Trace-based Performance Analysis of Scheduling Bags of Tasks in Grids

S.D. Anoep

Delft University of Technology

Trace-based Performance Analysis of Scheduling Bags of Tasks in Grids

Master's Thesis in Computer Science

Parallel and Distributed Systems Group
Faculty of Electrical Engineering, Mathematics, and Computer Science
Delft University of Technology

S.D. Anoep

15th January 2009

Author
S.D. Anoep
Title
Trace-based Performance Analysis of Scheduling Bags of Tasks in Grids
MSc presentation
17th December 2008

Graduation Committee

prof.dr.ir. H.J. Sips (chair)	Delft University of Technology
dr.ir. D.H.J. Epema	Delft University of Technology
A. Iosup, MSc	Delft University of Technology
dr.ir. M. van Gijzen	Delft University of Technology

Abstract

Grid computing promises large scale computing facilities based on distributed systems. Much research has been done on the subject of increasing the performance of grids. We believe that an adequate performance analysis of grids requires knowledge of the workload and the architecture of the grid. Currently, researchers assume that grids are similar to other distributed systems, such as massively parallel computers. However, workloads in grids differ from other distributed systems, because they consist for a significant part of bags-of-tasks.

This research presents a method to model the workload of grids realistically, which enables us to analyze the performance of those systems. We have created a flexible workload model that is specifically tailed for grids. The model explicitly handles bag-of-tasks, which comprise the majority of grid workloads. This workload model has been built using a vast amount of workload trace data from seven real-world grids.

The workload model enables us to conduct a performance analysis in which we analyze the impact of several workload characteristics, task selection and scheduling policies, and resource management architectures on system performance. We use simulations to systematically and realistically investigate the system performance in various scenarios.

This research has resulted in a grid performance analysis toolbox, a software package that allows researchers to analyze, model, and generate workloads of grids. In addition, we have contributed trace data and analysis to the community by means of the Grid Workloads Archive, an on-line archive of trace data analyzed with our toolbox.

Preface

This thesis contains the research I have done for my Master of Science in Computer Science at Delft University of Technology. This research was done at the Parallel and Distributed Systems group and it presents the design and implementation of a trace-based workload model that is used to analyze the performance of scheduling bags of tasks in grids. This work was done in the context of the Virtual Laboratory for E-Science project.

Parts of Chapter 3 to Chapter 6 have been submitted to and accepted by *the International Symposium on High Performance Distributed Computing 2008*, see reference [40]. In addition, the work on the Grid Workloads Archive, as discussed in Chapter 2, has been published in the journal *Future Generation Computer Systems*, see reference [39].

I would like to thank my supervisor Dick Epema for his advice and guidance in this project. I am very thankful to Alexandru Iosup, for the support throughout this project and for showing me how research is performed in the academic world. I would like to thank the people and institutions that supplied the grid workload traces, without which this research would not have been possible. Special thanks goes out to my parents and sister for supporting me throughout my academical career, especially when my self-motivation was nowhere to be found.

Shanny Anoep

Delft, The Netherlands
15th January 2009

Contents

Chapter 1

Introduction

Computational grids have become increasingly popular in a variety of fields, from the academic world with research grids such as DAS-2 [14] and Grid'5000 [31], production grids as Nordugrid [53] and LCG [44] to commercial grids employed by enterprises as Sun [2] and Amazon [1]. The combination of (clusters of) processors into computational grids has proven to be an effective way to supply computational resources to a large group of users, but still has room for improvement concerning the performance of grids.

In this thesis we present an investigation on scheduling bags of tasks in grids, in order to gain a better understanding on the factors that impact the performance of grids. This section gives an introduction to this research. First, in Section 1.1 we define the problem statement of this thesis, the need for performance analysis of scheduling bags of tasks in grids. In Section 1.2 we discuss the research questions we need to answer in order to solve the stated problem. In Section 1.3, we list the technical objectives that are necessary to perform a good analysis of scheduling bags in grids. Finally, in Section 1.5 we outline the structure of this thesis.

1.1 Problem statement

Computational grids consist of large collections of heterogeneous resources, distributed geographically and across administrative domains, causing performance analysis of such a complex and distributed system to be difficult. In order to analyze the performance of the grid, we need to have insight into the hardware architecture of the grid, such as the grid topology, communication links, processor speed, etc. In addition, we need to understand the software, more specifically the grid middleware.

The resource management software that enables the users of a grid to utilize resources is crucial to the performance of the grid. Job schedulers are an important component of grid middleware, as they determine which resources a job is going to use and when these resources are available to the job. Therefore, much research has been done on better scheduling methods, in order to improve the performance of grids.

In order to assess the performance of grids, we need to evaluate the combination of grid hardware architecture and the resource management software. For a performance evaluation, we need to supply input to a system, called the workload, and assess the performance by calculating a number of performance metrics. The workload entails the resource usage of users that execute their applications on the grid.

In order to evaluate the performance of grids, we need to have insight into the workload of computational grids, as the supplied workload has a significant effect on the outcome of the performance evaluation [23]. Therefore, it is necessary to characterize the workload of real grids, so we can use this information in our performance evaluation. By supplying realistic workloads to the system under investigation, we can analyze the performance of grids adequately.

Much research has been done in characterizing the workload of parallel computing environments [11, 42, 49]. This research has been used to evaluate the performance of supercomputers and clusters. Computational grids provide the same type of resources as clusters, but on a larger and more heterogeneous scale. It is not uncommon for grids to consist of a collection of clusters of computing resources. It seems plausible that the research on parallel workloads could be used to do the same for computational grids.

However, we show that the workload for grids is different from parallel workloads, and therefore needs to be evaluated differently from parallel computing environments. Analysis of real-world grid workloads have shown that, unlike parallel supercomputing environments, the workloads of grid systems contain a high fraction of single-node tasks [34]. In addition, a major part of these tasks can be identified as belonging to larger sets of tasks [37, 38], called bags of tasks (BoT) or batches. A common form of BoTs are parameter sweep applications, in which the same application is executed many times, but with different parameters.

The presence of BoTs in grid workloads has an important impact on the performance of the grid. Analysis of BoTs in three traces of grid systems have shown that the BoT submissions account for up to 96% of consumed CPU time [37]. The same analysis indicates that the performance of BoT submissions is worse than invididual task submissions. Since BoTs account for a significant fraction of consumed CPU time of the entire workload, a poor performance of BoT submissions will likely affect the performance of the entire grid negatively.

We believe that BoTs are an important enough feature of grid workloads to explicitly include in performance evaluation of grids. If we can identify the specific characteristics of BoTs that lead to poor performance, it is possible to tailor resource scheduling policies or give recommendations to users how to submit their BoTs, thereby improving the performance of grids from the perspective of the user and the system. The user will be more satisfied, because his application finishes sooner, while the grid resources will be utilized more efficiently.

As grid workloads are different from parallel computing environment workloads, and the workload has a significant effect on the performance metrics, we cannot simply use parallel workloads for the performance analysis of BoTs. We need a workload characterization method that represents real grid workloads, including BoTs. With the availability of grid-like workloads, we can assess the performance of BoTs in various grid architectures and various resource schedulers, in order to identify the problems with BoTs.

This thesis investigates the problem of scheduling BoTs in grids by presenting a performance analysis of grid scheduling algorithms under realistic grid workloads. We create our own workload model to characterize grid workloads, with explicit inclusion of BoTs in our model. We further evaluate the performance of various grid scheduling algorithms under workloads with BoTs, that reflect the workloads in real-world grid systems.

1.2 Research questions

The overall research goal of this thesis is to assess the performance of scheduling BoTs in grids. Therefore, we need to identify the specific characteristic of BoTs that may lead to poor performance, as well as the impact of the grid resource management architecture and scheduling policies. We assume that the performance of BoTs is affected by using different scheduling policies or by changing the characteristics of BoT submissions, such as number of tasks or the runtime of tasks. We also assume that the resource management architecture has a significant impact on the performance of scheduling BoTs in grids.

To achieve our research goal we need a method of testing various grid schedulers on a set of grid architectures under different workloads. Thus we need to answer the following research questions:

How can we characterize the workloads of grids realistically? To evaluate task scheduling policies effectively, we need realistic workloads that can be submitted to a (simulated) grid or to measure its performance. The result of a performance evaluation depends strongly on the used workload [23]. The use of a workload that does not correctly represent the real situation may lead to inaccurate performance measurements.

How can we evaluate grid scheduling policies on different architectures? It is desirable to do a performance analysis, such that the results are applicable to different grid architectures. Since there is no de-facto standard for grid architectures, the results of a performance analysis on one system do not necessarily hold for other systems, especially since grid architecture are large and complex with several hardware and software layers.

What is the performance of grid scheduling policies under different realistic workloads? We also need to able to change the properties of workloads, so that grid schedulers can be evaluated in different scenarios. We also want to be able to change individual aspects of workloads, so we can assess their impact on the performance. For example, we are interested in differences in performance when the sizes of bags are varied, or the performance when the grid system is heavily loaded.

These requirements give reason to create a new grid workload model, based on data from real-world grids, recorded in so-called workload traces. We create a statistical model to capture the characteristics of workloads in real grids. With this synthetic workload model, we can generate a number of workloads with different parameters, which gives us a flexible yet realistic way to evaluate grid schedulers. Our model only includes characteristics that can be deduced from workload traces.

We further use simulations of different grid architectures to evaluate the performance of grid schedulers. Simulations allows us to do the performance analysis in a number of different scenarios in a reasonable amount of time.

1.3 Technical objectives

In order to answer the research questions stated in the previous section, we need to achieve a number of technical objectives. By building new software tools we aim to make it easier for researchers to analyze, model and generate workload data. We also want to simplify the task of executing a performance analysis of different grid architectures under varying workloads. Therefore, we state the following technical objectives that we want to achieve as part of this research:

Workload trace analysis Workload trace data, as recorded by grid middleware, is usually difficult to interpret for humans, because it contains large amounts of raw data in log files or databases. In order to understand the characteristics of the workload we need tools that transform this raw data into human-readable output, in the form of statistical metrics and graphs. With such an analysis administrators of grid systems have more insight in the performance and usage of their system, and researchers can gain a better understanding of workloads.

Availability of grid workload data The research on performance analysis of grid systems depends on the availability of reliable, readily available grid workload data in a standard format. The Parallel Workloads Archive has been important in the field of parallel supercomputers by supplying workload traces in a standard format. We want workload traces of grid systems to be available to other researchers in a similar way, with the addition of detailed human-readable analysis reports.

Modeling workload trace data The creation of a workload model validated by large amounts of workload trace data is computationally expensive and requires knowledge on statistical methods. For this research, we want to have a software tool that allows us to model an arbitrary number of workload traces flexibly and easily. We want to be able to examine different modeling methods for several workload characteristics. With this tool researchers can build a realistic and flexible workload model, tailored to their needs.

Synthetic workload generation For performance analysis of grid systems we are going to use synthetic workload traces, based on a workload model. Therefore we need a software tool that can generate realistic workloads with varying characteristics. For instance, we may be looking for workload traces that will lead to a certain amount of utilization of a given system. Our workload generation tool needs to be able to generate workloads that satisfy such requirements.

Simulation of grids under varying workloads After generating workloads with varying characteristics, we want to analyze the impact of these workloads on different grid architectures. Therefore, we need a software simulator that can simulate workloads on these architectures. This simulator should be easy-to-use, and flexible in terms of adding new architectures and accepting workload input.

1.4 Terminology

This section gives a few definitions to establish the terminology used in this work.

Grids consist of collections of heterogeneous *nodes*, which are computer systems with processors, memory, network interfaces, and local disk space. For simplicity, these nodes can be divided into *computational* and *storage* nodes. Computational nodes are optimized for calculations, while storage nodes offer data access and storage. Nodes can be coupled into *sites*, for example cluster systems. Clusters typically consist of a large number of computational nodes, supplemented with a few storage nodes.

The workload of computational grids is generated by the users that execute applications on the grid. Grid applications are built from a set of individual *tasks* or *tasks*, possibly with dependencies. Tasks specify a program to execute, as well as *resources* and input files needed for completion. For example, resource requirements can be specified in terms of number of nodes, processors speeds and storage space needed. Tasks may belong to a larger set of tasks, called a *bag-of-tasks* (BoT). Users can submit tasks to the resource management software of the grid, which decides when and on which node the task is scheduled for execution, according to the implemented task *scheduling policy*.

We identify two types of grids: computational grids and data grids. In computational grids, the individual nodes together serve as an aggregate computational resource for a wide range of applications. On the other hand, data grids are focused on providing storage and retrieval of data that exceeds single-node storage capacity or needs to be distributed geographically. In this work, we are limit the scope of our research to computational grids.

Computational grids are defined as the hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities [26]. Basically, computational grids provide the architecture to share heterogeneous resource data, compute cycles, or instruments. These resources could be distributed across different locations and administrative domains [27]. The possibility of sharing resources enables applications that exceed the capacity of normal workstations or clusters. Computational grids can also be used for parallel computation, requiring large numbers of processors, more than available at a single physical location.

1.5 Thesis outline

In this thesis we assess the performance of scheduling BoTs in grids with different architectures under a variety of workloads. We have created a workload model, that includes the notion of BoTs and user behaviour. Furthermore, we use simulations to assess the performance of scheduling BoTs in grids with workloads generated from our workload model. The material in this thesis is structured as follows. In Chapter 2 we present the tools that we have created and used in this research. In Chapter 3 we discuss the theory behind our workload model, the characteristics that we include in our model, and our modeling method for these characteristics. In Chapter 4 we present the results of workload modeling using workload traces. In Chapter 5 we present our simulation and scheduling model to be used for our performance analysis. In Chapter 6 we present the results of the simulation experiments. In Chapter 7 we discuss our conclusions and recommendations for future work.

Chapter 2

A Grid Performance Analysis Toolbox

In this work, we want to investigate the performance of grids by investigating workloads, scheduling polices, and resource management architecture. We do this by analyzing and modeling workload data, which is made difficult by the heterogeneity of grid resource management software and architecture in current grid systems. In order to answer our research questions adequately and efficiently, we need to make processing workload data for performance analysis easier. We have created software tools that makes this easier, a toolbox that is the result of achieving the technical objectives stated in Section 1.3.

In this chapter we present our grid performance analysis toolbox, which provides the necessary tools for workload analysis, modeling and generation and simulation of grid systems. This toolbox is developed to analyze workload traces recorded by grid resource managers, to model the workload in these traces and to generate synthetic workloads. Using these tools we have modeled workload traces from real grids, for which the results are presented in Chapter 4. The generated workloads are used in simulations of grid systems, for which the results are presented in Chapter 5.1. This toolbox has also been used to create content for the Grid Workloads Archive and to support the performance analysis of grid architectures under varying workloads. A paper has been published in the journal *Future Generation Computer Systems* describing the Grid Workloads Archive in more detail [39].

The grid performance analysis toolbox consists of four different tools: the trace converter, the report generator, the modeler and the simulator. An overview is given in Figure 2.1. The outline of this chapter is as follows. In Section 2.1 we present the Grid Workloads Archive, for which the toolbox is used to create its content. In Section 2.2, we discuss the our workload trace converter. In Section 2.3 we discuss the workload analysis reporting features. In Section 2.4 we present our workload modeling tool. Finally, in Section 2.5 we discuss DGSim, our grid simulator, which is used in this work to assess the performance impact of batch submissions.

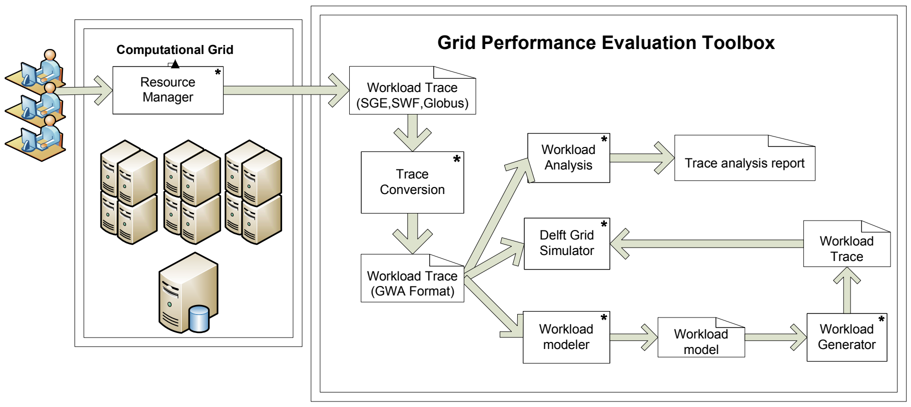


Figure 2.1: Overview of the grid performance analysis toolbox

2.1 The Grid Workloads Archive

Currently little is known about grid workloads, not unlike parallel production environments a few decades ago. We listed as a technical objective in Section 1.3 that we want to increase the availability of grid workload data and analysis reports. The Grid Workloads Archive (GWA) [39] was created from the need for readily available workload data on grid environments for researchers and administrators. At the moment, Grid middleware tools log information about workloads, but due to political issues and the lack of a standard format, it is difficult for grid researchers to get access to and study this data. This is a major issue in grid research, as performance evaluations lack a realistic basis is workload data is not available.

The GWA solves these problems in the form of a freely accessible website that contains grid workload traces, trace analysis reports, software tools for collecting and processing workload traces and serves as a community for grid workload research.

The GWA contains seven workload traces of grid production and experimental environments in a new text-based workload trace format, the Grid Workload Format. This trace format is similar to the well-known Standard Workloads Format (SWF), which is used by the Parallel Workloads Archive [56], but with grid-specific modifications. This format is designed with simplicity for processing and extensibility in mind. The traces are also available as a SQLite database for processing data with SQL queries, instead of text-based file processing.

In addition to the raw workload data, which is suitable for machine-processing, the GWA also presents detailed trace analysis reports on those traces, created with our performance analysis toolbox. For details on these reports, see Section 2.3.

The GWA aims to provide real-world grid workload traces for grid researchers, grid system administrators and grid system designers. The traces vary in grid system size, system architecture, number of users, jobs, etc. Therefore the GWA ranks the traces in several categories, making it easier to select a workload trace.

For researchers, the contents of the GWA provide a better understanding of grid workloads and give a realistic basis for performance analysis of grid middleware or architecture. Several workload traces in the GWA have already been used in analysis of grid workloads [34], and in performance analysis of grid scheduling policies [40].

The workload traces and analysis reports give administrators of grid systems the opportunity to compare the performance of their system with similar systems. They can check whether their systems evolve in a similar fashion, or whether the user behaviour is different. Grid system designers can use the GWA to support their decision for certain middleware, by looking at the performance of other systems.

Our performance analysis toolbox has been used to create a major part of the content on the GWA. Our workload trace converter is used to convert all workload traces to the Grid Workload Format, the analysis report generator was used to generate the human-readable information on the workload traces, which includes modeling results from the workload modeler for a number of traces. These tools are all discussed individually in subsequent paragraphs.

2.2 Workload trace converter

The most important input for our grid performance analysis toolbox are workload traces as provided by resource managers. With different grid architectures being used, the types of resource managers differ from system to system, and also their output formats differ. The workload trace converter was a result of achieving the technical objective of analyzing raw workload data into human readable form, stated in Section 1.3.

Because we do not want to have separate trace handling code for each input format in our toolbox, we have used of a standard workload trace format in all tools, specified as the Grid Workload Format (GWF) [32]. Traces recorded by resource management software have to be converted to this format, which is handled by the workload trace converter.

The workload trace converter can handle traces of grid-level resource managers, cluster-level managers, but also standard trace formats. Currently, we can convert traces from a range of data sources, log files from cluster-level resource managers (e.g. Sub Grid Engine), log files from grid resource managers (e.g. Globus GRAM). Different input formats are usable, as the traces from the Sun Grid Engine resource manager are supplied as simple text files, while the Globus logs are XML-based. In addition, we can also convert traces that are available in the Standard Workloads Format.

The conversion code is easily extendible to other formats. Any trace format that is specified as a text file with one line of information per jobs, can be added to the conversion process with few additions to the code.

The conversion tool also produces a SQLite database, that contains the exact same information as the workload trace. SQLite is a file-based database, which can be queried and updated by means of an API, eliminating the need for a server process. The advantage of having a SQL-compatible database is the relative ease with which analysis of the trace can be performed. With an SQLite database, we can execute SQL queries on the workload, as opposed to writing format-specific text-processing and analysis code.

2.3 Workload analysis report generator

In addition to the workload traces and databases, the performance analysis toolbox also produces human-readable output. For users and administrators of a grid it is interesting to know the workload characteristics of their system. With this information, users can tune their applications and administrators can tune the grid system to get better performance. Unprocessed workload traces contain a lot of raw data, which is difficult to interpret for humans. The workload analysis report generator was a result of achieving the technical objective of analyzing raw workload data into human readable form, stated in Section 1.3.

We have developed a workload analysis tool that generates a detailed report, using only a workload trace and a simple configuration file as input. The workload trace must be supplied in the GWF format, so every trace that can be converted to this format can be analyzed. Traces in other formats can be converted using our workload trace converter, such that they can be analyzed. A simple configuration file is used to define the input and output file location.

The report generator is built from a set of analysis scripts, written in Python. Each script analyzes a different aspect of the workload trace, for instance system utilization, user behaviour or batch characteristics. Each script outputs to text files containing detailed statistical data, such as minimum and maximum values, average, standard deviation for different performance metrics or workload characteristics. These output files are also readable for humans.

The output files of the analysis scripts are then parsed to create a concise yet comprehensive workload analysis report. The workload analysis report summarizes system-wide information, such as utilization, arrival rate or throughput, but also job-level characteristics, such as job runtimes or memory usage. Table 2.1 gives an overview of the calculated metrics. The metrics in the report are visualized with charts and plots, clarified by a textual description. If the workload modeling tool has been used on the same workload trace, the parameters and graphs of fitted distributions are also given in the report.

