeks for the complete experiments, with the duration proportional with the system size), and corresponds to the average cluster size for academic and small production clusters as reported by Kee et al. [Kee04] (note that in Section 4.3.2 we show that the average cluster size evolves very slowly with time).

The Tested Middleware Stacks

In these experiments we test four complete grid middleware stacks; Table 7.2 summarizes their structure. Each middleware stack includes a user job manager that can handle workflows (thus, they can also handle simpler types of jobs); we call this user user job manager a grid workflow engine (GWFE). We now discuss the four grid middleware stacks.

Table 7.2: *The characteristics of the four tested grid middleware stacks.*

Middleware Stack	Description
DAGMan	DAGMan + Condor v. 6.8.3
Karajan	Karajan + Globus GT4 + Condor v. 6.8.3
Gen-Condor	Generic GWFE + Condor v. 6.8.3
Gen-SGE	Generic GWFE + Sun Grid Engine v. 6.1

Table 7.3: *The job characteristics of the seven workflow-based workloads.*

Wl. Type	Description	Characteristics N	Characteristics L
S-1	sequence of tasks	30-50	30-50
S-2	parallel fork/join	30-50	5-15
S-3	bag-of-tasks	30-50	3
C-1	sameprob method [Alm92]	30-50	5-15
C-2	samepred method [Kas08]	30-50	5-15
C-3	layrprob method [Yan94]	30-50	5-10
C-4	layrpred method [Kas08]	30-50	5-10

The first middleware stack, DAGMan, comprises the DAGMan GWFE and the Condor resource manager. DAGMan is the default workflow engine for Condor.

The second middleware stack, Karajan, comprises the Karajan GWFE, the Globus GT4 grid middleware, and Condor as the cluster resource manager. Karajan is the default workflow engine for Globus.

The last two middleware stacks comprise our Generic GWFE operating on top of either Condor or the Sun Grid Engine (SGE) [Gen01] as cluster resource managers. The Generic GWFE is a simplistic implementation of a multi-threaded grid workflow engine. Each workflow is managed by a thread. Whenever a task in the workflow becomes "active", Generic attempts to submit it to a free resource using the services of the middleware layer below it, in this case, Condor or the SGE. Generic is not fault-tolerant, and does not have any of the advanced capabilities of a production GWFE. Thus, Generic can be seen as the baseline GWFE, and is equivalent with the job management tools written by the system administrator or by the experienced users, e.g., the `prun` tool in the DAS grid.

The Workloads

The system workload is a key part of any performance-related test [Jai91]; in particular, the simulation work of Ahmad and Kwok [Kwo99] and our recent investigation of the performance delivered by a GWFE in a development environment [Ost08] show that the properties of the workload's workflows have an important impact on the performance of the scheduling algorithms or engines.

We define seven testing workloads based on the following information sources: the only existing analysis of the traces of a real grid workflow engine [Ost08], the GWFE community's research, the characteristics and the models of generic grid workloads (see Section 4.4.2), and the workflow scheduling community's test workloads [Alm92; Yan94; Kwo99]. Table 7.3 summarizes the characteristics of the seven workloads used by our testing method; the first three (S-1, S-2, and S-3) are simple workload types, the last four (C-1 through C-4) are complex workload types. The first two workload types are instances of the chain-of-tasks (S-1) and of the master-worker (S-2) paradigms; S-3 is a hybrid in which the root node forks into several chains-of-tasks of short length that join at the end. The last four workload types have been defined by the workflow scheduling community [Kas08] and have been used in several simulation-based workflow scheduling studies [Alm92; Yan94; Kwo99].

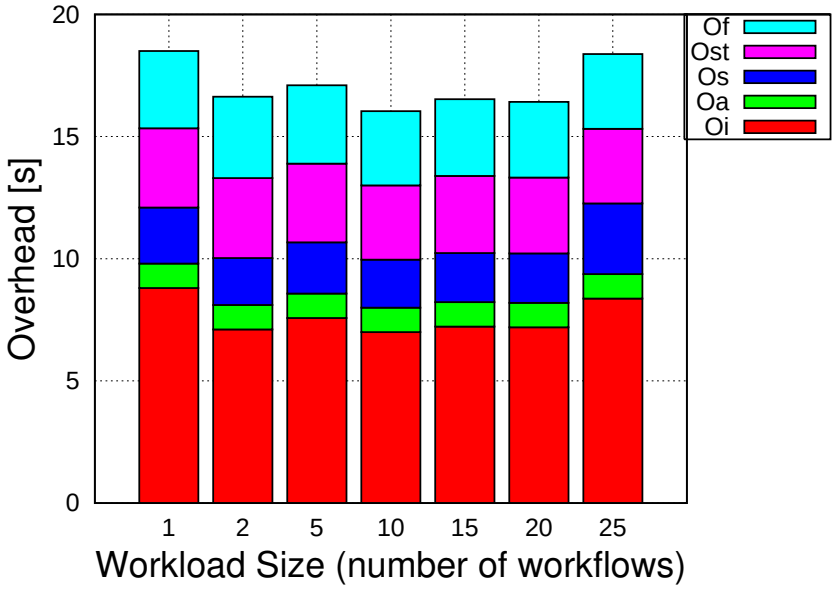


Figure 7.2: *The overhead breakdown for seven workload sizes (from 1 to 25 workflows).*

We now adapt in three steps these seven workload types to the task of testing GWFEs. First, we set for each workload type the characteristics of the workflows as shown in the columns N and L of Table 7.3. The values for the number of nodes (N) and for the graph traversal height L (see Chapter 2) correspond to the real values encountered in at least one real system [Ost08]. Second, to make these test workloads suitable for testing other settings, we use workloads comprising a number of workflows (the *workload size*) and an average workflow task duration (the *task size*) that are proportional with the size and the performance of the tested system. Note that by combining multiple workflows into the same workload we can compensate for the limited range of the values of N. The task sizes encountered in practice may range from seconds to below 5 minutes for workflow tasks [Ost08; Sin05], or from a few minutes to a few days for non-workflow grid jobs [Ios08d; Ios08e] or for workflows with at most 3-4 tasks. Third, we define an arrival scenario that is suitable for stress-testing and is also typical to grids: bursty arrivals in which many workflows are submitted to the system in just a few minutes [Li07e; Ios08d; Ios08e].

7.3.3 The Experimental Results

In this section we analyze the overhead and the scalability of the centralized grid middleware stacks.

The Overhead

To characterize the GWFE overhead we perform two sets of experiments. To better isolate the overhead, in each of these tests we execute jobs that take below 1 second to complete (i.e., "hello world" jobs); note that 1 second is the resolution of the grid middleware stack's accounting and logging.

First, we assess the impact of the middleware stack on the total overhead and on the overhead components (see Section 2.5) for one workload type, C-4. Figure 7.2 shows the total overhead and the overhead breakdown only for DAGMan; we have obtained similar results for all the other middleware stacks. The overhead is significant: over 15 seconds per task. From the overhead components, the most

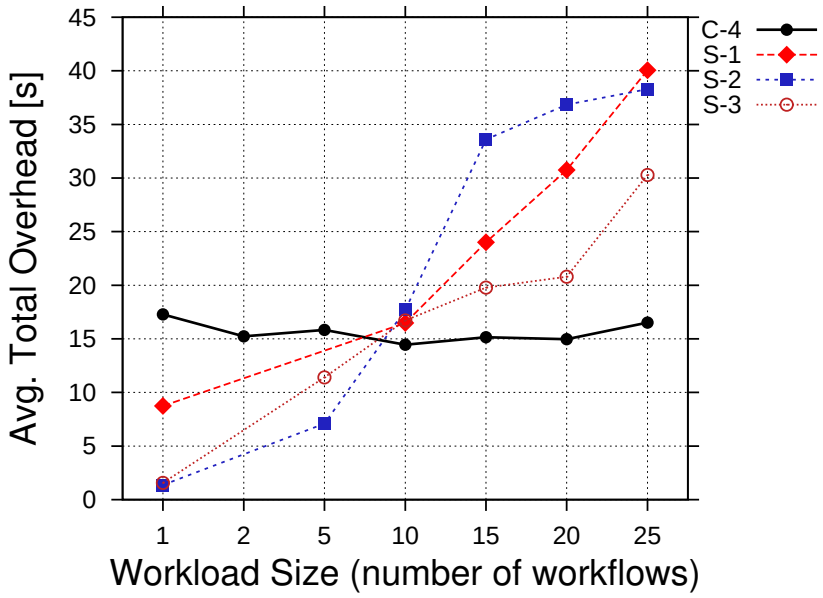


Figure 7.3: *The impact of the workload type on the total overhead.*

important is O_i, followed by O_f and O_{st}. The influence of each overhead component does not vary greatly for various workload sizes. Thus, better performance can be expected through improvements in the speed of updating the state of finished tasks and of their successors.

Second, we assess the impact of the workload type on the total overhead for DAGMan. Figure 7.3 shows the total overhead for four workload types; we have obtained similar results for the other three workload types. For C-4, the total overhead does not vary significantly with the increase of the workload size. We attribute this behavior to the characteristics of the C-4 workload, which does not stress the system. Unlike C-4, the workloads comprising simple workflows stress the system, and their total overhead increases with the workload size. For S-1 and for S-3, the total overhead threatens to render the system unresponsive for workload sizes of above 50 workflows.

The Gateway Scalability

For scalability, we note that the main bottleneck in a busy system is often the gateway [Ios06b; Cas07].

The gateway resource consumption for the centralized middleware stacks under workloads comprising 10 workflows each is depicted in Figure 7.4 for the workloads C-4 and S-2; we have obtained similar results for all the other workload types. Karajan imposes the highest usage of any resource for C-4, and of the CPU, memory, and of the number of sockets for S-2. It leads to the full utilization of the processor, to a number of 300-500 concurrently opened sockets (a common limit in today's operating systems is 1024), and to 50-60% memory usage. This indicates that Karajan would scale with difficulty to larger workload sizes. DAGMan has lower memory consumption and starts fewer processes than the other three middleware stacks; its CPU and socket are average. Thus, DAGMan would scale much easier than Karajan, in particular with regard to the memory usage (which it heavily optimizes). Finally, Gen-SGE consumes fewer gateway resources than Gen-Condor.

Figure 7.5 shows how the CPU consumption increases with the workload size for DAGMan. Over 25 workflows per workload (500-1000 tasks) the gateway of the centralized system is at over 90% load throughout most of the workload execution, and is effectively the system bottleneck.

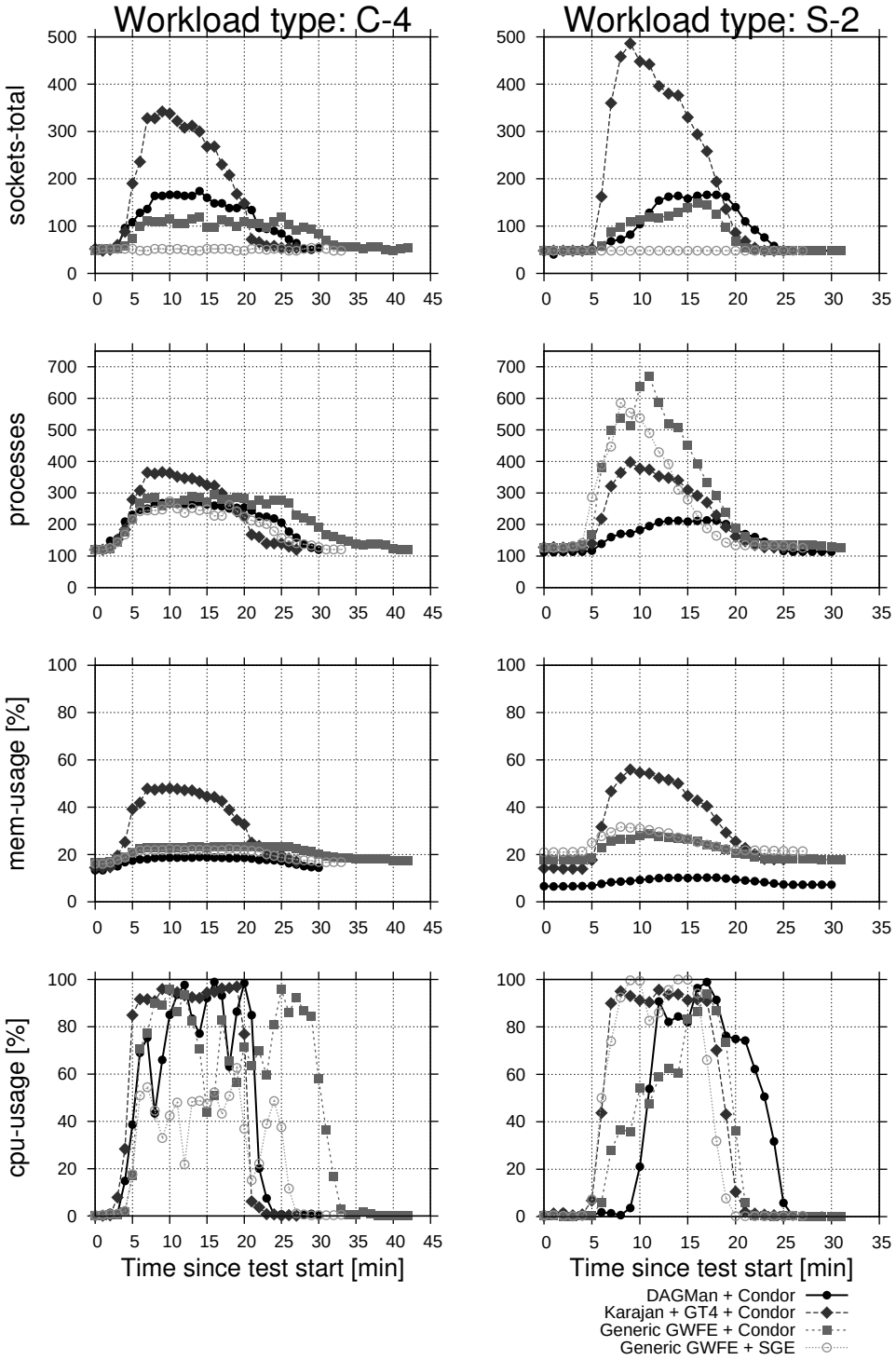


Figure 7.4: *The gateway resource consumption for the four tested middleware stacks: (*left column*) for the C-4 workload, and (*right column*) for the S-2 workload. Each row depicts the consumption of one type of resource.*

Summary of Experimental Results

The job overhead of the tested middleware stacks was above 10 seconds, and for many workloads typical in grids (e.g., bags-of-tasks) grew proportionally to the number of jobs. Similarly, the gateway

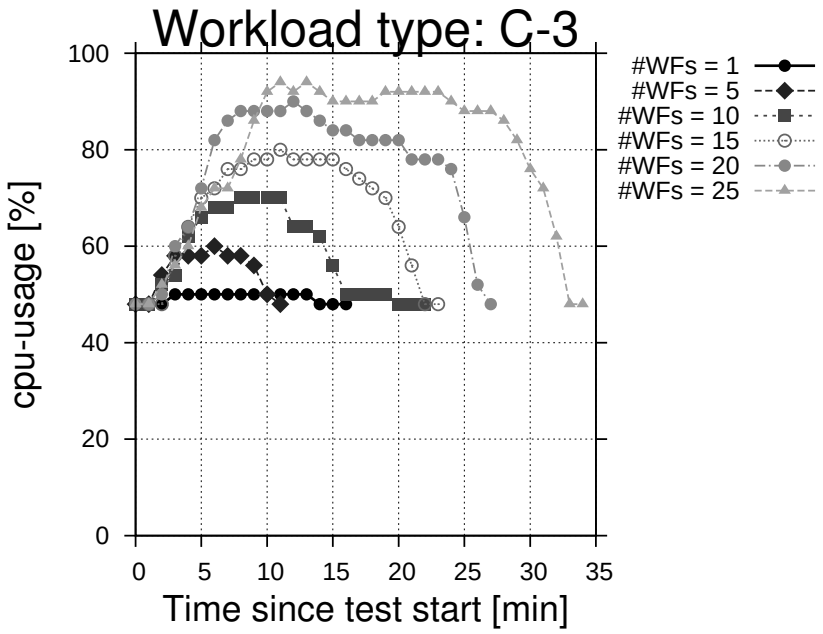


Figure 7.5: *The CPU consumption on the gateway as it depends on the workload size.*

resource consumption increases proportionally to the number of jobs. We attribute a part of this increase to the concurrent resource use by the jobs. Moreover, some of the tested grid middleware stacks were consuming significantly more gateway resources than others. With grids bursts including tens of thousands of jobs submitted at the same time to the same gateway (in Section 3.4.3 we show that over 20,000 jobs may arrive in the course of one day, and in our previous work [Ios06a] we show that hourly arrivals bursts can measure over 8,000 jobs), the central architecture is not a viable alternative for the most used grid middleware stacks.

7.4 Are the Current Grid Inter-Operation Architectures Sufficient?

In this section we assess the potential of the architectures presented in Section 7.2.1 to satisfy the requirements of grid inter-operation. We first formulate the requirements, then present a qualitative assessment of the architectural spectrum based on these requirements.

7.4.1 Requirements for Grid Inter-Operation

We now present five categories of requirements for grid inter-operation.

1. Resource Selection

There is little incentive in the inter-operation of grids if the overall achieved performance is lower than for the individual grids operating in isolation. We have shown in our previous work [Ios06a] that the resource selection is the point where the real performance differs the most from the performance predicted by simulation: jobs are routed to the wrong queues, and have much higher wait time than the optimum achieved by an omni-scient omni-potent scheduling system. The resource selection worsens significantly in face of high load.

The resource selection requirement for grid inter-operation is that the system must be able to select the appropriate resources under high load.

2. Gateway Ownership

One of the main issues in inter-operating grids is the control of the gateways. This involves on the one hand the maintenance and operation burden, and on the other hand gives complete executive power over the users of the gateway (including the subordinate gateways, if they exist). The burden is not trivial, and includes the costs and the effort for keeping the system running and for providing adequate administrative support. The annual operation costs of the gateway are from 100% of the system purchase price and upwards [Dav04]. The power conveyed by gateway ownership may lead to abuse, sometimes unwilling. The gateway enforces the local resource allocation policies and controls the logging of the system activity; thus, abuses can easily occur and are difficult to prove later-on. The unwilling and occasional abuse may result from the difficulty of maintaining the priority information for a large community of users.

An orthogonal problem to gateway control is that of multi-gateway composition. Many grids are created with the resources shared by groups that belong to one or more organizations. As a result, these grids must follow closely the real organizational structure between these groups. A common real organizational structure is the hierarchical relation between the components of the organization.

There are three gateway ownership requirements for grid inter-operation. First, there should be no root gateway, unless all the grid participants agree to operate under a central authority. Second, subordinate gateways must keep control of the resources they manage, e.g., they should have the ability to refuse requests from gateways to which they report. Third, the relations among gateways must follow closely the real organizational structure between their owners, and in particular must support the hierarchical structure (priority is given to the first requirement if conflicts with it occur).

3. Scalability

We define the scalability of the system as the ability of the system to operate within normal performance limits when faced with an increased number of users, jobs, and resources. The number of users increases over time; given the lifetime of many grid systems, we can consider that the users do not leave the system. We discuss about accommodating new users later in this section. Given that the total grid workloads are much higher in volume than anything that any single cluster can handle, and that grid inter-operation may lead to workload migration, a scalability problem occurs. We have shown in Section 7.3 that one gateway may not be sufficient for the volumes of jobs present in grids; while the gateway could be implemented in parallel, high maintenance costs and Amdahl's law limit the maximum achievable power for a gateway. We have shown in Section 4.3.2 that the number of clusters in a grid increases over time, at a rate of over 20% per year.

The scalability requirement is that the grid architecture must be scalable. For this, the architecture must support the addition of new users, the execution of workloads increasing in volume (number of jobs), and the addition and removal of resources of clusters (the addition and removal of resources from individual clusters is considered a feature of the cluster resource management systems).

4. Trust and Accounting

We define trust as the relation between participants that allows the sharing of information between the participants, and the decisions of one participant to be accepted and considered the correct decision

by the other participants. Trust is critical for the achievable performance of inter-operated grids: if information cannot be shared with any other participant, the job routing and scheduling processes are blind; if the routing and scheduling decisions of a participant are questioned, the load cannot be balanced. While trust and security are not equivalent, the former is the precursor of the latter: once a trust relationship exists, finding a security mechanism is straightforward (though optimizing it for the typical grid workloads may be difficult). The trust requirement for grid inter-operation is that a relation of trust must not exist between every participant to allow for every participant to use any other's resources (otherwise the user and the resource scalability are compromised).

Accounting is the operation of recording the resource use such that it can be later verified by anyone entitled, e.g., the resource owner, the resource user. The accounting requirement for grid inter-operation is that there should be three ways to verify a resource usage record if the user and the resource are not under the direct control of the same gateway: one based on the resource user records, one based on the resource owner records, and one by a third party.

5. Reliability

We define reliability as the ability of the system to survive the failure of its components. While the replication of the gateway services (e.g., the Condor High-Availability Daemons [Sil06]) is a solution for the failure of the gateway processing hardware, a network failure may prove fatal in certain architectures.

The reliability requirement for grid inter-operation is that the architecture must allow for the failure of the gateways with minimal performance disruption (the same ability related to resource failures is considered a feature of the cluster resource management systems).

7.4.2 A Qualitative Evaluation of the Grid Inter-Operation Architectures

We now evaluate the ability of the architectures introduced in Section 7.2.1 to satisfy the requirements of grid inter-operation.

Independent clusters

The independent clusters architecture has in general poor ability to handle the grid inter-operation requirements: it handles well the root ownership and the reliability requirements, but badly the other categories, and in particular the resource selection.

This architecture covers well the root ownership requirements: there is no root gateway and the individual gateways have full autonomy. However, this architecture cannot build hierarchical structures. The reliability of this architecture is good: the failure of one gateway affects only its local load. However, with grid workloads exhibiting often job arrival bursts, and with the system load being linked to its failure probability [Cas82; Lin90; Tan93], the affected load in case of a failure could easily dominate the overall load of the failed gateway.

Resource selection is the most important requirement that this architecture fails to meet: jobs are kept at their arrival place, which leads to poor load balancing and to performance problems in the face of high load. The scalability of this architecture is bad: each system gateway has the potential of becoming the system bottleneck; unlike the centralized architecture, this depends on the workload (that is, on the number of times high load occurs at a single gateway, and on the duration of the high load). The characteristics of the typical grid workloads make the independent clusters architecture unscalable. For similar reasons this architecture also fails to meet the scalability requirements.

The architecture performs very badly for trust and accounting. The users need to have accounts on each cluster they need to use, and there is no third party that can verify the use of remote resources.

Centralized meta-scheduler

The centralized meta-scheduler architecture offers a mix of very good and very bad ability to deal with the requirement categories: it handles very well resource selection and trust and accounting, but very badly the other three categories.

The ability of the centralized meta-scheduler to select the appropriate resources under high load in unparalleled under the same operation mechanism, as the central gateway has the highest amount of system information and of control. This architecture also handles the trust and accounting requirements optimally: only one gateway needs to be trusted by everybody else; an increase in the number of users and of gateways can be handled by the central gateway transparently for the other grid participants.

The root ownership requirements are broken by this architecture: there exists a root gateway, and it does not support a hierarchical structure. The scalability requirement is also broken: a central meta-scheduler can quickly become a bottleneck leading to unnecessarily low system utilization. Similarly, the reliability of the centralized architecture is compromised: the root gateway is a single point of failure leading to the break-down of the combined system. The reliability problem can be reduced with smart operation mechanism design (e.g., fall-back to an independent clusters architecture when the central gateway is down, building self-healing mechanisms such as the election a new central gateway).

Hierarchical K-level meta-scheduler

The hierarchical architecture does not excel but also does severely disappoint in addressing any of the grid inter-operation requirements.

Hierarchical architectures can offer good resource selection (though worse than the centralized systems), scalability (though the gateways become bottlenecks as they go higher in the hierarchy), and trust and accounting (though there exists a trade-off between accepting the decisions of the superior and keeping control of the local resources).

Similar to the centralized architecture, in this architecture the root ownership is an important problem. However, the importance of this selection diminishes with the increase of the number of hierarchical levels (K) and with the use of a smart operation mechanism (e.g., in Section 8.3 we propose keeping the load as low in the hierarchy as possible). This architecture also supports natively the hierarchical structure.

The reliability problems of the central architecture are shared to some degree by this architecture. Depending on the hierarchy depth and on the location of the failure, the impact of one failure ranges from similar to the independent clusters to similar to the centralized meta-scheduler. The impact of failures can be reduced with smart operation mechanism design (e.g., keeping the load low).

Distributed meta-scheduler

The ability to deal with the requirement categories of this architecture is almost a complement to that of the centralized architecture: this architecture handles very well resource ownership, scalability, and reliability, but very badly the trust and accounting issues. The resource selection of this architecture has not been previously evaluated under workloads that can appear in grids. We are the first to evaluate this aspect, in Chapter 8.

Table 7.4: *A qualitative comparison of the architectures for grid inter-operation. Each table cell marked with '+', '-', or '?' symbols shows the best possible capability of an architecture to address a certain feature, with the architecture given by the row and the feature given by the column of the cell. The capability ranges from '- -' (very bad) to '++' (very good); the '?' symbol specifies where the capability has not yet been assessed.*

Architecture	Resource Selection	Gateway Ownership	Scalability	Trust & Accounting	Reliability
Indep. clusters	- -	+	-	- -	+
Centralized	++	- -	- -	++	- -
Hierarchical	+	-	+	+	-
Distributed	?	++	++	- -	++

The root ownership is well handled: there is no root gateway and local resource control is maintained. However, the hierarchical structures may be built only through artifice in the operation mechanism.

The scalability of decentralized architectures is one of the reason for their expansion into the world of large-scale distributed systems, e.g., peer-to-peer networks [Mil03; Ris06]. However, there exists a trade-off between the number of links between the gateways (scalability), and the resource selection performance. Similarly, the reliability of this architecture is very good: the failure of one gateway does not disrupt the operation of the others. The impact of the failure on the local load can be even eliminated with smart mechanism design (e.g., job replication).

The trust and accounting are the obvious weak spots of this architecture: each participant needs to trust each other participant they are directly connected to (with a trade-off between performance and the trust burden), and the accounting requirement is not fulfilled.

7.4.3 Summary

We summarize our findings into a qualitative comparison between architectures; Table 7.4 depicts the results of this qualitative comparison. None of the architectures can satisfy all the requirements of grid inter-operation. The centralized architecture is the worst architectural choice, and in particular delivers poor performance due to its resource selection capabilities. The centralized, the hierarchical, and the distributed architectures each have very good or good abilities in two or three of the five requirement classes, respectively, but are not well equipped for dealing with the remaining requirements.

7.5 A Hybrid Hierarchical-Distributed Architecture

We have concluded in the previous section that none of the existing architectures can fulfill the requirements of grid inter-operation. We propose in this section a new architecture that aims to address this problem.

7.5.1 Overview and Generative Process

We propose a hybrid hierarchical-distributed meta-scheduler architecture, the only hybridization possible between the architectures introduced in Section 7.2.1. In this architecture, each grid is managed by a hierarchical K-Level meta-scheduler. In addition, the root meta-schedulers can share the load between each other. Other load-sharing links can also be established to improve the performance, the scalability, and the reliability of the system.

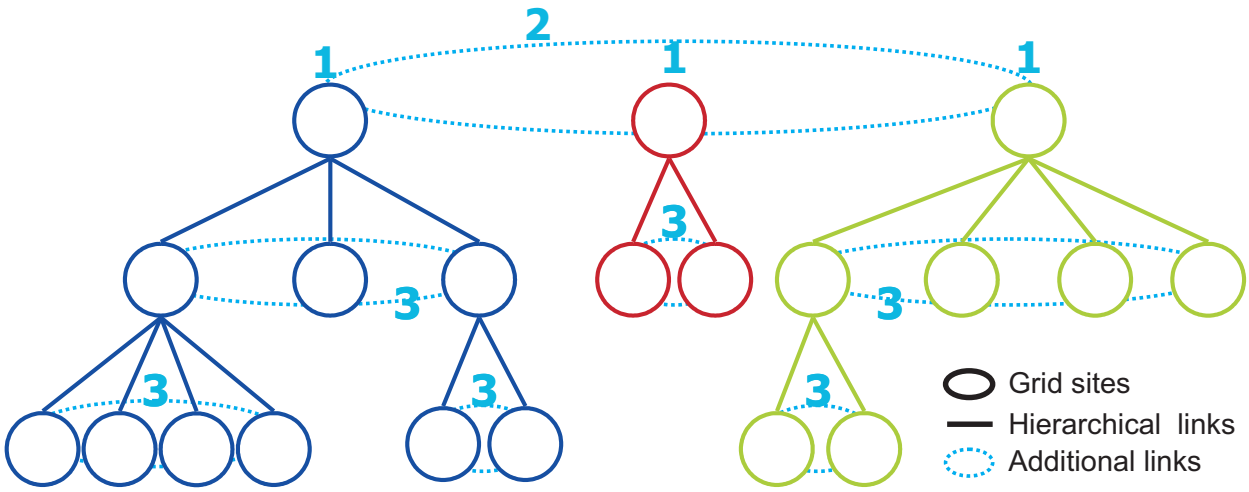


Figure 7.6: *The hybrid hierarchical-distributed architecture is built in three steps: (1) Start from multiple grids, each with a hierarchical architecture; (2) Build additional links to let the grid roots exchange load; (3) Build additional links to let siblings exchange load.*

We now present a generative process that describes how the grid clusters and other administrative units, from hereon *sites*, are connected. We aim to create a *network of sites* that manage the available resources, on top of and independently of the local cluster resource managers. The generative process is depicted in Figure 7.6.