Currently, the workload analysis report is generated in HTML and LaTeX format. Figure 2.2 shows a screenshot of the report in HTML, as rendered by a web browser. The report generating code has a single interface with multiple back-ends, in this case HTML and LaTeX. This makes it easy to add another output format for the analysis report.

Category	Metrics
System Utilization	Overall system utilization (min,max,avg)
Job arrival rate	Busiest day/week/months (min,max,avg)
Job arrival rate	Arriving Jobs/hour (min,max,avg)
Job characteristics	Number of CPUs per (min,max,avg)
Job characteristics	Runtime (min,max,avg)
Job characteristics	Memory usage (min,max,avg)
Sequential/parallel jobs	Number of sequential/parallel jobs
Sequential/parallel jobs	Consumed CPU time of sequential/parallel jobs
User characteristics	Top 10 users by number of jobs (with sys. util., job char., arrivals)
User characteristics	Top 10 users by consumed CPU time (with sys. util., job char., arrivals)
Group characteristics	Top 10 groups by number of jobs (with sys. util., job char., arrivals)
Group characteristics	Top 10 groups by consumed CPU time (with sys. util., job char., arrivals)
Performance analysis	Number of waiting/running jobs per day (min,max,avg)
Performance analysis	Job throughput per day (min,max,avg)
Performance analysis	Number of completed jobs per day (min,max,avg)
Workload model	Model parameters and goodness-of-fit test results

Table 2.1: Overview of calculated metrics in the workload analysis reports

2.4 Workload modeler

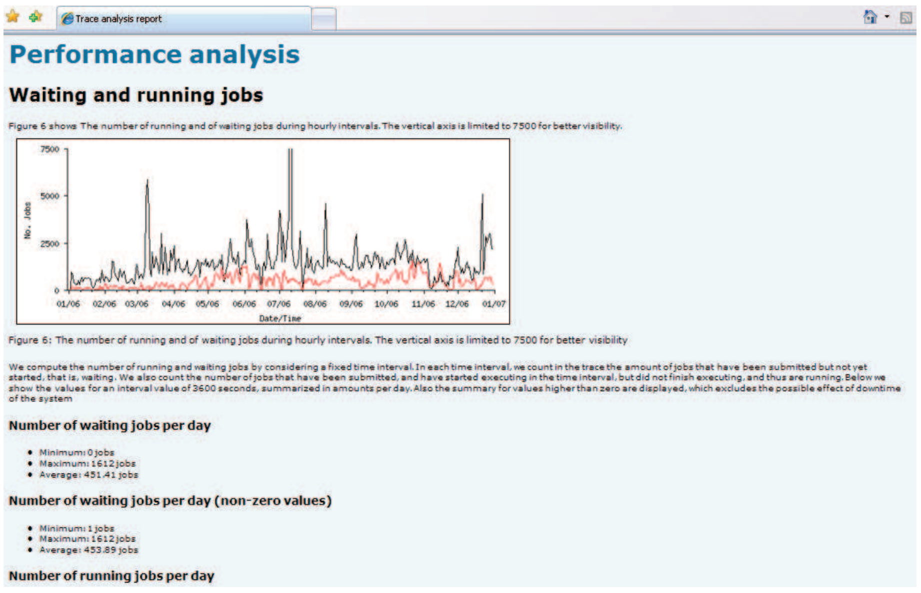
As we stated in our technical objectives in Section 1.3, examining different modeling methods for workload data is difficult and computationally expensive for large traces. We have created the workload modeler tool, with which grid researchers can model grid workload trace data flexibly and easily.

The workload modeler is used to find parameters for our grid workload model, presented in Chapter 3, such that we have a model instance that reflects the characteristics of a given workload trace. With a model instance, we can generate synthetic workloads that are similar to the input trace. This is important for performance analysis, because we need realistic workloads as input for our simulations.

The workload modeler tool is programmed as a combination of Python and R code. The Python code is mainly concerned with preprocessing the workload trace that is supplied as input. The input traces must be supplied in the GWF format. The modeling code is written in the statistical programming language R, which does the computationally intensive work of fitting data to statistical distributions.

The preprocessing code splits the GWF workload trace into different data sets, each representing a different characteristic we wish to model. Examples of characteristics are: job interarrival times, job runtime, requested runtime, batch size. This preprocessing is necessary, because some of the data needs to be transformed before supplying it as input to the modeling code. In addition, by isolating only the useful data, the parsing of input data takes considerably less time. This is important, since workload trace files can be quite large; a trace with jobs recorded over an entire year can contain around 1 Gigabyte of data.

The modeling code implements our chosen procedure of modeling workloads, explained in Section 4.1. The modeling code takes as input a data vector represent-



Performance analysis

Waiting and running jobs

Figure 6 shows The number of running and of waiting jobs during hourly intervals. The vertical axis is limited to 7500 for better visibility.

Figure 6: The number of running and of waiting jobs during hourly intervals. The vertical axis is limited to 7500 for better visibility

We compute the number of running and waiting jobs by considering a fixed time interval. In each time interval, we count in the trace the amount of jobs that have been submitted but not yet started, that is, waiting. We also count the number of jobs that have been submitted, and have started executing in the time interval, but did not finish executing, and thus are running. Below we show the values for an interval value of 3600 seconds, summarized in amounts per day. Also the summary for values higher than zero are displayed, which excludes the possible effect of downtime of the system

Number of waiting jobs per day

- Minimum: 0 jobs
- Maximum: 1612 jobs
- Average: 451.41 jobs

Number of waiting jobs per day (non-zero values)

- Minimum: 1 jobs
- Maximum: 1612 jobs
- Average: 453.89 jobs

Number of running jobs per day

Figure 2.2: A screenshot of the workload analysis report in HTML format as rendered by a web browser

ing some workload characteristic. The input data is transformed logarithmically to reduce fitting time. The input data is fitted to five different heavy-tailed probability distributions: the normal, log-normal, gamma, exponential and Weibull distribution. Subsequently, the goodness-of-fit is assessed using the Kolmogorov-Smirnov test, so we can determine which distribution fits the data best of all. In addition, the workload modeler produces plots of the input data along with the fitted distributions, so we can visually inspect the goodness-of-fit.

Because the preprocessing and modeling code is decoupled, it is easy to add new characteristics to be modeled. Only the preprocessing code needs to be modified to supply the correct data. The modeling code needs a one-dimensional data vector as input, so this part needs no modification to add new characteristics. The modeling code can also be reused for other types of data than workload traces. For example, the workload modeler has also been used to model availability data of grid systems, for which the results are presented in [37].

The workload modeler provides the means to model important characteristics of a workload trace, in a easy-to-use and efficient manner. This tool has been used in this research to model workloads of seven grid systems. The modeling results of workload traces from real grids in Chapter 4 have all been created with this tool.

2.5 DGSim: A Simulator of workloads in multi-cluster systems

The Delft Grid Simulator (DGSim) is created with the goal of simplifying the simulation of multi-grid systems [15, 41]. DGSim focuses on simulation of many grid resource management architectures, on generating and replaying grid workloads, and on helping with simulation management. We have used DGSim to achieve our technical objective of simulating grids in various scenarios, stated in Section 1.3. Current simulation tools lack automated experiment setup and management, as well as realistic workload generation.

DGSim allows us to replay a generated workload on different simulated grid architectures with relative ease. Alternative grid simulators such as SimGrid [46], GangSim [18], and GridSim [6] focus on providing only a low-level simulation framework. Currently DGSim can simulate six grid resource management architectures: independent clusters operated with job-push, independent clusters operated with matchmaking, centralized meta-scheduling with job pull, centralized meta-scheduling with job-push, federated clusters with matchmaking, and inter-operated grids with delegated matchmaking.

With DGSim the simulation of multiple grids with multiple workloads is made easy, thanks to the concept of experiments and scenarios. A scenario is a group of simulations covering a single real-world phenomenon. For instance, a scenario could be the simulation of multiple grid architectures with a given system utilization. An experiment is a set of scenarios, so we can study a family of phenomena, in the previous example it could be a simulation of a number of different values for system utilization. By defining a group of simulations as experiments and scenarios, DGSim provides a user-friendly way to perform large-scale simulations of multi-grid environments.

The simulator can load workload traces in the Grid Workload Format and in the Standard Workloads Format. The simulator includes a workload trace generator, which achieves our technical objective of being able to generate synthetic realistic workloads for simulation, stated in Section 1.3. These workloads can be generated based on our workload model explained in Chapter 3 and the Lublin-Feitelson workload model [49].

We can examine the impact of various workload characteristics by varying the parameters in the workload trace generation. For instance, when the cluster size of the grid is known, we can generate workloads for it with a chosen system utilization. The loaded traces are replayed on the the different architectures, while storing the results in a database for later processing.

DGSim has been initially created by Iosup et al. [15, 41], but has been extended as part of this research. We have added the functionality to run simulations of grids on grids, thereby strongly reducing the turnaround time of running simulations. By separating the simulations in a scenario in individual processes, and submitting these to a grid, we have been able to reduce total simulation time for an experiment from an order of days to and order of hours. In addition, various scheduling and selection policies we want to investigate in our performance analysis have been added to the simulator.

Chapter 3

A Comprehensive Grid Workload Model

For our performance analysis of scheduling BoTs in grids, our first objective is to find a method to characterize workload of grid systems. In this chapter we answer our research question stated in Section 1.2: How can we characterize the workloads of grids realistically? We have chosen to create a new workload model, that explicitly includes BoTs, such that we can evaluate the impact of specific characteristics of grid workloads on the performance of different scheduling policies realistically. With the creation of our workload model, we aim to increase the understanding of grid workloads and offer an extra tool for researchers to generate synthetic realistic workloads.

This chapter presents our workload model that we will use in for our performance analysis of scheduling BoTs in grid systems, and is organized as follows. In Section 3.1 we discuss various techniques for characterizing workloads, and elaborate on our decision to use a trace-based workload model. In Section 3.2 we discuss the requirements for our workload model, namely the desired properties for our model, as well as the granularity of the model. Our workload model can be split up into three levels: the user-level model, the BoT-level model and the task-level model. In Section 3.3 we present a family of workload models, based on a combination of these three levels and we discuss the characteristics that we wish to model in each of these levels. We also motivate which of the workload models we are going to use in our simulations. Parts of this section have been published in a paper, which was submitted to and accepted by the ACM International Symposium on High Performance Distributed Computing 2008 [40]. In Section 3.4, previous research on workload modeling for parallel systems and grids is reviewed, in order to find out how workload models have evolved over time.

3.1 Workload characterization techniques

The workload of a grid system can be defined as the resource demand of the tasks submitted to the system over some period of time. In large scale grid systems, the resource manager (e.g. scheduler) records for each task the user's resource usage requests and related information in a trace or log file (from hereon denoted by traces). The related information typically includes timing information, such as the time the task was submitted, the time the task has started, and when it was completed. Usually, the recorded traces also contain more detailed information about resource usage, for example memory usage or the number of processors. These traces can be used to reconstruct the workload that was put on the grid system, e.g. for accounting purposes.

The correctness of a performance evaluation depends on the representativeness of the supplied workload. Only when the evaluation is performed with the workload that has similar characteristics to the actual workload, we can be confident that the calculated performance metrics will be accurate.

Furthermore, we are often interested whether our results hold for various load conditions, such as high or low system utilization, or for various system configurations, such as the number of nodes in the system. In addition, we are interested in the performance of grids for a extended period of time, in the order of years.

In the following paragraphs we discuss three alternatives for workload characterization: live load, trace-driven simulation and trace-based modeling.

3.1.1 Live load

A first alternative to creating a workload model is experimenting with live load. In this case we evaluate the system with the workload at the time the experiment is performed. An advantage of this approach is that it uses a sample of the real workload. Because it is only a sample, the question arises whether the sample is representative for workload observed for longer periods of time. Another disadvantage is that results of such experiments are difficult to reproduce, and are also inflexible. It is impossible to change characteristics of the workload.

3.1.2 Trace-driven simulation

The second approach is to replay the tasks recorded in a trace. In this case, the tasks are submitted to the simulation environment as they appear in the trace [63]. This method has also been used for web server workload and memory simulations [3, 62].

Replaying traces has the advantage that a representative workload of a given system is used in the simulation, with the ability to reproduce results of the evaluation experiments.

On the other hand, it is still difficult to change characteristics of the workload, in order to generalize the evaluation results to other configurations. In particular, the scaling of workload traces in terms of timing behaviour or number of explicitly rocessors is difficult in lack of a model [20].

3.1.3 Trace-based workload model

The third approach is to create a workload model, that captures the most interesting characteristics of the actual workload. This workload model is then used to generate a stream of tasks, that reflects the workload recorded in the trace. A workload model can be derived from one or more traces by analyzing the data on timing and resource usage, and by fitting statistical distributions on these data [11,42,49].

The use of a workload model has the advantage that the parameters of the model can be adjusted, thereby creating many different scenarios. It is also possible to generate workload for longer periods of time than was recorded in the trace.

We have chosen to characterize the workload using a trace-based model, which proves to be representative, allows for evaluation in different scenarios, and is scalable over time. In the next section, we lay out the process of modeling the workload, based on traces of tasks.

3.2 Model requirements

This section discusses the desired properties of a workload model that will be used for performance analysis. We also discuss the model granularity, or the types of characteristics of a workload we wish to include in our model.

3.2.1 Desired properties

What we seek for in our workload model is flexibility, reproducibility and accuracy. The model should be flexible enough to evaluate a grid system in different situations, for example by varying the utilization or the fraction of tasks belonging to BoTs. For this reason, we wish to keep our model as simple as possible. Using complex mathematical constructs may achieve better accuracy in representing the original workload, but this will make it difficult to change the model.

The model should be reproducible, or we should have the ability to reproduce two indistinguishable workloads from the model. This allows us to reproduce evaluation results at different moments in time, or to verify the results on other systems. By creating statistical models for various characteristics of the workload, we can achieve reproducible results.

Ultimately, the workload model should accurately represent the original workload. This means that a workload generated from the model should have similar, but not necessarily identical, characteristics to the original workload [24,28,36]. For example, if the original workload has a high fraction of serial tasks with long runtimes, this should also be the case with the generated workload. Of course, a trade-off is made between level of accuracy and simplicity of the model.

3.2.2 Model granularity

We aim to create a model that is simple enough to understand and change, yet is comprehensive enough to represent the workload of a real grid system. A workload model can be specified at different levels or granularity. It is possible to create fine-grained workload models that include CPU instruction-level information about the executed tasks. modeling at this level could capture complex characteristics of the individual tasks. However, such a model would require large amounts of computation power in simulations, as the number of instruction runs in the billions per second. Furthermore, information at this level is not available in grid traces.

To achieve a simple and comprehensive workload model, we model essential characteristics of submitted tasks, task parallelism, task runtime and task requested runtime. These characteristics have been modeled for workloads in parallel computers, [11, 42, 49]. In addition, our model also includes the behaviour of BoTs, which are a significant part of grid workloads [37].

Besides the task-level characteristics, we are also interested in user behaviour of the grid system under study. Tasks are submitted by a pool of users, which are organized in groups or virtual organizations. The analysis of grid traces has shown that a small subset of users have a significant impact on the workload [34]. Therefore, we also model the distribution of tasks over users. In this work we also discuss and compare different options for user-level models.

3.3 A family of grid workload models

This section presents a family of grid workload models. One of these workload models is used in simulations, while another is explored to see if it leads to better results. First, we give an overview of our family of models, which is based on a combination of three models. Subsequently, we discuss the task-level, the BoT-level and the user-level model separately.

3.3.1 Overview

Workloads for large grid systems are complex, because they are generated by a large group of users for a wide range of applications. There may be many relations between tasks in the workloads, such as the existence of BoTs, or relations between characteristics of tasks, such as the runtime and number of used processors. Workloads also tend to show cyclic behaviour when it comes to the arrival process, since tasks are submitted by users who are generally less active after working hours and in weekends.

A workload model that wants to capture all the complexities of a real workload, will need many parameters in order to represent everything. Researchers have resorted to phase-type probability distributions, such as hyper-gamma distribution to model task runtimes [49], or more complex mathematical constructs as Markov-modulated Poisson processes to model the arrivals of tasks in grids [48].

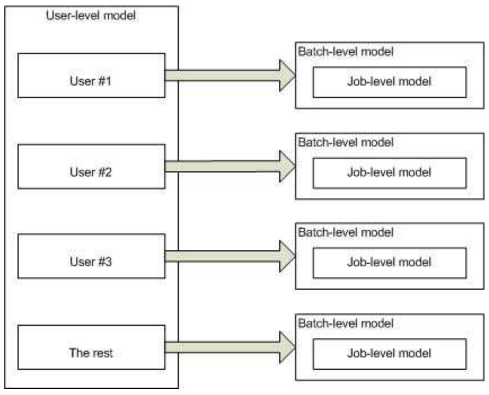


Figure 3.1: An illustration of the explicit user level model with three top users and one general user

The more parameters a workload model has, the more difficult it becomes to use. When complex mathematical constructs are used, it will be more difficult to deduce how to change parameters of a model in order to get a desired workload. However, a over-simplified model may not be able to represent the original workload correctly.

We have opted for an approach in which we can vary the number of parameters of our model. By using knowledge about the submission behaviour of users, which are responsible for creating the workload, and our knowledge on the presence of BoTs, we present a family of workload models.

The family of workload models is defined at three levels: task-level, BoT-level and user-level. The task-level model contains the attributes for individual tasks, such as arrival times, runtime and number of processors. The BoT-level model considers groups of tasks, thereby modeling the number of tasks in a BoT, the runtime variability and task arrivals within a BoT. Finally, the user-level contains the distribution of tasks over users, for both BoTs and single tasks.

This division in to three levels is based on the submission behaviour of users. We consider a grid workload as created by a set of users, who submit a set of BoTs, that are sets of tasks. If we can model the BoT characteristics for each users, and we can model the task characteristics of tasks within a BoT, we can "recreate" the real workload.

Creating the perfect representative workload model would mean modeling the behaviour of each user explicitly, which would result in a large amount of parameters of the workload model, as grids may have hundreds to thousands of unique users. Instead, we choose to only model a set of dominant users separately, and considering the rest as one generic user. Figure 3.1 shows an example of such a model with three users explicitly modeled, and the rest in one general model.

The number of users that will be modeled separately determines the number of parameters of the workload model. In Table 3.1, we summarize the number of parameters that the different variants of our grid workload model could have.

model	# parameters	user-level	BoT-level	task-level
M1	k	-	-	yes
M2	$k+m$	-	yes	yes
M3	$(k+m)+(n+1)$	implicit	yes	yes
M4	$(k+m)*(n+1)$	explicit	yes	yes

Table 3.1: Comparison of workload models, by number of parameters

Suppose we have a task-level model $M1$ with k parameters, a BoT-level model $M2$ with m parameters, and two user-level models $M3$ and $M4$. The implicit user-level model does not model any user separately, it considers all the tasks as originating from one user. The difference between implicit and explicit user-level models is further explained in Section 3.3.4.

The model $M3$ offers the best trade-off between scalability in the number of parameters, and the representativeness of the workload model. Adding a task-level parameter or increasing the number of top users and groups affects model $M3$ the least in terms of number of parameters, compared to model $M4$. Therefore, in simulations we will use the implicit user-level combined with the BoT-level and task-level model.

However, we want to see whether the explicit user-level model would lead to a more representative model, so we will also explore the modeling results for model $M4$.

3.3.2 Task-level model

This section discusses various methods used to model a number of task-level workload characteristics: degree of parallelism, runtime, requested runtime. Traces from resource managers typically contain enough information to model these characteristics. We discuss modeling methods based on traces used by other researchers, mainly for parallel computer workloads, and our choice for modeling each characteristic for grid systems.

Task size The degree of parallelism denotes the number of processors used by each task, also reffered to as the task size. In parallel production environments, power-of-two task sizes predominate, the cause of this has been much debated. However, Lublin and Feitelson [49] and Cirne and Berman [11] both model power-of-two task sizes explicitly. In addition, Lublin and Feitelson [49] also made a clear distinction between sequential and parallel task sizes.

Our workload model will take the latter approach, modeling sequential and parallel tasks separately, while accounting for the prevalence of power-of-two task sizes in parallel tasks. The task size is determined by a distribution of task sizes for all parallel tasks, combined with two proabilities $p1$ and $p2$. The tasks sizes are first logarithmically transformed with base 2.

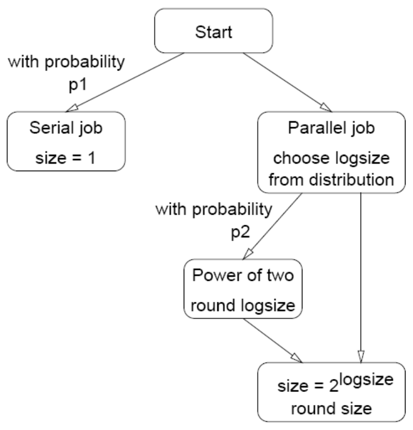


Figure 3.2: Procedure for generating the task size from the model

When generating a task size, we first determine whether the task is sequential or parallel using $p1$. If it is parallel, we take two additional steps. First, we use our distribution to generate a logarithm of the task size. Using $p2$ we determine whether this task size will be a power-of-2. If so, we round the generated task size to an integer. By raising this number to the power of two, we obtain the actual task size. Figure 3.2 shows the procedure for generating the task size from this model.

Task runtime After a user submits tasks to the grid system, the task is usually placed in a waiting queue of the resource manager. After a suitable execution location is found, the task starts execution. The task runtime is defined as the difference between the end time and the start time of the task. The runtime of a task denotes the time it has occupied resources of the system, therefore it is an important feature to include in our model. In parallel environments, the task runtimes increases with increasing task sizes [17]. However, since the majority of tasks in grids are sequential [34], we will model the task runtime independently.

Requested time When submitting a task to the grid system, usually the user has the option to specify the requested time for the task. This is an estimation of the runtime of the task that is submitted. This information can be used by resource managers to optimize scheduling of tasks in the grid. For example, backfilling schedulers use the requested time to advance short tasks in their queue, allowing waiting time of longer tasks to be filled with short tasks. The accuracy of the requested times have significant impact on backfilling schedulers [10]. This makes it and interesting feature of tasks to include in our workload model.

3.3.3 BoT-level model

With our task-level model we only consider individual tasks, however they may be identified as part of a group of tasks. Iosup et al. [37] defined three types of task groupings: BoT submissions, continued submissions and bursty submissions. Iosup et al. analyzed three grid traces on the characteristics of these groupings. Their analysis of real-world grid traces shows that BoTs contribute up to 96% of the consumed CPU time and that BoT submissions have high idle times. These results motivate the need for detailed analysis of the performance of BoTs in grids.

We model the arrivals of BoTs, instead of the arrivals of single tasks. For each BoT arrival, we decide the number of tasks in the BoT, the arrivals of tasks within a BoT and the runtime variability of tasks within a BoT.

The data on BoTs are not as easily extracted from workload traces as the task-level characteristics. The workload traces typically only record the data for each task separately, most traces do not record whether tasks belong to a BoT. Therefore, we first have to identify which tasks belong to a BoT, before we can extract the data on which we can fit distributions.

We use the definition of BoT submissions as given by Iosup et al. [37] in their analysis of BoTs in grids. A BoT is a group of task submissions of the same user, where the interarrival time of two consecutive tasks is smaller than 120 seconds. With this definition we can identify all BoTs in a workload trace, and calculate the necessary information for each BoT by examining its tasks.

BoT arrivals BoTs are submitted by large groups of users to the grid system, leading to a distribution of task arrivals over time.

The arrival of BoTs can be modeled by the amount of BoTs that are submitted in a given time interval, the arrival rate. For example, Cirne and Berman [11] present a workload model that estimated the number of arrivals for every 10 minutes. Alternatively, the task arrivals have been characterized by the interarrival time between two consecutive tasks. Lublin and Feitelson [49] and Jann et al. [42] modeled task arrivals in parallel production environments using interarrival times. The same approach can be taken to model the arrival of BoT submission, instead of single tasks.

The analysis of grid traces shows that periodic patterns may appear [34], such as daily and weekly cycles. In addition, several other moments of high contention (spikes) are occurring. This behaviour is difficult to capture in a statistical model. To work around this difficulty, Jann et al. [42] clustered the tasks by size, modeling each category separately.

Another approach was taken by Lublin and Feitelson [49], as they modeled the rather stationary interrarival times during peak hours, and the variation caused by the daily cycle separately. For the peak interarrival times, a statistical distribution is fitted to the original data. In addition, for each hour in a day a slot weight is calculated, which determines the relative rate at which tasks arrive in that hour.

We will use the modeling method of Lublin and Feitelson to model the arrivals of

BoTs by finding a distribution for the interarrival times of BoTs, while preserving the cyclic behaviour.

The daily cycle is modeled by the slot weight method. First, the BoTs are ordered on time of submission. Thereafter, the BoTs are divided in to 48 slots, one for each half hour. For each slot, we calculate the fraction of BoTs that arrives in this slot over the total number of BoTs, which we will call the arrival rate. The submission times are shifted by 6 hours, such that the distribution becomes unimodal. We want to find a distribution that gives for each slot the fraction of BoTs that arrives in that half hour.

With this distribution we can calculate slot weights, that determine "how fast" time passes in that slot. The weights are calculated by dividing the arrival rate for each slot, by the average arrival rate. This number indicates the virtual speedup or slowdown of each half hour. When we sample an interarrival time, we use this virtual time, such that slots with higher weights will contain more BoTs.

BoT sizes For new BoTs, we need a model for the distribution of BoT sizes, the number of tasks that is part of the BoT. This distribution is heavy-tailed [37], so we model it similar to the heavy-tailed task-level characteristics, such as task runtime.

BoT task interarrivals For task arrivals within a BoT, we are looking for the distribution of interarrival times within BoTs. For each BoT in a workload trace, we calculcate the interarrival times for the tasks within the BoT. We try to find a distribution that fits these interarrival times for all BoTs.

Runtime variability We assume that tasks in a BoT are related when it concerns the runtime of tasks. Therefore, we use the task-level runtime distribution to sample a single runtime for the entire BoT from the task-level model. We extend this model with the notion of variability, so we can control the variance of runtime of tasks within a BoT.

By finding a distribution for the runtime variance, we can generate the runtimes for tasks in a BoT. First, we generate a sample from the task runtime distribution as specified in the task-level model for each BoT. Then we take a sample from the runtime variance distribution. We can now generate runtimes for all the tasks in the BoT by sampling from a normal distribution with the mean as the generated task runtime and the sampled runtime variance.

3.3.4 User-level model

In parallel production environments, a small group of users dominate the workload in terms of the number of submitted tasks and consumed CPU time [17]. The dominant users often run a set of applications repeatedly. Similar user behaviour has also been reported in grid environments [34, 47].

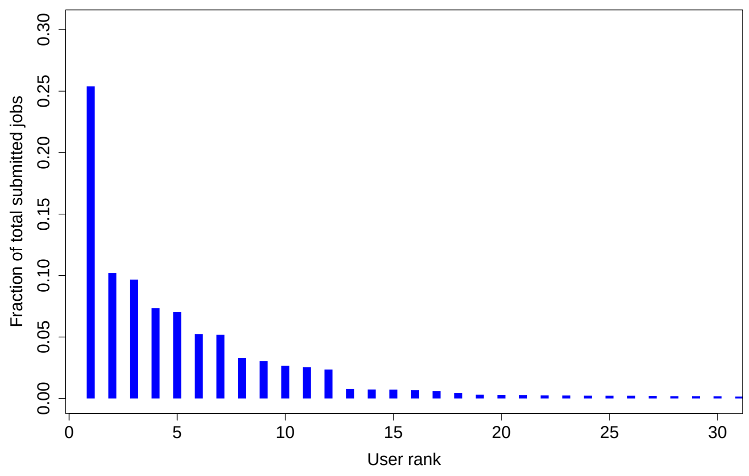


Figure 3.3: The user ranking based on the fraction of submitted tasks per user of the total number of tasks for the Grid'5000 system.

The question arises how to incorporate this behaviour in a workload model. Calzarossa and Serazzi [7] propose modeling individual user behaviour using Markov-chain models. The user behaviour is identified based on the commands the users executed on a parallel computer system. An alternate approach is presented by Song et al. [59], where users are clustered in groups, based on the size and runtime of their submitted tasks. The tasks for each group are then modeled separately with statistical distributions.

Our workload model takes a different approach to include the user behaviour. Since a small number of users dominate the workload, we can model the tasks of these users separate from the rest of the users. We assume that the submission behaviour and the resource usage of these power users will be significantly different from the other users.

Users are ranked based on their fraction of their submitted tasks of the total number of tasks submitted to the grid, from high to low. The top users belong to the minimal group of consecutive users, ordered by rank, that collectively consume more than a maximum proportion p of all tasks.

For example, Figure 3.3 shows a user ranking of a real grid system, Grid'5000, and their fraction of submitted tasks. For maximum proportion $p = 50\%$, rank 1 to 5 are the highest ranked users that together submit more than 50%, so they are considered the top users. The rest of the users are considered as the 'normal' users, so they will be grouped together, in a general user model.

The distribution of tasks over users can be determined either explicitly or implicitly. In the explicit model, a BoT- and task-level model is created for each of the top users. In addition, the general user model represents the BoTs of the rest of the users.

For k top users, this means that $k + 1$ task-level model instances need to be derived from the trace. Subsequently, each task-level model instance is used to generate a stream of tasks. By combining the tasks, the workload for all users is generated.

Alternatively, in the implicit model, a single general BoT- and task-level model is used to generate tasks for the workload. However, for each task we will determine whether the task belongs to one of the top users or the general user, based on the fraction of tasks these users submitted. The implicit model is more concise than the explicit model, therefore it will be used in our workload model.

To model the distribution of tasks over user in the implicit model, we choose to model the ranking of users. We want to have a function that takes as input the user rank and gives the fraction of tasks of a user as output. We will model this function with a power-law distribution, as our analysis shows that the user ranking of real systems has similar characteristics to these kind of distributions.

3.4 Related work

On the subject of workload modeling, much work has been done in characterizing the workload of parallel computing environments [16, 22, 42, 49]. This section discusses the evolution of parallel workload models by describing the different characteristics and modeling methods used by other researchers.

One of the first workload models for parallel systems was introduced by Feitelson for use in the evaluation of gang scheduling algorithms [22]. Feitelson modeled the runtimes of tasks with a hyperexponential distribution and investigated the correlation of runtime with the task size. In addition, the interarrival times of task submissions were assumed to be exponentially distributed. Feitelson acknowledged the difference between tasks in BoTs and normal tasks, as he used the same model for both, but used different data to find parameters for both task classes.

Jann et al. [42] present a model with runtimes and interarrival times characterized with hyper-Erlang distributions, fitted by moment matching. Jann et al. divided the tasks in a number of classes, based on the task size. This results in a complex model, which is difficult to change, but was nearly identical to the original data.

Both mentioned models only characterized rigid tasks, or tasks requiring a fixed number of nodes. Downey [16] proposes a model for moldable tasks. The model gives a speedup function, which derives the correlation between runtime and task size.

Cirne and Berman [11] present a model which tries to include the daily cycle of task arrivals. They modeled the arrivals by matching polynomials of high degree with the arrival rate of tasks. However, they did not achieve good results when compared to the original data. Cirne and Berman also modeled the requested runtime, which is interesting for backfilling schedulers. Another addition is the modeling of the cancellation rate and cancellation lag of submitted tasks.

Lublin and Feitelson [49] also modeled the daily cycle of arriving tasks, and achieved good results when compared to the original data. They used maximum likelihood estimation to fit Gamma distributions to the interarrival times and the daily fluctations in task arrivals. Their model also included task runtimes, and the correlation between runtimes and task size. Again, this resulted in a model that fits the data well, at the expense of a complex model.

The above mentioned workload models were based on traces of parallel systems. Li et al. [47] characterized the workload of the DAS-2 grid by fitting distributions on trace data. Their work modeled task runtime, task size, cancellation rate, memory usage, as well a correlation between runtime and requested runtime of tasks. However, they only used the trace data from one grid, therefore making it difficult to generalize for other systems.

Song et al. [59] introduce a workload model that identifies different types of users. These users are clustered based on the sizes and runtimes of the tasks they submitted. For each cluster, a separate workload model is created. This model did not include arrivals of tasks, only the runtime and size of tasks were modeled.

Our work extends the existing research by modeling the workload using seven traces of grids, as opposed to traditional parallel systems. We aim to create a simpler and more flexible model, so we can evaluate different scenarios. We include BoTs of tasks in our workload model, as well as the traditional characteristics of runtime, size, requested runtime, arrivals.

Chapter 4

Modeling Results for Real Grid Workload Traces

Now that we have presented a comprehensive grid workload model, we have a method to characterize grid workloads. One of our research questions stated we wanted to characterize workloads realistically. To provide our workload model with a realistic basis, we use grid workload trace data to find parameters for our model. In this chapter we discuss the results of modeling seven workload traces, based on the family of workload models described in Section 3.3. By using seven workload traces from real grid systems, we are able to create an "average system" workload model that can be used to generate realistic workloads, without resorting to a single system as a reference. These realistic workloads will in turn serve as input for the performance analysis of scheduling BoTs in grids.

First, in Section 4.1 we present a procedure for modeling workloads based on traces of real grid systems. In Section 4.2 we present a description of the used traces and discuss the grids that from which these traces originated. In Section 4.3 we discuss the modeling results of fitted distributions for each trace for the implicit user-level workload model, listing the parameters of the distributions and the goodness-of-fit test results. This is the workload model that will be used in our performance analysis. In Section 4.3.3 we present the parameters for the explicit user-level model, in which we model several dominant user individually on the BoT- and task-level. Parts of this chapter - the work on modeling procedure and modeling results for workload traces - have been published in a paper, which was submitted to and accepted by the ACM International Symposium on High Performance Distributed Computing 2008 [40].

4.1 A procedure for modeling workloads

In this section we will present our modeling method for the various workload characteristics using statistical distributions. First we discuss which distributions will be used to fit the workload data. Thereafter we discuss methods of fitting a statistical distribution to a dataset. We also look into goodness-of-fit tests, with which we can determine the accuracy of a fitted distribution. Finallly, we present methods to validate a generated workload with the original trace data, and how to compare the workload model with other workload models.

The results of the actual fitting of distributions, goodness-of-fit tests and comparison with other workload models and the original trace are presented in Section 4.

4.1.1 Candidate distributions

We assume that the individual characteristics of tasks, as described in detail in Section 3.3.2, are not randomly distributed, but rather samples from an unknown probability distribution. If we can identify this distribution, or a similar distribution, we can emulate the process that generates the samples, and therefore reconstruct the characteristics of the workload. Our assumption is enforced by previous workload models [11, 42, 49], that achieved good results by modeling workload characteristics using a variety of well-known statistical distributions.

A probability distribution describes the distribution of possible outcomes of a random variable. For example, if the random variable represents the gender of newborns, we can describe the probability distribution as follows:

$$Pr(female) = 0.53, Pr(male) = 0.47$$

Each possible outcome is assigned a probability of occurrence, possibly derived from historical data. This is an simple example of a discrete probability distribution, from which later on we can predict the gender of a next newborn.

In the previous example, there are only two possible outcomes, so describing the distribution is trivial. This is not the case with variables that have a large range of possible outcomes, such as task runtimes in a grid system. As runtimes may vary from zero seconds to multiple hours or even days, it is inconvenient to describe the probability of occurrence for each possible outcome. With a histogram it is possible to visualize the occurrence or frequency of a random variable. For illustration, Figure 4.1 shows the histogram for task runtimes in the DAS-2 grid.

A probability density function (PDF) can also be used to describe the frequency of possible outcomes. As input the PDF takes an outcome of a discrete random variable, and gives as output the probability of occurrence of that outcome. An equivalent way of describing is the use of the cumulative distribution function (CDF). The CDF gives for every real number x the probability that the outcome does not exceed x. Looking at the histogram of runtimes, it seems implausible to find a single function that fits the data exactly. Therefore we have to resort an approximation. What we need is a distribution that has PDF (or CDF) with a shape similar to the original data. It is not necessary to include every single irregularity.

In previous work, other researchers have found that well-known heavy-tailed distributions, for example the gamma distributions, are good candidates for fitting workload data [11,49]. Therefore we will focus our attention on a variety of heavy-tailed distributions. In addition, our aim is to keep the model as simple as possible,

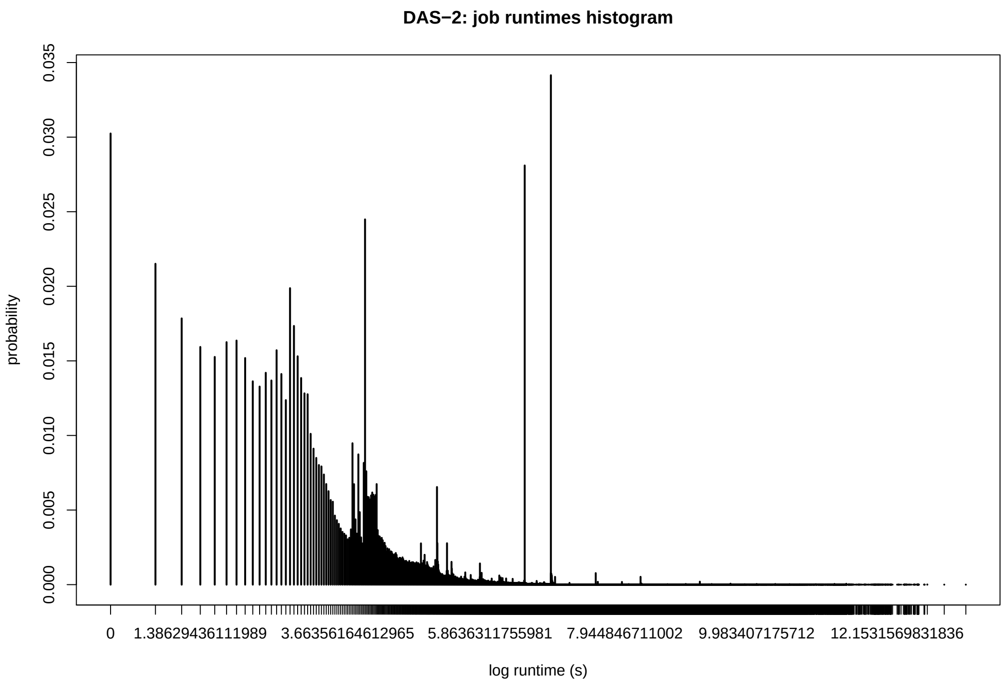


Figure 4.1: Histogram of task runtimes in the DAS-2 system. The runtimes are depicted on a logarithmic scale with base 2.

therefore we restrict ourselves to distributions with no more than two parameters. Each candidate distribution is discussed individually in the following paragraphs.

Normal distribution The normal or Gaussian distribution has been used in a variety of scientific area, from astronomy to psychology. The normal distribution has interesting statistical properties , for example its parameters denote the mean and the standard deviation of the distribution. The normal distribution has also been called the “bell curve”, since its symmetric shapes resembles that of a bell. The probability density function is defined as follows, with μ denoting the mean and σ the standard deviation:

$$f(x;\mu,\sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Log-normal distribution The log-normal distribution is related to the normal distribution in the sense that if x is a sample from a normal distribution, $y = e^x$ is a sample from a log-normal distribution. The probability density function of the log-normal distribution is defined as:

$$f(x;\mu,\sigma) = \frac{e^{-(\ln x-\mu)^2/(2\sigma^2)}}{x\sigma\sqrt{2\pi}}, x > 0$$

Exponential distribution The exponential distribution is a popular distribution, for example in the field of reliability engineering. It is a mathematically simple distribution, with only one parameter called the rate parameter, making it relatively to estimate parameters for a given sample. Its probability density function is defined as follows, with λ denoting the rate parameter:

$$f(x;\lambda) = \lambda e^{-\lambda x}, x \geq 0$$

Hyper-exponential distribution The hyper-exponential distribution is a combination of multiple exponential distributions. This enables modeling of multi-modal datasets, which has multiple peaks when visualized in a histogram. We consider a two-stage hyper-exponential distribution, having two rate parameters for each distribution and one proportion parameter, denoting the proportion of the distribution that belongs to the first. The probability density function is defined as:

$$f(x;\lambda_1;\lambda_2) = p\lambda_1 e^{-\lambda_1 x} + (1-p)\lambda_2 e^{-\lambda_2 x}, x \geq 0$$

Gamma distribution The gamma distribution is a generalized version of the exponential distribution. It represents the sum of k exponentially distributed random variables each with a mean of θ. When $k = 1$, the gamma distribution is equal to the exponential distribution. The gamma distribution is more flexible than the exponential distribution, making it possible to represent a larger range of datasets. The probability density function is defined as:

$$f(x;k,\theta) = x^{k-1}\frac{e^{-x/\theta}}{\theta^k \Gamma(k)}, x, k, \theta > 0$$

$$\Gamma(k) = \int_0^\infty t^{k-1} e^{-t} dt$$

Weibull distribution The Weibull distribution is commonly used in the area of lifetime or failure analysis. It is a flexible distribution, because it can represent the normal and exponential distribution by setting its parameters accordingly. The probability density function is defined as:

$$f(x;k,\lambda) = \frac{k}{\lambda}\left(\frac{x}{\lambda}\right)^{k-1} e^{-(x/\lambda)^k}$$

4.1.2 Fitting a distribution to data

The candidate distributions need to be fitted to the data obtained in the traces. The traces provide us with a set of observed values, such as the task runtime or size for all tasks, which has some unknown distribution. We assume that one of our candidate distributions can approximate this unknown distribution, therefore we need a method to estimate the parameters for the candidate distributions. This estimation is performed for every candidate distribution, thereafter the goodness of fit of the estimated is assessed. The distribution that resembles most the distribution of the original data, is chosen for the workload model.

Maximum likelihood estimation

Two commonly used methods for estimating distribution parameters are moment matching and maximum likelihood estimation. The first tries to estimate the distribution parameters by calculating the moments, such as mean and variance, of the original data. The moments of a distribution can typically be expressed as a function of its parameters. By calculating the moments of the original data and inverting the functional relation with the parameters, we can obtain an estimation of the distribution parameters. A problem with this method is that moment matching is sensitive to outliers, therefore is not suitable to match distributions with fat tails. However, moment matching methods have been used to fit distributions on to workload data [42].

We have chosen the second option, maximum likelihood estimation, which tries to optimize the likelihood function. Given a data set and a distribution with parameter vector θ, the likelihood function calculates the probability that this data set is a sample of this combination of distribution and parameters. The parameter vector with the highest probability is the best fit of the distribution to the original data. Essentially, we are looking for the maximum of the likelihood function. The likelihood function is defined as follows for a set of samples $X_1, \ldots, X_n$, parameter vector θ, and probability density function f:

$$L(X_1, .., X_n; \theta) = \prod_{i=1}^{n} f(X_i; \theta)$$

To speed up the optimizing of the likelihood function, the logarithm of the likelihood is used. This transforms the product into a sum, thereby reducing the number of computations required. For the normal, log-normal and exponential distributions the standard optimum finding algorithm can be used, by differentiating the function with respect to θ and equating to zero. For the other distributions, Newton-type optimization algorithms are used to find the parameters that have the maximum likelihood.