First, sites are added according to administrative and political agreements, and *parent-child* (hierarchical) *links* are established. Thus, a hierarchy of sites is formed, in which the individual grid clusters are leaves of the hierarchical tree. Each site administers directly its local resources and the workloads of its local users. However, a site may have no local resources, or no local users, or both. A site with no local resources can be installed for a research laboratory with no local computing resources. A site without local users can be installed for an on-demand computing center. A site without users or resources serves only administrative purposes.

Second, the roots of the hierarchical grid structures are linked in a decentralized network. This allows the load sharing between grids; depending on the operational mechanism, it also improves the performance, the scalability, and the reliability of the inter-operated grid.

Third, *sibling links* can be formed between sites at the same hierarchical level and operating under the same authority (parent site), supplementary to the hierarchical links. This allows the creation of operation mechanisms that off-load the parent gateways by keeping the load as low as possible in the hierarchy.

7.5.2 Is This Architecture Sufficient?

The root ownership requirements are very well covered by the hybrid architecture. There is no root gateway, with the limitations of the hierarchical architecture being eliminated by the decentralized architecture used to inter-connect the roots of each grid. The resource control is supported natively due to the distributed structure of each of the hierarchical levels; however, this architectures also supports the case where the resource control must be subordinate to the parent gateway for performance or organizational reasons. The hierarchical structure is supported natively due to the hierarchical

structure of the individual grids.

The scalability requirements are also very well covered: the architecture has at least the scalability of the hierarchical architecture, and the scalability of the gateways at the same hierarchical level is improved through the creation of the decentralized network. A similar line of reasoning applies for the reliability requirements.

The hybrid architecture handles well the trust and accounting requirements. Starting from the capabilities of the hierarchical architectures, the trust is not compromised due to the hybridization with a decentralized architecture, as the links are only formed between gateways at the same level and operating under the same parent. For accounting, the hierarchical structure ensures the three-way verification (see point 4 in Section 7.4.1) for any transaction that passes through a parent gateway. For the others, in case of problems there exists a parent that can mediate or be used for future transactions for all levels except for the root level, which is assumed to be trustworthy (otherwise there would be no grid inter-operation).

We conclude that this hybrid architecture has very good potential to fulfill the root ownership, the scalability, the trust and accounting, and the reliability requirements. We evaluate its resource selection capabilities in Chapter 8.

7.6 Concluding remarks

The decision to inter-operate grids leads to non-trivial design choices with respect to resource selection, gateway ownership, scalability, trust and accounting, and reliability. This decision can be further put on hold for an already deployed system, when the switch to a new system architecture would be required.

To address this complex problem, we first survey the existing approaches for (multi-)grid resource management with a focus on two aspects: the architecture and the operation mechanism. We identify four main architectural and three main operation mechanism classes, and review twenty real systems with respect to these two aspects. Then, we assess the practical limitations of the most used (multi-)grid architecture: the centralized architecture. We find that the overhead and the scalability issues exhibited by four such solutions when tested in a real environment make the central architecture less attractive in practice. The results of the survey and of the real experiments enable a qualitative comparison of the four architectural classes. We find that none of the current architectures is able to address all the requirements of grid inter-operation. Finally, to address these requirements we propose a new architecture for grid inter-operation, a hybrid between the hierarchical and the decentralized architectures.

Chapter 8

Inter-Operating Grids through Delegated MatchMaking

8.1 Overview

In this chapter* we present our Delegated MatchMaking approach for grid inter-operation.

8.1.1 Motivation and Problem Statement

We have argued throughout this dissertation that the grid vision of a single computing utility has yet to materialize. In the previous chapter we have analyzed several architectural alternatives. In this chapter we seek the answer to the question *What is an efficient way to inter-operate grids?* in light of the other issues raised in Section 7.4: gateway ownership, scalability, trust and accounting, and reliability.

8.1.2 Key Ideas and Selected Results

In the previous chapter we have presented several alternatives for grid inter-operation, including a novel hybrid hierarchical/decentralized architecture. We build in this chapter on this architecture. Our key idea is to operate this architecture through the *Delegated MatchMaking* mechanism, that is, through delegating requests for resources up-and-down the hierarchy, and within the completely decentralized networks. When resource request matches are found, the matched resources are delegated from the resource owner to the resource requester. By delegating resources to jobs instead of the traditional migration of jobs to resources, we lower the administrative overhead of managing user/group accounts on each site where they can use resources and we enable the execution of more application types than those supported at the individual grid sites. Our architecture can be used as an addition to existing (local) resource managers. We call the hybrid architecture operated through Delegated MatchMaking mechanism the *Delegated MatchMaking approach*.

We assess in this chapter the performance of our architecture, and compare it against five architectural alternatives (from the architectures presented in Chapter 7): two independent clusters architectures, two centralized architectures, and one decentralized architecture. Our experiments use a simulated system with 20 clusters and over 3000 resources. The workloads used throughout the experiments are either real long-term grid traces, or synthetic traces that reflect the properties of grid workloads. Our study shows that:

*This chapter is based on previous work published in the ACM/IEEE Conference on High Performance Networking and Computing (SC'07) [Ios07c]. This work was later invited to and appeared in the special edition of the Journal of Scientific Programming, "Best Papers from SuperComputing 2007" [Ios08b].

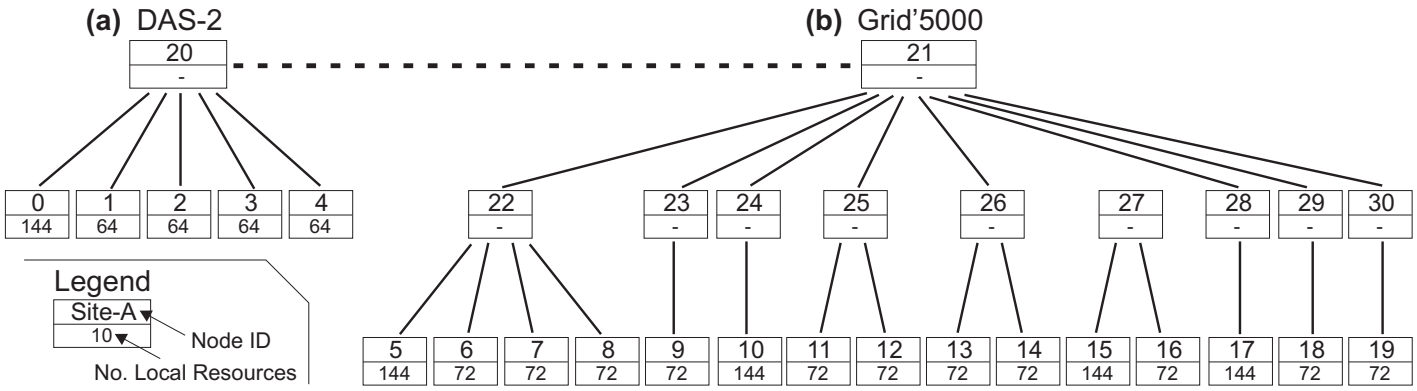


Figure 8.1: *The logical structure of the dual-grid system composed of (a) the DAS, and (b) Grid'5000. Leaves in this structure represent actual clusters of resources. Nodes labelled 20 through 30 are administrative-only.*

1. The Delegated MatchMaking approach achieves a good load balance for any system load, and in particular for high system loads.

2. The Delegated MatchMaking approach achieves a significant increase in goodput [Bas99] and a reduction of the average job wait time, when compared to centralized and decentralized approaches. Furthermore, when facing severe imbalance between the loads of the system's composing grids, our architecture achieves a much better performance than its alternatives, while keeping most of the load in the grids from which it originates.

3. The overhead of the Delegated MatchMaking approach, expressed in number of messages, remains low, even for high system loads.

8.1.3 Organization of this chapter

The remainder of this chapter is structured as follows. In Section 8.2 we present a motivating scenario: the inter-operation of the DAS and Grid5000 grids. We introduce in Section 8.3 our Delegated MatchMaking approach for grid inter-operation. After discussing the experimental setup in Section 8.4, we evaluate this approach in Section 8.5. Finally, we discuss in Section 8.6 the impact on our findings of two practical aspects not dealt with in the experiments.

8.2 The Motivating Scenario: Inter-Operating the DAS and Grid'5000

We consider as a motivating scenario the inter-operation of two grid environments, the DAS [Bal00] and Grid'5000 [Bol06].

8.2.1 The Dual-Grid System: Structure and Goals

The DAS-2 environment (see Figure 8.1a) is a wide-area distributed system consisting of 400 processors located at five Dutch Universities (the cluster sizes range from 64 to 144). The users, a scientific community sized around 300, are associated with one of the five clusters (the home cluster), but a grid infrastructure grants DAS-2 users access to any of the clusters. Each cluster is managed by an independent local cluster manager. The cluster owners may decide to partially or to completely take

away the cluster resources for limited periods of time. The DAS-2 workload comprises a large variety of applications, from single-CPU jobs to parallel jobs that may span across clusters. Jobs can arrive directly at the local clusters managers, or to the KOALA grid meta-scheduler [Moh05b].

The Grid'5000 environment (see Figure 8.1b) is an experimental grid platform consisting of 9 sites, geographically distributed in France. Each site comprises one or several clusters, for a total of 15 clusters and over 2750 processors inside Grid'5000. The users, a community of over 600 scientists, are associated with a site, and have access to any of the Grid'5000 resources through a grid infrastructure. Each individual site is managed by an independent local cluster manager, the OAR [Cap05], which has advance reservation capabilities. The other system characteristics, e.g., the cluster ownership and the workload, are similar to those of the DAS-2.

The hypothetical combined environment that is formed by inter-operating the DAS-2 and Grid'5000 comprises 20 clusters, and over 3000 processors. The goal of this combined environment is to increase the performance–*reduce the job slowdown, even in a highly utilized system.* The performance should be higher than that of the individual systems, taken separately. However, in achieving this goal we have to ensure that:

1. The load is kept local as much as possible, that is, jobs submitted in one grid should not burden the other if this can be avoided (the "keep the load local" policy).

2. The inter-connection should not require that each user, or even that each group, should have an account on each cluster they wish to use.

3. The clusters should continue running their existing resource management systems.

8.2.2 Load Imbalance in Grids

A fundamental premise of our delegated matchmaking architecture is that there exists a load imbalance between different parts of the dual-grid system. We show in this section that this imbalance actually exists.

We want to assess the imbalance between the loads of individual clusters. To this end, we analyze two long-term and complete traces of the DAS-2 and of the Grid'5000 systems, taken from the Grid Workloads Archive (GWA) [Th08; Ios08d]: traces `GWA-T-1` and `GWA-T-2`, respectively. The traces, with respectively over 1,000,000 and over 750,000 jobs, contain for each job information about the cluster of arrival, the arrival time, the duration, the number of processors, etc.

We define the *normalized daily load* of a cluster as the number of job arrivals over a day divided by the number of processors in the cluster during that period. We define the *hourly load* of a cluster as the number of job arrivals during hourly intervals. We distinguish two types of imbalance between the cluster loads, overall and temporary. We define the *overall imbalance* between two clusters over a period of time as the ratio between their normalized daily loads cumulated until the end of the period. We define the *temporary imbalance* between two clusters over a period of time as the maximum value of the ratio between the hourly loads of the two clusters, computed for each hour in the time period. The overall imbalance characterizes the load imbalance over a large period of time, while accounting for the differences in cluster size. The temporary imbalance characterizes the load imbalance over relatively short periods of time, regardless of the cluster sizes. The imbalance metrics proposed here characterize well the imbalance of a multi-cluster system when the collection of random variables describing the sizes of the arriving jobs for each cluster are independent and identically distributed.

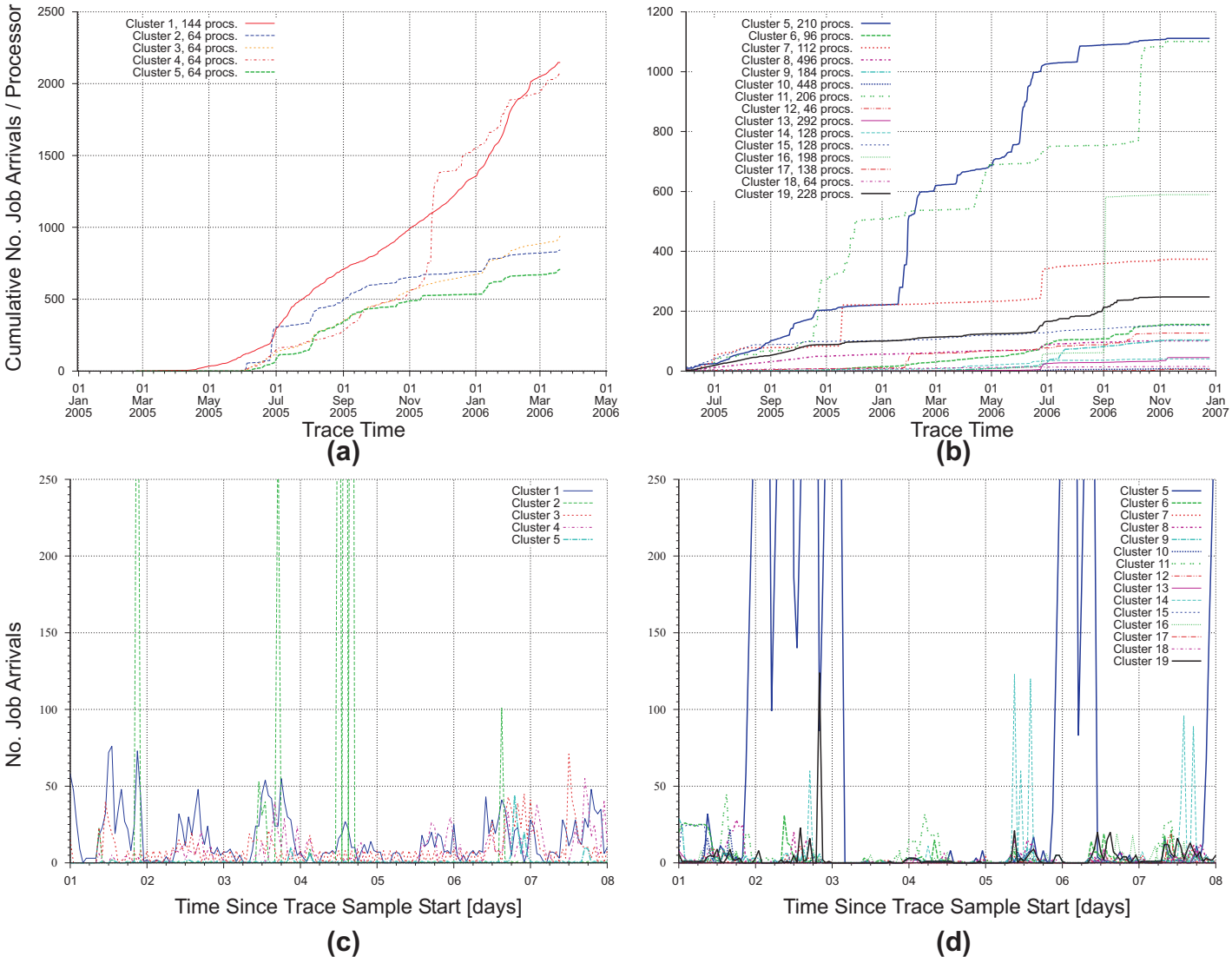


Figure 8.2: *Load imbalance between clusters of the same grid. (a) and (b) the cumulative normalized daily load of the clusters in the DAS-2 and Grid'5000 systems over time.; (c) and (d) the hourly load of the clusters in the DAS-2 and Grid'5000 systems over time. A higher imbalance is indicated in each of the four sub-graphs by more vertical space between the top-most and the bottom-most curves.*

We evaluate the imbalance for the DAS-2 and the Grid5000 systems independently. We use the original workload traces recorded in the Grid Workloads Archive (traces `GWA-T-1` and `GWA-T-2`); we replace each parallel job of size n present in the original trace with n jobs of size 1 with all the other properties (such as the job runtime) copied from the parallel job. Figure 8.2(a) shows the cumulative normalized daily load of the DAS-2 system, over a year, from 2005-03-20 to 2006-03-21. The right-

most value indicates the average number of jobs served by a single processor during this period. The maximum overall load imbalance between the clusters of the DAS-2 system is above 3:1. Figure 8.2(c) shows the hourly load of the DAS-2 system, over a week, starting from 2005-06-01. During the interval 2pm-3pm of 2005-06-04, there are over a thousand jobs arriving at cluster 2 and only one at cluster 5. The maximum temporary load imbalance between the clusters of the DAS-2 system is over 1000:1. We have obtained similar results for the Grid'5000 traces, as shown by Figures 8.2(b) and 8.2(d). We conclude that there exists a great potential to reduce the delays through load balancing across DAS-2 and Grid'5000.

8.3 The Delegated MatchMaking Approach

In this section we present our Delegated MatchMaking approach for grid inter-operation.

8.3.1 Architecture and Local Operation

We employ the hybrid hierarchical-distributed architecture introduced in Section 7.5. For the motivating scenario, we create the hierarchical links between the sites as in Figure 8.1. Additionally, the site pairs 0-4, 5-8, 11-12, 13-14, 15-16, 22-30, and 20-21 are also inter-connected with sibling links, respectively. Sites 20 through 30 have been installed for administrative purposes. To avoid ownership and maintenance problems, there is in fact no root of the hierarchical tree. Instead, sites 20 and 21 serve as roots for each of the two grids, and are connected through a sibling link.

We assume that each of the grid's clusters uses a Condor-like resource management system. This assumption allows us to consider in our architecture only the mechanisms by which the clusters are inter-operated, while benefiting from the local resource management features of Condor [Tha05a]: complete administrative control over owned resources (resources can reject jobs), high tolerance to resource failures, the ability to dynamically add/remove computing resources (through matchmaking and glide-in). This also ensures that the administrators of the grid clusters will understand easily our architecture as it uses concepts from the Condor world, such as matchmaking. However, the assumption of Condor-like resource management systems can be relaxed to allow any cluster resource management system that supports the dynamic addition and removal of resources.

Similarly to Condor, in our architecture each cluster is managed by a *site manager* (SM), which is responsible for gathering information about the local resources and jobs, informing resources about their match with a job and vice-versa, and maintaining the resource leasing information. According to this definition, our SM is equivalent to Condor's Negotiator, Collector, and Accountant components combined. Each resource is managed by a *resource manager* (RM), which will mainly be occupied with starting and running user jobs. Each user has (at least) one permanent *job manager* (JM), which acts as an application-centric scheduler [Ber96] that obtains resources from the local SM. Our RM and JM definitions correspond to those of Condor's Start and Sched daemons, respectively. In addition to the Condor-specific functions, in our architecture a site manager is also responsible for communicating with other site managers. We discuss in Section 8.6.1 several ways to implement the SM extensions required by the DMM approach.

8.3.2 The Delegated MatchMaking Mechanism

We now present through an example the operational mechanism of the Delegated MatchMaking approach, the *Delegated MatchMaking mechanism* for obtaining remote resources. In the example, a

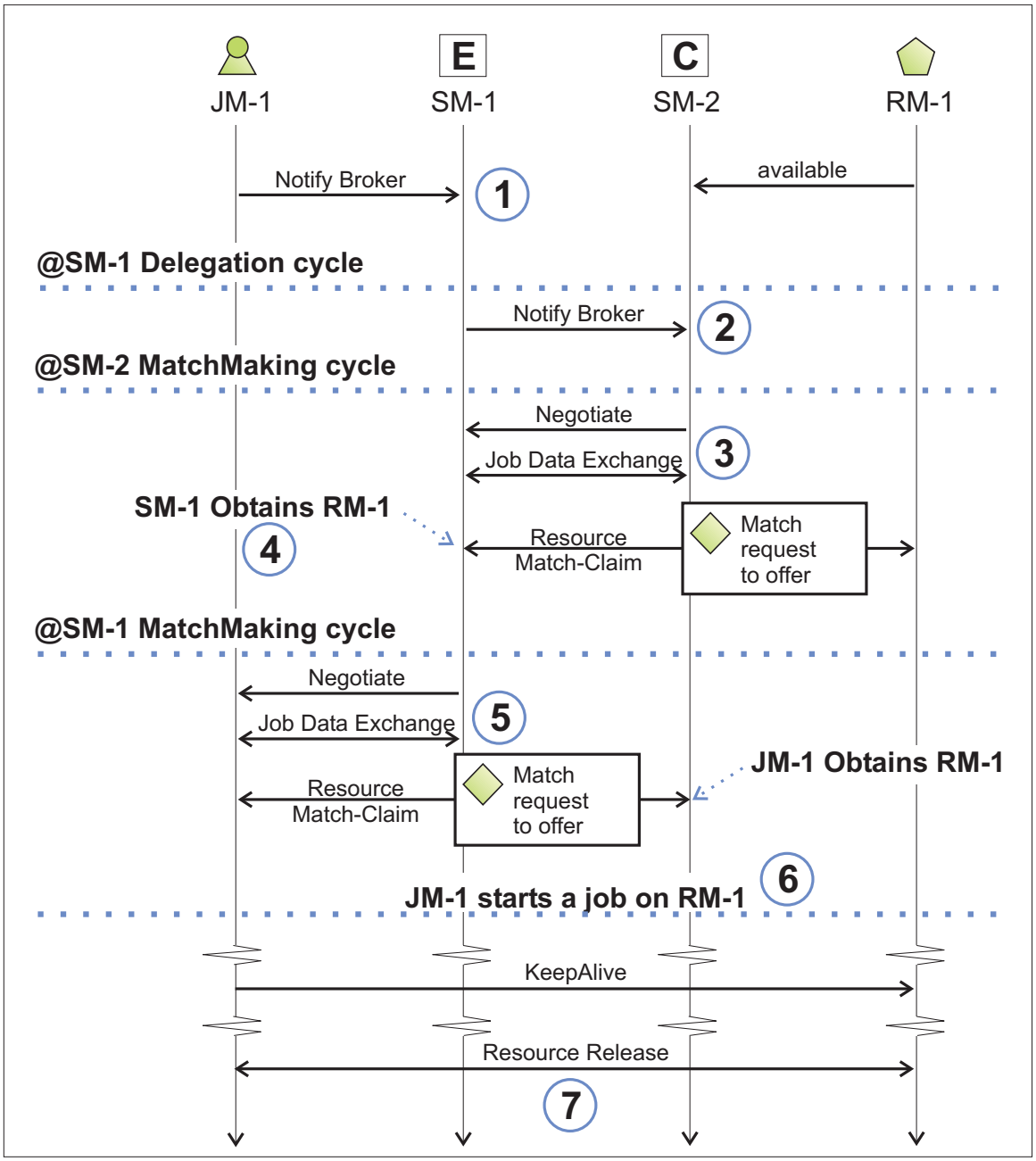


Figure 8.3: *The delegated matchmaking mechanism, during a successful match.*

system with two clusters (site managers SM-1 and SM-2) serves the requests of one user (job manager JM-1). Figure 8.3 depicts the Delegated MatchMaking mechanism during a successful match. The job manager JM-1 informs its site manager SM-1 about the needed resources, by sending a *resource request* (Step 1 in Figure 8.3). The resource request includes the type and the number of resources required by JM-1. At its next delegation cycle, the site manager SM-1 establishes that it cannot serve locally this request, and decides to delegate it. SM-1 selects then contacts SM-2 for this delegation

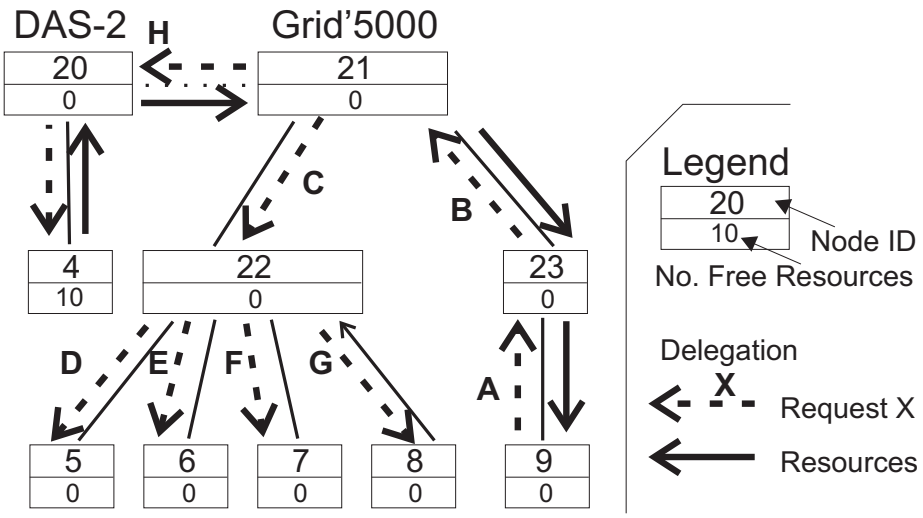


Figure 8.4: *A worst-case performance scenario for delegated matchmaking.*

(Step 2). To make its selection, SM-1 uses its target site ordering policy (see Section 8.3.3). During its next matchmaking cycle, SM-2 finds enough free resources, and delegates them to SM-1 through its local resource management protocol (Step 3). Then, SM-1 claims the resources, and adds them to its local environment (Step 4). At this point, a *delegation chain* has been created, with SM-2 being the *delegation source* and SM-1 the *delegation sink*. During its next matchmaking cycle, SM-1 handles JM-1's request using its own resource management protocol (Step 5). Upon receiving the resources, JM-1 starts the user's job(s) on RM-1 (Step 6). Finally, after the user job(s) have finished, JM-1 releases the resource, by informing both RM-1 and SM-1 (Step 7). The resource release information is transmitted backwards along the delegation chain: SM-1 informs SM-2 of the resource release. If SM-2's local resource management is Condor-like, RM-1 will also inform SM-2 of its release.

During Steps 1-7, several parties (e.g., JM-1, SM-1, SM-2, and RM-1) are involved in a string of negotiations. The main potential failures occurring in these multi-party negotiations are addressed as follows. First, an SM may not find suitable resources, both locally or through delegation. In this case, the SM sends back to the delegation requester (another SM) a `DelegationReject` message. Upon receiving a `DelegationReject` message, an SM will attempt to select and contact another SM for delegation (restart from Step 2 in Figure 8.3). Second, to prevent routing loops, and for efficiency reasons, the delegation chains are limited to a maximum length, which we call the delegation time-to-live (DTTL). Before delegating a resource request, the SM decreases its DTTL by 1. A resource request with a DTTL equal to 0 cannot be delegated. To ensure that routing loops do not occur in a more efficient way, the SMs can also retain a list of resource requests they have seen (e.g., during the past hour). Third, we account for the case when the user changes his intentions, and cancels the resource request. In this case, JM-1 is still accountable for the time during which the resource management was delegated from SM-2 to SM-1, and charges the user for this time. To prevent being charged, the user can select a reduced DTTL, or even a DTTL of 0. However, depending on the local conditions the requests with a DTTL of 0 can wait more for available resources than requests for which resources can be delegated.

The Delegated MatchMaking promises to significantly improve the performance of the system, by

occupying otherwise unused free resources with waiting jobs (through load sharing or load balancing, depending on the system policy configuration discussed later in Section 8.3.3). However, it can also worsen the performance of the system, by poor resource selection and by poor delegation routing. The resource selection is mostly influenced by the load management algorithm (discussed in Section 8.3.3). Figure 8.4 shows a worst-case performance scenario for delegation routing. Delegation requests in the figure are ordered in time in their lexicographical order (i.e., delegation A occurs before delegation B). Site 9 issues a delegation request to site 23. The site schedulers base their decision only on local information. Due to the lack of information, sites are unable to find the appropriate candidate (here, site 4), and unnecessary delegation requests occur. This leads in turn to messaging overheads, and to increased waiting times, due to waiting for the delegation and matchmaking cycles of the target sites. Additionally, the decision to delegate resource requests can lead to a suboptimal number of delegations, either too few or too many. All these load management decisions influence decisively the way the delegated matchmaking mechanism is used. We dedicate therefore the next section to load management.

8.3.3 The Delegated MatchMaking Policies

To manage the load, we use two independent algorithms: the delegation algorithm, and the local requests dispatching algorithm. We describe them and their associated policies below.

The *delegation algorithm* selects what part of the load to delegate[1], and the site manager from which the resources necessary to serve the load can be obtained. This algorithm is executed whenever the current system load is over an administrator-specified *delegation threshold*, and also at fixed intervals of time. First, requests are ordered according to a customizable *delegation policy*, e.g., FCFS. Then, the algorithm tries to delegate all the requests, in order, until the local load gets below the threshold that triggered the delegation alarm. The algorithm has to select for each request a possible target from which to bring resources locally. By design, the potential targets must be selected from the site's neighborhood. The neighbors are ordered according to a customizable *target site ordering policy*, which may take into account information about the current status of the target (e.g., its number of free resources), and an administrator selection of the request-to-target fitting (e.g., Best Fit). Upon finding the best target, the delegation protocol is initiated.