4.1.3 Goodness-of-fit

After the fitting of distributions with the maximum likelihood estimation method, we will assess the goodness of fit with the Kolmogorov-Smirnov test (KS test). The KS test tries to determine if two data sets differ significantly. The one-sample KS test tests the hypothesis that a given sample belongs to a specified continuous distribution. The cumulative distribution of the null hypothesis is compared with the empirical distribution function of the sample. Given a set of N observed values

$X_1, \ldots, X_n$, the empirical distribution function is defined as:

$$F_n(x) = \frac{\text{number of elements in the sample} \leq x}{n}$$

The KS test calculates the maximum distance between the hypothesized cumulative distribution function and the empirical distribution function of the sample. The KS test for a given cumulative distribution function $F(x)$ and empirical distribution function $F_n(x)$ is defined as follows:

$$D = \sup |F_n(x) - F(x)|$$

For large values of D, the hypothesis that the empirical distribution function and the cumulative distribution are identical is rejected. The value of D also gives a measure to compare several distributions to the same sample. The distributions that results in the lowest value of D in the KS test, is the distribution that fits the data best.

The KS test has the disadvantage that it only applies to fully-specificed continuous distributions, unlike the χ^2 test that can also test against discrete distributions like Poisson. Fortunately, we only test against continuous distributions with known cumulative distribution functions. Another disadvantage is that the KS test is more sensitive at the center of the distribution, than at the tails. This problem could be circumvented by using the Anderson-Darling test, but this test is only available for a few distributions, where the KS test does not depend on the distribution being tested.

4.2 Description of used workload traces

We have used seven grid traces from the Grid Workloads Archive [32], which makes (anonymized) grid workload traces available to researchers. The traces are available on-line in the Grid Workload Format, which is a standard format for grid workload traces.

The traces contain enough information to fit statistical distributions for all the desired characteristics discussed in Chapter 3. Before we present the results of the modeling, we give general information about the systems under study, as well as specific remarks on the used traces. A summary of system and trace characteristics for each trace is given in Table 4.1.

Distributed ASCI Supercomputer 2 This trace originates from the Distributed ASCI Supercomputer 2, or DAS-2 [14], which is a research grid consisting of five clusters, connected through a wide-area network. Each cluster is located at a different Dutch university, with a total number of 400 CPUs in the entire grid. It must be noted that this system is not heterogeneous, each cluster is built from dual Pentium II nodes, connected locally through a Myrinet network.

system	period	# of CPUs	# of tasks	total consumed CPU time
DAS-2	1.5 years	400	1,124,772	68y 230d 21h
NGS	3 years	378	631,737	269y 226d 3h
AuverGrid	1 year	475	404,176	277y 226d 7h
SHARCNET	1 year	6,828	1,195,242	3782y 15d 7h
LCG	11 days	24,515	188041	53y 179d 7h
NorduGrid	2 years	˜2000	781,370	2443y 220d 15h
Grid'5000	2.5 years	˜2500	951,234	651y 241d 19h

Table 4.1: Summary of system characteristics for the seven traces

Since DAS-2 is a research grid, the workload is composed of a large variety of different application types, from simple single CPU tasks to complex co-allocated MPI tasks. Tasks can be submitted directly to local resource managers or via Grid gateways that interface with local resource managers.

The system utilization is relatively low in the recorded period, on average 10 percent, as a consequence of the policy that the system should be left as free as possible. During working hours, the tasks are limited to 15 minutes of runtime, longer tasks are allowed during nights and in the weekend.

National Grid Service This trace contains tasks submitted to the National Grid Service (NGS) [52], which stems from a collaboration of different universities in the United Kingdom. It is the first production-level grid to be used for academic research in the UK. We have obtained traces from the four dedicated computing clusters present in NGS.

At time of writing, no information was available about the number of nodes in the system. From the trace we 'reverse-engineered' the number of nodes by calculating the maximum number of simultaneously occupied nodes in a short interval. According to our calculation the NGS has at least 378 CPUs available.

AuverGrid This trace contains tasks from the AuverGrid [4], a multi-site grid that is part of the EGEE project. This grid employs the LCG middleware as the grid's infrastructure. The trace that we used contains tasks from five different sites in the grid, with a total of 475 CPUs available. The grid is mainly used for bioinformatics, high energy physics and computer graphics applications.

When we look at the system utilization of AuverGrid, it proves to be a production-level grid, with an average of around 60 percent. It is remarkable that this trace only contains sequential tasks, with an average runtime around 7 hours. No requested runtime data was recorded in the AuverGrid trace.

The user information in this trace is unfortunately incomplete, as the user that submitted a task could not always be identified uniquely. This has the consequence that this trace can not be used for modeling the user behaviour.

SHARCNET This trace contains submitted tasks of SHARCNET [58], a multi-instutional high performance computing network that spans 16 leading academic institutions in South Central Ontario, Canada. The SHARCNET grid consists of 10 clusters containing a total of 6,828 CPUs. The SHARCNET trace contains 1,195,242 tasks, recorded over a period of 13 months.

LHC Computing Grid The LHC Computing Grid (LCG) [44] is the origin of this trace. The LCG is a large grid built to support the Large Hadron Collider, currently being built at CERN. It is used to store and analyze large volumes of data that originates from the Large Hadron Collider. The LCG trace contains only sequential tasks, and no requested runtime data.

NorduGrid This trace contains tasks submitted to the NorduGrid [53], a large-scale production grid. In NorduGrid, non-dedicated resources are connected using the Advanced Resource Connector (ARC) as Grid middleware. Over 75 clusters have been added over time to the infrastructure. In these logs, the information concerning the grid tasks is logged locally, then transferred to a central database voluntarily.

Grid'5000 This trace is recorded at Grid'5000 [31], an experimental grid plaform consisting of 9 sites geographically distributed across France. The sites are multi-cluster systems, with a total of 15 clusters in the entire grid. The main objective of this platform is to allow experiments in all the software layers between the network protocols up to the applications.

4.3 Workload modeling results

This section presents the results of modeling the workload recorded in the seven traces. We use the implicit user-level workload model, where the BoT-level and task-level characteristics are fitted based on data for all users. The user submission behaviour is implicitly modeled with a user ranking, which determines to which user a BoT belongs.

For each workload characteristic, we briefly discuss specific details of the modeling method. Thereafter, we present the goodness-of-fit test results and the fitted distribution parameters for each workload characteristic. From the goodness-of-fit test results we derive the best fit. For illustration of the fits to original data, we have plotted the fitted distribution for some of the systems.

Subsequently, we present the parameters and distributions for our generic grid workload model, which tries to summarize the results of the invididual traces in one workload model instance that can be used for workload generation. The workload model is then validated against the original data in the traces, of which the results are discussed.

system	lognormal	exp	normal	Weibull	hyperexp	gamma
das2	0.07	0.28	**0.04**	0.05	0.30	0.06
auvergrid	0.16	0.24	0.16	0.16	0.26	**0.15**
ngs	0.38	0.36	**0.20**	0.37	0.36	0.38
lcg	**0.14**	0.44	0.19	0.17	0.47	0.16
sharcnet	0.14	0.25	**0.08**	0.09	0.24	0.12
grid5000	0.08	0.29	**0.07**	0.08	0.31	0.07
nordugrid	0.11	0.33	**0.07**	0.07	0.29	0.10
average	0.16	0.31	**0.12**	0.14	0.32	0.15

Table 4.2: Goodness-of-fit test results for task runtimes for all seven traces

All distributions are fitted with the maximum likelihood estimation, with the exception of hyper-exponential which uses the Expectation Maximization algorithm. The goodness-of-fit is assessed with the Kolmogorov-Smirnov (KS) test. For an explanation of these methods, see Section 4.1.

4.3.1 Task-level model

At the task-level we model three characteristiscs: task runtimes, task requested runtimes and task size. We use the data of all the tasks in a trace to model these three characteristics.

Task runtimes

For the task runtimes the same holds as for the task interarrival times, because the task runtimes are continuous and span a large of values. For this reason, we first apply a logarithmic transformation with base 2 to the data.

Subsequently, we fitted the candidate distributions: gamma, exponential, hyper-exponential, normal, log-normal and Weibull. The goodness-of-fit results for all seven systems are displayed in Table 4.2, which shows the D-statistic of the KS test. Figure 4.2 shows the cumulative distribution function of task runtimes with the best fits for the SHARCNET trace.

For the task runtimes, the NGS system shows poor results for the goodness-of-fit results. Again, as with the task interarrival times, this can be explained by the high proportion of zero values, in this case around 40% of all tasks. This behaviour is not present in the other traces.

The distribution that fits the runtimes best differs for the seven systems. The normal distribution is the best fit for DAS-2, SHARCNET, NGS and Grid5000. Also on average, the normal distribution provides the best fits for all systems. The parameters for the normal distribution are given in Table 4.3.

	lognormal		exp	normal		Weibull		hyperexp			gamma	
system	mean	sd	rate	mean	sd	shape	scale	p	rate1	rate2	shape	rate
das2	1.28	0.49	0.26	1.78	3.87	2.56	4.50	1.00	0.18	0.27	4.97	1.24
auvergrid	1.88	0.64	0.14	3.47	7.39	2.49	8.54	3.39	0.11	0.28	3.55	0.47
ngs	1.52	0.68	0.28	3.50	3.51	1.97	6.17	0.30	0.20	28.78	2.86	0.52
lcg	1.82	0.34	0.15	2.22	6.51	3.15	7.29	1.00	0.11	0.17	9.18	1.41
sharcnet	1.90	0.50	0.13	3.06	7.45	2.70	8.39	6.24	0.11	0.22	4.73	0.63
grid5000	1.57	0.46	0.20	2.31	4.97	2.78	5.91	1.00	0.14	0.21	5.82	1.10
nordugrid	2.15	0.36	0.11	2.76	9.04	3.90	10.04	9.56	0.10	0.16	9.11	1.00

Table 4.3: Fitted distribution parameters for task runtimes for all seven traces

Task requested runtimes

For the task requested runtimes we used the same approach as with the task runtime, so first we apply the logarithmic transformation with base 2. Subsequently, we fitted the candidate distributions: gamma, exponential, hyper-exponential, normal, log-normal and Weibull. The goodness-of-fit results for all seven systems are displayed in Table 4.4, which shows the D-statistic of the Kolmogorov-Smirnov test. Figure 4.3 shows the cumulative distribution function of task requested runtimes with the best fits for the Grid'5000 trace.

For the task requested runtimes, the DAS-2, NGS and AuverGrid systems shows poor results for the goodness-of-fit results. Note that SHARCNET, LCG and AuverGrid did not record the requested runtime for tasks in their traces. This can be explained by the fact that the DAS-2, Grid5000, NGS and AuverGrid system have a default value for requested runtime if none is supplied by the user. This distorts the distribution of the requested runtime significantly. Even if these values were to be removed, a significant portion of requested runtime values approaches the default value, as users use this as a starting point for their own estimations.

We observe from the goodness-of-fit results that Weibull provides the best fit on average for the four systems. However, when we look at the parameters in Table 4.5, the log-normal distribution shows more consistent parameters, while the goodness-of-fit is only slightly better than Weibull. Therefore we choose log-normal as the distribution for requested runtimes.

Task size

For the task size, we employ the method proposed by Lublin and Feitelson [49]. First, we determine the probability that a task is sequential, simply by calculating the fraction of sequential tasks over all tasks. Table 4.6 lists the probabilities as $p1$ for each of the traces. Note that the AuverGrid and LCG traces do not contain any parallel tasks at all. The higher parallelism of the tasks for DAS-2 and Grid'5000 is probably due to the fact that these are research or experimental grids.

Secondly, we calculate the probability that a parallel task has a size equal to a power-of-2. These probabilities are also given in Table 4.6.

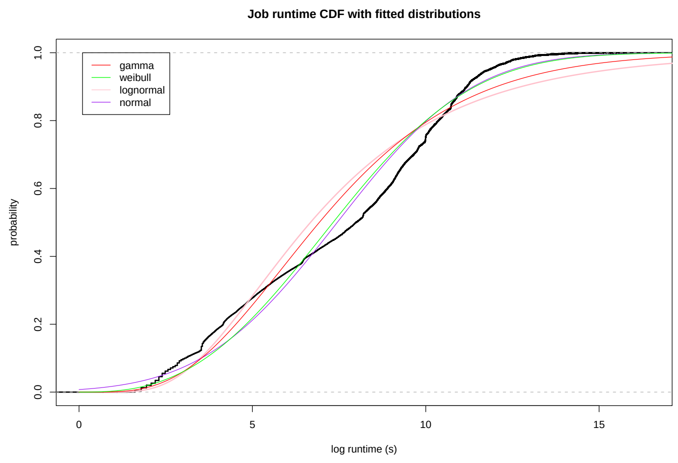


Figure 4.2: Cumulative distribution function for task runtimes in SHARCNET with fitted distributions

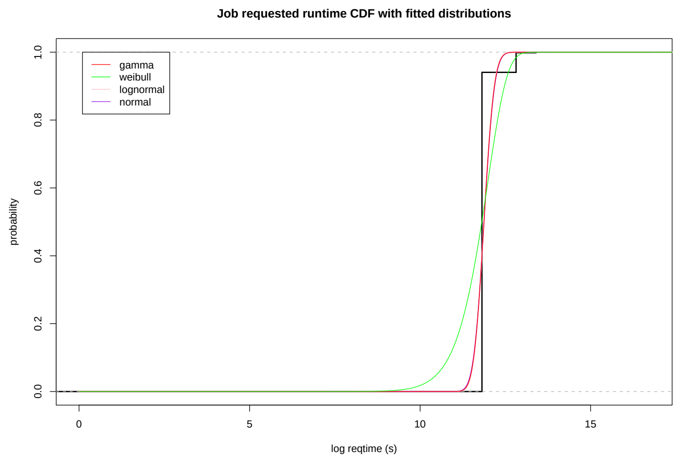


Figure 4.3: Cumulative distribution function for task requested run times in Grid'5000 with fitted distributions

system	lognormal	exp	normal	Weibull	hyperexp	gamma
das2	**0.39**	0.55	0.42	0.41	0.56	0.40
auvergrid	0.47	0.57	0.47	**0.46**	0.50	0.47
ngs	0.49	0.57	0.49	**0.47**	0.54	0.49
grid5000	0.32	0.55	0.32	0.32	0.54	**0.31**
average	0.42	0.56	0.42	**0.41**	0.54	0.42

Table 4.4: Goodness-of-fit test results for task requested runtime for all seven traces

The next step is to model the distribution of task sizes for the parallel tasks. As with the task runtimes and interarrival times, we first apply a logarithmic transformation with base 2 to the task size data. Combined with $p1$ and $p2$ we can use the distribution to generate task sizes.

Ofcourse, in order to generate the task size, we need to fit distributions on the original task sizes for all parallel tasks. We have done so for the following distributions: gamma, exponential, hyper-exponential, normal, log-normal and Weibull. The goodness-of-fit results for all seven systems are displayed in Table 4.7, which shows the results of the goodness-of-fit test. A lower value means a better fit. In Figure 4.4 we have plotted the CDF for the task size data of the DAS-2 trace, along with the fitted distributions.

For task sizes, the best distribution appears to be Weibull, providing the best fit on average for three out four traces. Table 4.8 shows the parameters for the fitted distributions for each of the traces.

4.3.2 BoT-level model

The BoT-level model is defined with four different characteristics, which are all modeled with a heavy-tailed probability distribution. The BoT arrivals are modeled in two parts, as the BoT interarrival times during peak hours and by determining the variations caused by the daily cycle. The interarrival times of tasks within BoTs are modeled separately. The BoT-level characteristics we have modeled are the BoT size and the runtime variability within BoTs.

BoT interarrival times

The BoT interarrival times are extracted from the trace files by ordering the BoTs on time of submission, and calculating the time difference between each consecutive BoT. We only consider BoTs that arrive during the peak hours, which we define as between 8 AM and 17 PM. This results in a large vector of interarrival times, which is continuous and spans a large range of values. We are looking for a probability distribution that characterizes this vector of interarrival times. To reduce the range and the effect of extreme values, we first apply a logarithmic transformation with base 2 on this data. This does not affect the quality of the fitting of distributions.

	lognormal		exp	normal		Weibull		hyperexp			gamma	
system	mean	sd	rate	mean	sd	shape	scale	p	rate1	rate2	shape	rate
das2	2.29	0.12	0.10	1.28	9.91	5.74	10.50	1.00	0.07	0.10	67.30	6.79
auvergrid	2.88	0.08	0.06	1.13	17.80	45.25	18.07	1.12	0.05	0.13	188.12	10.57
ngs	2.84	0.09	0.06	1.30	17.19	21.69	17.61	5.83	0.05	0.09	135.11	7.86
grid5000	2.52	0.15	0.08	1.75	12.52	7.68	13.26	2.88	0.08	0.09	46.45	3.71

Table 4.5: Fitted distribution parameters for task requested runtime for all seven traces

Subsequently, we fitted the candidate distributions: gamma, exponential, hyper-exponential, normal, log-normal and Weibull. The goodness-of-fit results for all seven systems are displayed in Table 4.9. A low value for goodness-of-fit test indicates a good fit. Figure 4.5 shows the cumulative distribution function of task interarrival times with the best fits for the DAS-2 trace.

From the goodness-of-fit results we observe that the Weibull distribution fits the data best over all systems, with the exception of NorduGrid, where the normal distribution provides a slightly better fit. The parameters for the normal distribution are fairly consistent across the different systems.

Because of the good fitting results, we choose the Weibull distribution to model the interarrival times. The mean μ and standard deviation σ parameters for each system are given in Table 4.10, along with parameters for all other distributions.

Daily cycle

The daily cycle is modeled by finding a distribution for the fraction of BoTs that arrive in a certain timeslot of the day. We divide days in slots of half an hour. For each of the 48 slots, we want to find a distribution that gives the fraction of tasks that arrive during that time. All BoT submissions in the traces are ordered on time, then divided into 48 classes, so we can calculate the fraction of tasks.

We fitted the candidate distributions: gamma, exponential, hyper-exponential, normal, log-normal and Weibull. The goodness-of-fit results for all seven systems are displayed in Table 4.11, which shows the results for the goodness-of-fit test.

The goodness-of-fit test results show that Weibull distribution is the best fit on average, but the differences between gamma and normal distribution are small enough to look closer. When we look at the parameters for those three distributions in Table 4.12, then we see that the parameters do not differ much in consistency. Therefore the Weibull distribution still comes out as the winner for modeling the daily cycle.

BoT task interarrival times

The BoT task interarrival times are extracted from the trace files by ordering tasks withing BoTs on time of submission, and calculating the time difference between

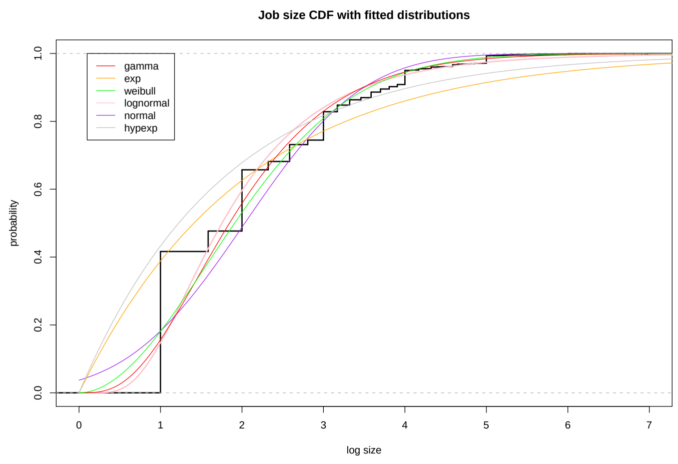


Figure 4.4: Cumulative distribution function for task sizes in DAS-2 with fitted distributions

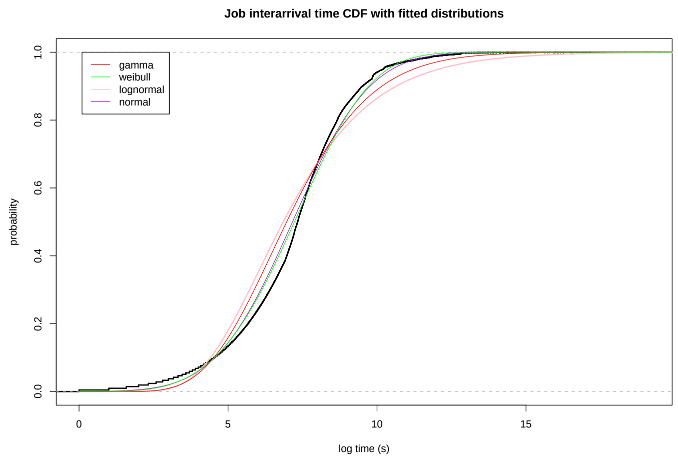


Figure 4.5: Cumulative distribution function for BoT interarrival times in DAS-2 with fitted distributions

system	$p1$	$p2$
DAS-2	0.34	0.75
AuverGrid	1.00	N/A
NGS	0.95	0.91
SHARCNET	0.90	0.71
LCG	1.00	N/A
Grid5000	0.54	0.84
NorduGrid	0.99	0.64

Table 4.6: Fraction of sequential tasks over all tasks and fraction of power-of-2 tasks over the parallel tasks in the traces

each consecutive task in the BoT. This results in a large vector of interarrival times, which is continuous and spans a large range of values. To reduce the range and the effect of extreme values, we first apply a logarithmic transformation with base 2 on this data.

Subsequently, we fitted the candidate distributions: gamma, exponential, hyper-exponential, normal, log-normal and Weibull. The goodness-of-fit results for all seven systems are displayed in Table 4.13. A low value for goodness-of-fit test indicates a good fit. In Figure 4.6 we have plotted the BoT task interarrival times in the LCG trace, along with the fitted distributions.

The goodness-of-fit test results show that the normal distribution is the best fit on average and the best fit for four of seven traces. The parameters for the normal distribution are given in Table 4.14, along with parameters for all the other distributions.

BoT sizes

BoT sizes are extracted from the traces, subsequently logarithmically transformed with base 2. We fitted the candidate distributions: gamma, exponential, hyper-exponential, normal, log-normal and Weibull. The goodness-of-fit results for all seven systems are displayed in Table 4.11, which shows the results for the goodness-of-fit test.

Subsequently, we fitted the candidate distributions: gamma, exponential, hyper-exponential, normal, log-normal and Weibull. The goodness-of-fit results for all seven systems are displayed in Table 4.15. A low value for goodness-of-fit test indicates a good fit. In Figure 4.7 we have plotted the sizes of BoTs in the LCG trace, along with the fitted distributions.

The goodness-of-fit test results show that Weibull distribution is the best fit on average, but the differences between gamma and normal distribution are small enough to look closer. When we look at the parameters for those three distributions in Table 4.16, then we see that the parameters do not differ much in consistency. Therefore the Weibull distribution still comes out as the winner.

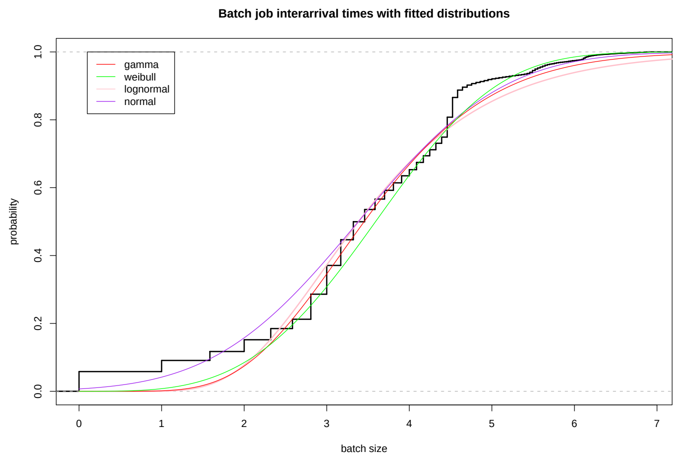


Figure 4.6: Cumulative distribution function for BoT task interarrival times in LCG with fitted distributions

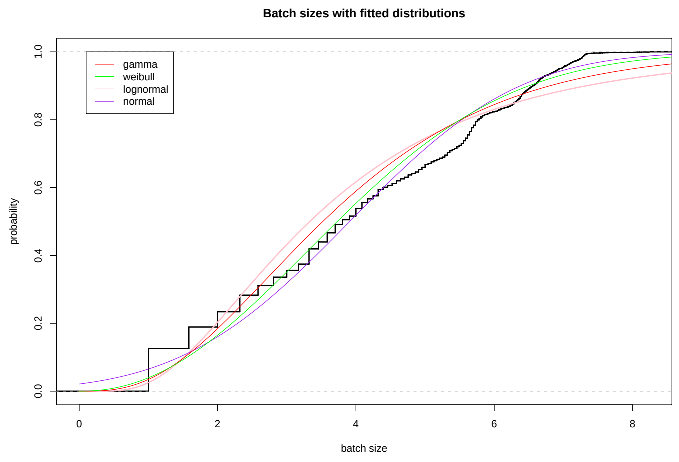


Figure 4.7: Cumulative distribution function for BoT sizes in LCG with fitted distributions

system	lognormal	exp	normal	Weibull	hyperexp	gamma
das2	0.27	0.39	**0.23**	0.24	0.43	0.26
ngs	0.28	0.32	**0.22**	0.23	0.36	0.26
sharcnet	0.16	0.36	0.19	**0.16**	0.44	0.17
grid5000	0.46	0.43	0.45	**0.42**	0.70	0.46
nordugrid	0.14	0.32	0.16	0.15	0.30	**0.12**
average	0.26	0.36	0.25	**0.24**	0.45	0.26

Table 4.7: Goodness-of-fit test results for task sizes for all seven traces

Runtime variability

For each BoT in a trace, we calculate the variance of runtimes for the tasks within a BoT. This gives a vector of variances for each BoT, that we want to model with one of the candidate distributions.

We fitted the variance data with the candidate distributions: gamma, exponential, hyper-exponential, normal, log-normal and Weibull. The goodness-of-fit results for all seven systems are displayed in Table 4.15. A low value for goodness-of-fit test indicates a good fit. In Figure 4.8 we have plotted the runtime variances of BoTs in NorduGrid, along with the fitted distributions.

The goodness-of-fit test results show that Weibull distribution is the best fit on average, but the differences between Weibull and normal distribution are small enough to look closer. When we look at the parameters for those two distributions in Table 4.16, then we see that the parameters for the normal distribution are more consistent across the different systems. Therefore we choose the normal distribution to model the variance of runtimes within BoTs.

4.3.3 User-level model

In Section 3.3.4, we distinguished between two types of user-level models: implicit and explicit. The first uses a ranking of users, which is used to distribute tasks over users, while the latter creates individual BoT-level and task-level models for a number of top users.

We present results for both types of user-level modeling, based on the seven real-world grid traces, and discuss our findings afterwards.

M3: User ranking for implicit user model

We model the distribution of users over BoTs with an user ranking coupled with a probability distribution. The users are ranked based on number of submitted BoTs, with rank 1 for the user for the highest number of submitted BoTs. The lowest rank is equal to the number of unique users in the trace. The probabilities we wish to couple with the rank, are calculated by dividing the number of submitted BoTs for

	lognormal		exp	normal		Weibull		hyperexp			gamma	
system	mean	sd	rate	mean	sd	shape	scale	p	rate1	rate2	shape	rate
das2	0.56	0.54	0.49	1.14	2.03	1.92	2.31	1.00	0.27	0.57	3.54	1.74
ngs	0.85	0.52	0.38	1.17	2.63	2.41	2.97	1.00	0.23	0.45	4.36	1.66
sharcnet	1.00	0.52	0.32	1.60	3.10	2.07	3.52	1.00	0.18	0.40	4.03	1.30
grid5000	0.32	0.62	0.56	1.67	1.79	1.30	1.97	0.60	0.36	3.46	2.08	1.16
nordugrid	0.97	0.52	0.34	1.33	2.97	2.41	3.36	0.51	0.32	0.32	4.26	1.44

Table 4.8: Fitted distribution parameters for task sizes for all seven traces

each user by the total number of BoTs in the trace. Note that the AuverGrid did not record user information correctly and therefore is not used in the user modeling.

The Zipf distribution is used in many fields for characterizing “rank data”, where the relative frequency of a given rank is determined by the distribution function. We have fitted the Zipf distribution onto our user ranking and relative frequencies (of tasks). The results are shown in Table 4.19, alpha represents the parameter of the Zipf distribution, while N stands for the number of unique users (and ranks). The parameters are consistent with each other over all systems. In Figure 4.9, the user ranking along with a fitted Zipf distribution is displayed for the LCG system.

M4: Results for explicit user-level model

The explicit user-level model, as discussed in Section 3.3, tries to capture the user behaviour into the workload model. As the users of a grid system generate the workload, capturing the submission behaviour of users in our model, we can generate representative workloads with the model.

The explicit user-level model takes a generative approach, by modeling the tasks by individual users, and combining the individual models to get a complete model. By modeling each source of the workloads individually, we aim to get a more realistic workload model. With this method, we hope to capture complexities that are otherwise difficult to capture in a all-in-one workload model.

As a small number of users dominate the entire workload, we choose to model the tasks of these users separately, and consider the rest of the users as the background workload.

To model the workload of individual users, we take the same approach as for the implicit user-level workload model, but with a different input. The procedure is as follows:

1. Identify dominant users (together at least 50% of submitted tasks) by analyzing trace files
2. Split trace files into multiple files by tasks of dominant users, and one file for background users
3. Run workload modeling tool on each newly created trace file

system	lognormal	exp	normal	Weibull	hyperexp	gamma
das2	0.13	0.37	0.07	**0.06**	0.37	0.11
auvergrid	0.12	0.34	0.07	**0.05**	0.30	0.10
ngs	0.16	0.42	0.10	**0.09**	0.39	0.14
lcg	0.11	0.35	0.06	**0.04**	0.31	0.09
sharcnet	0.13	0.38	0.08	**0.08**	0.37	0.11
grid5000	0.15	0.40	0.09	**0.09**	0.42	0.13
nordugrid	0.14	0.41	**0.08**	0.09	0.43	0.11
average	0.14	0.38	0.08	**0.07**	0.37	0.11

Table 4.9: Goodness-of-fit test results for BoT interarrival times for all seven systems

4. Evaluate fitted distribution parameters and goodness-of-fit test results

In Step 1 of the procedure we identify the dominant users. After ranking the users on number of submitted tasks, we select the top users that together consume greater than or equal to 50% of the total number of submitted tasks. Table 4.20 shows the for each trace, the number of dominant users that resulted from this analysis. Note that for the AuverGrid trace we did create a user-level model, because the user data in the trace was incomplete.

After splitting the original trace file, we run our workload modeling tool for each user-specific trace, and the background user trace. This results in the same number of parameters and goodness-of-fit test results for each user as for the entire user-level model. For brevity, we do not display all these results, but only summarize the goodness-of-fit results.

In Table 4.21 and Table 4.22 we present the goodness-of-fit results for the top users of the DAS-2 and the Grid'5000 systems. The colum "All" displays the Kolmogorov-Smirnov test results for all users together, each subsequent column displays the results for each dominant user, concluded by the results for the rest of the users together. We only displayed goodness-of-fit test results for these two systems and only for the distributions that were chosen for the BoT-level and task-level model, as displaying all distributions, for all users on all systems would be too elaborate.

We can conclude from the data in Table 4.21 and Table 4.22 that modeling the tasks of dominant users separately does not lead to good fits. The goodness-of-fit test results for the dominant users are either similar to that of all users or worse. There is only a small set of cases in which the modeling the dominant users separately leads to better fits. These conclusions were also found for the workload data of the other systems that we analyzed.

While the goodness-of-fit results need not be significantly better than with the implicit user method, these results do not promise much for this type of modeling. By modeling each user separately with equal or worse fits, it is unlikely that the combined model will represent the workload better than the original model.

	lognormal		exp	normal		Weibull		hyperexp			gamma	
system	mean	sd	rate	mean	sd	shape	scale	p	rate1	rate2	shape	rate
das2	1.92	0.34	0.14	2.03	7.16	4.06	7.91	0.34	0.14	0.14	10.38	1.44
auvergrid	1.83	0.38	0.15	2.07	6.57	3.87	7.33	1.22	0.13	0.21	8.60	1.29
ngs	2.02	0.28	0.13	1.80	7.79	4.94	8.48	3.72	0.12	0.15	15.28	1.95
lcg	1.72	0.37	0.17	1.80	5.82	4.05	6.48	3.97	0.15	0.24	9.03	1.54
sharcnet	1.90	0.33	0.14	1.91	6.94	4.17	7.65	1.91	0.14	0.16	10.98	1.57
grid5000	2.00	0.31	0.13	1.97	7.69	4.27	8.42	1.00	0.11	0.14	12.95	1.68
nordugrid	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 4.10: Fitted distribution parameters for BoT interarrival times for all seven traces

A further investigation of the trace data leads to a possible explanation for these results. It appears that dominant users submit a large number of tasks, but that the characteristics of these tasks do not vary as much as the rest of the users. This seems plausible as a certain user will be interested in a restricted set of applications. As a result the workload of a single user shows a preference for example for certain BoT sizes or runtimes.

Our modeling method uses heavy-tailed probability distributions, which is not ideal for data that has “spikes” in the distribution. Figure 4.10 shows an example by plotting the cumulative distribution function of fitted distributions with the original data for the BoT sizes of a dominant user in the NorduGrid system. We see that this user has a strong preference for BoTs of size between 8 and 9, which all distributions fail to model correctly.

For a representative model, heavy-tailed probability distributions are not sufficient. In order to reflect the spiky behaviour in the distribution of task characteristics, a modeling method is necessary that includes discrete components. We do not explore different modeling methods, as this is beyond the scope of our research.

We can conclude that modeling the workload of dominant users separately with the continuous probability distributions described in Section 4.1.1 does not yield good results. However, we do see potential in modeling the behaviour of users, but different, more discrete modeling methods will be necessary in order to achieve a representative model.

4.4 Generic workload model

Now that we explored different members of our family of workload models, specifically the different user-level models, we present our generic workload model. This workload model will be used a basis for our performance evaluation of scheduling BoTs in grids, of which the results are presented in Chapter 5. From the family of workload models presented in Section 3.3, we choose to use model M3, which uses an implicit user-level model, but includes BoT-level and task-level parameters. This model offers the best trade-off between scalability in the number of

system	lognormal	exp	normal	Weibull	hyperexp	gamma
das2	0.07	0.20	0.10	0.05	0.36	**0.04**
auvergrid	0.12	0.16	0.07	**0.07**	0.36	0.08
ngs	0.13	0.15	0.08	**0.07**	0.13	0.09
lcg	0.10	0.17	0.07	**0.05**	0.37	0.06
sharcnet	0.13	0.29	**0.04**	0.04	0.37	0.09
grid5000	0.10	0.16	0.10	**0.05**	0.36	0.05
nordugrid	0.09	0.15	0.08	**0.05**	0.13	0.05
average	0.11	0.18	0.08	**0.05**	0.30	0.06

Table 4.11: Goodness-of-fit test results for BoT dailycycle for all seven traces

	lognormal		exp	normal		Weibull		hyperexp			gamma	
system	mean	sd	rate	mean	sd	shape	scale	p	rate1	rate2	shape	rate
das2	2.80	0.71	0.05	11.54	19.93	1.82	22.63	1.72	0.02	0.98	2.62	0.13
auvergrid	2.87	0.84	0.04	13.33	22.45	1.72	25.49	1.61	0.02	1.17	2.10	0.09
ngs	2.85	0.88	0.04	13.80	22.45	1.64	25.42	5.59	0.04	0.96	1.95	0.09
lcg	2.82	0.80	0.05	12.66	21.06	1.71	23.86	9.32	0.02	1.02	2.20	0.10
sharcnet	3.11	0.66	0.04	11.37	25.58	2.44	28.99	3.12	0.02	1.02	3.56	0.14
grid5000	2.65	0.82	0.05	11.96	18.23	1.57	20.54	1.84	0.02	0.89	2.04	0.11
nordugrid	2.73	0.82	0.05	12.45	19.62	1.62	22.18	2.28	0.05	0.34	2.08	0.10

Table 4.12: Fitted distribution parameters for the daily cycle for all seven traces

parameters and the representativeness of the workload model.

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. We use the model M3 to model each characteristic of this average system and use the implicit user model to distribute tasks over users.

We use the seven traces from real grid systems to build the average system properties. In Section 4.3, we displayed the goodness-of-fit test results for all candidate distributions for each system and each characteristic. We select the distribution and parameter set for our average system as follows.

For each model characteristic, a candidate distribution that has the lowest average goodness-of-fit value over all seven traces is selected as the average system fit. When for two candidate distributions the difference of their goodness-of-fit values is below 0.01, the distribution closest to the average system fit is selected.

There are systems that showed poor results in fitting for a few characteristics. When calculating the average system parameters these systems were left out, because this is probably caused by anomalies in the trace data. We do not want this anomalies to have influence on the parameters of our average system. For task interarrival times, the trace data for the SharcNet system is ignored, because it had poor fits across all distributions.

system	lognormal	exp	normal	Weibull	hyperexp	gamma
das2	0.24	0.17	**0.10**	0.23	0.17	0.23
auvergrid	0.19	**0.18**	0.18	0.22	0.18	0.20
ngs	0.15	0.36	0.17	0.17	0.36	**0.12**
lcg	0.10	0.35	0.13	**0.09**	0.35	0.09
sharcnet	0.64	0.64	**0.36**	0.64	0.64	0.64
grid5000	0.40	0.28	**0.21**	0.37	0.28	0.39
nordugrid	0.19	0.19	**0.10**	0.17	0.20	0.18
average	0.27	0.31	**0.18**	0.27	0.31	0.26

Table 4.13: Goodness-of-fit test results for BoT task interarrival times for all seven traces

	lognormal		exp	normal		Weibull		hyperexp			gamma	
system	mean	sd	rate	mean	sd	shape	scale	p	rate1	rate2	shape	rate
das2	1.14	0.58	0.33	2.08	3.01	2.25	4.09	1.00	0.34	0.49	3.58	0.99
auvergrid	0.81	0.69	0.40	1.61	2.51	1.95	3.11	1.00	0.37	0.40	2.66	0.97
ngs	0.99	0.41	0.38	1.22	2.66	3.28	3.22	0.62	0.38	0.38	7.31	2.53
lcg	0.85	0.37	0.43	0.95	2.35	3.51	2.76	1.00	0.37	0.42	8.59	3.45
sharcnet	0.41	0.65	1.48	1.17	0.68	1.60	2.10	0.59	0.65	46.46	2.50	1.34
grid5000	0.39	0.68	0.75	1.34	1.33	1.59	2.09	1.00	0.37	0.86	2.34	1.26
nordugrid	0.64	0.59	0.50	1.30	2.00	2.03	2.53	1.00	0.31	0.52	3.31	1.48

Table 4.14: Fitted distribution parameters for BoT task interarrival times for all seven traces

For the percentage of parallel tasks, we also ignored the DAS-2 and Grid'5000 systems, because these are both experimental academic systems, that showed a relatively high preference for parallel tasks. The other production systems did not show this behaviour, we assume that a high percentage of sequential tasks is typical for grid systems.

The parameters of the average system represent the average of this set. The 'Avg' row in Table 4.23 presents the BoT-level and user-level parameters of the average system, while Table 4.24 shows the parameters for the task-level workload model. In addition to the parameters for the average system, we display the parameters of the best fitted distribution for each separate system and model characteristic.

Similarly to the average workload models built for other types of systems [49], 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 (the 'Avg' row), performance analysis results obtained by different researchers can be compared.

system	lognormal	exp	normal	Weibull	hyperexp	gamma
das2	0.19	0.32	0.16	**0.15**	0.39	0.18
auvergrid	0.26	0.41	0.22	**0.22**	0.51	0.25
ngs	0.23	0.39	0.20	**0.19**	0.49	0.22
lcg	0.12	0.23	**0.08**	0.09	0.23	0.09
sharcnet	0.24	0.33	0.24	**0.22**	0.38	0.22
grid5000	0.22	0.37	0.26	0.24	0.44	**0.21**
nordugrid	0.16	0.30	0.14	**0.12**	0.33	0.14
average	0.20	0.34	0.18	**0.18**	0.40	0.19

Table 4.15: Goodness-of-fit test results for BoT sizes for all seven traces

	lognormal		exp	normal		Weibull		hyperexp			gamma	
system	mean	sd	rate	mean	sd	shape	scale	p	rate1	rate2	shape	rate
das2	0.76	0.61	0.39	1.58	2.57	1.75	2.91	1.00	0.27	0.72	2.89	1.12
auvergrid	0.13	0.53	0.75	0.82	1.33	1.78	1.51	1.00	0.32	1.04	3.42	2.57
ngs	0.20	0.51	0.72	0.74	1.40	2.02	1.58	1.00	0.36	0.97	3.97	2.85
lcg	0.84	0.62	0.37	1.33	2.71	2.16	3.06	0.75	0.38	0.38	3.28	1.21
sharcnet	0.28	0.68	0.58	1.43	1.71	1.37	1.89	1.00	0.19	0.68	2.08	1.22
grid5000	0.19	0.61	0.66	1.18	1.51	1.46	1.68	1.00	0.23	0.83	2.47	1.64
nordugrid	0.47	0.61	0.52	1.14	1.92	1.80	2.17	1.00	0.29	0.57	2.91	1.51

Table 4.16: Fitted distribution parameters for BoT sizes for all seven traces

4.5 Discussion

Performance analysis of grid systems requires knowledge of the workload of such systems. We have chosen to use a workload model to characterize the workload of grid systems, which gives us a better understanding of the characteristics of grid workloads and allows us to analyze the impact of various aspects of workloads on grid systems.

Our workload model is based on real-world grid workload traces, which makes it useful to create synthetic and realistic traces. Our workload model is flexible, as we present a family of workload models, with different levels of modeling. With the task-level, BoT-level and user-level modeling, researchers can use the specific characteristics they need.

We have used proven workload modeling methods, the use of probability distributions, and tailored this method to grid-specific needs. First, by using workload data that originates from grid systems, that are significantly different from parallel supercomputing workloads. Second, by explicitly including a BoT-level model, as BoTs make up an important part of grid workloads.