The *local requests dispatching policy* deals with the ordering of resource requests, both local and delegated. Similarly to the Condor's matchmaking cycle, we call this algorithm periodically, at intervals normally longer than those of the delegation algorithm cycle. The administrator may select the local request dispatching policy.

We argue that our architecture is operated with a generic load management mechanism. The three policies defined above allow for many traditional scheduling algorithms, and in particular gives our architecture the ability to leverage existing well-established on-line scheduling algorithms [Sga96; Alb99]. The target site ordering policy enables the use in our architecture of many of the results in traditional networking/queuing theory [Kle75].

8.4 The Experimental Setup

In this section we present the experimental setup used to evaluate the Delegated MatchMaking approach: a simulated environment encompassing both the DAS-2 and Grid'5000 grids, for a total of

[1] We actually delegate the resource requests and the right to use the resources, and do not delegate the jobs (that is, the job description, the job input data, etc.).

20 sites and over 3000 processors. We first present an overview of the simulator. Then, we describe the used workloads, with an emphasis on the differences between the workloads of grids and of traditional parallel production environments [Fra99; Lub03]. Specifically, we discuss the implications of the high number of single-processor jobs sent to the grid in batches (Section 8.4.2). We then present the simulated architectures and the assumptions that are made for these experiments.

8.4.1 The Simulator

We have used the DGSim (see Chapter 6) to simulate the combined DAS-2 and Grid'5000 grid system (see Section 8.2). Each of the 20 clusters of the combined system receives an independent stream of jobs. Depending on each job's parallelism, one or several resources are assigned to it exclusively, from the time when the job starts until the time when the job finishes.

We attempt to evaluate a stable state of the simulated system. To this end, unless otherwise specified the last simulated event in each simulation is the arrival of the last job, all job streams considered together. This ensures that our simulation does not include the cool-down phase of the system, in which no more jobs arrive while the system finishes the remaining load. The inclusion of the cool-down phase may bias the performance metrics, especially if the last jobs queue at only few of the clusters. We do not perform a similar elimination for system warm-up, as (a) we cannot distinguish reliably between the warm-up period and the normal system behavior, and (b) given the long duration of the jobs (see the workloads description in Section 8.4.3), the start-up period is small compared to the remainder of the simulation, especially for high load levels.

The simulator assesses the following performance metrics:

Utilization, Wait and Response Time, Slowdown We consider in this work the average values of the system utilization (**U**), average job wait time (**AWT**), average job response time, and average job slowdown (**ASD**). For a review of these traditional performance metrics, we refer to [Fei98b].

Goodput, expressed as the total processing time of jobs that finish successfully, from the point of view of the grid resource manager (similar to the traditional definition [Bas99], but taking into consideration that all grid jobs are executed "remotely" from the user's perspective). For the DMM architecture, we also measure the goodput of jobs running on resources obtained through delegated matchmaking. Furthermore, we account for goodput obtained on resources delegated from the same site (*intra-site goodput*), from the same grid (*intra-grid goodput*), and between the grids (*inter-grid goodput*).

Finished Jobs (JF%), expressed as the percentage of jobs that finish, from the jobs in the trace. Due to the cool-down period elimination, the maximum value for this metric is lower than 100%.

Overhead We consider the overhead of an architecture as the number of messages it employs to manage the workload. There are five types of messages: Notify Broker, Negotiate, Job Data Exchange, Resource Match-Claim-Release, and DMM (the last specific to our architecture). The *Overhead* is then expressed as a set of five values, one for the number of messages of each type. Additionally, we consider for our architecture the *number of delegations of a job*, which is defined as the length of its delegation chain.

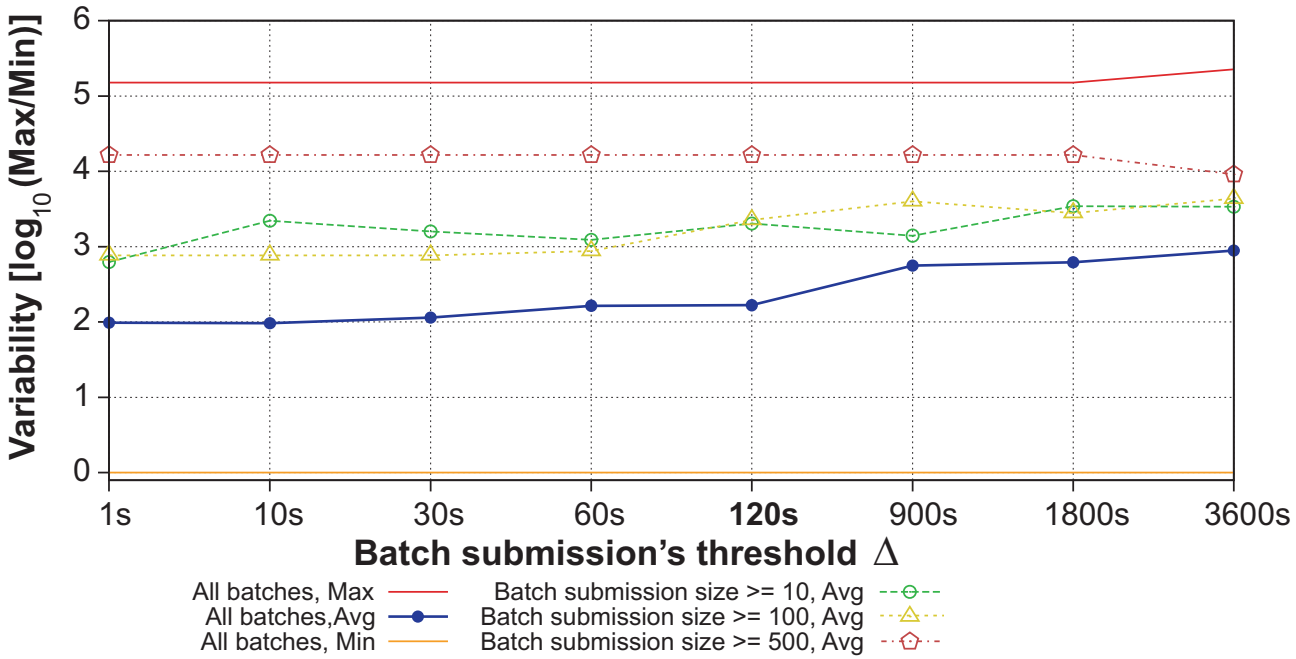


Figure 8.5: *The variability of the runtimes of jobs in batch submissions in grids.*

8.4.2 Intermezzo: Typical Grid Workloads

Two main factors contributing to the reduction of the performance of large-scaled shared systems, such as grids, are the overhead imposed by the system architecture (i.e., the messages, the mechanisms for ensuring access to resources etc.), and the queuing effects due to the random nature of the demand. While the former is under the system designer's control, the latter is dependent on the workload. Despite a strong dependency of performance on the system workload, most of the research in grid resource management does not employ realistic workloads (i.e., trace-based, or based on a validated workload model with realistic parameter values). For the few reported research results that attempt to use realistic workloads [Ham00; Smi00; Ran05], the traces considered have been taken from or modelled after the Parallel Workloads Archive. However, there exist significant differences between the parallel supercomputers workloads in the PWA and the workloads of real grid environments. In this section we present two important distinctions between them.

First, the percentage of "serial" (single-processor) jobs is much higher in grid traces than in the PWA traces. There exist 70%-100% single-processor jobs in grid traces (the percentage grows to 99-100% in most production grid environments), but only 20%-30% in the PWA traces [Lub03; Ios06a; Ios08d]. There are two potential consequences: on the one hand, the resource managers become much more loaded, due to the higher number of jobs. On the other hand, the resource managers can be much simpler, since individual single-processor jobs raise fewer scheduling issues. We have quantified in Section 7.3 the practical limits of centralized architectures.

Second, the grid single-processor jobs typically represent instances of conveniently parallel jobs, or batch submissions. A batch submission is a set of jobs ordered by the time when they arrive in the system, where each job was submitted at most Δ seconds after the first job ($\Delta = 120s$ is considered the most significant). The batch submissions are usually managed by batch engines; thus, the individual jobs arrive in the system independently. In a recent study, Iosup et al. [Ios07d] show that 70% of the

jobs, accounting for 80% of the consumed processor time, are part of batch submissions. Figure 8.5 shows that the runtime of jobs belonging to the same batch submission varies on average by at least two orders of magnitude, and that the variability increases towards five orders of magnitude as the size of the batch reaches 500 or more jobs. The predominance of batch submissions and their jobs' high runtime variability have an important impact on the operation of a large number of today's cluster and grid schedulers. The user must submit many jobs as a batch submission with a *single* runtime estimate. Due to the high runtime variability, the user cannot estimate the runtime of individual jobs, other than specifying a large value, typically the largest value allowed by the system. As a result, the scheduling schemes relying on user estimates, e.g., all backfilling variants [Mu'01], are severely affected.

8.4.3 The Workloads

The workloads used in our experiments are either traces collected from the individual grids starting at identical moments in time, or synthetic traces that reflect the properties of grid workloads. We use two alternatives to obtain the workloads: we either use data from real grid traces or generate synthetic workloads based on the properties of real grid traces. To the best of our knowledge, ours is the first study that takes into account the difference between the parallel supercomputers workloads (comprising mostly parallel jobs), and the workloads in real grid environments (comprising almost exclusively single-node jobs).

To validate our approach (Section 8.5.1), we use traces collected for each of the simulated clusters, starting at identical moments in time (the grids from which the traces originate are described in Section 8.2). However, these traces raise two problems. First, they incur a load below 20% of the combined system [Ios06a]. Second, they represent the workload of research grid environments, which contains many more parallel jobs than in a production grid.

To address both these issues, we employ a model-based trace generation for the rest of our experiments. We use the grid adaptation of the Lublin and Feitelson (Grid-LF) model described in Section 4.4.1. Using the Grid-LF model, we generate streams of rigid jobs (that is, whose size is fixed at the job's arrival in the system) for each cluster.

By modifying from the default the Grid-LF model the parameters that characterize the job inter-arrival time during peak hours, we are able to generate a load of a given level (e.g., 70%), for a system of known size (e.g., 128 processors), during a specified period (e.g., 1 month). Using this approach, we generate 10 sets of 20 synthetic job streams (one per simulated cluster) for each of the following load levels: 10%, 30%, 50%-100% in increments of 10%, 95%, 98%, 120%, 150%, and 200%. We call the *default load levels* the following nine load levels: 10%, 30%, 50%, 60%, 70%, 80%, 90%, 95%, and 98%. The results reported in Sections 8.5.2, 8.5.3, and 8.5.4 are for workloads with a duration of 1 day, for a total of 953 to 39550 jobs per set (11827 jobs per set, on average). We have repeated some of the experiments in Sections 8.5.2 and 8.5.4 for traces with the duration of 1 week and 1 month, and obtained similar results.

8.4.4 The Simulated Architectures

For the simulation of the `DMM` architecture, unless otherwise noted, we use a delegation threshold of 1.0 and a matchmaking cycle of 300s. Throughout the experiments, we employ a FCFS delegation policy, a target site ordering policy that considers only the directly connected neighbors of a site, and a FCFS local requests dispatching policy. We simulate five alternative architecture models, described below and summarized in Table 8.1.

System	Architecture	Operation
`condor`	Independent	Matchmaking
`sep-c`	Independent	Job routing
`cern`	Centralized	Job pull
`central`	Centralized	Job routing
`fcondor`	Federated	Matchmaking
`DMM`	Distributed	Matchmaking

Table 8.1: *Simulated meta-scheduling architectures.*

1. **`cern`** This is a centralized meta-scheduler architecture with job pull operation, in which users submit all their jobs to a central queue. Whenever they have free resources, sites pull jobs from the central queue. Jobs are pulled in the order they arrive in the system.

2. **`condor`** This is an independent clusters architecture with matchmaking operation that simulates a Condor-like architecture. Unlike the real system, the emulation does not prioritize users through a fair-sharing mechanism [Tha05a]. Instead, at each matchmaking round jobs are considered in the order of arrival in the system. The matchmaking cycle occurs every 300s, the default value for Condor (see `NEGOTIATOR_INTERVAL` in the Condor manual).

3. **`fcondor`** This is a federated clusters architecture with matchmaking operation that simulates a Condor-like architecture with flocking capabilities [Epe96]. The user's job manager will switch to a new site manager whenever the current site manager cannot solve all of its resource demands. This simulation model also includes the concept of fair-sharing employed by Condor in practice [Tha05a]. At each matchmaking round, users are sorted by their past usage, which is reduced (decayed) with time. Then, users are served in order, and for each user all feasible demands are solved. Similarly to `condor`, the matchmaking cycle occurs every 300s. The performance of the `fcondor` simulator corresponds to an optimistic performance estimation of a real Condor system with flocking, for two reasons. First, in the `fcondor` simulator, we allow any job manager to connect to any site manager. This potentially reduces the average job wait time, especially when the grids receive imbalanced load, as JMs can use SMs otherwise unavailable. Second, jobs that cannot be temporarily served are bypassed by jobs that can. Given that 95% of the jobs are single-processor, this results in sequential jobs being executed before parallel jobs, when the system is highly loaded. Then, the resource fragmentation and the average wait time decrease, and the utilization increases.

4. **`central`** This is a centralized grid meta-scheduler architecture with job push operation. Users submit all their jobs to a central queue. As soon as jobs arrive, the queue dispatches them on sites with free resources. Jobs stay in the queue until free resources are found. The information about the number of free resources is gathered periodically by a monitoring service.

5. **`sep-c`** This is an independent clusters architecture with job push operation. Each cluster receives a stream of jobs and stores them into a central cluster queue. From this queue, the jobs are pushed to the free resources. We assume the ideal case where the free resources are detected as soon as they become idle.

8.4.5 The Assumptions

In our simulations we make the following assumptions:

Assumption 1: No network overhead We assume that the network between the clusters is perfect and has zero latency. Given the average job runtime of one hour, we argue that this assumption has little effect. However, we do present the number of messages used by our architecture to manage the workload.

Assumption 2: Identical processors To isolate the effects of the resource management solutions, we assume identical processors across all clusters. However, the system is heterogeneous in number of processors per cluster.

Assumption 3: Ideal FCFS scheduling policy at cluster-level We assume that each site employs a FCFS policy (i.e., *without* backfilling) using an ideal, infinite-capacity, cluster gateway. Backfilling systems are effective when many parallel jobs exist in the system, and when accurate job runtime predictions are given by the users. This situation is uncommon in grids, and also in many large-scale parallel computing environments [Lee04; Tsa05]. The infinite-capacity cluster gateway favors the architectures where many of the jobs gather in one location (e.g., centralized architectures, overloaded clusters in independent clusters architectures).

Assumption 4: Processors as scheduling unit In Condor, multi-processor machines can be used as sets of single-processor machines without performance penalties. We assume that this feature is available regardless of the modelled alternative architectures. Note that this increases the performance of the `cern`, `sep-c`, and `central` architectures, and has no effect on the Condor-based `condor`, `fcondor`, and `DMM`.

Assumption 5: No background load We assume that all the load arrives through the grid inter-operation architecture, with no jobs bypassing it.

In light of our model for cluster-based grids (see Section 4.3.1) and of the model for desktop grids of Kondo et al. [Kon07b], an environment with resource failures is equivalent to a smaller environment, provided that the average resource availability duration is with high probability not lower than the runtime of the jobs. This requirement holds for the workloads considered in this work; thus, the assumption that there are no resource failures holds for our experiments.

Another assumption, the use of delegated resources with negligible overhead, is made for the experiments in Section 8.5, but later relaxed in Section 8.6.2.

8.5 The Experimental Results

In this section we present the experimental evaluation of the `DMM` approach and of five of its alternatives for inter-operating grids. Our experiments are aimed at characterizing:

- The behavior of the `DMM` approach and of its alternatives, for a duration of one year (Section 8.5.1);
- The performance of the `DMM` approach, and of its alternatives, under various load levels (Section 8.5.2);

Simulator	No.Jobs [kJobs]	AWT [s]	ASD	Goodput [CPUyr]	JF [%]
condor	455.4	7,681	1,610	117	100
sep-c	455.4	1,938	590	117	100
cern	455.4	44	6	117	100
fcondor	455.4	2,570	255	117	100
DMM	455.4	1,283	298	117	100

Table 8.2: *Performance results when running real long-term traces.*

- The effects of an imbalance between the loads of different grids on the performance of the DMM approach and of its alternatives (Section 8.5.3);
- The influence of the DMM mechanism's delegation threshold parameter on the performance of the system (Section 8.5.4);
- The overhead of the DMM mechanism (Section 8.5.5);

8.5.1 Preliminary Real Trace-Based Evaluation

For this experiment, we first assess the behavior of the DMM approach and of its alternatives when managing a contiguous subset of 4 months from the real grid traces described in Section 8.2, starting from 01/11/2005. Only for this experiment, we do not stop the simulation upon the arrival of the last job, and we do include the cool-down period. However, no jobs arrive after the 4-months limit.

We define the *workload management messages* as the NotifyBroker, the Negotiate, the Job Data Exchange, and the Resource Match-Claim-Release messages used by the delegated matchmaking mechanism (see Section 8.3.2). We further define the *resource use messages* as the messages used to maintain and control the resource during the execution of the job, e.g., the KeepAlive messages exchanged between the JM and the RM and in all Condor-based systems while jobs are being run by the RM, to ensure that the RM (JM) are continuing their loan (use) agreement. We use these definitions first in this section, where the system load is aligned temporally with the messaging to understand the behavior of the DMM mechanism, and in Section 8.5.5, where we characterize the overhead of the DMM mechanism.

Figure 8.6 shows the system behavior over a selected period of two days. During this period, all architectures are capable of starting all the incoming jobs. However, the DMM and the fcondor reduce the most the jobs' waiting time (for these two architectures, the curves of the job starts follow closely those of the job arrivals, with a small delay). In this figure, the independent clusters architecture condor introduces big delays. This can be explained by the bottom row of Figure 8.6, depicting the number of messages used by the DMM to manage the workload. The DMM messages, used to delegate work, appear mostly when the condor introduces large delays: around hours 10-13 and 33-39 (since the start of the two days period). It is in these periods that the number of arriving jobs rises at some cluster, and delegation or a similar mechanism is needed to alleviate the problem. Note that the number of jobs needs not be very high for a delegation to become desirable: around hour 33 and 39 (since start) a small cluster becomes suffocated, while much larger ones are free.

Table 8.2 shows the performance results for running the real traces over the whole 4-months period. All architectures successfully manage all the load. However, the ASD and the AWT vary greatly across architectures. The cern has the smallest ASD and AWT, by a large margin. This is

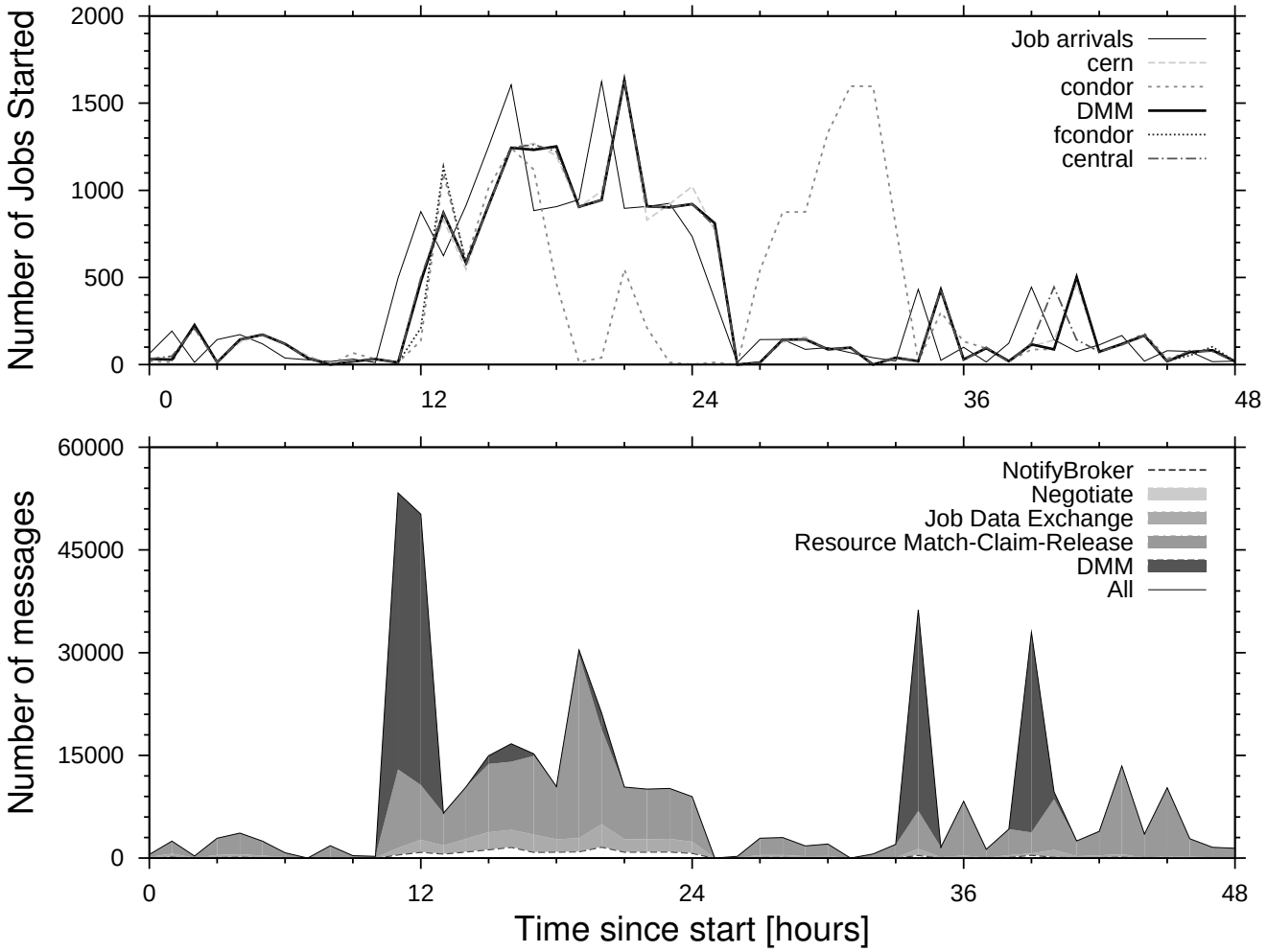


Figure 8.6: *System behavior over a selected period of two days. Top graph: comparison of the number of jobs started by the* `DMM` *and by its alternatives. Bottom graph: number of workload management messages for the* `DMM`*.*

because, unlike the alternatives, it is centralized, and operates in a lightly loaded system, where little job routing is needed. Looking at the ASD, the `fcondor` and the `DMM` have similar performance, and are both better than the independent clusters architectures (`condor` and `sep-c`). For both `condor` and `sep-c` the job response time is dominated by the time spent waiting for resources; thus, their ASD is high. Independently of whether we use AWT or ASD, in this setting the `condor` has a lower performance than `sep-c`: the time spent waiting for the next matchmaking cycle affects negatively the performance of `condor`.

We have repeated the experiments for a one-year sample with the same starting point, 01/11/2005. We have obtained similar results, with the notable exception of `fcondor`, whose AWT degraded significantly. We attribute this poor performance to the flocking target selection: if the target SM is also very loaded, `fcondor` wastes significant time, since the JM will have to wait for the target SM's next matchmaking cycle to discover that the latter cannot fulfill any demands. This gives further reasons for the development of a dynamic target selection mechanism such as the `DMM`.

8.5.2 The Influence of the Load Level

In this section we assess the performance of the DMM architecture and of its alternatives under various load levels. We report the results for the default load levels (defined in Section 8.4.3). Figure 8.7 shows the performance of the DMM architecture and of its alternatives, under the default load levels. Starting with a medium system utilization (50%) and towards higher loads, DMM offers better goodput than the alternatives. The largest difference is achieved for a load of 80%. At this load, DMM offers 32% more goodput than the fcondor. We attribute this difference to DMM's better target site selection policy, and quicker rejection of delegations by loaded sites that are incorrectly targeted. The centralized meta-schedulers, cern and central, offer in lower goodput values when the load is medium or higher.

The AWT and the ASD of DMM remain similar to that of central meta-scheduler architectures, regardless of the load level. However, DMM incurs lower AWT and ASD than fcondor at loads over 70%, and much better JF% than cern and central. DMM and fcondor manage to finish many more jobs than their alternatives, in the same time: up to twice more jobs finished for load levels below 60%, and up to four times more jobs finished for loads from 60% up to 98%. We attribute these large differences to the scheduling mechanism. The centralized architectures (cern and central) operated with FCFS may keep many jobs blocked in the queue when the oldest job cannot find its needed resources. The decentralized architectures (DMM and fcondor) act as *natural backfilling* algorithms, that is, they delegate the blocked jobs and dispatch the others. The large delegated jobs will not starve, due to the DTTL (see Section 8.3.2). Finally, there is a difference of 0%-4% in JF% between DMM and fcondor, in favor of DMM.

The independent cluster architectures, sep-c and condor, are outperformed by the other architectures for all load levels and for all performance metrics.

The additional performance comes at a cost: the additional messages sent by the DMM architecture for its delegations. Figure 8.7 also shows the number of delegations per job. Surprisingly, this overhead is relatively constant for all loads below 90%. This suggests that under realistic grid workloads with medium to high load levels, the DMM manages to find suitable delegation targets efficiently, while using a completely decentralized routing algorithm. Above 80% load, the system is overloaded, and DMM struggles to find good delegation targets, which increases the number of delegations per job in proportion with the load level increase.

8.5.3 The Influence of the Inter-Grids Load Imbalance

In this section we present the effects of an imbalance between the loads of the two grids on the performance of the DMM architecture and of its alternatives. We simulate an imbalance between the load of the DAS-2, which we keep fixed at 60%, and that of Grid'5000, which we vary from 60% to 200%. Note that at load levels higher than 120% for Grid'5000, the two-grid system is overloaded.

Figure 8.8 shows the performance of the DMM for various imbalanced system loads. The figure uses a logarithmic scale for the average wait time and for the average slowdown. At 60%/100% load, the system starts to be overloaded, and all architectures but the DMM "suffocate", i.e., they are unable to start all jobs. At 60%/150%, when the system is truly saturated, only DMM can finish more than 80% of the load. The DMM architecture is superior to its alternatives both in goodput and in percentage of finished jobs. Compared to its best alternative, fcondor, DMM achieves up to 60% more goodput, and finishes up to 26% more jobs. The cern architecture achieves lower ASD by *not* starting most of its incoming workload.

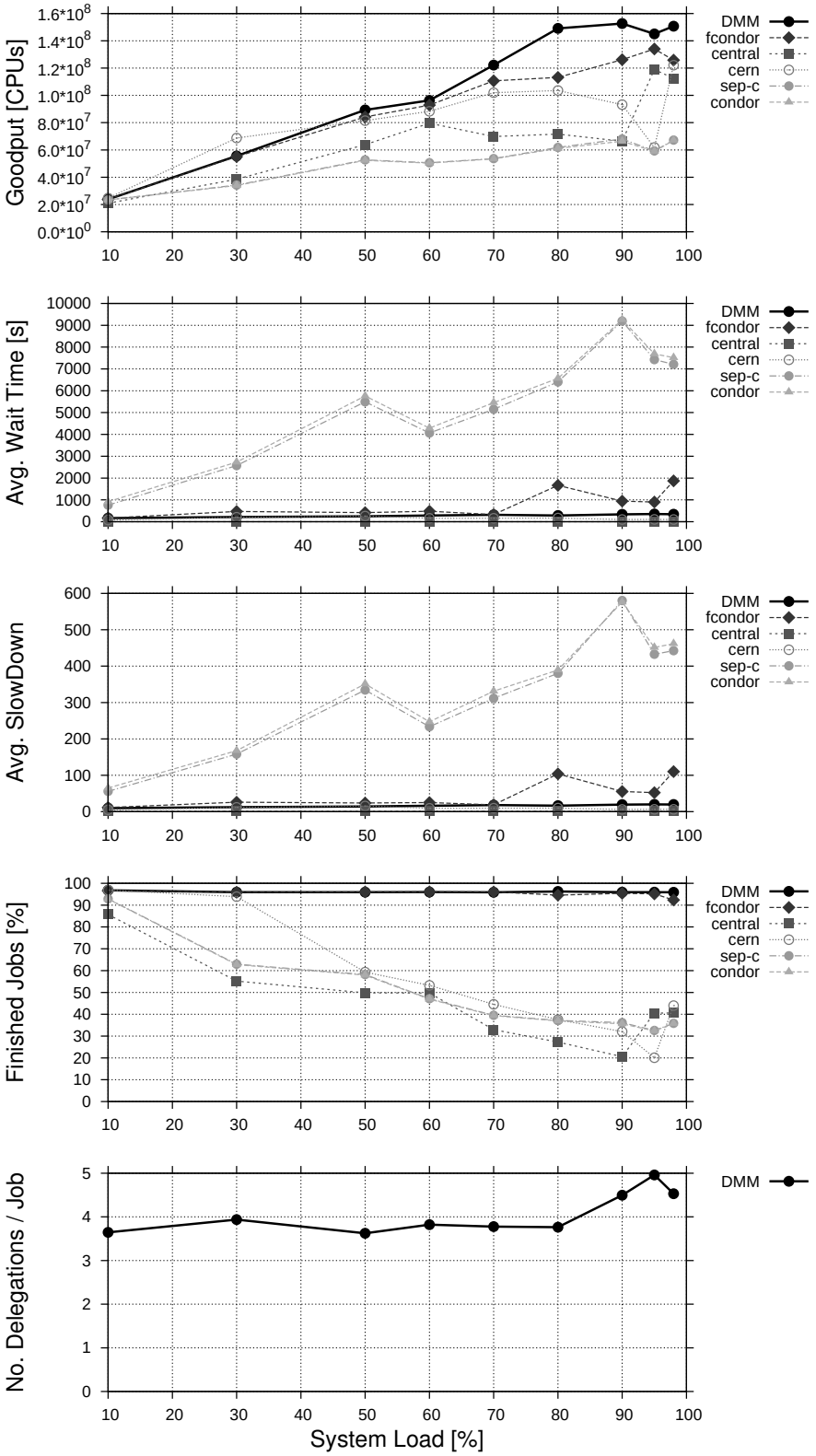


Figure 8.7: *The performance of* **DMM** *and of alternative architectures, for various system loads.*

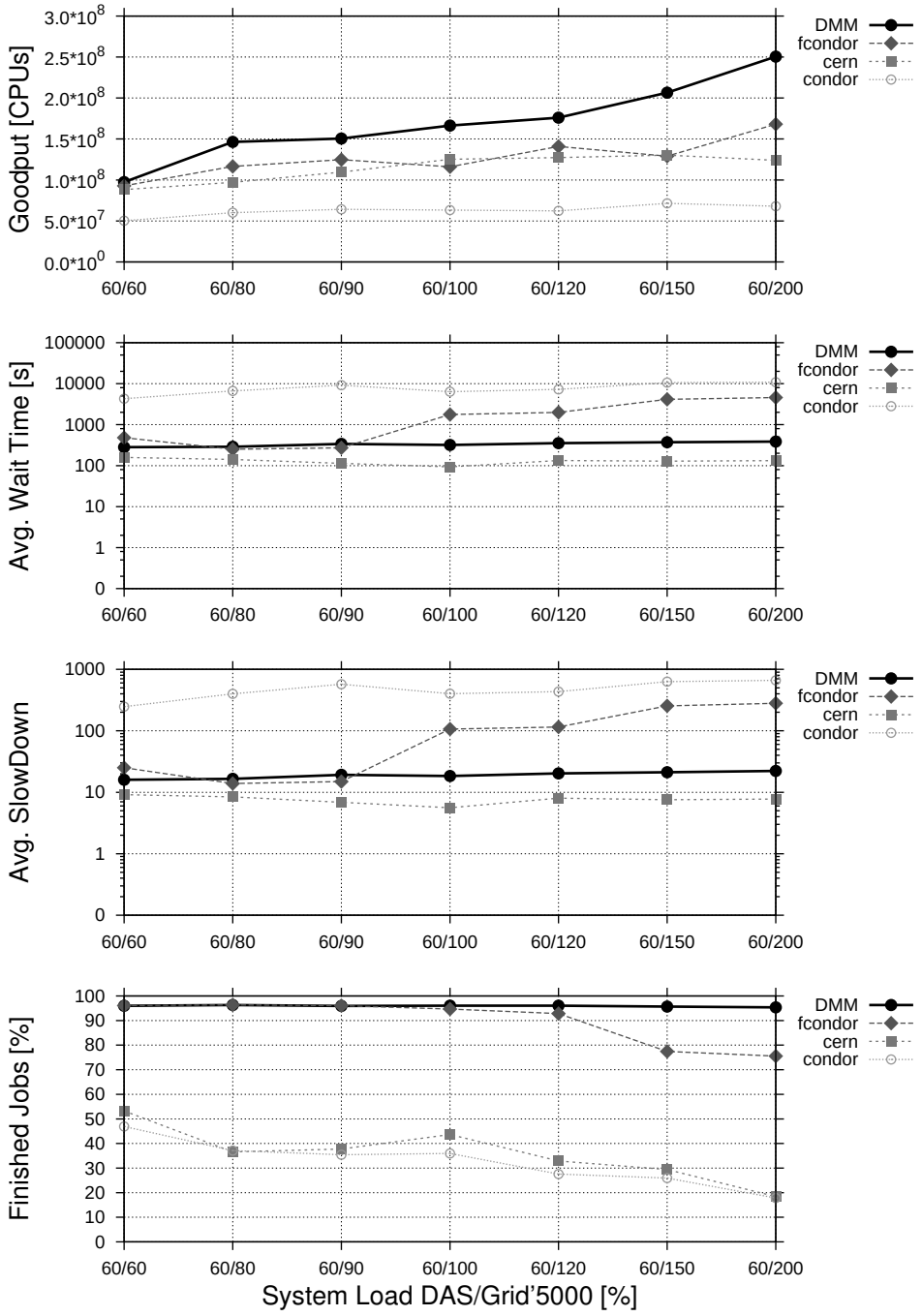


Figure 8.8: *The performance of* DMM *and of alternative architectures, for various imbalanced system loads.*

Similarly to the case of balanced load, the number of delegations per job is relatively constant for imbalanced loads of up to 60%/100%. Afterwards, the number of delegations per job increases in proportion with the load level increase, but at a higher rate than for the balanced load case.

To better understand the cause for the performance of the DMM, we show in Figure 8.9 the breakdown of the goodput components for various imbalanced system loads. According to the "keep the load local" policy (defined in Section 8.2.1), the goodput on resources delegated between sites is low for per-grid loads below 100%. However, as soon as Grid'5000 becomes overloaded, the inter-grid dele-

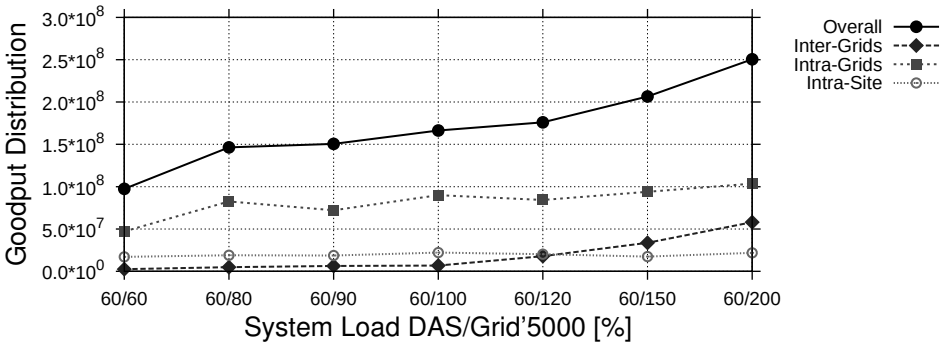


Figure 8.9: *The components of goodput for various imbalanced system loads, with DMM: overall, inter-grid, intra-grid, and intra-site goodput.*

gations become frequent, and the inter-grid goodput rises, to up to 37% from the goodput obtained on delegated resources. A similar effect can be observed for the intra-grid goodput, and for the intra-site goodput.

8.5.4 The Influence of the Delegation Threshold

The moments when the DMM architecture issues delegations and the number of requested resources depend on the delegation threshold. We therefore assess in this section the influence of the delegation threshold on the performance of the system.

Figure 8.10 shows the performance of the DMM architecture for values of the delegation threshold ranging from 0.60 to 1.25, and for six load levels ranging from 10% to 98%. A system administrator attempting to tune the system performance while keeping the overhead reduced, should select the best delegation threshold for the predicted system load. For a lightly loaded system, with a load of 30%, setting a delegation threshold higher than 1.0 leads to a quick degradation of the system performance. For a system load of 70%, considered high in systems that can run parallel jobs [Jon99], the best delegation threshold is 1.0, as it offers both the best goodput, and the lowest AWT and ASD.

8.5.5 The Message Overhead

We have defined the message types used by the DMM in Section 8.5.1. We now assess the message overhead of the DMM in two sets of scenarios: when the system load is varied, and when the delegation threshold is varied.

Figure 8.11(a) shows the distributions of the number of workload management and of the `Resource Use` messages, for various system loads (see Section 8.5.1 for message type descriptions). DMM adds only up to 19% more messages to the workload management messages for load levels below 60%, but up to 97% for higher load levels. However, the majority of messages in Condor-based systems are `KeepAlive` messages; when taking them into consideration, the messaging overhead incurred by DMM is at most 16%.

Figure 8.11(b) shows the distribution of the messages in the DMM architecture, for various values of the delegation threshold and when the system's load level is set to 70%. The number of DMM messages accounts for 7% to 16% of the total number of messages, and decreases with the growth of

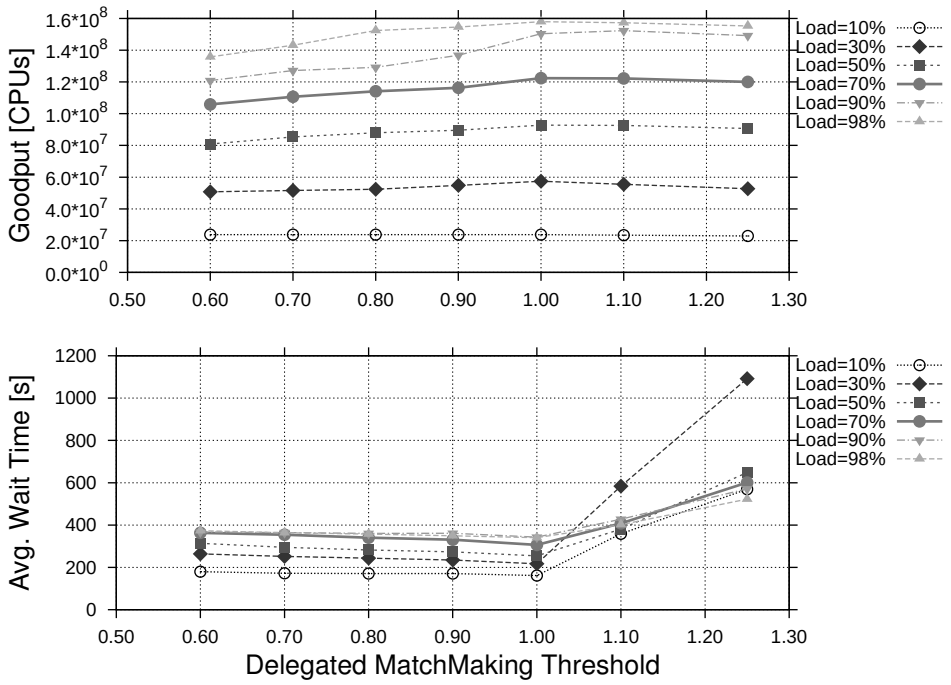


Figure 8.10: *The performance of the* `DMM` *architecture for various values of the delegation threshold.*

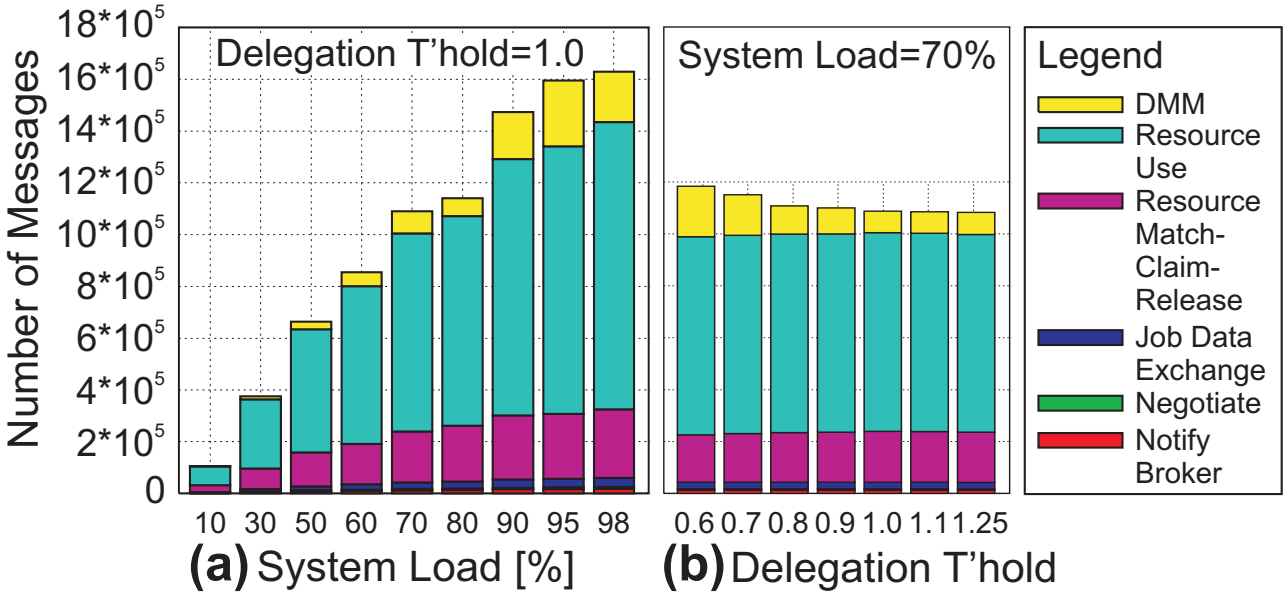


Figure 8.11: *The distribution of the messages in the* `DMM` *architecture: (a) for various system load levels and a Delegation T'hold of 1.0; (b) for various Delegation T'hold values and a 70% system load level.*

the delegation threshold. The workload management overhead when the `KeepAlive` messages are not considered grows from 35% (threshold 1.0) to 86% (threshold 0.6); the overhead is lower when

Binding	Duration [s]	Implementation
Ideal	0	-
Light	5-10	Condor Glide-Ins
Heavy	30-180	VMware, Xen, Amazon EC2

Table 8.3: *The resource binding duration for the ideal, the light, and the heavy binding.*

considering the `KeepAlive` messages.

8.6 Discussion: The Delegated MatchMaking in Practice

In this section we discuss two practical aspects of the `DMM` approach: the implementation of the site manager and the additional overheads due to the use of delegated resources.

8.6.1 Implementing the DMM Site Manager

The site manager extensions described in Section 8.3 can be implemented in three ways:

1. **Tight Coupling** An extension module providing all the necessary functionality is implemented as part of the cluster resource management system (CRMS). The main benefit of this approach is the low communication overhead (it operates inside the CRMS). However, this approach requires access to the source code of the CRMS and "root"-like access rights, and needs to be implemented for each CRMS.

2. **Loose Coupling** An extension module providing all the necessary functionality operates independently of the CRMS. The extension module polls periodically the cluster queue to determine the current load and to act upon it. This approach allows for a portable implementation (most CRMSs support the DRMAA interface for job/resource management), but has the inherent problems of producer/consumer systems: the trade-off between good performance and low system overhead is system- and (worse) load-dependent.

3. **Interposition** An extension module providing all the necessary functionality is interposed between the users and the CRMS, that is, all the jobs that enter the system pass directly through the extension module. This approach has the advantage of not being as intrusive at the tight coupling, and of reacting quicker to load changes than the loose coupling. However, it can also become the system bottleneck, and it has much higher communication overhead than the tightly coupled approach. Also, similarly to the tight coupling, normal coupling requires an implementation for each CRMS (it needs to intercept the messaging between the CRMS's submission and scheduling modules).

8.6.2 Additional DMM Overhead

In the experimental part presented in Section 8.5 we have not taken into account the duration of the resource binding procedure, or the time needed to transfer data to and from the jobs. We argue in this section that these would not affect greatly our experimental findings.

We begin with the resource binding. The ratio between the duration of the binding procedure and the average job runtime can be used as a rough estimation of the desirability of `DMM`. We identify three cases of resource binding. In the *ideal binding* case, this procedure requires no additional setup, and is

instantaneous (except for the exchange of setup messages between the JM and the RM, which can take 2-3 round-trips). In the *light binding* case, a lightweight virtualization mechanism, e.g., the Condor glide-in [Fre01; Tha05a], is deployed on the remote resources before they can be added to the user's resource pool. In the *heavy binding* case, the resource binding process requires the instantiation of a heavy virtualization mechanism, e.g., a full-blown VMware or Xen virtual machine. Table 8.3 shows the durations for the ideal, the light, and the heavy binding. We have obtained realistic values for the duration of light binding by measuring in a real environment the time it takes for a Condor glide-in to be ready for operation. Similarly, the duration of heavy binding has been reported for several of the most-used virtual machines, e.g., VMware and Xen [Krs04; Bol06; Irw06]. In addition, we have measured the resource binding duration for Amazon Elastic Computing Cloud (EC2) resources to an average of 50 seconds, with a standard deviation below 5%. Thus, for the workloads considered in this work, there is no significant increase in the average waiting time due to the use of resource binding.

A similar argument holds for the data transfers. In Section 4.4.2 we have seen two classes of transferred files: the small class below 10MB, and the large class where 50% of the transferred files are above 200MB. The files below 10MB and below 200MB are similar to the ideal and light, and to the heavy resource binding cases, respectively. In dealing with the files larger than 200MB there are two important difference from the treatment of VMs. First, while the time needed to deploy a VM is limited to at most 5-6 minutes and therefore has a limited impact on the system performance, the distribution of file sizes has been often modeled with heavy-tailed distributions, that is, there are enough large files to count [Cro97; Iam06]. Second, the interest of the users in specific files is not always uniform; Iamnitchi et al. [Iam06] show that high-energy physics files are accessed in grids with a power-law-like distribution. Thus, we reach for the case of files below 10MB a similar conclusion to that of resource binding: there is no significant increase in the average waiting time due to their use. However, more work is required to establish the impact of larger files on remote execution.

8.7 Related Work

The work presented in this chapter is situated at the intersection of two directions: the design of distributed grid inter-operation approaches and the performance evaluation of such approaches. We present in this section the related work for each of these two directions.

8.7.1 The Design of Distributed Grid Inter-Operation Approaches

We have presented in Section 7.2.3 many approaches that can (also) be employed for grid inter-operation. We now trace the evolution of ideas in the design of distributed inter-operation approaches, which has seen recently increase in activity [But03; Fu03; And03; Irw06; Sid06; de 08].

Peer-to-Peer Approaches

The seminal work of Epema et al. [Epe96] led to the design of the Condor flocking mechanism based on static and manual configuration. A peer-to-peer overlay network based on Pastry is employed to provide the dynamic (re-)configuration of the flocking mechanism [But03; But06], effectively decentralizing it. The peer-to-peer overlay is used to quickly find the clusters with free resources and which are close in terms of network latency. Independently of peer-to-peer Condor flocking, OurGrid was designed as a peer-to-peer grid inter-operation architecture [And03]. The authors focus on incentives

for participation and fair-use, with clusters running the remote jobs of other clusters and giving priority to jobs coming from the most important mutual contributors. In comparison with the DMM, the peer-to-peer approaches raise the trust issue: they require that each cluster knows and trusts each other cluster in the grid. In addition, the peer-to-peer Condor flocking also requires that each cluster knows about each job submitter (user).

Market-Based Approaches

The seminal work of Fu, Chase, et al. [Fu03] led to the design of the Secure Highly Available Resource Peering (SHARP) system, a market-based federated approach. In SHARP, each resource owner (e.g., cluster, grid) emits tickets that are bought by prospective users; the system allows tickets to be split/composed. The users obtain resource leases of fixed duration by claiming the tickets. SHARP allows tickets to be oversubscribed, that is, that the resource owners sell more tickets than their available resources. The Shirako system [Irw06] is an extension of the SHARP system for networked computing systems, e.g., inter-operated grids. It provides the management of the leased resources (installing a runtime environment and executing jobs), co-allocation features, and application-level scheduling. It also provides re-negotiation mechanisms to extend leases without reinstalling the runtime environment. Shirako is the closest approach to our DMM; the main difference and the main disadvantage of Shirako is its way of using of the market-based paradigm: the resource owners may issue too many tickets, leading to degraded service (in proportion to the oversubscription, down to 30% for very high loads [Fu03]). Another disadvantage in comparison with the DMM is the overhead associated with the expired leases: for Shirako the lease duration is set by the resource owner and the renewal requires a complex re-negotiation process, while for DMM the lease renewal requires only one KeepAlive message; the difference is important due to the inability of grid users to know the job runtime beforehand, and to the network latency between remote sites.

The InterGrid approach [de 08] also uses a market-based paradigm, along the lines of the Internet ISP-to-ISP peering relationships [Met01]. Their architecture is similar to that of the DMM, but the use of hierarchical links is in their architecture optional. They employ market-based protocols for the use of remote resources (i.e., remuneration for consumed resources is part of the protocol). The InterGrid was not yet implemented or evaluated, and no policies for operating it have as of yet been proposed.

8.7.2 The Performance Evaluation of Grid Inter-Operation Approaches

This chapter also includes a comparative evaluation of grid inter-operation alternatives. Several such evaluations have been attempted in the past [Wan85; Sha03; Ern04; Ran05]; in comparison with these studies, ours is the broadest in size and scope, and the only one to use real and realistic grid workloads.

The seminal theoretical study by Wang and Morris [Wan85] compares several load balancing approaches for distributed computing systems. They find that the pull operation (called in their study server-initiative) outperforms the push operation (source-initiative) when using the same information. On the negative side, besides the workload model (Poisson arrivals and exponential/deterministic service time, which are inappropriate models for real grid workloads), their framework does not cover the real architectures considered in our work (e.g., hierarchical systems, more refined source-initiative policies such as Condor flocking).

The study of Shan, Oliker, and Biswas [Sha03] compares a multi-cluster system with centralized or independent clusters management systems under the workload of several parallel supercomputers. A similar study is performed on a theoretical system comprising four parallel production environments

situated in different time zones [Ern04]. This latter study uses workload traces from the Parallel Workloads Archive, from which parts are selected so that the peak load periods of the four systems complement each other; thus, this study evaluates the maximum benefit due to the use of grid computing. A third study compares for a multi-cluster grid a federated architecture with ideal information about the system load with an independent clusters architecture [Ran05]; again, the workloads are selected parts of traces from the Parallel Workloads Archive. Unsurprisingly, given the high performance penalty incurred by the independent clusters architecture, the findings of these three studies are in agreement, and are similar to those of our experimental results. In addition, our study is the first to compare several design choices for the centralized and for the independent clusters architectures.

8.8 Concluding remarks

The next step in the evolution of grids is to inter-operate several grids into a single computing infrastructure, to serve larger and more diverse communities of scientists. This raises additional challenges to traditional resource management, e.g., load management between separate administrative domains. In this chapter we have proposed DMM, a novel approach for inter-operating grids. Our hybrid hierarchical/distributed architecture allows the interconnection of several grids, without requiring the operation of a central point of the hierarchy. In DMM, when a user's request cannot be satisfied locally, remote resources are transparently added to the user's site through delegated matchmaking.

We have evaluated with simulations the performance of the DMM approach, and compared it against that of five alternative architectures. A key aspect of this research is that the workloads used throughout the experiments are either real long-term grid traces, or synthetic traces that reflect the properties of grid workloads. The analysis of system performance under balanced load between grids shows that our architecture can accommodate equally well low and high system loads. In addition, the results show that for high system utilization the DMM offers a better goodput and a lower average wait time than the considered alternatives. This difference in performance increases when the inter-operated grids experience high and imbalanced loads.

These results demonstrate that the DMM architecture can result in significant performance and administrative advantages. We expect that this work will simplify the current efforts in inter-operating the DAS and Grid'5000 systems, which are currently under way. From a technical point of view, we also intend to extend our simulations to a more heterogeneous platform, to account for resource and job failures, and to investigate the impact of existing and unmovable load at the cluster level. Finally, we hope that this architecture will become a useful step for sharing resources across grids.

Chapter 9

Lowering (But Not Closing) the Curtains

In the past decade, computers have become one of the major fabrics of society, with a significant impact on the economic output of the wealthiest of countries. The integration trend emerging from the evolution of widely used distributed systems has been followed for computers by the Internet (for information sharing) and by the vision of the Grid (for computing capacity sharing). However, while the Internet is already a global presence, the vision of the Grid has yet to materialize: many grids have been deployed, but the vast majority still work in isolation. The inter-operation of these grids is the next logical step towards *the* Grid. However, the grid inter-operation problem, introduced in Chapter 1, is difficult.

The objective of this thesis is the study of grid inter-operation mechanisms. However, the lack of a research toolbox complicates the research approach. As a result, our research follows two steps: the creation of a toolbox for grid inter-operation research, and using the toolbox to create a method for grid inter-operation. We have introduced the challenges addressed by our research in Chapter 1. A basic grid model was then formulated in Chapter 2. Chapters 3, 4, 5, and 6 then introduced four tools that form the research toolbox: the Grid Workloads Archive, a comprehensive grid model, GrenchMark, and DGSim, respectively. A taxonomy for grid inter-operation alternatives based on the grid inter-operation requirements was introduced in Chapter 7. Based on the classification of real systems within the taxonomy, and on results obtained from the research toolbox, the need for new grid inter-operation mechanisms was demonstrated. A novel grid inter-operation system (architecture and mechanism) was introduced in Chapter 8. The evaluation of the new system and the comparison with five existing systems are made possible by the use of the research toolbox. To conclude, we have addressed in this thesis the grid inter-operation problem.

In the remainder of this chapter we make a review of the research questions and of the main results (Section 9.1), present nine selected lessons drawn from our experience and results (Section 9.2), and discuss three future directions for continuing this research (Section 9.3). Finally, we give a summary of the results dissemination in Section 9.4.

9.1 Review of Research Questions and Main Results

In this section we review our answers to the three research questions introduced in Chapter 1. We first present the main results which contribute to these answers, then we summarize these answers.

9.1.1 Main Results

How are real grids used? In Chapter 3 we have introduced the Grid Workloads Archive, a virtual meeting place where the grid community can archive and exchange grid workload traces. The GWA already collects traces from nine well-known grid environments, with a total content of more than 2000 users submitting more than 7 million jobs over a period of over 13 operational years, and with working

environments spanning over 130 sites comprising 10,000 resources. Given its size, the GWA already offers a good basis for the performance evaluation of grids and other large-scale distributed computing systems. The GWA has already been the basis of numerous research studies, for instance [Rai07; Yua08; Ran08b; Li08; Rza08].

What are the characteristics of the resource (un)availability in large-scale environments? In Chapter 4 we have introduced the concept of separating between the short- and the long-term changes of resource availability: the resource dynamics and the resource evolution, respectively. We have presented analysis results and models for each of the two characteristics. In one of the papers on which Chapter 4 is based [Ios07e] we have shown that the resource dynamics have an important impact on the system performance even for a low average system load.

What are the characteristics of grid workloads? In Chapter 4 we have explained the important difference between grid workloads and workloads of traditional parallel environments (e.g., clusters, supercomputers): the dominance of the bags-of-tasks comprising single-processor jobs. We have introduced a grid workload model with two components: one for a mix of parallel and single-processor jobs, and one for single-processor jobs with a focus on their grouping into bags-of-tasks. In one of the papers on which Chapter 4 is based [Ios06a] we have also shown that the distinction between grid workloads and the workloads of traditional parallel environments makes previous simulation approaches overestimate the performance of grids.

How to build an adequate testing framework for grids? In Chapter 5 we have introduced the GrenchMark grid testing framework, which automates the testing process from workload specification to obtaining results. We have also implemented, validated, and deployed a reference implementation of the GrenchMark framework. Chapter 5 presents a summary of the over twenty-five scenarios in which the reference implementation has been used, including the testing of environments of hundreds to thousands of resources managed by Condor or Globus. Notably, after being tested with GrenchMark, the Koala grid scheduler has been released to serve the DAS community of over 300 Dutch e-scientists.

How to build an adequate simulation framework for grids? In Chapter 6 we have introduced the DGSim simulation framework, which focuses on the specific requirements of the grid resource management simulation process: automatic generation of realistic grid systems and workloads, support for typical grid resource management components, and automation of the typical experiment. We have also implemented, validated, and deployed a reference implementation of the DGSim framework.

What are the alternatives for grid inter-operation? In Chapter 7 we survey the alternatives for grid inter-operation. We identify two aspects that define the ability of a grid inter-operation system to fulfill the requirements of grid inter-operation: the architecture and the operation mechanism. We develop around these aspects a taxonomy with four main architectural classes and three main operational classes; we also map twenty of the most-used real systems to this taxonomy. We analyze the limitations of the architectural aspect to fulfill the requirements of grid inter-operation. Last, we propose a novel architecture that hybridizes elements of the two most promising alternatives from the taxonomy.

Under which practical circumstances do the centralized architectures become the system bottleneck? To understand the suitability of centralized architectures, we assess in Chapter 7 the overhead and the scalability of these approaches in practice (i.e., in a real environment and using GrenchMark for testing). Our results give evidence that the centralized approaches cannot deal with the load bursts that occur in grids.

9.1.2 Answers to the Research Questions

How to inter-operate grids? In Chapter 8 we introduce the Delegated MatchMaking approach, which combines the hybrid hierarchical-decentralized architecture from Chapter 7 with a novel operation mechanism. The latter, called the Delegated MatchMaking mechanism, has two main components. First, it can delegate requests for resources up-and-down the hierarchy, and within the completely decentralized networks. Second, when resource request matches are found, the matched resources are delegated from the resource owner to the resource requester (using the first component). Through trace-based simulations we evaluate the Delegated MatchMaking approach, and find that it delivers good performance at a reasonably low overhead. We also compare our approach with five alternatives closely modeling real systems, and find that our approach delivers the best overall performance, in particular when the system load is high or when there exists a significant imbalance between the load of the grids participating in the inter-operation.