We explored two different modeling methods for the user-level modeling, concluding that explicit modeling of dominant users with probability distributions is problematic and does not yield good fitting results. The implicit user-level modeling did lead to good results, it is our main choice for modeling user behaviour.

system	lognormal	exp	normal	Weibull	hyperexp	gamma
das2	0.11	0.12	0.09	**0.06**	0.12	0.07
auvergrid	0.15	0.18	**0.07**	0.10	0.35	0.12
ngs	0.08	0.16	0.13	0.08	0.21	**0.06**
lcg	0.07	0.34	0.08	0.07	0.27	**0.07**
sharcnet	0.09	0.20	0.08	**0.06**	0.36	0.07
grid5000	0.10	0.17	**0.06**	0.07	0.15	0.07
nordugrid	0.11	0.28	**0.05**	0.05	0.36	0.09
average	0.10	0.21	0.08	**0.07**	0.26	0.08

Table 4.17: Goodness-of-fit test results for BoT runtime variance for all seven traces

	lognormal		exp	normal		Weibull		hyperexp			gamma	
system	mean	sd	rate	mean	sd	shape	scale	p	rate1	rate2	shape	rate
das2	1.95	0.85	0.11	5.69	8.90	1.64	10.21	1.00	0.12	0.15	2.03	0.22
auvergrid	2.25	0.82	0.09	6.41	11.73	1.91	13.41	3.30	0.05	1.03	2.30	0.19
ngs	1.51	0.90	0.17	4.58	6.06	1.40	6.76	1.00	0.07	0.20	1.79	0.29
lcg	2.45	0.46	0.08	4.87	12.63	2.85	14.21	4.33	0.06	0.12	5.76	0.45
sharcnet	2.30	0.70	0.08	6.54	12.04	1.93	13.65	2.72	0.04	1.02	2.75	0.23
grid5000	2.04	0.82	0.10	5.54	9.56	1.79	10.90	2.03	0.10	0.70	2.25	0.23
nordugrid	2.59	0.50	0.07	5.72	14.76	2.86	16.58	6.05	0.04	1.01	5.14	0.35

Table 4.18: Fitted distribution parameters for BoT runtime variance for all seven traces

Using the modeling results for seven workload traces, we defined a theoretical "average" system, for which we presented the distribution and respective parameters that represent this system. This workload model instance can be used to generate realistic synthetic workloads, without resorting to the specifics of a single trace. We use this workload model in our performance analysis of different scheduling models for BoTs in grid systems.

system	alpha	N
DAS-2	1.25	333
AuverGrid	N/A	N/A
NGS	1.30	379
SHARCNET	1.25	412
LCG	1.32	216
Grid5000	1.39	481
NorduGrid	1.36	387

Table 4.19: Parameters for the fitted Zipf distribution on the user ranking data

trace	# dominant users
das2	4
auvergrid	-
ngs	4
lcg	3
sharcnet	5
grid5000	3
nordugrid	3

Table 4.20: Number of dominant users for each analyzed trace

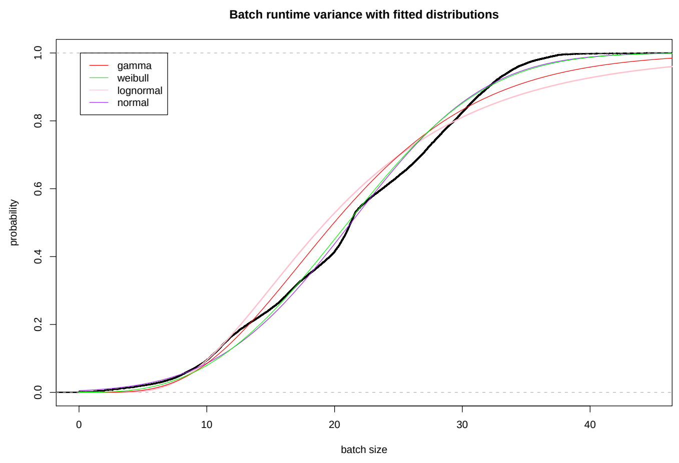


Figure 4.8: Cumulative distribution function for BoT runtime variance in NorduGrid with fitted distributions

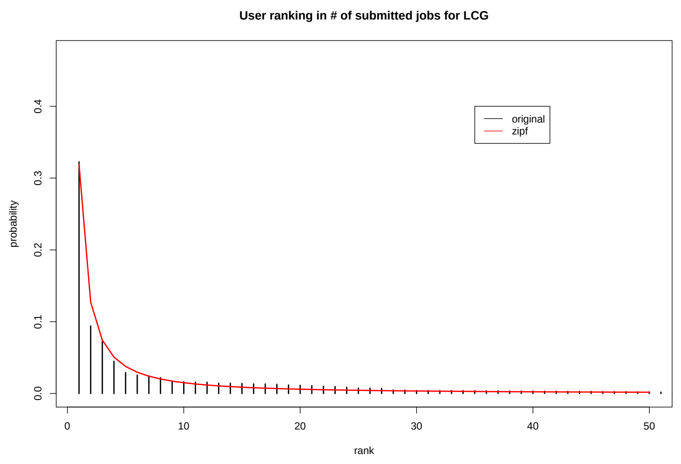


Figure 4.9: The user ranking data for the LCG system with fitted Zipf distribution

Grid'5000		Users				
	dist.	All	U584 (25%)	U30 (10%)	U414 (10%)	U-other (55%)
Requested runtime	lognormal	0.32	0.54	0.51	0.53	0.30
Runtime	normal	0.07	0.25	0.15	0.38	0.05
CPUs	Weibull	0.42	n/a	0.53	0.58	0.34
BoT sizes	Weibull	0.24	0.19	0.20	0.09	0.25
BoT interarrival	normal	0.09	0.13	0.03	0.06	0.09
BoT daily cycle	normal	0.05	0.05	0.08	0.08	0.05
BoT task interarrival	normal	0.21	0.20	0.39	0.25	0.22
Runtime variability	normal	0.06	0.14	0.10	0.13	0.07

Table 4.21: Kolmogorov-Smirnov goodness-of-fit test results for trace data for each dominant users in the Grid'5000 system

DAS-2		Users					
	dist.	All	U66 (18%)	U84(13%)	U239(8%)	U173(8%)	U-other (53%)
Requested runtime	lognormal	0.08	0.30	n/a	0.54	n/a	0.39
Runtime	normal	0.04	0.10	0.19	0.07	0.40	0.06
CPUs	Weibull	0.24	0.19	0.46	0.47	0.12	0.22
BoT sizes	Weibull	0.15	0.08	0.29	0.25	0.12	0.18
BoT interarrival	normal	0.07	0.07	0.09	0.12	0.08	0.06
BoT daily cycle	normal	0.04	0.05	0.04	0.07	0.08	0.04
BoT task interarrival	normal	0.10	0.11	0.46	0.13	0.30	0.09
Runtime variability	normal	0.09	0.12	0.16	0.12	0.33	0.10

Table 4.22: Kolmogorov-Smirnov goodness-of-fit test results for trace data for each dominant users in the DAS-2 system

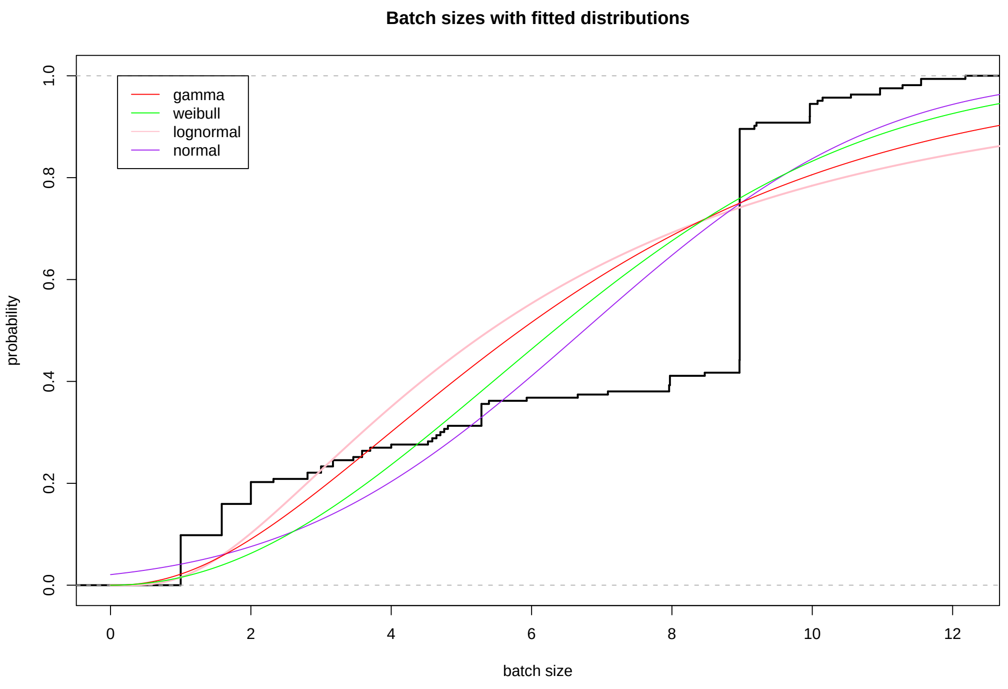


Figure 4.10: Explicit user modeling for BoT sizes for a dominant user in the NorduGrid trace.

Trace	User	Bag-Of-Tasks			
ID	Ranking	IAT	Daily Cycle	Size	Task IAT
DAS-2	Z(1.25,333)	W(4.06,7.91)	G(2.62,0.13)	W(1.75,2.91)	N(2.08,3.01)
AuverGrid	n/a	W(3.87,7.33)	W(1.72,25.49)	W(1.78,1.51)	E(0.40)
NGS	Z(1.30,379)	W(4.94,8.48)	W(1.64,25.42)	W(2.02,1.58)	W(3.28,3.22)
LCG	Z(1.32,216)	W(4.05,6.48)	W(1.71,23.86)	N(1.33,2.71)	N(0.95,2.35)
SharcNet	Z(1.25,412)	W(4.17,7.65)	W(2.44,28.99)	W(1.37,1.89)	N(1.17,0.68)
Grid'5000	Z(1.39,481)	W(4.27,8.42)	W(1.57,20.54)	G(2.47,1.64)	N(1.34,1.33)
NorduGrid	Z(1.36,387)	N(1.97,8.00)	W(1.62,22.18)	W(1.80,2.17)	N(1.30,2.00)
Avg	Z(1.31,368)	W(4.25,7.86)	W(1.79,24.16)	W(1.76,2.11)	N(3.11,1,89)

Table 4.23: The parameter values for the best fits of the statistical distributions to the BoT model for the seven studied traces. N, LN, W, and G stand for the normal, lognormal, Weibull, and gamma distributions, respectively. Z stands for the Zipf distribution with two parameters: α and the number of unique users (ranks).

Trace	Task-level				
ID	Task Runtime	Runtime var.	% Parallel tasks	% Power-of-2	Task size
DAS-2	N(1.78,3.87)	W(1.64,10.21)	0.34	0.75	N(1.14,2.03)
AuverGrid	G(3.55,0.47)	N(6.41,11.73)	1.00	n/a	n/a
NGS	N(3.50,3.51)	G(1.79,0.29)	0.95	0.91	N(1.17,2.63)
LCG	LN(1.82,0.34)	W(2.85,14.21)	0.90	n/a	n/a
SharcNet	N(3.06,7.45)	W(1.93,13.65)	1.00	0.71	W(2.07,3.52)
Grid'5000	N(2.31,4.97)	N(5.54,9.56)	0.54	0.84	W(1.30,1.97)
NorduGrid	N(2.76,9.04)	W(2.86,16.58)	0.99	0.64	G(4.26,1.44)
Avg	N(2.73,6.1)	W(2.05,12.25)	0.94	0.77	W(2.02,2.82)

Table 4.24: The parameter values for the best fits of the statistical distributions to the BoT model for the seven studied traces. N, LN, W, and G stand for the normal, lognormal, Weibull, and gamma distributions, respectively.

Chapter 5

A Model for Analyzing the Performance of Scheduling Bags Of Tasks in Grids

For our performance analysis, we want to investigate the impact of the workload, the scheduling policies and the resource management architecture. Our workload model gives us a way to characterize workloads, this chapter deals with scheduling policies and resource management architectures. We answer the research question of how we can evaluate grid scheduling policies on different architectures. We do this by choosing a performance analysis method, simulations, and proposing a system and scheduling model for BoTs in grids on which we base our simulations. This model for analyzing the performance of scheduling BoTs in grids is the answer to our research question: How can we evaluate grid scheduling policies on different architectures?

In this chapter, we propose a performance analysis method that realistically and systematically evaluates scheduling BoTs of jobs in grids. In Section 5.1 we discuss three performance analysis methods to clarify our choice for using simulation in this performance analysis. In Section 5.2 we present our scheduling model, system and job model, the scheduling policies and resource management architectures we are going to evaluate using simulation. In Section 5.3 we review related work on this subject. In the next chapter we present the results of executing the performance analysis. Parts of this chapter - the work on our scheduling model - have been published in a paper, which was submitted to and accepted by the ACM International Symposium on High Performance Distributed Computing 2008 [40].

5.1 Performance analysis of grid systems

Performance analysis of grid systems can be performed with the following methods: mathematical modeling, experiments on real systems, and simulations of grid systems. We have opted for simulations for our research. For each method, we discuss the advantages and disadvantages of grid systems, with the goal of motivating our choice for simulations for our performance analysis.

5.1.1 Mathematical modeling

A computational grid is a large and complex system, consisting of numerous software and hardware components, located at different layers in the architecture. Due to this complexity, it is difficult to determine the effects of changes in the configuration of the system to the performance of the grid.

A mathematical model of a grid system would make it possible to determine the effects of changing one aspect of the system on the performance of the grid. However, a mathematical modeling method for such large and complex systems has not been found yet. Therefore, grid components are usually evaluated by experiments, either on real systems or on simulated systems.

5.1.2 Experiments on real systems

The information in workload traces can be used in performance analysis of grid system. Let's assume we have a method to characterize the workload representatively for a given system. We can evaluate a grid system by generating a high number of synthetic jobs, based on our characterization method, and submitting this to the actual system. The GrenchMark framework [35] is an example of a tool that enables the submission of synthetic workloads to real grid systems for performance analysis.

Experiments on real systems have the advantage that the analysis includes all the complexities of the actual system, even if they are hidden. Nevertheless, in order to obtain repeatable and consistent experimental results, it is necessary to have a configurable environment, and to eliminate the influence from other users of the system. These requirements could be very costly and inconvenient, especially if the grid in question is a production system. Moreover, experimental results would only hold for the used system, so generalization of the results to systems with other hardware or software architectures is not possible.

5.1.3 Simulation of grid systems

Software simulation of grid systems offers a effective way to quickly analyze the performance of grid systems. It is not necessary to occupy a real grid, which may be costly, and the virtual system is more flexible and time efficient, as simulations can be performed faster than real-time.

Simulations have the advantage that different scenarios can be used easily, which makes it possible to evaluate several system configurations, or in our case resource management architecture and scheduling policies. It is also possible to speed up the experiments, because we are not bound by real-life hardware or network delays, and because simulations usually work with a simplified system and scheduling model.

Simulation has the disadvantage that the analysis results may be inaccurate, because not every aspect of the real system can be modeled. However, results of the simulation can be validated by running experiments on real systems afterwards.

Because of the advantages over experiments on real systems, we will use simulation for our research in this work. In Section 5.2 we present our system and scheduling model, that allow us to analyze various scheduling policies and resource management architectures in our simulations.

5.2 A scheduling model for bags of tasks

In this section we present a scheduling model for BoTs of tasks in grid computing systems consisting of four components. In Section 5.2.1, we describe the models of the system and of the tasks submitted to it. The system model considers clusters of resources. In Section 5.2.2 we present the resource management architectures, which specify the way clusters and their resources are organized in a grid system.

Deciding which tasks to run where is in our model a two-step process: first from the waiting tasks in the system an eligible set is created using one of the task selection policies (Section 5.2.3), and then the tasks from the eligible set are mapped to resources using one of the task scheduling policies (Section 5.2.4).

Our model extends the current state-of-the-art in two ways. First, the resource management architecture and the task selection policies have not been explicitly included in previous BoTs scheduling models.

Second, for the task scheduling policies, previous models [8,45] have considered that the resource performance (e.g., speed, SPECInt2006 value [60]) or the task runtime are not (accurately) known by the scheduling policy, but that at least one of them is accurately known. In contrast to these models, our model considers that at the same time the information about both the processor performance and the task runtime may be inaccurate or even missing.

5.2.1 System and task model

In our model of grid computing systems we assume that the computing resources (processors) are grouped in clusters. The processors may have different performance across clusters, but within the same cluster they are homogeneous. The workload of the system consists of tasks submitted by various users; each of the tasks is a bag of sequential tasks (possibly only one).

We employ the SPEC CPU benchmarks model for application execution time [60], that is, the time it takes to finish a task is inversely proportional to the performance of the processor it runs on (see also Section 6.1.1).

Upon their arrival into the system the tasks are queued, waiting for available resources on which to be executed. 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.

5.2.2 Resource management architectures

The complete set of clusters is operated as one grid computing system using a resource management architecture which dictates how tasks are distributed across the resources in the system. In our model, each of the clusters has its own local resource manager (RM), but other RMs may be employed to build a complete architecture. A global resource manager (GRM) is an RM that can submit tasks for execution to another RM.

Each RM in the system uses the same procedure upon the arrival of new BoTs, on the completion of tasks, and also periodically. The scheduling procedure is as follows. First, the RM calls the task-selection policy, which selects from the RM's current queue the eligible set of tasks. Then, the RM executes the eligible set using the task-scheduling policy, which in turn sorts the eligible set and/or ranks the resources to create a schedule. Only after all the tasks in the schedule are completed is a new eligible set generated.

In this work we use three resource management architectures, one based on independent clusters, one centralized, and one decentralized. Below we describe these architectures:

1. **SEParated Clusters (`sep-c`)** Each cluster operates separately with its own local RM and its own local queue to which tasks arrive. Each user can submit tasks to exactly one RM.

2. **Centralized Scheduler with Processor monitoring (`csp`)** Each cluster operates separately with its own local RM and its own local queue to which tasks arrive. In addition, a global RM with a global queue operates on top of the cluster RMs. The users submit tasks only to the global system queue. When the global RM observes that a cluster has idle resources, it moves some of the tasks on the global queue to that cluster. The information about the number of free processors is gathered periodically by a monitoring service.

3. **Condor-like, with Flocking (`fcondor`)** This models a Condor-like architecture with flocking capabilities [19]. Similarly to `sep-c`, each cluster operates separately with its own local RM and its own local queue to which tasks arrive. However, here each user can submit tasks to any RMs. A user keeps submitting tasks to the same RM while that RM starts the tasks immediately; when tasks start to be queued, the user will switch to another RM, in round-robin order.

In our model, a GRM may submit at most one task at a time to another RM; the target RM will in this case receive BoTs with one task. This ensures that the GRM can control the order in which its tasks are considered at the remote RM. Note that for `sep-c` there is no GRM, but there exist only local users.

5.2.3 Task selection policies

The RM uses its task selection policy to select from the RM's local queue the eligible set of tasks. The task scheduling policy is applied to this set of tasks to decide when and where to execute each task.

We investigate in this work seven task selection policies, the first two of which do not take into account the user who submits a task:

1. **`S-T`** The Select-Tasks policy selects all the tasks in the system. For each arriving BoT, the set of its tasks is added to the eligible set.

2. **`S-B`** The Select-BoTs policy selects BoTs of tasks in the order of their arrival. When the eligible set has become empty, the set of tasks of the selected BoT are considered the new eligible set.

3. **`S-U-Prio`** The Select-User-Priority policy assumes that each user in the system has a unique priority. It selects all the tasks of the user with the highest priority. Ideally, each user has a unique priority, which distinguishes the user's resource usage rights from any other's. In practice, a system will be configured with just a few (e.g., up to four) distinct priorities. We use in this work the ideal scenario; in many settings, its performance gives an upper bound of the performance of the practical scenario.

4. **`S-U-T`** The Select-User-Tasks policy aims at the equal sharing of resources among the system users: It first selects the user with the lowest resource consumption, and then it selects all the tasks of the selected user.

5. **`S-U-B`** Similarly to `S-U-T`, the Select-User-Batch policy aims at the equal sharing of resources among the system users: It first selects the user with the lowest resource consumption, then it orders the selected user's BoTs in their order of arrival into an ordered set. From this set, the tasks of the first BoT are considered the new eligible set.

6. **`S-U-GRR`** The Select-User-Global-Round-Robin algorithm selects the next user in round-robin order. It then adds all waiting tasks of this user to the eligible set. If a user submits additional BoTs during his turn, the corresponding tasks will not be considered for selection during this turn.

7. **`S-U-RR`** The Select-User-Round-Robin policy is a variation of S-U-GRR, where only one task is selected per user at each round. Under S-U-RR, a BoT of size N will receive complete service after exactly N rounds. This algorithm is similar to the WFQ algorithm for scheduling packets over the network [55], with the main difference of tasks not having a known runtime at selection time, as opposed to network packets having a pre-assigned number of bits to transfer.

		Task Info.		
		K	H	U
Resource Info.	K	ECT [50], FPLT [51] MaxMin [8]	ECT-P*	FPF*
	H	DFPLT [13] MQD [45]	-	-
	U	STFR*	-	RR [30] WQR [13]

Table 5.1: An information availability framework. K, H, and U stand for information known a-priori, based on historical data, and unknown, respectively. The scheduling policies marked with $\star$ have not been previously studied in the context of BoT scheduling.

For all task-selection policies that require it, the resource consumption is computed as the sum of the past usage and usage of currently running tasks. The usage of a task is computed as the CPU time spent by it until its completion (the current moment) for tasks that have (not yet) completed. Given the range of the runtimes of the tasks in our traces and experiments, a precision on the order of seconds is enough for computing and accurately measuring these CPU time values.