What is the possible gain of inter-operating existing grids? In Chapter 8, the evaluation of the Delegated MatchMaking approach, and the comparison with two of the independent clusters approaches (which can be seen as instances of grids operating independently), reveal that the best grid inter-operation approach can finish more jobs than the non-inter-operated approach and/or exhibit lower job response time, under high load. Additionally, the imbalance of load between the grids leads to more benefits for inter-operated approaches where load is shared between grids.

How to study grid inter-operation? We have shown in this dissertation that it is possible to study grid inter-operation within the framework introduced in Section 1.3. The main component of this framework is the toolbox for grid inter-operation research presented in Chapters 3, 4, 5, and 6.

9.2 "Grid Inter-Operation Works" and Other Lessons

We have covered a wide array of topics in this thesis, and each topic has revealed one or more important lessons about the use of the methods and tools that we have developed for the study of grid inter-operation mechanisms. We present in the following nine of them, two main and seven secondary lessons.

The first main lesson of this work is that *grid inter-operation works.* In Chapter 8, we show that the performance loss due to the lack of grid inter-operation is important: with our grid inter-operation solution more jobs were finished than with a solution that completely disregards inter-operation, under high load. Even the use of a centralized approach can lead to significant improvements (if used according to the guidelines given in Section 7.4.2).

The second main lesson is that *building the research toolbox is as important as the research using its tools.* While good intuition and work practices can help alleviate the lack of a good toolbox, in the long term the research output is limited, for the researcher but most importantly for the research community. In Chapters 3, 4, 5, and 6 we have introduced four tools that proved critical for the study of grid inter-operation mechanisms.

From hereon we present the lessons in the order of the chapters they are related to.

Lesson 3. There is no replacement for system workload traces. Many have fallen into the trap of trying to characterize/model without access to real data the workloads of large-scale systems built with emerging technologies. We have also fallen in the past into this trap by assuming that the grid workloads will comprise mainly parallel jobs; we were assuming that the parallel production community, tired of the high wait time at their (supposedly overcrowded) facilities, will move to the Grid. We learned that we were wrong, with seven of the largest current production grids being

dominated by single-processor jobs (see Chapter 4).

Lesson 4. Models of cluster resource dynamics are applicable for multi-cluster grids. In Chapter 4 we have investigated the resource dynamics of a multi-cluster grid. Our findings have shown that failure bursts are almost always contained within the cluster where the first failure occurs. Thus, previous models for cluster resource dynamics can be coupled with a model for failure location to produce realistic models for multi-cluster resource dynamics.

Lesson 5. Do not forget about resource evolution (especially if you care about resource dynamics). The existence of resource evolution has been largely ignored so far. In Chapter 4 we have investigated the resource evolution of a multi-cluster grids over a period of three years. We have found that while the cluster size tends to increase slowly over the years, the number of clusters present in such a system increases quickly. As a result, the system of tomorrow will require much more scalability from the grid inter-operation mechanism than the system of today. Thus, when designing a (multi-)grid system, the concern for canceling the effect of resource dynamics should be doubled by a similar effort for resource evolution.

Lesson 6. Traditional grid scheduling performs badly for realistic grid workloads. In Chapter 4 we have characterized and modeled the characteristics of grid workloads. We found that grid workloads are dominated by bags-of-tasks with high runtime variability. The current grid schedulers require that the user provides a single runtime value for the whole bag-of-tasks; even if the user knew the runtime of each task in the bag, he would not be able to transmit it to the grid scheduler. In recent work, we have also shown that many typical predictors fail to give reasonable runtime predictions [Ios08e]. Thus, an important assumption of traditional grid schedulers, that the job runtime is given by the user (and is reasonably accurate) or it can be easily predicted, does not hold in reality, and the resulting schedule is adversely affected.

Lesson 7. Testing real systems requires real and realistic workloads for meaningful results. From all the parameters of the testing process, the input workload has the most important impact on the validity of the results. In Section 7.3 we show that different workloads with the same overall characteristics (for example the number of jobs) lead to very different overhead and scalability results, for instance a system that was scalable for one type of workload suffocated for another. In Chapter 5 we introduce GrenchMark, the first testing framework built around the concept of testing with real and realistic workloads.

Lesson 8. For grid simulation, implementing the system model is not the time-consuming task; instead, the majority of the time is spent in setting up the environment, and in managing the results. Implementing the system model is a time-consuming aspect of simulation. However, for (multi-)grids, the environment includes tens to hundred of clusters, hundreds of users, thousands of resources, and tens of thousands of jobs; setting up this environment for the hundreds of instances required by the simulation methodology is at best a tedious process. The results produced by hundreds of (multi-)grid simulations also need to be collected and processed. Thus, the majority of time is actually spent in the setup and results management tasks, unless they are automated. In Chapter 6 we propose a grid simulation framework that does just that.

Lesson 9. Solutions for abstract models are not solutions for real systems (though understanding the difference between the two can help). In Chapter 7 we use real measurements to determine the practical limits for centralized architectures. A centralized system can be modeled (incorrectly) as using an infinitely powerful gateway, that is, disregarding the impact of the real load on the performance of the real gateways. Such models will inevitably conclude that a centralized system delivers better performance than all its alternatives, under the same scheduling prerequisites. In practice, the central gateway will suffocate under high load. It is no surprise that such studies would also be disregarded

by the system administrators.

9.3 Further directions

The framework for the study of grid inter-operation mechanisms developed in this thesis offers room for further research in the area of grid inter-operation. We identify below eight such research directions grouped in two classes depending on the use of the current framework: direct or through extensions.

9.3.1 Direct use of the current framework

Application of Delegated MatchMaking in a real environment. Many things can go wrong when leaving the safety of the simulated environment for the perils of real systems. Real systems are more complicated, and often require considerations different from the simulation studies (see Lesson 9 in the previous section). One of the motivating factors for this study was the desire to inter-operate the DAS and Grid'5000; we have solved this task in a simulated environment, and are looking forward to implement and deploy the Delegated MatchMaking architecture and mechanism in this dual-grid environment.

How many clusters are best? and other system structure related questions. We have shown in this thesis that grid inter-operation is to be preferred to leaving grids to operate independently. However, we have not yet answered the question *How many clusters are best?*, in the sense that the system administrator still does not know how many clusters to employ (and in consequence, what is the average cluster size) that would lead to the optimal performance vs. overhead trade-off. Additional questions can be formulated in this direction: *Does it help to have more clusters of the same size, or is cluster size imbalance to be preferred?*, *What is the optimal topology for a certain group of clusters?*, and *What is the optimal topology for a certain resource evolution process?*

Using other performance metrics. We have investigated the performance of various systems according to the traditional metrics, which are mostly focused around averages. However, other metrics are of interest in a highly automated system such as an inter-operated grid, e.g., the stability of the system performance, the adaptation of the traditional performance metrics for systems with dynamic resource availability, the performance delivered per VO or per user, the performance delivered to the worst-served user. In particular, the first and the last groups of metrics are of interest when the fairness of resource use is considered [Wie07].

Tuning the framework. We have analyzed grid inter-operation alternatives starting from strong assumptions about the characteristics of the grid resources and workloads. While these assumptions are the result of detailed analysis and modeling of real traces, tuning the framework for other types of workloads (such as other mixes of parallel jobs and bags-of-tasks) and for other types of systems (for instance with a different resource evolution) may be useful for studying other (multi-)grids.

9.3.2 Use of the current framework with extensions

Mechanisms for decentralization Our study of the Delegated MatchMaking approach has followed only the fully connected decentralization of nodes at the same hierarchical level and

operating under the same parent, and the same mechanism for the hierarchy roots. While this mechanism has proven enough for the Delegated MatchMaking approach to outperform its alternatives, more efficient mechanisms should be investigated. We have performed an early investigation of peer-to-peer grid inter-operation mechanisms [Ios08a].

Scheduling Our study of the Delegated MatchMaking approach has followed only the FCFS scheduling approach at the cluster level. However, the seminal work [HB00; Yan06] on scheduling jobs with unknown duration, and the numerous results in job runtime prediction [Mu'01; Li04] may enable better scheduling approaches. Also, building on-line scheduling algorithms [Sga96; Alb99] that exploit the grid inter-operation is an open research direction. We have taken first steps in these directions with the exploration of the design space of scheduling policies for bags-of-tasks [Ios08e].

New workloads Our study has made the strong assumption that the grid workloads comprise either unitary jobs or composite jobs with bag-of-tasks structure. Recently, the use of grid workflows has led to a spur of workflow-related research [Sin05; Fah05; Dua05]; we expect that as a result, the grid workflow users community will grow, making such grid applications important. We have taken the first steps towards including realistic workflow data into our framework with an analysis of grid workloads that comprise exclusively workflows [Ost08], and in our analysis of grid workflow engines (Chapter 7).

Extension of the framework with SLA concepts. Our framework focuses on the operation of integrated systems. However, economic forces may require that the sharing of resources is governed by Service Level Agreements (SLAs) [Dum07b], that is, by agreements between two or more parties to share resources under specified limits and penalties. We have taken first steps in this direction with the study of various usage SLA policies [Dum07a]. We have moved further in this direction with a problem that has large potential for commercial exploitation: serving massively multiplayer online-games [Nae08].

9.4 Results dissemination

The Grid Workloads Archive can be reached online at:

`http://gwa.st.ewi.tudelft.nl`

The GrenchMark testing framework can be reached online at:

`http://grenchmark.st.ewi.tudelft.nl/`

Additionally, GrenchMark has been integrated with the distributed testing system DiPerF into ServMark, a framework for testing grid services that is part of the Globus Incubator Projects. The ServMark testing framework can be reached online at:

`http://dev.globus.org/wiki/Incubator/ServMark`

The DGSim grid simulation framework can be reached online at:

`http://www.st.ewi.tudelft.nl/~iosup/dgsim.php`

Bibliography

[Note] *For the convenience of the reader, each bibliographic entry was appended with a list of back-references to the page(s) where the entry is referenced.*

[Th07] The Parallel Workloads Archive Team . The parallel workloads archive logs, Aug 2007. [Online]. Available: http://www.cs.huji.ac.il/labs/parallel/workload/logs.html. 17, 21, 31, 33, 77

[Th08] The Grid Workloads Archive Team . The grid workloads archive logs, Apr 2008. [Online]. Available: http://gwa.ewi.tudelft.nl/. 101, 104, 131

[06] Teragrid, Mar 2006. 24

[07a] The open science grid project (OSG), Jul 2007. 2

[07b] Worldwide LHC Grid Computing. [Online] http://lcg.web.cern.ch/LCG/., Nov 2007. 24, 98

[Abr02] D. Abramson, R. Buyya, and J. Giddy. A computational economy for grid computing and its implementation in the nimrod-g resource broker. *Future Generation Comp Syst*, vol. 18(8):pp. 1061–1074, 2002. 10, 112

[Ach97] A. Acharya, G. Edjlali, and J. H. Saltz. The utility of exploiting idle workstations for parallel computation. In *Proc. of the ACM Int'l. Conf. on Measurement and Modeling of Computer Systems (SIGMETRICS '97)*, pp. 225–236. 1997. 36, 67

[AD95] R. H. Arpaci-Dusseau, A. C. Arpaci-Dusseau, A. Vahdat, L. T. Liu, T. E. Anderson, and D. A. Patterson. The interaction of parallel and sequential workloads on a network of workstations. In *SIGMETRICS*, pp. 267–278. 1995. 67

[Alb99] S. Albers and S. Leonardi. On-line algorithms. *ACM Comput Surv*, vol. 31(3es):p. 4, 1999. 136, 158

[Ald97] J. Aldrich. R. A. Fisher and the making of maximum likelihood 1912-1922. *Statistical Science*, vol. 12(3):pp. 162–176, 1997. 41

[Alm92] V. Almeida, I. M. M. Vasconcelos, J. N. C. Árabe, and D. A. Menascé. Using random task graphs to investigate the potential benefits of heterogeneity in parallel systems. In *ACM/IEEE Conference on High Performance Networking and Computing (SC)*, pp. 683–691. ACM Press, 1992. 117

[Alt85] T. Altiok. On the phase-type approximations of general distributions. *IIE Transactions*, vol. 17(2):pp. 110–116, 1985. 42

[And02] M. Andreolini, V. Cardellini, and M. Colajanni. Benchmarking models and tools for distributed web-server systems. In M. Calzarossa and S. Tucci, editors, *Performance Evaluation of Complex Systems: Techniques and Tools, Performance 2002, Tutorial Lectures*, vol. 2459 of *Lecture Notes in Computer Science*, pp. 208–235. Springer, 2002. 93

[And03] N. Andrade, W. Cirne, F. V. Brasileiro, and P. Roisenberg. OurGrid: An approach to easily assemble grids with equitable resource sharing. In *Int'l. Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP), Revised Selected Papers*, vol. 2862 of *Lecture Notes in Computer Science*, pp. 61–86. Springer, 2003. 2, 109, 112, 150

[And06] D. P. Anderson and G. Fedak. The computational and storage potential of volunteer computing. In *IEEE/ACM Intl. Symp. on Cluster Computing and the Grid (CCGrid)*, pp. 73–80. IEEE Computer Society, 2006. 67

[Ant07] G. Antoniu, L. Cudennec, M. Jan, and M. Duigou. Performance scalability of the JXTA P2P framework. In *Int'l. Parallel & Distributed Processing Symposium (IPDPS)*, pp. 1–10. IEEE Computer Society, 2007. 92, 93

[Arb66] R. A. Arbuckle. Computer analysis and thruput evaluation. *Computers and Automation*, vol. 15(1):pp. 12–15, Jan 1966. 68

[Ava05] S. Avallone, D. Emma, A. Pescapè, and G. Ventre. Performance evaluation of an open distributed platform for realistic traffic generation. *Perform Eval*, vol. 60(1-4):pp. 359–392, 2005. 93

[Bal00] H. Bal et al. The Distributed ASCI Supercomputer project. *Operating Systems Review*, vol. 34(4):pp. 76–96, 2000. 2, 15, 23, 72, 130

[Bal01] Z. Balaton and G. Gombas. GridLab Monitoring: Detailed Architecture Specification. EU Information Society Technologies Programme (IST) TR GridLab-11-D11.2-01-v1.2, 2001. Http://www.gridlab.org/Resources/Deliverables/D11.2.pdf. 30

[Ban01] J. S. Banks, J. Carson II, B. L. Nelson, and D. M. Nicol. *Discrete-Event System Simulation*. Prentice-Hall, Inc., New Jersey, USA, 3rd edn., 2001. 95

[Bar98] P. Barford and M. Crovella. Generating representative web workloads for network and server performance evaluation. In *Proc. of the ACM Int'l. Conf. on Measurement and Modeling of Computer Systems (SIGMETRICS '98)*, pp. 151–160. 1998. 40, 93

[Bar99] P. Barford and M. Crovella. Measuring web performance in the wide area. *Performance Evaluation Review*, vol. 27(2):pp. 37–48, 1999. 92, 93

[Bas99] J. Basney and M. Livny. Improving goodput by co-scheduling cpu and network capacity. *Int'l J of High Performance Computing Applications (HPCA)*, vol. 13(3), 1999. 130, 137

[Ber94] M. Berry. Public international benchmarks for parallel computers: PARKBENCH committee: Report-1. *Sci Program*, vol. 3(2):pp. 100–146, 1994. Chairman-Roger Hockney. 92

[Ber96] F. Berman and R. Wolski. Scheduling from the perspective of the application. In *IEEE Int'l. Symp. on High Performance Distributed Computing (HPDC)*, pp. 100–111. IEEE Computer Society, 1996. 133

[Ber03a] A. N. Berger. The economic effects of technological progress: Evidence from the banking industry. *Journal of Money, Credit and Banking*, vol. 35(2):pp. 141–176, Apr 2003. 1

[Ber03b] F. Berman, R. Wolski, H. Casanova, W. Cirne, H. Dail, M. Faerman, S. M. Figueira, J. Hayes, G. Obertelli, J. M. Schopf, G. Shao, S. Smallen, N. T. Spring, A. Su, and D. Zagorodnov. Adaptive computing on the grid using apples. *IEEE Trans Parallel Distrib Syst*, vol. 14(4):pp. 369–382, 2003. 112

[Bha03] R. Bhagwan, S. Savage, and G. M. Voelker. Understanding availability. In *Proc. of the Int'l. Workshop on Peer-to-Peer Systems (IPTPS'03)*, vol. 2735 of *Lecture Notes in Computer Science*, pp. 256–267. Springer Verlag, Berkeley, CA, USA, 2003. 36, 67

[BL92] T. Berners-Lee, R. Cailliau, J.-F. Groff, and B. Pollermann. World-wide web: The information universe. *Electronic Networking: Research, Applications and Policy*, vol. 1(2):pp. 74–82, 1992. 31

[Bob05] A. Bobbio, A. Horvath, and M. Telek. Matching three moments with minimal acyclic phase type distributions. *Stochastic Models*, vol. 21:pp. 303–326, 2005. 42

[Bol00] W. J. Bolosky, J. R. Douceur, D. Ely, and M. Theimer. Feasibility of a serverless distributed file system deployed on an existing set of desktop pcs. In *Proc. of the ACM Int'l. Conf. on Measurement and Modeling of Computer Systems (SIGMETRICS '00)*, pp. 34–43. 2000. 67

[Bol06] R. Bolze, F. Cappello, E. Caron, M. Dayde, F. Desprez, E. Jeannot, Y. Jegou, S. Lanteri, J. Leduc, N. Melab, G. Mornet, R. Namyst, P. Primet, B. Quetier, O. Richard, E.-G. Talbi, and T. Irena. Grid'5000: a large scale and highly reconfigurable experimental grid testbed. *International Journal of High Performance Computing Applications*, vol. 20(4):pp. 481–494, Nov 2006. 2, 23, 43, 73, 91, 130, 150

[Bow07] S. Bowers, T. M. McPhillips, M. Wu, and B. Ludäscher. Project histories: Managing data provenance across collection-oriented scientific workflow runs. In S. C. Boulakia and V. Tannen, editors, *Int'l. Workshop on Data Integration in the Life Sciences (DILS)*, vol. 4544 of *Lecture Notes in Computer Science*, pp. 122–138. Springer, 2007. 73

[Bru99] M. Brune, J. Gehring, A. Keller, and A. Reinefeld. Managing clusters of geographically distributed high-performance computers. *Concurrency - Practice and Experience*, vol. 11(15):pp. 887–911, 1999. 109, 111, 112

[Buc03] A. I. D. Bucur and D. H. J. Epema. Trace-based simulations of processor co-allocation policies in multiclusters. In *HPDC*, pp. 70–79. IEEE CS, 2003. 15, 102, 106

[Bun01] P. Buneman, S. Khanna, and W. C. Tan. Why and where: A characterization of data provenance. In J. V. den Bussche and V. Vianu, editors, *International Conference on Database Theory (ICDT)*, vol. 1973 of *Lecture Notes in Computer Science*, pp. 316–330. Springer, 2001. 72, 76

[Bus05] J.-M. Busca, F. Picconi, and P. Sens. Pastis: A highly-scalable multi-user peer-to-peer file system. In J. C. Cunha and P. D. Medeiros, editors, *Int'l. European Conference on Parallel and Distributed Computing (Euro-Par)*, vol. 3648, pp. 1173–1182. Springer, 2005. 92, 93, 94

[But03] A. R. Butt, R. Zhang, and Y. C. Hu. A self-organizing flock of condors. In *ACM/IEEE Conference on High Performance Networking and Computing (SC)*, p. 42. ACM Press, 2003. 150

[But06] A. R. Butt, R. Zhang, and Y. C. Hu. A self-organizing flock of condors. *J Parallel Distrib Comput*, vol. 66(1):pp. 145–161, 2006. 150

[Buy02] R. Buyya and M. M. Murshed. GridSim: a toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing. *Concurrency and Computation: Practice and Experience*, vol. 14(13-15):pp. 1175–1220, 2002. 9, 95, 106

[CAI07] CAIDA Team. The Cooperative Association for Internet Data Analysis, Aug 2007. 17, 33

[Cal93] M. Calzarossa and G. Serazzi. Workload characterization: a survey. *Proc of the IEEE*, vol. 81(8):pp. 1136–50, Aug 1993. 31

[Cal94] M. Calzarossa and G. Serazzi. Construction and use of multiclass workload models. *Perform Eval*, vol. 19(4):pp. 341–352, 1994. 68, 69, 70

[Cal97] K. L. Calvert, M. B. Doar, and E. W. Zegura. Modeling internet topology. *IEEE Communications Magazine*, vol. 35(6):pp. 160–163, June 1997. 68

[Cam04] D. G. Cameron, A. P. Millar, C. Nicholson, R. Carvajal-Schiaffino, K. Stockinger, and F. Zini. Analysis of scheduling and replica optimisation strategies for data grids using OptorSim. *J Grid Comput*, vol. 2(1):pp. 57–69, 2004. 95, 106

[Cap05] N. Capit, G. D. Costa, Y. Georgiou, G. Huard, C. Martin, G. Mounie, P. Neyron, and O. Richard. A batch scheduler with high level components. In *IEEE/ACM Intl. Symp. on Cluster Computing and the Grid (CCGrid)*, pp. 776–783. IEEE Computer Society, May 2005. 23, 44, 109, 112, 131

[Cap07] F. Cappello and H. E. Bal. Toward an international "computer science grid". In *IEEE/ACM Intl. Symp. on Cluster Computing and the Grid (CCGrid)*, pp. 3–12. IEEE Computer Society, 2007. 15, 16

[Car07] E. Caron, V. Garonne, and A. Tsaregorodtsev. Definition, modelling and simulation of a grid computing scheduling system for high throughput computing. *FGCS*, vol. 23(8):pp. 968–976, 2007. 95, 106

[Cas82] X. Castillo, S. R. McConnel, and D. P. Siewiorek. Derivation and calibration of a transient error reliability model. *IEEE Trans Computers*, vol. 31(7):pp. 658–671, 1982. 40, 66, 67, 123

[Cas00] H. Casanova, A. Legrand, D. Zagorodnov, and F. Berman. Heuristics for scheduling parameter sweep applications in grid environments. In *HCW*, pp. 349–363. 2000. 100, 106

[Cas06] H. Casanova. On the harmfulness of redundant batch requests. In *International Symposium on High-Performance Distributed Computing (HPDC)*, pp. 70–79. IEEE CS, 2006. 101, 106

[Cas07] H. Casanova. Benefits and drawbacks of redundant batch requests. *J Grid Comput*, vol. 5(2):pp. 235–250, 2007. 119

[Caw04] G. Cawood. Josh system design. Tech.Rep. EPCC-SUNDCG-SD D4.2, EPCC Sun Data and Compute Grids Project, 2004. 112

[Cen07] Center for HPC Cluster Resource Management and Scheduling. Maui HPC workload/resource trace respository, Aug 2007. 33

[Cha99] S. J. Chapin, W. Cirne, D. G. Feitelson, J. P. Jones, S. T. Leutenegger, U. Schwiegelshohn, W. Smith, and D. Talby. Benchmarks and standards for the evaluation of parallel job schedulers. In D. G. Feitelson and L. Rudolph, editors, *Int'l. Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP), Revised Selected Papers*, vol. 1659 of *Lecture Notes in Computer Science*, pp. 67–90. Springer, 1999. 33, 68

[Cho00] K. Cho, K. Mitsuya, and A. Kato. Traffic data repository at the wide project. In *ATEC'00: Proceedings of the Annual Technical Conference on 2000 USENIX Annual Technical Conference*, pp. 51–51. USENIX Association, Berkeley, CA, USA, 2000. 33

[Chu03] B. N. Chun, D. E. Culler, T. Roscoe, A. C. Bavier, L. L. Peterson, M. Wawrzoniak, and M. Bowman. PlanetLab: an overlay testbed for broad-coverage services. *Computer Communication Review*, vol. 33(3):pp. 3–12, 2003. 91

[Chu04] G. Chun, H. Dail, H. Casanova, and A. Snavely. Benchmark probes for grid assessment. In *Int'l. Parallel & Distributed Processing Symposium (IPDPS)*. IEEE Computer Society, 2004. 92, 93

[Cle07] L. Clementi, M. Rambadt, R. Menday, and J. Reetz. Unicore deployment within the deisa supercomputing grid infrastructure. In W. Lehner, N. Meyer, A. Streit, and C. Stewart, editors, *Euro-Par 2006 Workshops: Parallel Processing, CoreGRID 2006, UNICORE Summit 2006, Petascale Computational Biology and Bioinformatics, Dresden, Germany, August 29-September 1, 2006, Revised Selected Papers*, vol. 4375 of *Lecture Notes in Computer Science*, pp. 264–273. Springer, 2007. 109

[Clu07] Cluster Resources Inc. Moab workload manager administrator's guide. Tech. Doc. v.5.0, Jan 2007. 109, 111, 112

[Cof72] E. G. Coffman Jr. and R. L. Graham. Optimal scheduling for two-processor systems. *Acta Inf*, vol. 1:pp. 200–213, 1972. 12

[Coh03] B. Cohen. Incentives build robustness in bittorrent. In *Workshop on Economics of Peer-to-Peer Systems*. Berkeley, USA, May 2003. http://bittorrent.com/. 93

[Cor89] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. The MIT Press and McGraw-Hill Book Company, 1989. 12

[Cro97] M. Crovella and A. Bestavros. Self-similarity in world wide web traffic: evidence and possible causes. *IEEE/ACM Trans Netw*, vol. 5(6):pp. 835–846, 1997. 40, 150

[Cza98] K. Czajkowski, I. T. Foster, N. T. Karonis, C. Kesselman, S. Martin, W. Smith, and S. Tuecke. A resource management architecture for metacomputing systems. In D. G. Feitelson and L. Rudolph, editors, *Int'l. Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP), Revised Selected Papers*, vol. 1459 of *Lecture Notes in Computer Science*, pp. 62–82. Springer, 1998. 10, 77, 112, 114

[Cza99] K. Czajkowski, I. T. Foster, and C. Kesselman. Resource co-allocation in computational grids. In *IEEE Int'l. Symp. on High Performance Distributed Computing (HPDC)*. IEEE Computer Society, 1999. 21, 89, 102

[Dab04] F. Dabek, J. Li, E. Sit, J. Robertson, M. F. Kaashoek, and R. Morris. Designing a DHT for low latency and high throughput. In *Symposium on Networked Systems Design and Implementation (NSDI)*, pp. 85–98. USENIX, 2004. 92, 93, 94

[Dan07] P. Danzig, J. Mogul, V. Paxson, and M. Schwartz. The Internet Traffic Archive, Aug 2007. 17, 33

[Dat07] DatCat Team. The DatCat Internet Measurement Data Catalog , Aug 2007. 33

[Dav04] A. Davies. Computational intermediation and the evolution of computation as a commodity. *Applied Economics*, vol. 36(11):pp. 1131–1142, 2004. Available at http://ideas.repec.org/a/taf/applec/v36y2004i11p1131-1142.html. 122

[De 07] A. De Smet. The grid exerciser, Jul 2007. 93

[de 08] M. D. de Assuncao, R. Buyya, and S. Venugopal. InterGrid: a case for internetworking islands of grids. *Concurrency and Computation: Practice and Experience*, vol. 20(8):pp. 997–1024, 2008. 150, 151

[Doa96] M. Doar. A better model for generating test networks, 1996. In Proceedings of Globecom '96. 68

[Don95] J. J. Dongarra and H. D. Simon. High performance computing in the U.S. 1995- an analysis on the basis of the TOP500 list. Tech. rep., University of Tennessee, Knoxville, TN, USA, 1995. 92

[Don07] J. J. Dongarra. Top500 supercomputer sites, Aug 2007. 92

[Dow99a] A. B. Downey and D. G. Feitelson. The elusive goal of workload characterization. *Performance Evaluation Review*, vol. 26(4):pp. 14–29, 1999. 4

[Dow99b] A. B. Downey and D. G. Feitelson. The elusive goal of workload characterization. *Performance Evaluation Review*, vol. 26(4):pp. 14–29, 1999. 62

[Dua05] R. Duan, R. Prodan, and T. Fahringer. Dee: A distributed fault tolerant workflow enactment engine for grid computing. In *HPCC*, vol. 3726 of *LNCS*, pp. 704–716. Springer-Verlag, 2005. 158

[Dum05] C. Dumitrescu and I. T. Foster. GangSim: a simulator for grid scheduling studies. In *IEEE/ACM Intl. Symp. on Cluster Computing and the Grid (CCGrid)*, pp. 1151–1158. IEEE Computer Society, 2005. 95, 106

[Dum07a] C. Dumitrescu, A. Iosup, O. Sonmez, H. Mohamed, and D. Epema. Virtual domain sharing in e-Science based on usage service level agreements. In *Proc. of the CoreGRID Symposium (CG07)*. Springer, 2007. August 2006, Rennes, France. In conjunction with the Euro-Par 2007 Conference. 158

[Dum07b] C. Dumitrescu, I. Raicu, and I. T. Foster. The design, usage, and performance of gruber: A rid sage service level agreement based rok*r*ing infrastructure. *J Grid Comput*, vol. 5(1):pp. 99–126, 2007. 158

[Ebe97] C. Ebeling. *An Introduction to Reliability and Maintainability Engineering*. McGraw-Hill, Boston, MA, 1997. 43, 48

[Eer03] P. Eerola, B. Kónya, O. Smirnova, T. Ekelöf, M. Ellert, J. R. Hansen, J. L. Nielsen, A. Wäänänen, A. Konstantinov, J. Herrala, M. Tuisku, T. Myklebust, F. Ould-Saada, and B. Vinter. The NorduGrid production grid infrastructure, status and plans. In *Int'l. Conf. on Grid Computing (GRID)*, pp. 158–165. IEEE Computer Society, 2003. 2, 23