5.2.4 Task scheduling policies

There exist many scheduling policies for workloads with BoTs in distributed systems [8, 13, 30, 45]. We categorize such policies according to the information policy used for resources and tasks, where a piece of information can be either fully Known (`K`), known from Historical records (`H`), or fully Unknown (`U`). In this work we consider only two pieces of information (the *information set*): the performance of a processor and the execution time of a task (e.g., on a reference processor).

A task-scheduling policy can be characterized in terms of its information usage by a tuple (R, T), with R and T the information policy for resources and for tasks, respectively. We map several scheduling policies to this characterization in Table 5.1. Most of the existing scheduling policies are either (U, U) or (K, K); in practice, Condor [61] uses by default an (U, U) policy, AppLeS supports several (U, U) and (K, K) heuristics [5], MyGrid implements an (U, U) policy [12]. There are no scheduling policies of types (U, K), (K, U), (U, H), (H, U), and (H, H).

For evaluating BoT scheduling in grids, we propose one simple policy for each of the types (U, K), (K, U), and (K, H). The (U, K) and (K, U) policies give an upper bound of the achievable performance of (U, H) and (H, U) policies, respectively. The scheduling policies used in this work are described below:

1. **ECT** (K, K) The Earliest Completion Time policy assigns each task to the resource (cluster or processor) that leads to the earliest completion time possible. If the resource is a cluster, it also takes into account the cluster's queue when computing the earliest completion time. It is a Gantt chart-based

scheduling policy [9]; other examples of similar policies include MaxMin, MinMin, and (X)Sufferage [8].

2. FPLT (K, K) The Fastest Processor Largest Task policy assigns the largest task to the fastest processor available.

3. RR (U, U) The Round-robin Replication policy first assigns all the tasks to processors, in the initial order of the eligible set. After finishing all tasks in the eligible set, it replicates tasks at most once on the resources that become available, in round-robin order.

4. WQR (U, U) The Work Queue with Replication policy differs from RR in that it can replicate tasks several times instead of only once. The number of replicas is appended to name of the scheduling policy, e.g., WQR-1 replicates tasks once (and is identical to RR).

5. DFPLT (H, K) The Dynamic Fastest Processor Largest Task policy assumes that the resource performance is dynamic over time. On the completion of a task, the performance of the resource on which the task was executed is (re-)computed, and the resource receives a performance rank. This policy assigns the largest task to the resource with the highest performance rank.

6.ECT-P (K, H) The ECT with task runtime Prediction policy operates similarly to ECT, but uses predicted instead of real task runtime values.

7. STFR (U, K) The Shortest Task First with Replication always assigns the shortest task first. After finishing all tasks in the eligible set, it replicates tasks at most once on the resources that become available, in round-robin order.

8. FPF (K, U) The Fastest Processor First assigns the tasks in the initial order of the eligible set. Each task is assigned to the fastest available processor.

The information set can be extended to more dimensions than just two. However, the two selected pieces of information already foster non-trivial customization, e.g., the execution time of a task can be extended to include the task setup and removal. If the task execution model of the system does not allow for the decoupling of task data from the task execution, as is the case for many cluster managers used in practice, the execution time of a task can also include the data transfer time to/from the execution place. Similarly, the execution time can include the setup of a virtual environment that is needed for executing a task.

5.3 Related work

We have discussed throughout the text the research most closely related to ours for workload modeling [48, 49] and for design of batch scheduling policies [8, 13, 30, 45]. For the latter, the policies have been previously evaluated through simulation in independent clusters environments (i.e., `sep-c` in Section 5.2) using a per-batch scheduling algorithm (i.e., algorithm S-B), but the workload did not resemble the workloads of real grids. Closest to our work, Casanova et al. [8] compare the performance of several BoT scheduling policies in a `sep-c` environment. They also present one class of selection policies (`S-B`). In comparison with these results, our work focuses on the systematic evaluation of batch scheduling in grids (resource management architecture, scheduling algorithm, scheduling policy), and on using realistic workloads.

Chapter 6

Simulations of Scheduling Bags-Of-Tasks in Grids

In previous chapters, we have presented our workload model, with which we can generate synthetic realistic workloads with varying characteristics. We have also presented a system and scheduling model for simulations of scheduling BoTs on different resource management architectures, task selection and task scheduling policies. We are ready to answer the research question: what is the performance of grid scheduling policies under different realistic workloads? We answer this question by simulating the submission and processing of tasks onto three different grid resource management architectures.

In this chapter, we discuss the results of simulating the scheduling of BoTs in grids. In Section 6.1 we discuss our experimental setup, which simulator we used, the types of workloads and our simulation assumptions. In Section 6.2 we present the results of the simulation, by discussing the impact of task scheduling policies, task selection policies, workload characteristics and the resource management architecture. The simulation results in this chapter have been published in a paper, which was submitted to and accepted by the ACM International Symposium on High Performance Distributed Computing 2008 [40].

6.1 The experimental setup

This section describes the setup of the performance analysis experiments for which the results are presented in Section 6.2.

6.1.1 The simulator

The experiments are performed in a simulated environment consisting of two multi-cluster environments, the DAS and Grid'5000 grids, for a total of 20 clusters and over 3500 processors, corresponding to an existing grid environment in production. The simulation environment is the DGSim [33, 41] discrete event simulator for grids and for other large-scale distributed computing environments. The reason for using DGSim is threefold. First, DGSim already provides much of the simulated environment used in this work, e.g., the combined DAS and Grid'5000 system, the resource management architectures (see Section 5.2.2). Second, the scheduling component of DGSim can be easily extended with the features required for this work. DGSim was extended for this paper with the tasks-selection and tasks-scheduling policies described in Section 5.2. In addition, DGSim was extended to support heterogeneous processing speeds; the processing speed of the resources used in simulation correspond to the SPECInt2006 values [60] of the real DAS and Grid'5000 resources, and their relative performance ranges between 1.0 and 1.75. Third, when compared to the previous simulation tools,

DGSim reports a variety of performance metrics, such as the utilization, the per-tasks wait and response time, the per-tasks slowdown, the goodput (expressed as the total processing time of the tasks that finish successfully), and the finished tasks (expressed as the percentage of tasks that finish, from the tasks in the workload).

For this research, the simulator was extended to report the following BoT-related metrics:

Makespan (MS), which for a BoT is defined as the difference between the earliest time of submission of any of its tasks, and the latest time of completion of any of its tasks.

Normalized Schedule Length (NSL), which for a BoT is defined as the ratio of its Makespan and the sum of its tasks' runtime on a reference processor. The NSL is the extension of the slowdown used for tasks in traditional computing systems [25]. Lower NSL values are better, in particular NSL values below 1 are desired.

6.1.2 The workloads

Each of the 20 clusters of the combined system receives an independent stream of tasks (input workload). The input workloads used in our experiments are either one month-long traces collected from the individual grids starting at identical moments in time (*real traces*), or synthetic traces that reflect the properties of grid workloads (*realistic traces*). In [33] Iosup et al. argue that most research on scheduling in grids uses unrealistic synthetic traces for experimental purposes. We use throughout this article the formulation *realistic traces* in place of *realistic synthetic traces* to accentuate the difference between the traces used in this work and the unrealistic synthetic traces.

The need for realistic traces is twofold. First, traces coming from one system cannot be used unmodified on a different system [21,29] (e.g., the task submission depends on the original circumstances); the seven traces used in this work originate from seven different grids. Furthermore, modifying the real traces (e.g., by scaling or duplicating their tasks) may lead to input that does not actually represent a realistic trace; scheduling results are highly sensitive to such changes [21]. Second, given the size of the explored design space (see Section 6.2), performing the experiments for each real trace becomes unmanageable. The use of real traces is required for validation purposes, to give evidence that results obtained with real and with realistic traces are similar.

Unless otherwise noted, the realistic traces use the parameter values of the average system given in row ”Avg” of Table 4.24 and Table 4.23. Our workloads have between 20,000 and 175,000 tasks, with an average of over 60,000 tasks (the median is over 64,000 tasks). Thus, our experimental workloads fulfill the realism and the size requirements for the accurate simulation of schedulers [29].

6.1.3 Simulation assumptions

We have made the following five assumptions in the experiments (a full description and motivation for the first four is given in [33]).

Assumption 1: No network overhead First, we assume that the network between the clusters is perfect and has zero latency.

Assumption 2: FCFS scheduling policy at cluster-level Second, because all tasks are sequential, all sites employ the FCFS policy, *without* backfilling.

Assumption 3: Processors as scheduling unit Third, multi-processor machines (which do occur in the DAS and Grid’5000) behave and can be used as sets of single-processor machines without a performance penalty.

Assumption 4: No background load Fourth, all load arrives directly at the RMs of the resource management architecture which is in place, with no tasks bypassing those RMs.

Assumption 5: No resource failures Finally, we assume that there are no resource failures, because in light of the model for cluster-based grids [37] and of the model for desktop grids of Kondo et al. [43], an environment with resource failures is equivalent to a smaller environment, provided that the average resource availability duration is not lower than the runtime of the tasks.

Section	Res.Mgmt. Architecture	Selection Policy	Scheduling Policy	Workload Characteristics	System Load	Information Inaccuracy
Section 6.2.1	`csp`	`S-T`	**all**	real, realistic	∼25%, 20-95%	no
Section 6.2.2	`csp`	`S-T`	all	**synthetic**	60%	no
Section 6.2.3	`csp`	**all**	`FPLT`	realistic	20-95%	no
Section 6.2.4	**all**	`S-T`	`FPLT`	realistic	20-95%	no

Table 6.1: The design space coverage of the experiments presented in this section. The characteristics in bold indicate the main focus of each section.

6.2 The performance of bags of tasks

In this section we present the results of an analysis of the performance of bags of tasks in grids. We cover with over 1200 trace-based simulations a design space with five axes and more than 2 million points: We consider 7 task-scheduling policies, various parameters of our workload model with 7 number of values of each, 8 task-selection policies, and 3 resource management architectures. The simulations are performed using 3 types of workloads (real, realistic, synthetic), for 7 system utilization levels.

To explore this design space efficiently, we perform a set of experiments for each parameter the impact on performance of varying the parameter along one of the axes of the design space. Table 6.1 shows an overview of what each experiment covers. The five experiments presented in this section are aimed at evaluating the impact on performance of:

- The task scheduling policy
- The workload characteristics
- The task selection policy
- The resource management architecture

Most of the results present average values of the BoT MS and/or NSL metrics under various system loads, and as such are mostly useful to the system administrator or to the user submitting a workload with characteristics closely match those of the average workload. However, Sections 6.2.1 and 6.2.2 present detailed results for different BoT sizes and task runtimes, therefore they are also useful to understanding the performance perceived by users whose workload characteristics are significantly different than the average workload.

6.2.1 The impact of the task scheduling policy

In this section we assess the impact on performance of the scheduling policy using the setup described in the first line of Table 6.1. Our main findings are summarized below.

Small and Medium-Sized batches have a high NSL even under low load (Figure 6.1) We simulate the task-scheduling policies in a `csp/S-T` system under real load of 25%. Figure 6.1 shows that even with this low load, under `csp/S-T` the batches of sizes 5-9 and 10-19 have a higher than average NSL, while the average MS increases monotonically with the batch size. This indicates that their users get higher than expected wait times for their tasks compared to other system users. The main beneficiaries are the users submitting very small (size 2-4) or very large batches (size over 200).

Comparison of policies with different information types Figure 6.2 shows the performance of the scheduling policies under realistic load varying between

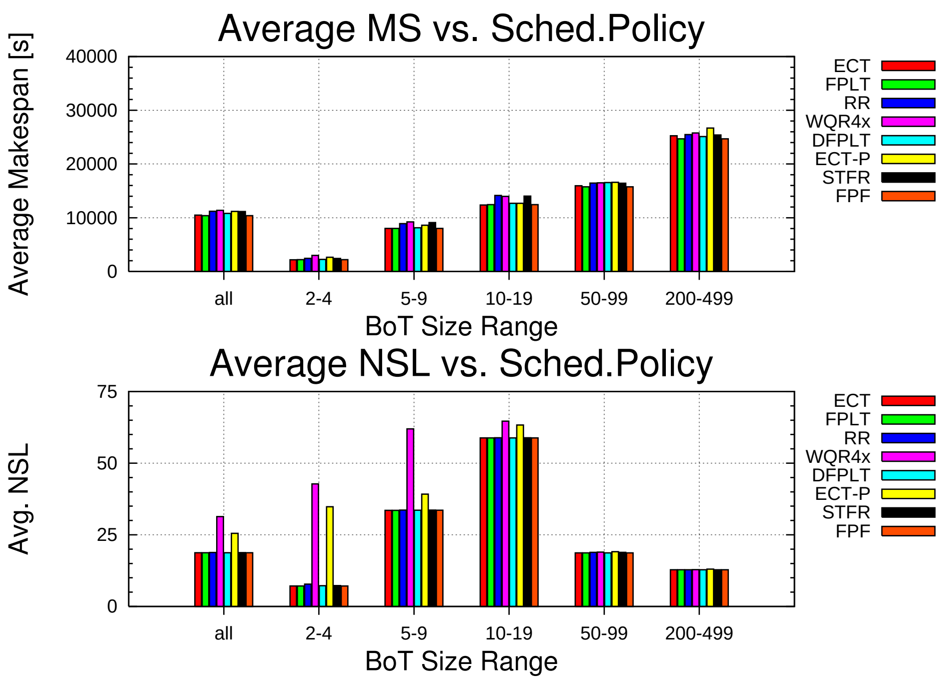


Figure 6.1: The performance of the scheduling policies in large-scale distributed computing systems with `csp/S-T`, under *real* load.

20% and 95%. The (K, K) policies have the best balance between MS and NSL. Moreover, `FPLT` has the lowest overall MS, and `ECT` has the second lowest overall NSL. As the policies that include unknown (U) information models rely on the quality of their uninformed heuristic, their performance ranges from surprisingly good to surprisingly poor. The (U, K) policy `STFR` has the best overall NSL, but average MS. The (K, U) policy `FPF` has the lowest overall MS, but poor NSL. The (U, U) policy `RR` has a good MS, but poor NSL. The `WQR4x` has poor MS and NSL, especially at high load. The prediction-based policies (those that have at least one information model of type H, i.e., `DFPLT` and `ECT-P`), perform consistently worse than the other scheduling policies.

Design guidelines for scheduling policies The transition from information type K to type H can be costly: `FPLT` outperforms `DFPLT` (transition (K, K) to (H, K)), and `ECT` outperforms `ECT-P` (transition (K, K) to (K, H)). However, it may be necessary due to real system requirements. While we have not explored in this paper the complete transition from K via H to U, the comparison of policies with different information types above shows evidence that when changing to information types H and U, it is better to do so for the piece of information that has the lowest variation. In our simulated system, the variation in resource performance is low (i.e., the ratio between the performance of the fastest and the slowest processor is below 2), whereas the task runtime variation is high (reaches values over a thousand). Thus, for the system we simulated, policies of type $(U, *)$ are

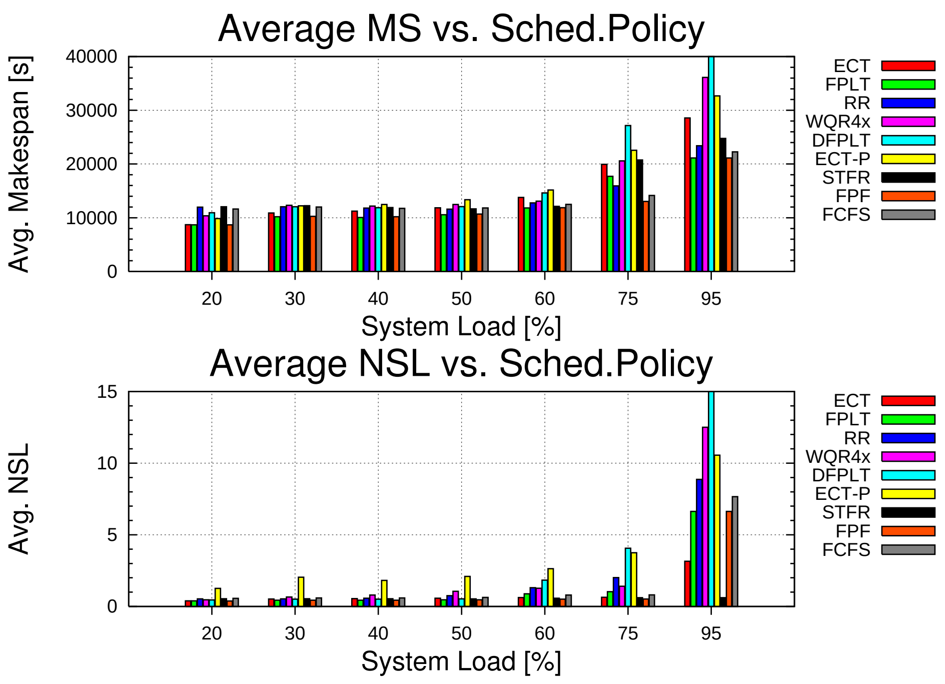


Figure 6.2: The performance of the scheduling policies in large-scale distributed computing systems with `csp/S-T`, under *realistic* load.

preferable.

6.2.2 The impact of the workload characteristics

In this section we assess the impact on performance of the workload characteristics using the setup described in second line of Table 6.1. For each such characteristic, we generate a synthetic workload imposing a system load of 60% using the workload model described in Section 4.4, so that the values of the characteristics have the desired statistical properties, for instance a median BoT size of 25. Our main findings are summarized below.

Burstiness leads to poor performance in `csp/S-T` (Figure 6.3) We vary the BoT arrival pattern with as possibilities a daily cycle based on the workload model (Realistic), the pattern with all BoTs arriving at the beginning of the simulation (”All at T=0”), and the pattern with all BoT interarrival times being equal (Evenly Spread).

Figure 6.3 shows that the extreme of burstiness (”All at T=0”) leads to much higher average MS and NSL values when compared to the other two patterns. The Evenly Spread case is more favorable than the Realistic case for four of the scheduling policies by about 10%, and for `WQR4x` (`FPLT`) even by 20% (48%).

Impact of the BoT size (Figure 6.4) From the realistic workload model proposed in Section 4.4, we vary the BoT size to achieve median values between 10

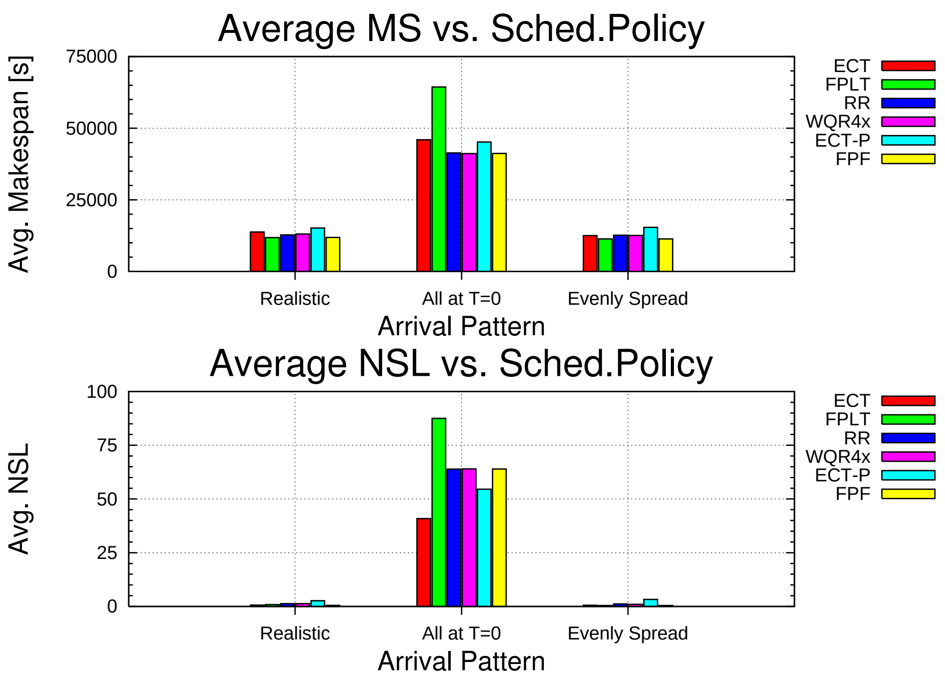


Figure 6.3: The performance of the scheduling policies in large-scale distributed computing systems with `csp/S-T`, under various BoT arrival patterns.

and 50, while keeping the coefficient of variation of the BoT size constant. Figure 6.4 shows that the MS increases with the increase of the median of the BoT size. Relative to a BoT median size of 10, the MS increase ranges from 37% for `ECT` to over 80% for `WQR4x`, which indicates that for some policies, the average MS is less dependent from the BoT size than for others. The average NSL decreases for `ECT-P` and `STFR` with the increase of the BoT size; the other policies do not exhibit this performance improvement trend.

Longer tasks lead to better NSL (Figure 6.5) From the realistic workload model proposed in Section 4.4, we vary the task runtime to achieve mean values between 3 minutes and 3 hours. Figure 6.5 shows that the NSL tends to decrease with the increase of task runtime.

6.2.3 The impact of the task selection policy

In this section we assess the impact of the task-selection policy on the system performance. We use the setup described in the fourth line of Table 6.1: a `csp` resource management architecture, the `FPLT` (K, K) task-scheduling policy, and realistic workloads that subject the system to a load between 20% and 95%. Our main findings are described below.