[Ell07] M. Ellert et al. Advanced Resource Connector middleware for lightweight computational grids. *Future Generation Comp Syst*, vol. 23(2):pp. 219–240, February 2007. 23, 112

[Epe96] D. Epema, M. Livny, R. van Dantzig, X. Evers, and J. Pruyne. A worldwide flock of Condors: Load sharing among workstation clusters. *Future Generation Comp Syst*, vol. 12:pp. 53–65, 1996. 112, 113, 140, 150

[Ern02] C. Ernemann, V. Hamscher, U. Schwiegelshohn, R. Yahyapour, and A. Streit. On advantages of grid computing for parallel job scheduling. In *IEEE Int'l. Symp. on Cluster Computing and the Grid (CCGrid)*, pp. 39–51. IEEE Computer Society, 2002. 36

[Ern04] C. Ernemann, V. Hamscher, and R. Yahyapour. Benefits of global grid computing for job scheduling. In R. Buyya, editor, *GRID*, pp. 374–379. IEEE Computer Society, 2004. 36, 151, 152

[Err02] A. Erramilli, M. Roughan, D. Veitch, and W. Willinger. Self-similar traffic and network dynamics. *Proc of the IEEE*, vol. 90(5):pp. 800–19, 2002. 37

[Erw02] D. W. Erwin. Unicore - a grid computing environment. *Concurrency and Computation: Practice and Experience*, vol. 14(13-15):pp. 1395–1410, 2002. 10, 112

[Eva00] M. Evans, N. Hastings, and B. Peacock. *Statistical Distributions*. Wiley, 3rd edn., 2000. 39

[Fah05] T. Fahringer, A. Jugravu, S. Pllana, R. Prodan, C. S. Jr., and H. L. Truong. ASKALON: a tool set for cluster and grid computing. *Concurrency and Computation: Practice and Experience*, vol. 17(2-4):pp. 143–169, 2005. 158

[Fei96] D. G. Feitelson. Packing schemes for gang scheduling. In D. G. Feitelson and L. Rudolph, editors, *Int'l. Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP), Revised Selected Papers*, vol. 1162 of *Lecture Notes in Computer Science*, pp. 89–110. Springer, 1996. 68, 69

[Fei97] D. G. Feitelson. Memory usage in the LANL CM-5 workload. In D. G. Feitelson and L. Rudolph, editors, *Int'l. Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP), Revised Selected Papers*, vol. 1291 of *Lecture Notes in Computer Science*, pp. 78–94. Springer, 1997. 31

[Fei98a] D. G. Feitelson and A. Mu'alem Weil. Utilization and predictability in scheduling the IBM SP2 with backfilling. In *12th Intl. Parallel Processing Symp. (IPPS)*, pp. 542–546. Apr 1998. 31

[Fei98b] D. G. Feitelson and L. Rudolph. Metrics and benchmarking for parallel job scheduling. In D. G. Feitelson and L. Rudolph, editors, *Int'l. Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP), Revised Selected Papers*, vol. 1459 of *Lecture Notes in Computer Science*, pp. 1–24. Springer, 1998. 75, 76, 137

[Fei02] D. G. Feitelson. Workload modeling for performance evaluation. In M. Calzarossa and S. Tucci, editors, *Performance Evaluation of Complex Systems: Techniques and Tools, Performance 2002, Tutorial Lectures*, vol. 2459 of *Lecture Notes in Computer Science*, pp. 114–141. Springer, 2002. 68

[Fei07] D. G. Feitelson and B. Nitzberg. Job characteristics of a production parallel scientific workload on the NASA Ames iPSC/860. In D. G. Feitelson and L. Rudolph, editors, *Int'l. Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP), Revised Selected Papers*, vol. 949 of *Lecture Notes in Computer Science*, pp. 337–360. Springer, 2007. 31, 68, 69

[Fer72] D. Ferrari. Workload characterization and selection in computer performance measurement. *IEEE Computer*, vol. 5(4):pp. 18–24, Jul/Aug 1972. 31

[Fis95] P. A. Fishwick. Simulation model design. In *WSC*, pp. 209–211. ACM Press, 1995. 95

[Fos98] I. Foster and C. Kesselman, editors. *The Grid: Blueprint for a New Computing Infrastructure*, chap. Computational Grids, pp. 15–52. Morgan-Kaufmann, Jul 1998. 1, 2, 13, 84

[Fos04] I. Foster et al. The Grid2003 production grid: Principles and practice. In *IEEE Int'l. Symp. on High Performance Distributed Computing (HPDC)*, pp. 236–245. IEEE Computer Society, 2004. 24

[Fra99] H. Franke, J. Jann, J. E. Moreira, P. Pattnaik, and M. A. Jette. An evaluation of parallel job scheduling for ASCI Blue-Pacific. In *ACM/IEEE Conference on High Performance Networking and Computing (SC)*. ACM Press, 1999. 137

[Fre01] J. Frey, T. Tannenbaum, M. Livny, I. T. Foster, and S. Tuecke. Condor-g: A computation management agent for multi-institutional grids. In *IEEE Int'l. Symp. on High Performance Distributed Computing (HPDC)*, pp. 55–. IEEE Computer Society, 2001. 10, 112, 150

[Fu03] Y. Fu, J. S. Chase, B. N. Chun, S. Schwab, and A. Vahdat. Sharp: an architecture for secure resource peering. In *ACM Symposium on Operating Systems Principles (SOSP)*, pp. 133–148. ACM Press, 2003. 150, 151

[Fu07] S. Fu and C.-Z. Xu. Exploring event correlation for failure prediction in coalitions of clusters. In B. Verastegui, editor, *Proc. of the ACM/IEEE Conference on High Performance Networking and Computing (SC 2007), November 10-16, 2007, Reno, Nevada, USA*, p. 41. 2007. 36

[Fuj99] R. M. Fujimoto. *Parallel and Distribution Simulation Systems*. John Wiley & Sons, Inc., NY, USA, 1999. 95, 106

[Gar06] P. Garbacki, A. Iosup, D. H. J. Epema, and M. van Steen. 2Fast: Collaborative downloads in p2p networks. In *Peer-to-Peer Computing*, pp. 23–30. IEEE Computer Society, 2006. 92, 93, 94

[Gen01] W. Gentzsch. Sun grid engine: Towards creating a compute power grid. In *IEEE/ACM Intl. Symp. on Cluster Computing and the Grid (CCGrid)*, pp. 35–39. IEEE Computer Society, 2001. 10, 117

[God07] B. Godfrey and I. Stoica. Repository of availability traces (RAT), Aug 2007. 33

[Gom06] Gomez-Salvador, Musso, Stocker, and Turunen. Labour productivity developments in the Euro area. Occasional paper series no.53, ECB, 2006. 1

[Gra86] J. Gray. Why do computers stop and what can be done about it? In *Symposium on Reliability in Distributed Software and Database Systems*, pp. 3–12. 1986. 66, 67

[Gra90] J. Gray. A Census of Tandem System Availability Between 1985 and 1990. In *IEEE Trans. on Reliability*, vol. 39, pp. 409–418. October 1990. 43, 66, 67

[Gum03] P. K. Gummadi, R. J. Dunn, S. Saroiu, S. D. Gribble, H. M. Levy, and J. Zahorjan. Measurement, modeling, and analysis of a peer-to-peer file-sharing workload. In *19th ACM Symposium on Operating Systems Principles (SOSP)*, pp. 314–329. Bolton Landing, NY, USA, Oct 2003. 36

[Hal05] E. Halepovic and R. Deters. The JXTA performance model and evaluation. *Future Generation Comp Syst*, vol. 21(3):pp. 377–390, 2005. 92, 93

[Ham00] V. Hamscher, U. Schwiegelshohn, A. Streit, and R. Yahyapour. Evaluation of job-scheduling strategies for grid computing. In *GRID*, vol. 1971 of *Lecture Notes in Computer Science*, pp. 191–202. Springer Verlag, 2000. 138

[HB97] M. Harchol-Balter and A. B. Downey. Exploiting process lifetime distributions for dynamic load balancing. *ACM Trans Comput Syst*, vol. 15(3):pp. 253–285, 1997. 40, 68, 69, 70

[HB00] M. Harchol-Balter. Task assignment with unknown duration. In *ICDCS*, pp. 214–224. 2000. 158

[He05] L. He, S. A. Jarvis, D. P. Spooner, D. A. Bacigalupo, G. Tan, and G. R. Nudd. Mapping dag-based applications to multiclusters with background workload. In *CCGRID*, pp. 855–862. 2005. 106

[Hea02] T. Heath, R. P. Martin, and T. D. Nguyen. Improving cluster availability using workstation validation. In *Proc. of the ACM Int'l. Conf. on Measurement and Modeling of Computer Systems (SIGMETRICS '02)*, pp. 217–227. ACM Press, 2002. 66, 67

[Hey02] T. Hey and A. E. Trefethen. The uk e-science core programme and the grid. *Future Generation Comp Syst*, vol. 18(8):pp. 1017–1031, 2002. 2

[Hey05] A. J. G. Hey and G. Fox. Special issue: Grids and web services for e-science. *Concurrency - Practice and Experience*, vol. 17(2-4):pp. 317–322, 2005. 1

[Hoa07a] W. Hoarau, S. Tixeuil, and L. Silva. Fault-injection and dependability benchmarking for grid computing middleware. In S. Gorlatch and M. Danelutto, editors, *Integrated Research in GRID Computing*, pp. 119–134. Springer, 2007. 92, 93

[Hoa07b] W. Hoarau, S. Tixeuil, and F. Vauchelles. Fail-fci: Versatile fault-injection. *Future Generation Comput Syst*, vol. 23(7):pp. 913–919, 2007. 93

[Hor07] G. Horn, A. Kvalbein, J. Blomskøld, and E. Nilsen. An empirical comparison of generators for self similar simulated traffic. *Perform Eval*, vol. 64(2):pp. 162–190, 2007. 93

[Hot96] S. Hotovy. Workload evolution on the Cornell Theory Center IBM SP2. In D. G. Feitelson and L. Rudolph, editors, *Int'l. Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP), Revised Selected Papers*, vol. 1162 of *Lecture Notes in Computer Science*, pp. 27–40. Springer, 1996. 31, 88

[HS03] J. Hernandez-Serrano, M. Soriano, J. Pegueroles, and F. Rico-Novella. Heuristic method for synthetic traffic generation for multicast networked games. In *Communication Systems and Networks*. Acta Press, 2003. 41

[Hue04] E. Huedo, R. S. Montero, and I. M. Llorente. A framework for adaptive execution in grids. *Softw, Pract Exper*, vol. 34(7):pp. 631–651, 2004. 112, 114

[Hum06] M. Humphrey. Altair's pbs - altair's pbs professional update. In *ACM/IEEE Conference on High Performance Networking and Computing (SC)*, p. 28. ACM Press, 2006. 10

[i2006] i2010 High Level Group. The economic impact of ICT: evidence and questions. Issue paper on the economic impact of ict, i2010 European Information Society, Apr 2006. 1

[Iam06] A. Iamnitchi, S. Doraimani, and G. Garzoglio. Filecules in high-energy physics: Characteristics and impact on resource management. In *International Symposium on High-Performance Distributed Computing (HPDC)*, pp. 69–80. IEEE CS, 2006. 58, 150

[Ios06a] A. Iosup, C. Dumitrescu, D. H. J. Epema, H. Li, and L. Wolters. How are real grids used? the analysis of four grid traces and its implications. In *GRID*, pp. 262–269. IEEE Computer Society, 2006. 4, 6, 17, 29, 31, 35, 37, 62, 68, 71, 73, 77, 90, 92, 97, 115, 116, 121, 138, 139, 154

[Ios06b] A. Iosup and D. H. J. Epema. GrenchMark: A framework for analyzing, testing, and comparing grids. In *CCGRID*, pp. 313–320. IEEE Computer Society, 2006. 4, 6, 9, 17, 29, 30, 31, 71, 72, 78, 82, 83, 119

[Ios06c] A. Iosup, D. H. J. Epema, C. Franke, A. Papaspyrou, L. Schley, B. Song, and R. Yahyapour. On grid performance evaluation using synthetic workloads. In E. Frachtenberg and U. Schwiegelshohn, editors, *JSSPP*, vol. 4376 of *Lecture Notes in Computer Science*, pp. 232–255. Springer, 2006. 21, 36, 75

[Ios06d] A. Iosup, P. Garbacki, and D. H. J. Epema. Provisioning and scheduling resources for world-wide data-sharing services. In *e-Science*, p. 84. IEEE Computer Society, 2006. 29

[Ios06e] A. Iosup, P. Garbacki, J. A. Pouwelse, and D. H. J. Epema. Correlating topology and path characteristics of overlay networks and the internet. In *CCGRID*, p. 10. IEEE Computer Society, 2006. 93

[Ios07a] A. Iosup and D. H. J. Epema. GrenchMark: a framework for testing large-scale distributed computing systems. *Perform Eval*, Sep 2007. (submitted). 6, 71

[Ios07b] A. Iosup, D. H. J. Epema, P. Couvares, A. Karp, and M. Livny. Build-and-test workloads for grid middleware: Problem, analysis, and applications. In *CCGRID*, pp. 205–213. IEEE Computer Society, 2007. 17, 31, 71

[Ios07c] A. Iosup, D. H. J. Epema, T. Tannenbaum, M. Farrellee, and M. Livny. Inter-operating grids through delegated matchmaking. In *ACM/IEEE Conference on High Performance Networking and Computing (SC)*, p. 13. ACM Press, 2007. 4, 6, 7, 8, 9, 29, 101, 104, 105, 109, 129

[Ios07d] A. Iosup, M. Jan, O. O. Sonmez, and D. H. J. Epema. The characteristics and performance of groups of jobs in grids. In A.-M. Kermarrec, L. Bougé, and T. Priol, editors, *Euro-Par*, vol. 4641 of *Lecture Notes in Computer Science*, pp. 382–393. Springer, 2007. 4, 6, 29, 35, 92, 138

[Ios07e] A. Iosup, M. Jan, O. O. Sonmez, and D. H. J. Epema. On the dynamic resource availability in grids. In *GRID*, pp. 26–33. IEEE Computer Society, 2007. 4, 6, 21, 30, 35, 40, 66, 100, 101, 104, 116, 154

[Ios08a] A. Iosup, D. H. Epema, and A. Kertesz. Investigating peer-to-peer meta-brokering in grids. Tech. Rep. TR-0170, Institute on Resource Management and Scheduling, Aug 2008. 158

[Ios08b] A. Iosup, D. H. J. Epema, T. Tannenbaum, M. Farrellee, and M. Livny. Inter-operating grids through delegated matchmaking. *Sci Prog, Special Edition "Best Paper Award at SuperComputing 2007"*, pp. 1–21, Feb 2008. 7, 8, 109, 129

[Ios08c] A. Iosup, H. Li, C. Dumitrescu, L. Wolters, and D. Epema. The Grid Workload Format , Apr 2008. [Online]. Available: http://gwa.ewi.tudelft.nl/TheGridWorkloadFormat_v001.pdf. 21

[Ios08d] A. Iosup, H. Li, M. Jan, S. Anoep, C. Dumitrescu, L. Wolters, and D. Epema. The grid workloads archive. *Future Generation Comp Syst*, vol. 24(7), Feb 2008. 6, 17, 31, 44, 57, 75, 77, 91, 118, 131, 138

[Ios08e] A. Iosup, O. Sonmez, S. Anoep, and D. H. Epema. The performance of bags-of-tasks in large-scale distributed systems. In *IEEE Int'l. Symp. on High Performance Distributed Computing (HPDC)*, pp. 97–108. ACM Press, 2008. 6, 9, 35, 68, 69, 70, 118, 156, 158

[Ios08f] A. Iosup, O. O. Sonmez, and D. H. J. Epema. DGSim: Comparing grid resource management architectures through trace-based simulation. In E. Luque, T. Margalef, and D. Benitez, editors, *Euro-Par*, vol. 5168 of *Lecture Notes in Computer Science*, pp. 13–25. Springer, 2008. 7, 95

[Ios08g] A. Iosup, C. Stratan, and D. H. J. Epema. Grid workflow engines: a study of their functionality, performance, and reliability. In *GRID*, pp. 25–32. IEEE Computer Society, 2008. 6, 7, 9, 109

[Irw06] D. E. Irwin, J. S. Chase, L. E. Grit, A. R. Yumerefendi, D. Becker, and K. Yocum. Sharing networked resources with brokered leases. In *USENIX Annual Technical Conference, General Track*, pp. 199–212. USENIX Association, 2006. 150, 151

[Iye82] R. K. Iyer, S. E. Butner, and E. J. McCluskey. A statistical failure/load relationship: Results of a multicomputer study. *IEEE Trans Computers*, vol. 31(7):pp. 697–706, 1982. 66, 67

[Jac01] D. B. Jackson, Q. Snell, and M. J. Clement. Core algorithms of the maui scheduler. In *Int'l. Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP), Revised Selected Papers*, vol. 2221 of *Lecture Notes in Computer Science*, pp. 87–102. Springer, 2001. 10

[Jai91] R. Jain. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling,*. May 1991. 31, 68, 117

[Jan97] J. Jann, P. Pattnaik, H. Franke, F. Wang, J. Skovira, and J. Riordan. Modeling of workload in mpps. In D. G. Feitelson and L. Rudolph, editors, *Int'l. Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP), Revised Selected Papers*, vol. 1291 of *Lecture Notes in Computer Science*, pp. 95–116. Springer, 1997. 40, 68, 69, 92

[Jin01] S. Jin and A. Bestavros. Gismo: a generator of internet streaming media objects and workloads. *SIGMETRICS Performance Evaluation Review*, vol. 29(3):pp. 2–10, 2001. 40

[Joh89] M. A. Johnson and M. R. Taaffe. Matching moments to phase distributions: Mixtures of Erlang distributions of common order. *Communications in Statistics - Stochastic Models*, vol. 5(4):pp. 711–743, 1989. 42

[Joh90] M. A. Johnson and M. R. Taaffe. Matching moments to phase distributions: Density function shapes. *Communications in Statistics - Stochastic Models*, vol. 6(2):pp. 283–306, 1990. 42

[Jon99] J. P. Jones and B. Nitzberg. Scheduling for parallel supercomputing: A historical perspective of achievable utilization. In *Int'l. Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP), Revised Selected Papers*, vol. 1659 of *Lecture Notes in Computer Science*, pp. 1–16. Springer, 1999. 147

[Jon04] W. M. Jones, L. W. Pang, D. C. Stanzione Jr., and W. B. Ligon III. Job communication characterization and its impact on meta-scheduling co-allocated jobs in a mini-grid. In *IPDPS*. IEEE CS, 2004. 95, 106

[Jor02] D. Jorgenson, M. Ho, and K. Stiroh. Information Technology, education, and the sources of economic growth across U.S. industries. White paper, U.S. Federal Reserve Board, 2002. 1

[JXT07] JXTA Community. JXTA Bench Project, Jul 2007. 93

[Kal99] M. Kalyanakrishnam, Z. Kalbarczyk, and R. K. Iyer. Failure data analysis of a lan of windows nt based computers. In *Symposium on Reliable Distributed Systems*, pp. 178–187. 1999. 67

[Kap99] N. H. Kapadia and J. A. B. Fortes. Punch: An architecture for web-enabled wide-area network-computing. *Cluster Computing*, vol. 2(2):pp. 153–164, 1999. 112

[Kar03] N. T. Karonis, B. R. Toonen, and I. T. Foster. Mpich-g2: A grid-enabled implementation of the message passing interface. *J Parallel Distrib Comput*, vol. 63(5):pp. 551–563, 2003. 11

[Kas08] H. Kasahara. Standard Task Graph Set. [Online] Available: http://www.kasahara.elec.waseda.ac.jp/schedule/, Apr 2008. 117

[Kee04] Y.-S. Kee, H. Casanova, and A. A. Chien. Realistic modeling and svnthesis of resources for computational grids. In *Proc. of the ACM/IEEE Conference on High Performance Networking and Computing (SC 2004), 6-12 November 2004, Pittsburgh, PA, USA, CD-Rom*, p. 54. IEEE Computer Society, 2004. 52, 68, 116

[Ken02] C. Kenyon and G. Cheliotis. Architecture requirements for commercializing grid resources. In *IEEE Int'l. Symp. on High Performance Distributed Computing (HPDC)*, pp. 215–224. IEEE Computer Society, 2002. 84

[Ken03] C. Kenyon and G. Cheliotis. Creating services with hard guarantees from cycle-harvesting systems. In *IEEE/ACM Intl. Symp. on Cluster Computing and the Grid (CCGrid)*, pp. 224–231. IEEE Computer Society, 2003. 84

[Kha06] O. Khalili, J. He, C. Olschanowsky, A. Snavely, and H. Casanova. Measuring the performance and reliability of production computational grids. In *GRID*, pp. 293–300. IEEE CS, 2006. 17, 71

[Kle75] L. Kleinrock. *Queueing Systems, Vol. 1: Theory*. Wiley, New York, 1975. 37, 38, 136

[Kle76] L. Kleinrock. *Queueing Systems, Vol. 2: Computer Applications*. Wiley, New York, 1976. 37, 39

[Kon07a] D. Kondo, F. Araujo, P. Malecot, P. Domingues, L. M. Silva, G. Fedak, and F. Cappello. Characterizing result errors in internet desktop grids. In A.-M. Kermarrec, L. Bougé, and T. Priol, editors, *Int'l. European Conference on Parallel and Distributed Computing (Euro-Par)*, vol. 4641 of *Lecture Notes in Computer Science*, pp. 361–371. Springer, 2007. 35, 67

[Kon07b] D. Kondo, G. Fedak, F. Cappello, A. A. Chien, and H. Casanova. Characterizing resource availability in enterprise desktop grids. *Future Generation Comp Syst*, vol. 23(7):pp. 888–903, 2007. 33, 35, 36, 67, 141

[Kon07c] D. Kondo, G. Fedak, F. Cappello, A. A. Chien, and H. Casanova. The desktop grid availability traces archive, Aug 2007. 33

[Kra05] D. Kranzlmüller. The egee project - a multidisciplinary, production-level grid. In L. T. Yang, O. F. Rana, B. D. Martino, and J. Dongarra, editors, *Proc. of the Int'l. Conf. on High Performance Computing and Communications (HPCC)*, vol. 3726 of *Lecture Notes in Computer Science*, p. 2. Springer, 2005. 2

[Krs04] I. Krsul, A. Ganguly, J. Zhang, J. A. B. Fortes, and R. J. O. Figueiredo. Vmplants: Providing and managing virtual machine execution environments for grid computing. In *ACM/IEEE Conference on High Performance Networking and Computing (SC)*, p. 7. IEEE Computer Society, 2004. 150

[Kur07] K. Kurowski, J. Nabrzyski, A. Oleksiak, and J. Weglarz. Grid scheduling simulations with GSSIM. In *SRMPDS*. 2007. 95, 100, 106

[Kwo99] Y.-K. Kwok and I. Ahmad. Benchmarking and comparison of the task graph scheduling algorithms. *J Parallel Distrib Comput*, vol. 59(3):pp. 381–422, 1999. 117

[Lam05] W. Lammers. *Adding Support for New Application Types to the KOALA Grid Scheduler*. Master's thesis, Delft University of Technology, Delft, NL, Oct 2005. 72

[Law00] A. M. Law and W. D. Kelton. *Simulation Modeling and Analysis*. McGraw Hill, NY, USA, 3rd edn., 2000. 95

[Lee04] C. B. Lee, Y. Schwartzman, J. Hardy, and A. Snavely. Are user runtime estimates inherently inaccurate?. In *Int'l. Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP), Revised Selected Papers*, vol. 3277 of *Lecture Notes in Computer Science*, pp. 253–263. Springer, 2004. 141

[Leg00] I. Legrand and H. B. Newman. The MONARC toolset for simulating large network-distributed processing systems. In *WSC*, pp. 1794–1801. 2000. 106

[Leg03] A. Legrand, L. Marchal, and H. Casanova. Scheduling distributed applications: the simgrid simulation framework. In *IEEE/ACM Intl. Symp. on Cluster Computing and the Grid (CCGrid)*, pp. 138–145. IEEE Computer Society, 2003. 106

[Lel86] W. E. Leland and T. J. Ott. Load-balancing heuristics and process behavior. In *Proc. of the ACM Int'l. Conf. on Measurement and Modeling of Computer Systems (SIGMETRICS '86)*, pp. 54–69. 1986. 68, 69

[Lel94] W. E. Leland, M. S. Taqqu, W. Willinger, and D. V. Wilson. On the self-similar nature of ethernet traffic (extended version). *IEEE/ACM Trans Netw*, vol. 2(1):pp. 1–15, 1994. 37, 40, 68

[Li04] H. Li, D. L. Groep, J. Templon, and L. Wolters. Predicting job start times on clusters. In *IEEE/ACM Intl. Symp. on Cluster Computing and the Grid (CCGrid)*, pp. 301–308. IEEE Computer Society, 2004. 29, 158

[Li05] H. Li, D. L. Groep, and L. Wolters. Workload characteristics of a multi-cluster supercomputer. In D. G. Feitelson, L. Rudolph, and U. Schwiegelshohn, editors, *Int'l. Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP), Revised Selected Papers*, vol. 3277 of *Lecture Notes in Computer Science*, pp. 176–193. Springer, 2005. 37, 58, 68, 69, 88, 90

[Li06] H. Li, D. Groep, L. Wolters, and J. Templon. Job failure analysis and its implications in a large-scale production grid. In *IEEE Int'l. Conf. on e-Science and Grid Computing (e-Science)*, pp. 27–27. IEEE Computer Society, 2006. 17, 71

[Li07a] H. Li. Long range dependent job arrival process and its implications in grid environments. In *Proc. of MetroGrid Workshop, Int'l. Conference on Networks for Grid Applications (GridNets07)*. ACM Press, 2007. (in press). 29

[Li07b] H. Li, D. L. Groep, and L. Wolters. Mining performance data for metascheduling decision support in the grid. *Future Generation Comp Syst*, vol. 23(1):pp. 92–99, 2007. 29

[Li07c] H. Li, R. Heusdens, M. Muskulus, and L. Wolters. Analysis and synthesis of pseudo-periodic job arrivals in grids: A matching pursuit approach. In *IEEE/ACM Intl. Symp. on Cluster Computing and the Grid (CCGrid)*, pp. 183–196. IEEE Computer Society, 2007. 4, 29

[Li07d] H. Li and M. Muskulus. Analysis and modeling of job arrivals in a production grid. *SIGMETRICS Performance Evaluation Review*, vol. 34(4):pp. 59–70, 2007. 37, 69, 70

[Li07e] H. Li, M. Muskulus, and L. Wolters. Modeling job arrivals in a data-intensive grid. In E. Frachtenberg and U. Schwiegelshohn, editors, *Int'l. Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP), Revised Selected Papers*, vol. 4376 of *Lecture Notes in Computer Science*, pp. 210–231. Springer, 2007. 4, 29, 37, 57, 118

[Li07f] H. Li and L. Wolters. Towards a better understanding of workload dynamics on data-intensive clusters and grids. In *Int'l. Parallel & Distributed Processing Symposium (IPDPS)*, pp. 1–10. IEEE Computer Society, 2007. 4, 29, 37, 68, 115

[Li08] H. Li. *Workload Characterization, Modeling, and Prediction in Grid Computing*. Phd thesis, Leiden Universiteit, Mar 2008. 154

[Lil69] H. W. Lilliefors. On the kolmogorov-smirnov test for the exponential distribution with mean unknown. *Journal of the American Statistical Association*, vol. 64:pp. 387–389, 1969. 42

[Lin90] T.-T. Y. Lin and D. P. Siewiorek. Error log analysis: statistical modeling and heuristic trend analysis. In *IEEE Trans. on Reliability*, vol. 39, pp. 419–432. October 1990. 66, 67, 123

[Lit88] M. J. Litzkow, M. Livny, and M. W. Mutka. Condor - a hunter of idle workstations. In *ICDCS*, pp. 104–111. 1988. 112

[Lo98] V. M. Lo, J. Mache, and K. J. Windisch. A comparative study of real workload traces and synthetic workload models for parallel job scheduling. In D. G. Feitelson and L. Rudolph, editors, *Int'l. Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP), Revised Selected Papers*, vol. 1459 of *Lecture Notes in Computer Science*, pp. 25–46. Springer, 1998. 68

[Lon95] D. D. E. Long, A. Muir, and R. A. Golding. A longitudinal survey of internet host reliability. In *Symposium on Reliable Distributed Systems*, pp. 2–9. 1995. 67

[Lu03] D. Lu and P. A. Dinda. Synthesizing realistic computational grids. In *Proc. of the ACM/IEEE Conference on High Performance Networking and Computing (SC 2003)*, p. 16. ACM Press, 2003. 68

[Lub03] U. Lublin and D. G. Feitelson. The workload on parallel supercomputers: modeling the characteristics of rigid jobs. *J Parallel Distrib Comput*, vol. 63(11):pp. 1105–1122, 2003. 21, 37, 56, 57, 58, 62, 68, 69, 70, 71, 92, 101, 104, 137, 138

[Lus06] P. Luszczek, D. H. Bailey, J. Dongarra, J. Kepner, R. F. Lucas, R. Rabenseifner, and D. Takahashi. S12 - The HPC Challenge (HPCC) benchmark suite. In *ACM/IEEE Conference on High Performance Networking and Computing (SC)*, p. 213. ACM Press, 2006. 92, 93

[Mam05] M. Mambelli and al. Atlas data challenge production on Grid3, 2005. 24

[Mar05] L. Marchal, Y. Yang, H. Casanova, and Y. Robert. A realistic network/application model for scheduling divisible loads on large-scale platforms. In *Int'l. Parallel & Distributed Processing Symposium (IPDPS)*. IEEE Computer Society, 2005. 9

[Mat98] M. Matsumoto and T. Nishimura. Mersenne Twister: A 623-dimensionally equidistributed uniform pseudorandom number generator. *ACM Trans on Modeling and Computer Simulation*, vol. 8(1):pp. 3–30, 1998. 78

[Med01] A. Medina, A. Lakhina, I. Matta, and J. W. Byers. Brite: An approach to universal topology generation. In *MASCOTS*, pp. 346–357. IEEE Computer Society, 2001. 68

[Med05] E. Medernach. Workload analysis of a cluster in a grid environment. In D. G. Feitelson, E. Frachtenberg, L. Rudolph, and U. Schwiegelshohn, editors, *Int'l. Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP)*, vol. 3834 of *Lecture Notes in Computer Science*, pp. 36–61. Springer, 2005. 37, 69, 70

[Men03] D. A. Menasce, V. Almeida, R. H. Riedi, F. Ribeiro, R. C. Fonseca, and W. M. Jr. A hierarchical and multiscale approach to analyze e-business workloads. *Perform Eval*, vol. 54(1):pp. 33–57, 2003. 40

[Met01] C. Metz. On the wire: Interconnecting isp networks. *IEEE Internet Computing*, vol. 5(2):pp. 74–, 2001. 151

[Mil03] D. S. Milojicic, V. Kalogeraki, R. Lukose, K. Nagaraja, J. Pruyne, B. Richard, S. Rollins, and Z. Xu. Peer-to-peer computing. HP Tec.Rep. HPL-2002-57R1, Hewlett-Packard, Jul 2003. 125

[MOA07] MOAT Team. The NLANR Measurement and Network Analysis Group, Aug 2007. 33

[Moh05a] H. Mohamed and D. Epema. The design and implementation of the KOALA co-allocating grid scheduler. In P. M. A. Sloot, A. G. Hoekstra, T. Priol, A. Reinefeld, and M. Bubak, editors, *European Grid Conference (EGC)*, vol. 3470 of *Lecture Notes in Computer Science*, pp. 640–650. Springer, 2005. 15, 23, 27

[Moh05b] H. H. Mohamed and D. H. J. Epema. Experiences with the Koala co-allocating scheduler in multiclusters. In *IEEE/ACM Intl. Symp. on Cluster Computing and the Grid (CCGrid)*, pp. 784–791. IEEE Computer Society, 2005. 15, 72, 77, 78, 82, 83, 102, 109, 112, 131

[Moh07] H. Mohamed and D. Epema. KOALA: A Co-Allocating Grid Scheduler. *Concurrency and Computation: Practice and Experience*, 2007. (in print). 87

[Mu'01] A. W. Mu'alem and D. G. Feitelson. Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with backfilling. *IEEE Trans Parallel & Distributed Syst*, vol. 12(6):pp. 529–543, Jun 2001. 31, 68, 139, 158

[Nae08] V. Nae, A. Iosup, S. Podlipnig, R. Prodan, D. Epema, and T. Fahringer. Efficient management of data center resources for massively multiplayer online games. In *ACM/IEEE Conference on High Performance Networking and Computing (SC)*. ACM Press, 2008. (accepted). 158

[Nel84] L. S. Nelson. Technical aids. *Journal of Quality Technology*, vol. 16(4):pp. 238–239, Oct 1984. 80

[New03a] H. Newman, M. H. Ellisman, and J. A. Orcutt. Data-intensive e-science frontier research. *Commun ACM*, vol. 46(11):pp. 68–77, 2003. 1

[New03b] H. B. Newman, I. C. Legrand, P. Galvez, R. Voicu, and C. Cirstoiu. Monalisa : A distributed monitoring service architecture. In *Computing in High Energy and Nuclear Physics (CHEP03)*. 2003. 30, 116

[New05] M. E. J. Newman. Power laws, pareto distributions and zipf's law. *Contemporary Physics*, vol. 46:p. 323, 2005. 40, 62

[Nie05] R. van Nieuwpoort, J. Maassen, G. Wrzesinska, R. F. H. Hofman, C. J. H. Jacobs, T. Kielmann, and H. E. Bal. Ibis: a flexible and efficient java-based grid programming environment. *Concurrency - Practice and Experience*, vol. 17(7-8):pp. 1079–1107, 2005. 11

[Nor94] I. Norros. A storage model with self-similar input. *Queueing Syst*, vol. 16:pp. 387–396, 1994. 37

[NTT07] NTTT Team. Network Tools and Traffic Traces, Aug 2007. 33

[Nur05] D. Nurmi, J. Brevik, and R. Wolski. Modeling machine availability in enterprise and wide-area distributed computing environments. In *11th International Euro-Par Conference (Euro-Par 2005)*, vol. 3648 of *Lecture Notes in Computer Science*, pp. 432–441. Springer, Lisbon, Portugal, Aug 2005. 36

[Nus07] L. Nussbaum and O. Richard. Lightweight emulation to study peer-to-peer systems. *Concurrency and Computation: Practice and Experience - Special Issue on HotP2P 06*, 2007. 92, 93, 94

[Oli00] S. D. Oliner and D. E. Sichel. The resurgence of growth in the late 1990s: Is information technology the story? *Journal of Economic Perspectives*, vol. 14(fall):pp. 3–22, 2000. 1

[Opp03] D. L. Oppenheimer, A. Ganapathi, and D. A. Patterson. Why do internet services fail, and what can be done about it? In *USENIX Symposium on Internet Technologies and Systems*. 2003. 67

[Oso06] T. Osogami and M. Harchol-Balter. Closed form solutions for mapping general distributions to quasi-minimal ph distributions. *Perform Eval*, vol. 63(6):pp. 524–552, 2006. 41, 42

[Ost08] S. Ostermann, A. Iosup, R. Prodan, T. Fahringer, and D. Epema. On the characteristics of grid workflows. In *Proc. of the CoreGRID Workshop on Integrated Research in Grid Computing (CGIW'08)*, pp. 431–442. Apr 2008. Crete, GR. ISBN: 978-960-524-260-2. 13, 116, 117, 118, 158

[Pab07] Pablo Research Group. CADRE: A national facility for I/O characterization and optimization, Aug 2007. 33, 92

[Pav06] A. Pavlo, P. Couvares, R. Gietzel, A. Karp, I. D. Alderman, M. Livny, and C. Bacon. The NMI build & test laboratory: Continuous integration framework for distributed computing software. In *Conference on Systems Administration (LISA)*, pp. 263–273. USENIX, 2006. 92, 93

[Pax95] V. Paxson and S. Floyd. Wide area traffic: the failure of poisson modeling. *IEEE/ACM Trans Netw*, vol. 3(3):pp. 226–244, 1995. 37

[Pax98] V. Paxson, J. Mahdavi, A. Adams, and M. Mathis. An architecture for large-scale internet measurement. *IEEE Communications*, vol. 36(8):pp. 48–54, August 1998. 92, 93

[Per06] K. S. Perumalla. Parallel and distributed simulation: traditional techniques and recent advances. In *WSC*, pp. 84–95. ACM Press, 2006. 95

[Per07a] Performance and Architecture Research Lab. NMSU trace database, Aug 2007. 33

[Per07b] Performance Evaluation Laboratory. BYU trace distribution center, Aug 2007. 33

[Pet96] D. L. Peterson. Data center i/o patterns and power laws. In *Int. CMG Conference*, pp. 1034–1045. Computer Measurement Group, 1996. 40

[Pha03] S. Phatanapherom, P. Uthayopas, and V. Kachitvichyanukul. Fast simulation model for grid scheduling using HyperSim. In *WSC*, pp. 1494–1500. ACM Press, 2003. 106

[Pil02] D. Pilat, F. Lee, and B. van Ark. Production and use of ICT: a sectoral perspective on productivity growth in the OECD area. *OECD Economic Studies*, vol. 35, 2002. 1

[Pou05] J. A. Pouwelse, P. Garbacki, D. H. J. Epema, and H. J. Sips. The bittorrent p2p file-sharing system: Measurements and analysis. In M. Castro and R. van Renesse, editors, *Proc. of the Int'l. Workshop on Peer-to-Peer Systems (IPTPS'05)*, vol. 3640 of *Lecture Notes in Computer Science*, pp. 205–216. Springer Verlag, 2005. 36

[Rai06] I. Raicu, C. Dumitrescu, M. Ripeanu, and I. T. Foster. The design, performance, and use of DiPerF: An automated distributed performance evaluation framework. *J Grid Comput*, vol. 4(3):pp. 287–309, 2006. 92, 93

[Rai07] I. Raicu, Y. Zhao, C. Dumitrescu, I. T. Foster, and M. Wilde. Falkon: a fast and light-weight task execution framework. In B. Verastegui, editor, *ACM/IEEE Conference on High Performance Networking and Computing (SC)*, p. 43. ACM Press, 2007. 154

[Ram98] R. Raman, M. Livny, and M. H. Solomon. Matchmaking: Distributed resource management for high throughput computing. In *HPDC*, pp. 140–150. 1998. 113

[Ram07] A. Ramakrishnan, G. Singh, H. Zhao, E. Deelman, R. Sakellariou, K. Vahi, K. Blackburn, D. Meyers, and M. Samidi. Scheduling data-intensive workflows onto storage-constrained distributed resources. In *CCGRID*, pp. 401–409. IEEE CS, 2007. 106

[Ran02] K. Ranganathan and I. T. Foster. Decoupling computation and data scheduling in distributed data-intensive applications. In *HPDC*, pp. 352–358. IEEE CS, 2002. 95, 106

[Ran05] R. Ranjan, R. Buyya, and A. Harwood. A case for cooperative and incentive-based coupling of distributed clusters. In *CLUSTER*. IEEE CS, 2005. 106, 138, 151, 152

[Ran08a] R. Ranjan, A. Harwood, and R. Buyya. A case for cooperative and incentive-based federation of distributed clusters. *Future Generation Comp Syst*, vol. 24(4):pp. 280–295, 2008. 36

[Ran08b] R. Ranjan, M. Rahman, and R. Buyya. A decentralized and cooperative workflow scheduling algorithm. In *IEEE/ACM Intl. Symp. on Cluster Computing and the Grid (CCGrid)*, pp. 1–8. IEEE Computer Society, 2008. 154

[Ric04] A. Richards, J. Schmidt, P. M. Dew, F. Youhanaie, M. Ford, and N. Geddes. Production quality e-science grid. In *Proceedings of the UK e-Science All Hands Meeting*. Aug 2004. 93

[Ris06] J. Risson and T. Moors. Survey of research towards robust peer-to-peer networks: Search methods. *Computer Networks*, vol. 50(17):pp. 3485–3521, 2006. 125

[RM07] A. J. Rubio-Montero, E. Huedo, R. S. Montero, and I. M. Llorente. Management of virtual machines on globus grids using gridway. In *Int'l. Parallel & Distributed Processing Symposium (IPDPS)*, pp. 1–7. IEEE Computer Society, 2007. 112, 114

[Roo06] J. Roozenburg. *Secure Decentralized Swarm Discovery in Tribler*. Master's thesis, Delft University of Technology, Delft, NL, Nov 2006. 82, 83, 91

[Roo07] B. Rood and M. J. Lewis. Multi-state grid resource availability characterization. In *GRID*, pp. 42–49. IEEE Computer Society, 2007. 67

[Ros65] R. F. Rosin. Determining a computing center environment. *Commun ACM*, vol. 8(7):pp. 463–468, 1965. 57, 68

[Rza08] K. Rzadca. *Resource Management Models and Algorithms for Multi-Organizational Grids*. Phd thesis, INPG and PJIIT, Feb 2008. 154

[Sah03] R. K. Sahoo, A. J. Oliner, I. Rish, M. Gupta, J. E. Moreira, S. Ma, R. Vilalta, and A. Sivasubramaniam. Critical event prediction for proactive management in large-scale computer clusters. In *KDD*, pp. 426–435. ACM Press, 2003. 100

[Sah04] R. K. Sahoo, A. Sivasubramaniam, M. S. Squillante, and Y. Zhang. Failure data analysis of a large-scale heterogeneous server environment. In *Int'l. Conference on Dependable Systems and Networks (DSN)*, pp. 772–. IEEE Computer Society, 2004. 35, 67

[Sai03] P. Saiz, P. Buncic, and A. J. Peters. AliEn resource brokers. In *Computing in High Energy and Nuclear Physics*. 2003. Also available as CoRR cs.DC/0306068. 109, 112

[Sar05] R. G. Sargent. Verification and validation of simulation models. In *WSC*, pp. 130–143. ACM Press, 2005. 101, 102

[Sch86] L. D. Schroeder, D. L. Sqoquist, and P. E. Stephan. *Understanding regression analysis*. Sage Publications, 1986. Pp.31–32. 42

[Sch99] U. Schwiegelshohn and R. Yahyapour. Resource allocation and scheduling in metasystems. In *DCM*, vol. 1593 of *LNCS*, pp. 851–860. 1999. 109, 112

[Sch06] B. Schroeder and G. A. Gibson. A large-scale study of failures in high-performance computing systems. In *Int'l. Conference on Dependable Systems and Networks (DSN)*, pp. 249–258. IEEE Computer Society, 2006. 35, 40, 66, 67, 71

[Sch07] B. Schroeder and G. Gibson. The computer failure data repository (CFDR), Aug 2007. 17, 33

[Sga96] J. Sgall. On-line scheduling. In *Online Algorithms*, vol. 1442 of *LNCS*, pp. 196–231. 1996. 136, 158

[Sha03] H. Shan, L. Oliker, and R. Biswas. Job superscheduler architecture and performance in computational grid environments. In *ACM/IEEE Conference on High Performance Networking and Computing (SC)*, p. 44. ACM Press, 2003. 151

[She80] W. A. Shewhart. *Economic Control of Quality of Manufactured Product/50th Anniversary Commemorative Issue*. Amer Society for Quality, 1980. 80

[Sid06] M. Siddiqui, A. Villazón, and T. Fahringer. Grid allocation and reservation - grid capacity planning with negotiation-based advance reservation for optimized qos. In *ACM/IEEE Conference on High Performance Networking and Computing (SC)*, p. 103. ACM Press, 2006. 112, 150

[Sil06] M. Silberstein, G. Kliot, A. Sharov, A. Schuster, and M. Livny. Materializing highly available grids. In *IEEE Int'l. Symp. on High Performance Distributed Computing (HPDC)*, pp. 321–323. IEEE Computer Society, 2006. 123

[Sim04] C. R. Simpson Jr. and G. F. Riley. Neti@home: A distributed approach to collecting end-to-end network performance measurements. In C. Barakat and I. Pratt, editors, *Int'l. Workshop on Passive and Active Network Measurement (PAM)*, vol. 3015 of *Lecture Notes in Computer Science*, pp. 168–174. Springer, 2004. 92, 93

[Sim05] Y. Simmhan, B. Plale, and D. Gannon. A survey of data provenance in e-science. *SIGMOD Record*, vol. 34(3):pp. 31–36, 2005. 72, 76

[Sin05] G. Singh, C. Kesselman, and E. Deelman. Optimizing grid-based workflow execution. *J Grid Comput*, vol. 3(3-4):pp. 201–219, 2005. 118, 158

[Sma04] S. Smallen, C. Olschanowsky, K. Ericson, P. Beckman, and J. M. Schopf. The inca test harness and reporting framework. In *ACM/IEEE Conference on High Performance Networking and Computing (SC)*, p. 55. IEEE Computer Society, 2004. 92, 93

[Smi00] W. Smith, I. T. Foster, and V. E. Taylor. Scheduling with advanced reservations. In *Int'l. Parallel & Distributed Processing Symposium (IPDPS)*, pp. 127–132. IEEE Computer Society, 2000. 21, 138

[Son05] B. Song, C. Ernemann, and R. Yahyapour. User group-based workload analysis and modelling. In *IEEE/ACM Intl. Symp. on Cluster Computing and the Grid (CCGrid)*, pp. 953–961. IEEE Computer Society, 2005. 37, 69, 70

[Son06] O. Sonmez, H. Mohamed, and D. Epema. Communication-aware job placement policies for the koala grid scheduler. In *IEEE Int'l. Conf. on e-Science and Grid Computing (e-Science)*, pp. 79–86. IEEE Computer Society, 2006. 72, 78, 82, 83

[Soo04] T. Soong. *Fundamentals of Probability and Statistics for Engineers*. Wiley, 2004. 38

[SPE00] SPEC Team. Int2k benchmark. SPEC, 2000. [Online] Available: http://www.spec.org/cpu2000/. 10

[SPE08] SPECCPU Team. SPEC CPU2006. Standard Performance Evaluation Corporation, Mar 2008. [Online] Available: http://www.spec.org/cpu2006/. 10

[Str07] C. Stratan, C. Cirstoiu, and A. Iosup. On the accuracy of off-line monitoring information in grids. In *Proc. of the 17th Intl. Conference on Control Systems and Computer Science (CSCS-17)*. 2007. May, Bucharest, Romania. 4, 30

[Sul07] A. Sulistio, G. Poduval, R. Buyya, and C.-K. Tham. On incorporating differentiated levels of network service into GridSim. *FGCS*, vol. 23(4):pp. 606–615, 2007. 106

[Sur06] S. Sur, M. J. Koop, and D. K. Panda. Mpi and communication - high-performance and scalable mpi over infiniband with reduced memory usage: an in-depth performance analysis. In *ACM/IEEE Conference on High Performance Networking and Computing (SC)*, p. 105. ACM Press, 2006. 11

[Tak99] A. Takefusa, S. Matsuoka, H. Nakada, K. Aida, and U. Nagashima. Overview of a performance evaluation system for global computing scheduling algorithms. In *HPDC*. IEEE CS, 1999. 106

[Tal07] D. Talby, D. G. Feitelson, and A. Raveh. A co-plot analysis of logs and models of parallel workloads. *ACM Trans Model Comput Simul*, vol. 17(3), 2007. 56

[Tan93] D. Tang and R. K. Iyer. Dependability measurement and modeling of a multicomputer system. *IEEE Trans Computers*, vol. 42(1):pp. 62–75, 1993. 36, 40, 43, 66, 67, 123

[Tel03] M. Telek and A. Heindl. Matching moments for acyclic discrete and continuous phase-type distributions of second order. *International Journal of Simulation*, vol. 3(3-4):pp. 47–57, 2003. 42

[Tha05a] D. Thain, T. Tannenbaum, and M. Livny. Distributed computing in practice: the Condor experience. *Concurrency - Practice and Experience*, vol. 17(2-4):pp. 323–356, 2005. 10, 24, 36, 112, 113, 133, 140, 150

[Tha05b] D. Thain, T. Tannenbaum, and M. Livny. Distributed computing in practice: the Condor experience. *Concurrency and Computation: Practice and Experience*, vol. 17(2-4):pp. 323–356, 2005. 10, 77, 82

[Tha06] D. Thain, T. Tannenbaum, and M. Livny. How to measure a large open-source distributed system. *Concurrency and Computation: Practice and Experience*, vol. 18(15):pp. 1989–2019, 2006. 84, 113

[The07] The Metrics Project. Globus metrics. Tech.Rep. v1.4, Globus, May 2007. 87, 114

[Tri06] J. E. Triplett and B. P. Bosworth. *The New Economy and Beyond : Past, Present, and Future*, chap. Baumols disease has been cured : IT and multifactor productivity in US services industries, pp. 34–71. Edward Elgar Publishing Ltd., Jan 2006. 1

[Trö07] P. Tröger, H. Rajic, A. Haas, and P. Domagalski. Standardization of an api for distributed resource management systems. In *CCGRID*, pp. 619–626. IEEE Computer Society, 2007. 114

[Tru07] H. L. Truong, S. Dustdar, and T. Fahringer. Performance metrics and ontologies for grid workflows. *Future Generation Comp Syst*, vol. 23(6):pp. 760–772, 2007. 76

[Tsa05] D. Tsafrir, Y. Etsion, and D. G. Feitelson. Modeling user runtime estimates. In *Int'l. Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP), Revised Selected Papers*, vol. 3834 of *Lecture Notes in Computer Science*, pp. 1–35. Springer, 2005. 141

[Tso05] G. Tsouloupas and M. D. Dikaiakos. GridBench: A workbench for grid benchmarking. In P. M. A. Sloot, A. G. Hoekstra, T. Priol, A. Reinefeld, and M. Bubak, editors, *EGC*, vol. 3470 of *LNCS*, pp. 211–225. Springer, 2005. 92, 93

[Van02] R. F. Van Der Wijngaart and M. Frumkin. NAS Grid Benchmarks version 1.0. Tech.Rep. NAS-002-005, NASA, 2002. 92, 93

[Wan85] Y.-T. Wang and R. J. T. Morris. Load sharing in distributed systems. *IEEE Trans Computers*, vol. 34(3):pp. 204–217, 1985. 9, 151

[Wax88] B. W. Waxman. Routing of multipoint connections. *IEEE J Sel Areas Commun*, vol. 6:pp. 1617–1622, 1988. 68

[Wie07] A. Wierman. Fairness and classifications. *SIGMETRICS Performance Evaluation Review*, vol. 34(4):pp. 4–12, 2007. 157

[Wil00] W. Willinger. The discovery of self-similar traffic. In G. Haring, C. Lindemann, and M. Reiser, editors, *Performance Evaluation: Origins and Directions*, vol. 1769 of *Lecture Notes in Computer Science*, pp. 513–527. Springer, 2000. 37

[Win96] K. Windisch, V. Lo, R. Moore, D. Feitelson, and B. Nitzberg. A comparison of workload traces from two production parallel machines. In *6th Symp. Frontiers Massively Parallel Comput.*, pp. 319–326. Oct 1996. 31, 88

[Wrz05] G. Wrzesinska, R. van Nieuwpoort, J. Maassen, and H. E. Bal. Fault-tolerance, malleability and migration for divide-and-conquer applications on the grid. In *Int'l. Parallel & Distributed Processing Symposium (IPDPS)*. IEEE Computer Society, April 2005. 11, 23, 58, 75

[Wu01] X. Wu, V. E. Taylor, J. Geisler, X. Li, Z. Lan, R. L. Stevens, M. Hereld, and I. R. Judson. Design and development of the Prophesy performance database for distributed scientific applications. In *PPSC*. 2001. 92

[Yan94] T. Yang and A. Gerasoulis. Dsc: Scheduling parallel tasks on an unbounded number of processors. *IEEE Trans Parallel Distrib Syst*, vol. 5(9):pp. 951–967, 1994. 117

[Yan06] C.-W. Yang, A. Wierman, S. Shakkottai, and M. Harchol-Balter. Tail asymptotics for policies favoring short jobs in a many-flows regime. In R. A. Marie, P. B. Key, and E. Smirni, editors, *SIGMETRICS/Performance*, pp. 97–108. ACM, 2006. 158

[Yeo06] J. Yeo, D. Kotz, and T. Henderson. CRAWDAD: a community resource for archiving wireless data at dartmouth. *SIGCOMM Comput Commun Rev*, vol. 36(2):pp. 21–22, 2006. 17, 33

[Yu05] J. Yu and R. Buyya. A taxonomy of scientific workflow systems for grid computing. *ACM SIGMOD Rec*, vol. 34(3):pp. 44–49, 2005. 89

[Yua08] Y. Yuan, Y. Wu, G. Yang, and W. Zheng. Adaptive hybrid model for long term load prediction in computational grid. In *IEEE/ACM Intl. Symp. on Cluster Computing and the Grid (CCGrid)*, pp. 340–347. IEEE Computer Society, 2008. 154

[Zha04] Y. Zhang, M. S. Squillante, A. Sivasubramaniam, and R. K. Sahoo. Performance implications of failures in large-scale cluster scheduling. In *JSSPP*, vol. 3277 of *LNCS*, pp. 233–252. Springer-Verlag, 2004. 36, 43, 66, 67, 100

Validation of the Random Numbers Use

We have used in this thesis several synthetic workloads, such as those used for testing real systems in Chapters 5 and 7, and for simulations in Chapters 6 and 8. We demonstrate in this appendix that their use does not bias the reported results.

The synthetic workloads used in this work represent a stochastic process, that is, the job arrival process evolves according to probability distributions. To generate the workloads, random numbers are sampled from the directing probability distributions. Thus, the experiments that use synthetic workloads are based on the assumption that the sampled numbers are indeed random. In turn, this means that the random numbers used in this work are produced by good random number generators (RNGs), that is, RNGs that pass the standard DieHarder randomness test[1].

Our experiments do not require truly random numbers; instead, deterministic, pseudo-random number generators (PRNGs) are used, allowing experiments to be re-run for testing and verification purposes. A PRNG is an algorithm that (i) for every input generates a sequence of numbers $X = (x_1, x_2, ..., x_n)$ such that X has the properties of a random sequence of numbers, and (ii) always generates the same X_I for an input I. The latter requirement distinguishes the PRNG class from other RNGs.

There are many types of PRNGs[2][3][4][5][6][7][8][9]: linear congruential generators, lagged Fibonacci generators, linear feedback shift registers and generalized feedback shift registers; for more we refer to[10]. Table 1 shows the characteristics of seven common PRNGs. The values for the time needed to generate one million random numbers for the seven PRNGs are obtained using the CRNG Python implementation[11] on an Intel(R) Xeon(TM) CPU 2.40GHz, 2GB RAM, OS Linux 2.6, gcc 4.0.2. We have selected the MT19937 variant of the Mersenne Twister pseudo-random generator for its excellent random and good speed properties. The Mersenne Twister is a PRNG based on a generalized feedback shift regis-

[1] BROWN, R. The DieHarder RNG tests, Jul 2007. http://www.phy.duke.edu/~rgb/General/dieharder.php

[2] MATSUMOTO, M., and NISHIMURA, T. Mersenne Twister: A 623-dimensionally equidistributed uniform pseudo-random number generator. ACM Trans. on Modeling and Computer Simulation 8, 1 (1998), 3–30.

[3] LUESCHER, M. A portable high-quality random number generator for lattice field theory simulations. Computer Physics Communications 79 (1994), 100–110.

[4] KNUTH, D. E. The Art of Computer Programming: Seminumerical Algorithms, third ed. Addison Wesley, 1997.

[5] L'ECUYER, P. Good parameters and implementations for combined multiple recursive random number generators. Operations Research 47 (1999), 159–164.

[6] L'ECUYER, P. Maximally equidistributed combined Tausworthe generators. Mathematics of Computation 65 (1996), 203–213.

[7] L'ECUYER, P. Efficient and portable combined random number generators. Communications of the ACM 31 (1988), 742–749+774.

[8] WICHMANN, B., and HILL, I. Algorithm AS 183. an efficient and portable pseudo-random number generator. Applied Statistics 31 (1982), 188–190.

[9] PARK, S. K., and MILLER, K. W. Random number generators: Good ones are hard to find. Communications of the ACM 31 (1988), 1192–1201.

[10] MENEZES, A. J., VAN OORSCHOT, P. C., and VANSTONE, S. A. Handbook of Applied Cryptography. CRC Press, 1997.

[11] KRAULIS, P. Random-number generators (RNGs) implemented as Python extension types coded in C, Jul 2007.

Table 1: *Summary of the characteristics and the performance of seven PRNGs. The Period is an intrinsic characteristic of the algorithm. The Work Space is the number of data items that need to be present when generating the next random number. The Time is the time required to generate one million random numbers on a baseline computer (see text).*

Metric	MT19937 [2]	Ranlux [34]	MRG32k3a [5]	Taus88 [6]	LEcuyer [7]	W&H [8]	P&M [9]
Period	$\sim 10^{6000}$	$\sim 10^{171}$	$\sim 10^{57}$	$\sim 10^{26}$	$\sim 10^{18}$	$\sim 10^{12}$	$\sim 10^{9}$
Work Space	624	1000	-	3	-	-	-
Time [s]	26.32	2.80	8.93	29.85	12.90	5.46	20.20

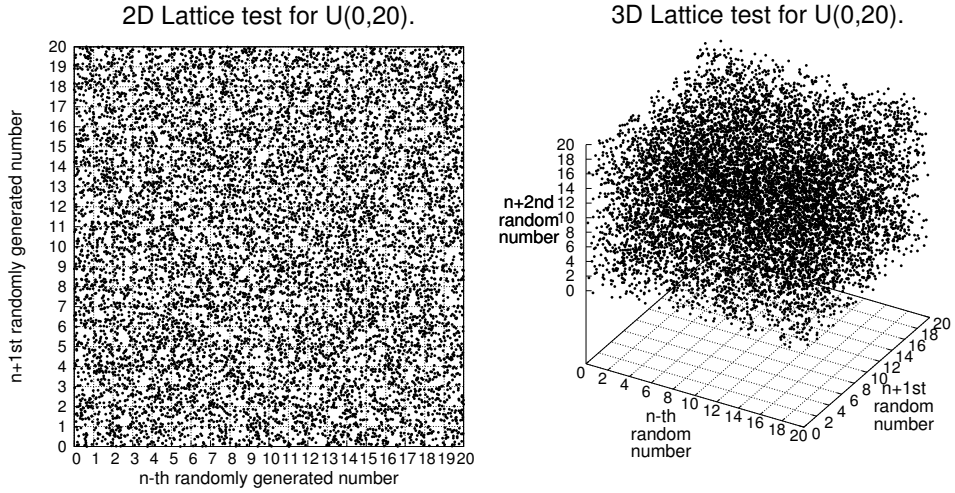


Figure 1: *The results of the 2D and 3D lattice tests.*

ter. The MT19937 variant has a period of $2^{19937}-1$, and 623-dimension equi-distribution up to 32 bits of accuracy[2]. These properties make MT19937 the current de-facto standard for statistical simulation in engineering[10].

There are many tests of the properties of (pseudo-)random number generators,9,10[4]. Tests of randomness usually assess the probability of a sequence of numbers to be generated from a specific random distribution. The MT19937 implementation used for all our experiments[11] passes the standard quantitative statistical distribution test, DieHarder[1]. We have also performed independently various tests, including a graphical CDF matching test, and a graphical lattice test; our tests confirm that MT19937 is a good PRNG. Below we describe one of our tests.

The graphical lattice test assesses the (lack of) auto-correlation in a sequence of randomly generated numbers. For this test, a random number sequence of small size (e.g., 10^2) is generated. Then, the n^{th} and the $n+1^{st}$ random number in each sequence are taken as the coordinates of a point in a 2-dimensional space, and the the n^{th}, $n+1^{st}$, and the $n+2^{nd}$ random number in each sequence are taken as the coordinates of a point in a 3-dimensional space. The resulting 2D and 3D graphs show if the consecutive numbers in the sequence are equally-spaced in the $[0,1)^2$ and the $[0,1)^3$ space, respectively. Figure 1 shows the resulting 2D and 3D graphs of the graphical lattice tests. The MT19937 implementation behaves as a good PRNG: the points are evenly distributed across the space, meaning that the auto-correlation between consecutive numbers is low.

Summary

A Framework for the Study of Grid Inter-Operation Mechanisms

The study of the history of computing infrastructures reveals an integration trend. For example, the explosive growth of the Internet in the 1990s was the result of an integration process started in the 1960s with the emerging networks of computers. By using the Internet, millions of users were capable of accessing information anytime and anywhere, much like other daily utilities such as water, electricity, and telephone. However, an important category of users remained under-served: the users with large computational and storage requirements, e.g., the scientists, the companies that focus on data analysis, and the governmental departments that manage the interaction between the state and the population (such as census, tax, and public health). Thus, in the mid-1990s, the vision of the Grid as a universal computing utility was formulated. The main benefits promised by the Grid are similar to those of other integration efforts: extended and optimized service of the integrated network, and significant reductions of maintenance and operation costs through sharing and better scheduling.

While the universal Grid has yet to be developed, large-scale distributed computing infrastructures that provide their users with seamless and secure access to computing resources, individually called Grid parts or simply *grids*, have been built throughout the world—in different countries, for different sciences, and both for production work and for computer-science research. At the same time, the main technological alternatives to grids, that is, supercomputers and large clusters, have evolved into much larger, scalable, and reliable systems. Thus, the integration of existing grids into larger infrastructures and finally into The Grid is key in keeping the grid vision attractive for its potential users.

The integration of grids raises a double challenge, one related with the efficient scaling of a distributed computing system, the second associated with the operation of a system across different ownership and administrative domains. Thus, many of the traditional approaches for inter-operating computer systems, such as those based on completely centralized or purely decentralized system architectures, are eliminated from the start. To mark the distinction between the typical problem of integrating smaller components into a larger system and the double challenge of grid integration, we call the latter *the problem of grid inter-operation*. In this thesis we approach the problem of grid inter-operation with two main objectives: to design a comprehensive framework for the study of grid inter-operation mechanisms, and to provide an initial but good solution for this problem.

Our framework provides both the theoretical support and the tools for finding new and improved solutions for this problem. The tools are assembled into a *research toolbox* for the study of grid inter-operation mechanisms. This research toolbox addresses two problems that have hampered the grid community in the past decade: the lack of knowledge about the workloads and resources of real grids, and the lack of tools for grid simulation and performance evaluation in real environments. Research using unrealistic characteristics or characteristics that are specific to other types of environments is being limited in scope and applicability, and may even miss the problems that are specific to grids. Thus, real data and realistic models of grid workloads and resources are critical for designing efficient and scalable architectures. Using for simulation and for performance evaluation in real environments

tools that have not been adapted to the requirements of grids leads to slower progress and to results that are difficult to compare. Thus, tools adapted to grids and aimed at producing results that can be shared with other researchers are needed.

The contents of this thesis is split into four logical parts: the introduction, a toolbox for grid inter-operation research, a method for grid inter-operation, and the conclusion.

We begin the thesis with an introduction to the problem of grid inter-operation that focuses on the challenges of grid inter-operation addressed by this thesis. In Chapter 1 we also present an overview of the framework for the study of grid inter-operation mechanisms introduced in this thesis. In Chapter 2 we introduce a basic model for grid inter-operation. This model, required to understand the remainder of the thesis, defines the components of a grid system, the types of applications that can be found in a grid, the system users, and the grid job execution model.

The toolbox for grid inter-operation research is described in Chapters 3, 4, 5, and 6, which we describe in turn. In Chapter 3 we present the Grid Workloads Archive (GWA). We design the GWA with a focus on building a grid workload data repository, and on establishing a community center around the archived data. One of the important design achievements is the formulation of a grid workload format for storing job-level information that can be extended for higher-level information such as co-allocated jobs or resource reservations. We develop a comprehensive set of tools for collecting, processing, and using grid workloads. To make the GWA accessible by non-expert users, we devise a mechanism for automated trace ranking and selection. So far, the GWA contains traces from nine well-known grid environments, with a total content of more than 2,000 users submitting more than 7 million jobs over a period of over 13 operational years, and with working environments spanning over 130 sites comprising 10,000 resources.

In Chapter 4 we describe the extension of the basic model for grid environments into a comprehensive model for (multi-)grids. By analyzing real data such as long-term system traces of real grids, we find that grid resources exhibit a highly dynamic availability both over the course of single days and over whole years. We also find that grid workloads are very different from the workloads of other related systems such as parallel production environments and distributed web servers. Based on the results of this analysis, we design and validate a comprehensive model for grid resource dynamics and evolution, and for grid workloads that include parallel jobs and/or bags-of-tasks.

In Chapter 5 we introduce the GrenchMark testing framework. The main focus of this framework is on testing large-scale distributed computing systems with synthetically generated yet realistic workloads. We test and validate our reference implementation of the GrenchMark framework, and show that GrenchMark has been successful in testing real multi-cluster grids and pools of resources. The experimental results show that a grid testing tool focusing on realistic workloads can indeed be used to assess important characteristics of real systems that are otherwise not available, such as scalability limits, overheads, and reliability.

To conclude the presentation of our grid research toolbox, in Chapter 6 we introduce the DGSim grid simulation framework. The main focus of this framework is on facilitating repeated simulations of multi-cluster and multi-grid environments under realistic workload. We test and validate our reference implementation of the DGSim framework, and show that DGSim has been successful as the simulation tool for several design space exploration studies of grid settings that are larger than the previous state-of-the-art.

The method for grid inter-operation and a solution for the grid inter-operation problem are described in Chapters 7 and 8, which we describe in turn. In Chapter 7 we study the existing alternatives for grid inter-operation, and introduce a novel architecture for grid inter-operation. We classify real grid systems according to their architectural and operational components. The practical limitations

of the centralized grid inter-operation approaches are evaluated in a real environment. These two preliminary steps allow us to assess the grid inter-operation ability of existing grid resource management systems; we find that this ability is limited. Thus, we introduce a novel architecture for grid inter-operation with a better potential of fulfilling the requirements of grid inter-operation. The architecture is a hybrid between hierarchical and purely decentralized architectures. The set of architectures investigated here provides a comprehensive architectural space for the problem of grid inter-operation.

In Chapter 8 we introduce a novel approach for grid inter-operation, Delegated MatchMaking. Our approach, which couples the hybrid architecture introduced in the previous chapter with a novel inter-operation mechanism, is compared with five alternatives through trace-based simulations, and is found to deliver the best performance especially when the system is heavily loaded. While many other mechanisms can be designed in the future, our experiments prove that the Delegated MatchMaking approach already is a good solution for the problem of grid inter-operation. Our experiments also demonstrate that the inter-operation of existing grids can lead to significant performance gains in comparison with leaving them operate independently.

At the end of this thesis, Chapter 9 summarizes our main achievements and presents future directions for this work. The direct use of the framework for the study of grid inter-operation mechanisms holds good promise for future research. In particular, "How many clusters are best?" and other related questions about the system structure can find answers under this framework, leading to important contributions to automating system provisioning and administration. With extensions, our framework can be used to investigate important classes of resource management problems, such as mechanisms and incentives for more system decentralization, scheduling for specific classes of applications or scheduling under less strict information availability assumptions, and guarantees for Quality-of-Service for commercial workloads. We have already taken initial steps in several of these directions.

Samenvatting

Een Raamwerk voor de Studie van Grid Inter-Operatiemechanismen

De bestudering van de geschiedenis van computerinfrastructuren laat een integratietrend zien. Zo was de explosieve groei van het Internet in de jaren negentig van de vorige eeuw het eindresultaat van een integratieproces dat begon in de jaren zestig met het ontstaan van computernetwerken. Via het Internet konden miljoenen gebruikers altijd en overal toegang krijgen tot informatie als ware het een nutsvoorziening zoals water, elektriciteit, en telefoon. Evenwel, een belangrijke categorie van gebruikers werden onvoldoende bediend: de gebruikers met grote reken- en opslageisen, waaronder wetenschappers, bedrijven die zich concentreren op data-analyse, en overheidsinstellingen die de interactie beheren tussen de overheid en de bevolking (zoals op het gebied van volkstellingen, belastingen, en volksgezondheid). Als gevolg daarvan werd in het midden van de jaren negentig de visie van het Grid als een universele computervoorziening geformuleerd. De belangrijkste beloften van het Grid zijn vergelijkbaar met die van andere integratie-inspanningen: uitgebreide en geoptimaliseerde dienstverlening van het geïntegreerde netwerk en een belangrijke reductie van de onderhouds- en exploitatiekosten door gemeenschappelijke gebruik en betere planning.

Terwijl het universele Grid nog steeds moet worden ontwikkeld, zijn grootschalige gedistribueerde computerinfrastructuren die hun gebruikers naadloze en beveiligde toegang tot IT-middelen bieden, individueel Grid-delen of gewoon *grids* genoemd, overal ter wereld gebouwd—in verschillende landen, voor verschillende wetenschappen, en zowel voor productiewerk als voor informatica-onderzoek. Terzelfder tijd zijn de belangrijkste technologische alternatieven voor de grids, namelijk supercomputers en grote clusters, geëvolueerd tot veel grotere, schaalbare, en betrouwbare systemen. Bijgevolg is de integratie van bestaande grids tot grotere infrastructuren en uiteindelijk tot *het Grid* de sleutel om de gridvisie aantrekkelijk te houden voor zijn potentiële gebruikers.

De integratie van grids levert een dubbele uitdaging op, één gerelateerd aan het efficiënt schalen van gedistribueerde computersystemen, en één gerelateerd aan de werking van een systeem gespreid over verschillende eigendoms- en administratieve domeinen. Derhalve kunnen veel van de traditionele benaderingen voor de samenwerking van computersystemen meteen al vervallen, zoals die welke zijn gebaseerd op volledig gecentraliseerde of zuiver gedecentraliseerde systeemarchitecturen. Om het onderscheid tussen het typische probleem van de integratie van kleinere componenten tot een groter systeem en de dubbele uitdaging van gridintegratie duidelijk te maken, noemen we dit laatste het *grid inter-operatieprobleem*. In dit proefschrift benaderen we dit probleem met twee hoofddoelen: het ontwerp van een algemeen raamwerk voor de studie van grid inter-operatiemechanismen, en het creëren van een eerste maar goede oplossing voor dit probleem.

Ons raamwerk biedt zowel de theoretische ondersteuning als gereedschap voor nieuwe en verbeterde oplossingen voor dit probleem. Dit gereedschap is opgenomen in een gereedsschapskist voor de studie van grid inter-operatiemechanismen. Deze onderzoeksgereedschapskist richt zich op twee problemen die de gridgemeenschap in de afgelopen tien jaren hebben gehinderd: het gebrek aan kennis over de werklasten en de *resources* van echte grids, en het ontbreken van instrumenten voor gridsimulatie en

evaluatie van de prestaties in echte omgevingen. Onderzoek dat gebruik maakt van onrealistische kenmerken of kenmerken die specifiek zijn voor andere soorten omgevingen is beperkt in reikwijdte en toepasbaarheid, en kan zelfs niet relevant zijn voor de specifieke problemen van grids. Derhalve zijn echte data en realistisch modellen van de werklasten en *resources* van grids essentieel voor het ontwerpen van efficiënte en schaalbare architecturen. Het gebruik van instrumenten voor de simulatie en evaluatie van de prestaties in echte omgevingen die niet zijn aangepast aan de eisen van grids leidt tot minder vooruitgang en tot resultaten die moeilijk te vergelijken zijn. Dus zijn tools nodig die zijn aangepast aan grids en gericht op het bereiken van resultaten die met andere onderzoekers kunnen worden gedeeld.

De inhoud van dit proefschrift is verdeeld in vier logische delen: de inleiding, een gereedschapskist voor grid inter-operatie onderzoek, een methode voor grid inter-operatie, en de conclusie.

We beginnen het proefschrift met een inleiding tot het probleem van de grid inter-operatie die zich richt op de uitdagingen die in dit proefschrift aan de orde komen. In Hoofdstuk 1 presenteren we ook een overzicht van het raamwerk voor de studie van grid inter-operatiemechanismen die in dit proefschrift worden geïntroduceerd. In Hoofdstuk 2 introduceren we een basismodel voor grid inter-operatie. Dit model, dat nodig is voor het begrijpen van de rest van het proefschrift, definieert de componenten van een gridsysteem, de soorten toepassingen die aangetroffen kunnen worden in een grid, de gebruikers van het systeem, en het gridjob executiemodel.

De gereedschapskist voor grid inter-operatie onderzoek wordt beschreven in de Hoofdstukken 3, 4, 5, en 6, die we nu ieder op hun beurt beschrijven. In Hoofdstuk 3 presenteren we de Grid Workloads Archive (GWA). We ontwerpen de GWA met een focus op het creëren van een opslagplaats voor gegevens van gridwerklasten, en op het creëren van een gemeenschap rondom deze gearchiveerde gegevens. Een van de belangrijkste resultaten is de formulering van een grid *trace format* voor het opslaan van gegevens op jobniveau dat kan worden uitgebreid met extra informatie zoals over co-allocatie en reserveringen. We hebben een uitgebreide verzameling van hulpmiddelen ontwikkeld voor het verzamelen, verwerken, en gebruiken van grid *traces*. Om de GWA toegankelijk te maken door niet-experts hebben we een mechanisme ontwikkeld voor de automatische rangschikking en selectie van *traces*. Tot dusver bevat de GWA *traces* van negen bekende grids, met een totale hoeveelheid gegevens over meer dan 2.000 gebruikers die meer dan 7 miljoen jobs hebben aangeboden over een periode van meer dan 13 operationele jaren, en met gridomgevingen die in totaal meer dan 130 locaties met 10.000 processoren omvatten.

In Hoofdstuk 4 beschrijven we de uitbreiding van het basismodel voor gridomgevingen tot een compleet model voor (multi-)grids. Door het analyseren van gegevens zoals lange-termijn *traces* van echte grids, komen we tot de bevinding dat grid *resources* een zeer dynamische beschikbaarheid hebben zowel over periodes van enkele dagen als over periodes van hele jaren. Ook wordt duidelijk dat gridwerklasten heel verschillend van de werklasten van verwante systemen, zoals parallelle productie-omgevingen en gedistribueerde webservers. Op basis van de resultaten van dit onderzoek ontwerpen en valideren we een alomvattend model voor de dynamiek en evolutie van grid *resources*, en voor gridwerklasten die parallelle jobs en/of jobs bestaande uit een groot aantal sequentiële taken bevatten.

In Hoofdstuk 5 introduceren wij het GrenchMark testraamwerk. De belangrijkste focus van dit raamwerk ligt op het testen van grootschalige gedistribueerde systemen met synthetisch gegenereerde maar realistische werklasten. We testen en valideren onze referentie-implementatie van GrenchMark, en laten zien dat GrenchMark succesvol is in het testen van echte multi-cluster grids. De experimentele resultaten tonen aan dat een grid testgereedschap gericht op realistische werklasten inderdaad kan worden gebruikt om belangrijke kenmerken van echte systemen, zoals de grenzen aan de schaalbaarheid, overheadkosten, en betrouwbaarheid, te bepalen die niet op een andere manier

verkregen kunnen worden.

Ter afsluiting van de presentatie van onze gereedschapskist voor gridonderzoek introduceren we in Hoofdstuk 6 het DGSim gridsimulatie raamwerk. De belangrijkste focus van dit raamwerk is het vergemakkelijken van herhaalde simulaties van multi-cluster en multi-gridomgevingen onder realistische werklasten. We testen en valideren onze referentie-implementatie van het DGSim raamwerk, en laten zien dat DGSim goed gebruikt kan worden als simulatieomgeving voor verscheidene grotere parameterstudies van grids dan tot nu toe mogelijk was.

De methode voor grid inter-operatie en een oplossing voor het grid inter-operatieprobleem worden beschreven in Hoofdstuk 7 en 8. In Hoofdstuk 7 bestuderen we de bestaande alternatieven voor grid inter-operatie, en voeren we van een nieuwe architectuur in voor grid inter-operatie. We classificeren echte gridsystemen op basis van hun architectuur en operationele componenten, en we evalueren de praktische beperkingen van gecentraliseerde benaderingen voor grid inter-operatie in een echte omgeving. Deze twee voorbereidende stappen stellen ons in staat om het vermogen tot grid inter-operatie van bestaande grid *resource-management* systemen te beoordelen; onze conclusie is dat dit vermogen beperkt is. Derhalve introduceren we een nieuwe architectuur voor grid inter-operatie met een beter potentieel om aan de eisen van grid inter-operatie te voldoen. De architectuur is een hybride tussen hiërarchische en zuiver gedecentraliseerde architecturen. De verzamelingen van architecturen die we hier onderzoeken is representatief voor de mogelijke architecturen voor het probleem van grid inter-operatie.

In Hoofdstuk 8 introduceren we een nieuwe benadering voor grid inter-operatie, *Delegated MatchMaking*. Onze aanpak, die de hybride architectuur geïntroduceerd in het vorige hoofdstuk koppelt met een nieuw *inter-operation* mechanisme, wordt vergeleken met vijf alternatieven via *trace-based* simulaties, en blijkt de beste prestaties te leveren, vooral wanneer het systeem zwaar belast is. Terwijl veel andere mechanismen in de toekomst ontworpen kunnen worden, laten onze experimenten zien dat de benadering met Delegated MatchMaking al een goede oplossing is voor het probleem van grid inter-operatie. Onze experimenten tonen eveneens aan dat de inter-operatie van bestaande grids kan leiden tot aanzienlijke prestatiewinst in vergelijking met het onafhankelijk van elkaar te laten werken van de grids.

Aan het einde van dit proefschrift geven we in Hoofdstuk 9 een overzicht van het verrichte onderzoek en presenteren we toekomstige richtingen voor dit werk. De directe toepassing van het raamwerk voor de studie van grid inter-operatiemechanismen houdt een goede belofte in voor toekomstig onderzoek. In het bijzonder kan de vraag "Hoeveel clusters vormen de beste configuratie?" en andere gerelateerde vragen over de systeemstructuur binnen dit raamwerk beantwoord worden, hetgeen kan leiden tot belangrijke bijdragen aan het automatiseren van het beschikbaar stellen en beheren van systemen. Met extensies kan ons raamwerk worden gebruikt om belangrijke klassen van *resource-management* problemen te onderzoeken, zoals mechanismen en prikkels voor meer decentralisatie van systemen, *scheduling* van specifieke klassen van applicaties of onder minder stricte veronderstellingen met betrekking tot de beschikbaarheid van informatie, en garanties voor de kwaliteit van de dienstverlening voor commerciële werklasten. We hebben de eerste stappen al gezet in verscheidene van deze richtingen.

Curriculum Vitae

Alexandru Iosup was born on the 12th of June 1980 in Bucharest, Romania. He received in 2003 a B.Sc./Eng. degree from the Politehnica University of Bucharest (UPB), and in 2004 an M.Sc. degree from the same university. He went on to become in 2004 a PhD student with the Faculty of Electrical Engineering, Mathematics, and Computer Science at Delft University of Technology (TU Delft), as a member of the Parallel and Distributed Systems group. In the summer of 2003 and in the spring of 2004 Alexandru Iosup was a visiting researcher with the grid group of Dr. Stephane Vialle at Supelec, Metz, France. In the fall of 2006 he was a visiting researcher with the Condor group of Dr. Miron Livny at the University of Wisconsin-Madison, USA. He was also a visiting researcher for shorter periods of time with Dr. Ramin Yahyapour (University of Dortmund, Germany), with Dr. Thomas Fahringer (University of Innsbruck, Austria), and with Dr. Nicolae Ţăpuş (UPB, Romania).

Alexandru Iosup's research interests are in the areas of parallel and distributed computing, with a focus on grids, clouds, and peer-to-peer systems and their application to e-Science and commercial workloads. He is the founder of the Grid Workloads Archive, the largest archive for workload traces of grid computing environments. He is a member of the team that performed the largest BitTorrent measurements and analysis to date. His work on grids was awarded the third place at the ACM Student Research Competition 2007, was nominated for a best-paper award at the ACM SuperComputing Conference 2007, and was invited for the special journal issue with the best papers of the EuroPar Conference 2008. He was the co-recipient of the IEEE P2P 2006 best-paper award for work on peer-to-peer systems. He received a Werner von Siemens Award in 2004 for his M.Sc. project.

Alexandru Iosup worked between 2001 and 2004 as a part-time software engineer, work that led to the creation of award-winning shareware games, and of solutions for the on-line monitoring of power transformers. Between 2002 and 2008 he was involved in various teaching activities, including the supervision of M.Sc. students. Throughout this period he has been an active promoter of games as an educational medium for engineering and technical material; as a result, several courses at both UPB and TU Delft now use educational material based on games. Alexandru Iosup was involved in the research community as conference program committee member, peer-reviewer, and scientific event organizer.

Alexandru Iosup's work attracted until December 2008 over 200 citations (Google Scholar), with an h-index of 8 (Publish-or-Perish). A complete list of Alexandru Iosup's publications can be found at: http://www.st.ewi.tudelft.nl/~iosup/; sixteen selected publications are listed below.

International (refereed) journals

1. A. Iosup, O. Sonmez, D. Epema, DGSim: A Simulation Framework for Comparing Grid and Cloud Resource Management Architectures, Concurrency and Computation: Practice and Experience, Elsevier, 2008. (invited)
2. A. Iosup, D.H.J.Epema, T. Tannenbaum, M. Farrellee, and M. Livny, Inter-Operating Grids through Delegated MatchMaking, In the Journal of Scientific Programming, Special Edition on Best Paper Award at SuperComputing 2007, IOS Press, 2008, vol. 16(2-3), pp. 233-253.
3. J.A. Pouwelse, P. Garbacki, J. Wang, A. Bakker, J. Yang, A. Iosup, D.H.J. Epema, M. Reinders, M. van Steen, and H.J. Sips, Tribler: A social-based peer-to-peer system, Concurrency and

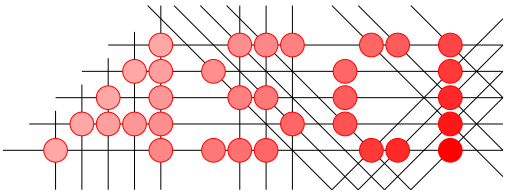
Computation: Practice and Experience, Elsevier, 2008, vol. 20(2), pp. 127–138.
(over 60 citations)

4. A. Iosup, H. Li, M. Jan, S. Anoep, C. Dumitrescu, L. Wolters, and D.H.J. Epema, The Grid Workloads Archive, Future Gen. Comp. Sys., Elsevier, 2008, vol. 24(7), pp. 672-686.

International (refereed) conferences

1. V. Nae, A. Iosup, S. Podlipnig, R. Prodan, D.H.J. Epema, and T. Fahringer (2008). Efficient Management of Data Center Resources for Massively Multiplayer Online Games. In ACM/IEEE Conference on High Performance Networking and Computing (SuperComputing 2008).
2. A. Iosup, O.O. Sonmez, S. Anoep, D.H.J. Epema (2008). The Performance of Bags-of-Tasks in Large-Scale Distributed Systems. In the International Symposium on High-Performance Distributed Computing (HPDC-17 2008), 23-27 June 2008, Boston, MA, USA, pp. 97-108.
3. C. Stratan, A. Iosup, and D.H.J. Epema (2008). A Performance Study of Grid Workflow Engines. In the IEEE/ACM International Conference on Grid (Grid2008), September 29th - October 1st, 2008, Tsukuba, Japan, pp. 25-32.
4. A. Iosup, O.O. Sonmez, and D.H.J. Epema (2008). DGSim: Comparing Grid Resource Management Architectures through Trace-Based Simulation. In the Int'l. Euro-Par Conference on Parallel Processing (Euro-Par 2008) 2008, pp. 13-25. Lecture Notes in Comp. Sci. 5168.
5. A. Iosup, D.H.J. Epema, T. Tannenbaum, M. Farallee, and M. Livny (2007). Inter-Operating Grids through Delegated Matchmaking. In ACM/IEEE Conference on High Performance Networking and Computing (SuperComputing 2007). pp.13-26. (over 5 citations)
6. A. Iosup and M. Jan and O.O. Sonmez and D.H.J. Epema (2007). On the Dynamic Resource Availability in Grids. In the IEEE/ACM Int'l Conference on Grid Computing (Grid2007), pp. 26-33. (over 15 citations)
7. A. Iosup, M. Jan, O.O. Sonmez, and D.H.J. Epema (2007). The Characteristics and Performance of Groups of Jobs in Grids. In the 13th Int'l. Euro-Par Conference on Parallel Processing (Euro-Par 2007), pp. 382-393. Lecture Notes in Comp. Sci. 4641. (over 5 citations)
8. A. Iosup, C. Dumitrescu, D.H.J. Epema, H. Li, and L. Wolters (2006). How Are Real Grids Used? The Analysis of Four Grid Traces and its Implications. In the IEEE/ACM International Conference on Grid Computing (Grid2006), pp. 262-269. (over 30 citations)
9. A. Iosup and D.H.J. Epema (2006). GrenchMark: A Framework for Analyzing, Testing, and Comparing Grids. In the IEEE/ACM Intl. Symposium on Cluster Computing and the Grid (CCGrid06), pp. 313-320. (over 20 citations)
10. P. Garbacki, A. Iosup, D.H.J. Epema, and M. van Steen (2006). 2Fast: Collaborative Downloads in P2P Networks (Best Paper Award). In the IEEE International Conference on Peer-to-Peer Computing (P2P 2006), pp. 23-30. (over 10 citations)
11. J. Pouwelse, P. Garbacki, J. Wang, A. Bakker, J. Yang, A. Iosup, D.H.J. Epema, M. Reinders, M. van Steen, and H. Sips, Chitraka: A Social-Based Peer-to-Peer System, In the International Workshop on Peer-to-Peer Systems (IPTPS'06), 27-28 February, 2006, Santa Barbara, CA, USA.
12. A. Iosup, D.H.J. Epema, C. Franke, A. Papaspyrou, L. Schley, B. Song, and R. Yahyapour (2006). On Grid Performance Evaluation using Synthetic Workloads. In the Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP), pp. 232-255. Lecture Notes in Comp. Sci. 4376. (over 5 citations)

Colophon



Advanced School for Computing and Imaging

This work was carried out in the ASCI graduate school.
ASCI dissertation series number **169**.

This manuscript was typeset by the author with the LaTeX 2ε Documentation System on a PC running Microsoft Windows XP, Red Hat Linux, and a GNU/Linux (via Cygwin). The text was edited using WinEdit and TeXnic Center, and compiled using the MikTeX toolkit. The tables were sometimes edited with LaTable, and then copy-pasted into the text editor. The illustrations and graphs were created with gnuplot, Corel Draw, Visio, Dia, and various pre- and post-processing scripts. The body type is 11 point Computer Modern Roman. Chapter and section titles are in various sizes of Adobe Helvetica-Narrow Bold. The monospace typeface used for verbatim text is Adobe Courier.

The cover was produced by Alexandru Iosup between 2007 and 2008. The image on the front cover represents the grid inter-operation as a necessity for the prosperity of the modern world—similar to airplanes—and as a research problem—similar to rocket science. The prosperity is implied by the presence of fruit-laden trees and by the bright colors.

The LaTeX 2ε page formatting is ensured by a modified version of the "It Took Me Years To Write" template by Leo Breebaart[1]. For more information see the page dedicated to this thesis:

http://www.st.ewi.tudelft.nl/~iosup/aiosup_phd_thesis.html

[1] Leo Breebaart, "It Took Me Years To Write" template for Delft University PhD Thesis, 2003. [Online] Available: http://www.kronto.org/thesis/ (checked Nov 2008).