The task-selection policy is important only in busy systems Figure 6.6 shows that for loads up to 50%, the performance of the system is almost identical for the

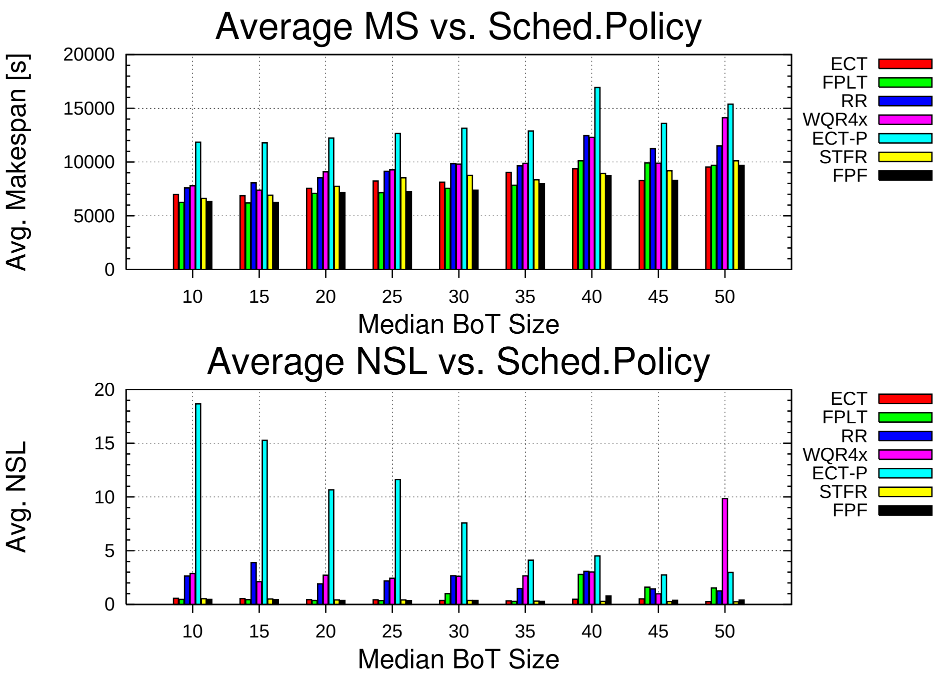


Figure 6.4: The performance of the scheduling policies in large-scale distributed computing systems with `csp`/`S-T`, for various median BoT sizes.

seven task-selection policies. This is not surprising: with a (K, K) scheduling policy, the system resources are harnessed efficiently when much spare capacity exists. The fact that `FLTP` does not use replication is also important in establishing a load of 50% as the threshold beyond which the task-selection policy does have an impact on performance. With a policy that does use replication, this threshold would be lower, in proportion to the average number of replications per task. Figure 6.6 also shows that the task-selection policy has an important impact on performance for loads of 50% and higher; this impact increases with the load.

`S-B` has much better performance than the other task-selection policies Figure 6.7 depicts the NSL of all the task-selection policies for various system load. For loads of 60% and higher, the average NSL (MS) for `S-B` is up to 16 (2) times lower than the average NSL (MS) of the other policies. In particular, `S-B` outperforms the task-selection policy most commonly used in practice, which is `S-T`. We attribute the differences to the design of the `S-B` policy, which greatly favors the small BoTs that are common in the workloads of large-scale distributed computing systems. This is confirmed by the set of columns corresponding to 95% load in Figure 6.7.

System fairness costs: accounting system or performance (Figure 6.6) Introducing fairness into a resource management system in general reduces the aggregate performance of the system. We compare the four task-selection policies studied in this paper that consider fairness: `S-U-T`, `S-U-BoT`, `S-U-GRR`, `S-U-RR`.

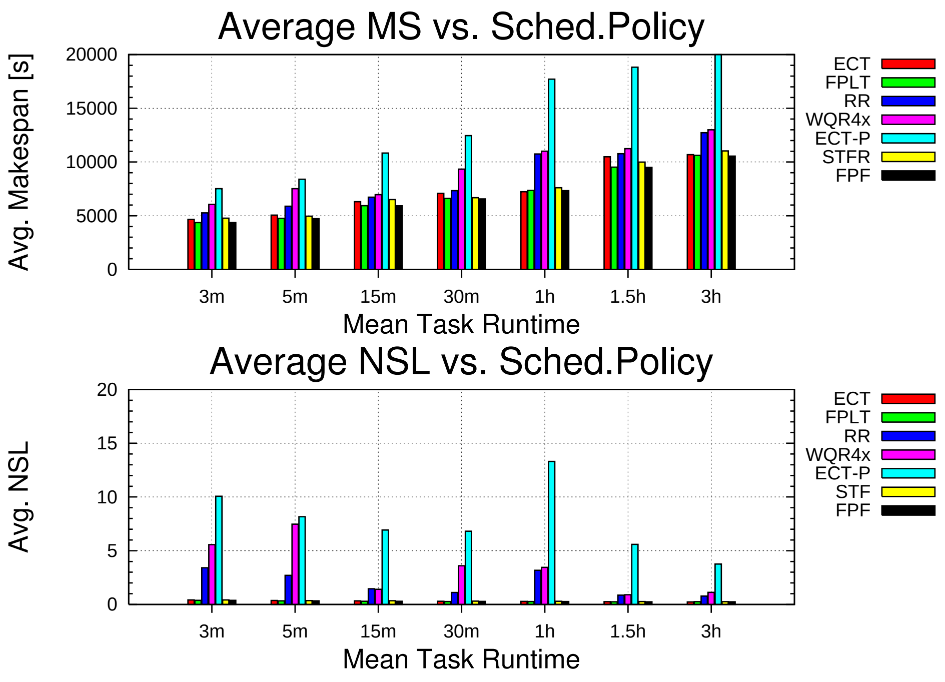


Figure 6.5: The performance of the scheduling policies in large-scale distributed computing systems with `csp`/`S-T`, for various mean task runtimes.

The first three of these may all lead to one user blocking the system for a long time regardless of what the other users do (e.g., when sending one or more large BoTs and getting selected); `S-U-RR` does not suffer from this problem if all the users submit equally large BoTs. While system blocking is a possibility, our study shows that this does not happen in practice often enough to be significant. From the four policies, `S-U-T` performs the best. The ordering of BoTs by arrival time employed by `S-U-BoT` does not lead to better performance. The round-robin ordering of users employed by `S-U-GRR`, which does not require an accounting system to be present, leads in turn to up to 15% higher NSL than `S-U-T`. Finally, `S-U-RR`, which also does not require an accounting system, has up to 45% higher NSL values than `S-U-T`. To conclude, to have fairness a system must either setup an accounting system, or it will pay in lower performance.

System QoS costs: level vs. performance (Figure 6.6) Offering guarantees about the response time of the submitted tasks is expensive in terms of performance in a system that does not support advance resource reservation. We compare the three task-selection policies presented in this paper that support QoS: `S-U-Prio`, `S-U-GRR`, `S-U-RR`. `S-U-Prio` guarantees that the tasks of the user with the highest priority from the users currently having queued tasks will be selected next. `S-U-GRR` guarantees that a user's task will be selected for execution at most n rounds after submission, where n is the total number of users in the system, both with and without queued tasks. `S-U-RR` guarantees that at least one of a user's

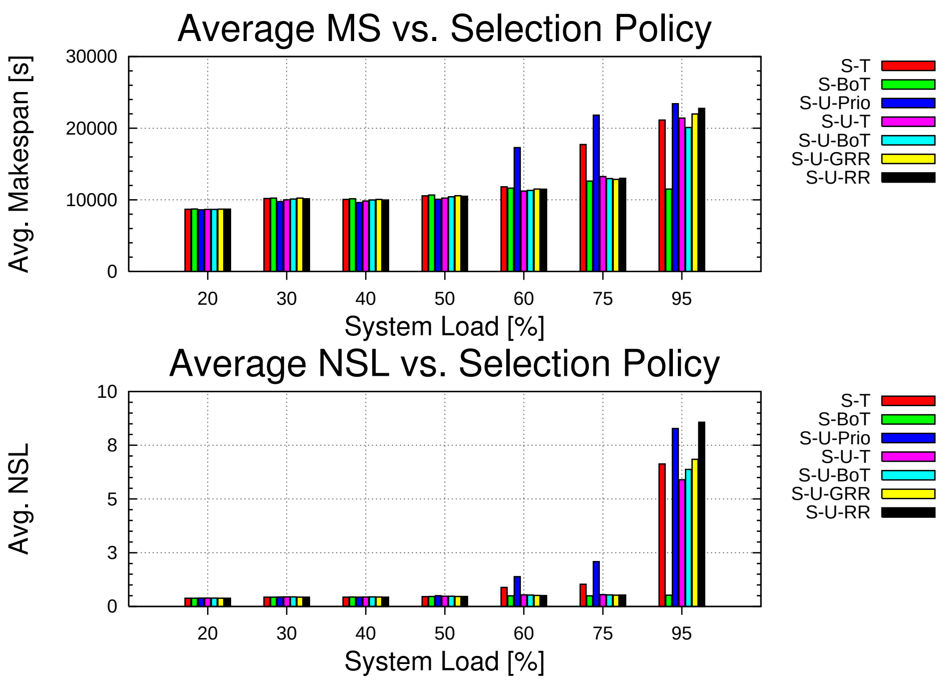


Figure 6.6: The performance of the task selection policies in large-scale distributed computing systems with `csp/FPLT`, under realistic load.

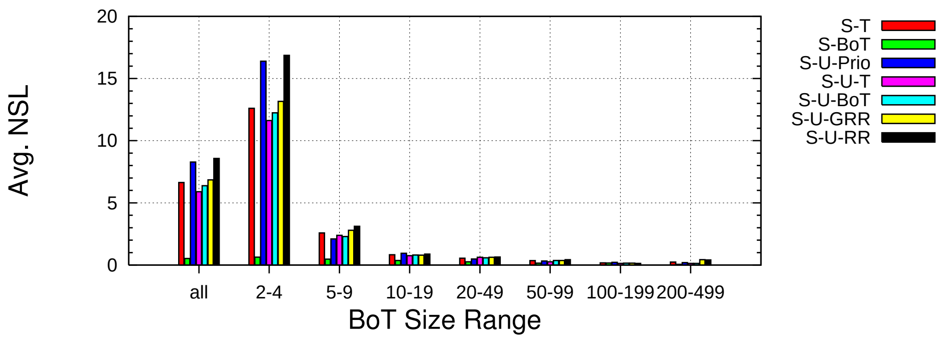


Figure 6.7: The NSL of the task selection policies for different BoT sizes in large-scale distributed computing systems with `csp/FPLT`, when a realistic load of 95% is imposed on the system.

queued tasks is selected in the next n rounds; the rounds of `S-U-RR` are on average much shorter than those of `S-U-GRR`, as `S-U-RR` selects only one task per round. `S-U-Prio` and `S-U-RR` have similar performance. Compared to `S-U-GRR`, their performance is lower, by up to 20%. To conclude, there is a trade-off between the

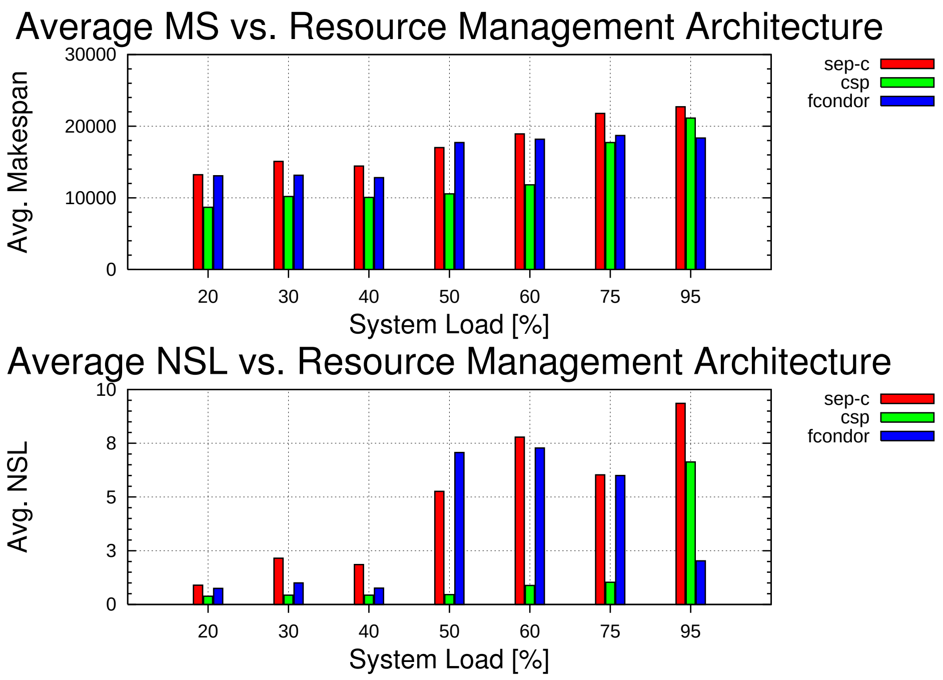


Figure 6.8: The impact of the system resource management architecture on performance under realistic load, with the `S-T()` selection (scheduling) policies.

weak QoS guarantees but higher performance of `S-U-GRR`, and the stronger QoS guarantees but lower performance of `S-U-Prio` and `S-U-RR`.

6.2.4 The impact of the resource management architecture

In this section we assess the impact of the resource management architecture on the system performance. We use the setup described in Table 6.1 (i.e., an `S-T` selection policy, and the `FPLT` (K, K) scheduling policy), and realistic workloads that subject the system to a load between 20% and 95%.

Centralized, separated, or distributed? (Figure 6.8) The centralized policy `csp` achieves the best performance. Conversely, the independent clusters policy `sep-c` achieves the worst performance. The distributed architecture `fcondor` exhibits mixed performance results: expectedly, for loads below 50% its performance is similar to that of `csp`, and, surprisingly, for loads above and including 50% its performance is similar to that of `sep-c`. We observe that for loads above and including 50% `fcondor` and `sep-c` cannot complete all tasks; for the 95% load they complete only 44% and 53% of the tasks, respectively. We attribute the inability of `fcondor` to complete more tasks to its fostering of ”natural competition”: most of the tasks that would get blocked in a central architecture (until other tasks are finished) are submitted in the distributed architecture to the clusters, where they compete with each other for occupying the idle processors.

6.3 Discussion

In this research we have investigated the performance of scheduling BoTs in grids in various situations. We focused our research on three important factors that decide the performance of a grid: the workloads, the resource management architecture, and the scheduling policies. We want to assess the impact of these factors systematically and realistically, so we can increase our understanding of grid performance and providing options to improve it.

We used a systematic approach to simulating the scheduling of BoTs in grid systems, so we can explore the design space of a grid system, from the middleware (scheduling policies) to resource management architecture. With our system and scheduling model, separating the job selection and scheduling policies, we could assess the impact of each separately. In addition, we simulated job submission to three different resource management architectures, that have been used in real production grid systems.

Our workload model, presented in Section 4.4, enabled to use to analyze scheduling performance with realistic, yet synthetic workloads. We were able to generate workloads that resemble real workloads, with the ability to change desired characteristics of BoTs or individual jobs. For instance, we have shown that burstiness in the BoT arrivals leads to poor performance.

Concluding, we have discussed the results for our performance analysis of scheduling BoTs in grids, based on realistic workloads and a novel system and scheduling model, with a systematic approach.

Chapter 7

Conclusions and Future Work

The overall research goal of this thesis is to investigate the performance of scheduling BoTs in grids. In Chapter 1 we formulated a number of research questions that we have answered in the subsequent chapters. We have introduced a new workload model for BoTs in grids, we have validated the model with real-world trace data and we have performed a performance analysis of scheduling BoTs in grids. This section presents the conclusions that we reached after performing this research, and our recommendations for future work on the subject of scheduling BoTs in grids.

7.1 Conclusions

The goal of this research was to gain a better understanding of the performance of grids. Grids are complex, heterogeneous and distributed systems, which makes it difficult to analyze the grid performance. Because of the complex nature of grids, many factors may affect their performance. We chose to focus on three of those factors: workload characteristics, resource scheduling policies, and resource management architecture. We wanted to investigate the impact on the performance of each of these factors, with special attention to BoTs, because they account for a significant part of the workload in grids.

Our investigation on the impact of these factors has led to three major results: a comprehensive workload model for characterizing grid workloads, a performance analysis of scheduling BoTs in grids by simulation, and a performance analysis software toolbox. For each of these three results we present our conclusions on the methods or techniques we used, and on the research results we achieved.

Grid workload modeling We presented a comprehensive and flexible family of workload models, designed to characterize grid workloads. By using three different modeling levels - tasks, BoTs and users - the model is well suited for researchers of grids to use in performance analysis. Our method of capturing important workload characteristics with statistical probability distributions has yielded good results, with the notable exception of the user-level modeling. The model was validated with workload trace data from seven real-world grid environments. We have also used workload traces to synthesize realistic workloads for a theoretical "average" grid system.

Performance analysis by simulations We proposed a systematic approach to analyze the impact of workloads characteristics, task scheduling policies and resource management architecture on system performance. Our grid workload model enabled us to investigate the performance of scheduling BoTs in grids in various scenarios, by generating synthetic yet realistic workloads. This led to new insights in the impact of BoT size, task runtimes and arrival patterns. Using the DGSim grid simulator, we were able to additionally analyze the impact of task selection and scheduling policies, and resource management architecture, in a reasonable amount of time. We concluded amongst others that task selection policies are important in busy systems and that S-B selection policies, where entire BoTs are selected for scheduling in order of arrival, achieve the best performance.

Performance analysis toolbox This research has also resulted in a performance analysis software toolbox that makes it easier for grid researchers and administrators to gain insight into grid workload characteristics. The toolbox provides workload format conversion, analysis, reporting, modeling and generation. Thanks to these tools and the availability of workload trace data from real grids, we were able to create the Grid Workloads Archive, making trace data and analysis reports publicly available to researchers worlwide.

To conclude, we have created a comprehensive workload model that helps researchers to gain a better understanding of grid workload characteristics and can be used to generate synthetic workloads for performance analysis. We have conducted a performance analysis of various task scheduling policies and resource management architectures, of which the results can be used in the design and tuning of grids to increase their efficiency. Finally, we have created a performance analysis toolbox that gives grid researchers and administrators the opportunity to obtain more insight into the workloads of grids.

7.2 Future Work

This work has already introduced new results in the field of scheduling BoTs in grids, but can be extended in the future. In this section we give recommendations for future work.

On the subject of workload modeling, we have chosen to use a select number of well-known probability distributions. For some combinations of workload traces and characteristics, this led to poor fitting results. It is possible that other distributions, are better in representing the workload characteristics of grids, for example Erlang-Coxian distributions [54]. This can lead to better results in fitting the workload model to trace data. Our workload modeling tool makes it relatively easy to try new probability distributions.

On the subject of performance analysis, reliability of grids is another important factor in the performance of a system, which we did not explore in this work. In a large and complex grid, failures of hardware or software components are inevitable, and will likely have a negative impact on the performance. We recommend that future performance analysis studies incorporate availability data, such as that represented in [37, 57] modeled in such a way that it can be used in simulations.

We would also like to evaluate more scheduling policies and resource management architectures than we were able to do in this work. Last, a performance analysis of scheduling in grids should also be carried out on a real grid, in order to validate the results achieved in simulations.

Bibliography

[1] Amazon elastic compute cloud. [Online] Available: `http://aws.amazon.com/ec2/`.

[2] Sun grid compute utility. [Online] Available: `http://sun.com/grid/`.

[3] M. Arlitt and C. Williamson. Trace-Driven Simulation of Document Caching Strategies for Internet Web Servers. *Simulation Journal*, 68(1):23–33, 1997.

[4] Auvergrid. [Online] Available: `http://www.auvergrid.fr/`.

[5] Francine Berman, Richard Wolski, Henri Casanova, Walfredo Cirne, Holly Dail, Marcio Faerman, Silvia M. Figueira, Jim Hayes, Graziano Obertelli, Jennifer M. Schopf, Gary Shao, Shava Smallen, Neil T. Spring, Alan Su, and Dmitrii Zagorodnov. Adaptive computing on the grid using apples. *IEEE Trans. Parallel Distrib. Syst.*, 14(4):369–382, 2003.

[6] Rajkumar Buyya and M. Manzur Murshed. GridSim: a toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing. *Concurrency and Computation: Practice and Experience*, 14(13-15):1175–1220, 2002.

[7] Maria Calzarossa and Giuseppe Serazzi. Construction and use of multiclass workload models. *Perform. Eval.*, 19(4):341–352, 1994.

[8] Henri Casanova, Arnaud Legrand, Dmitrii Zagorodnov, and Francine Berman. Heuristics for scheduling parameter sweep applications in grid environments. In *Heterogeneous Computing Workshop*, pages 349–363, 2000.

[9] Henri Casanova, Graziano Obertelli, Francine Berman, and Richard Wolski. The apples parameter sweep template: User-level middleware for the grid. In *ACM/IEEE Conference on High Performance Networking and Computing (SuperComputing)*, 2000.

[10] Su-Hui Chiang, Andrea C. Arpaci-Dusseau, and Mary K. Vernon. The impact of more accurate requested runtimes on production job scheduling performance. In Dror G. Feitelson, Larry Rudolph, and Uwe Schwiegelshohn, editors, *Proc. of the Job Scheduling Strategies for Parallel Processing, Workshop (JSSPP)*, volume 2537 of *Lecture Notes in Computer Science*, pages 103–127. Springer, 2002.

[11] W. Cirne and F. Berman. A comprehensive model of the supercomputer workload. In *WWC '01: Proceedings of the Workload Characterization, 2001. WWC-4. 2001 IEEE International Workshop on*, pages 140–148, Washington, DC, USA, 2001.

[12] Walfredo Cirne, Daniel Paranhos da Silva, Lauro Costa, Elizeu Santos-Neto, Francisco Vilar Brasileiro, Jacques Philippe Sauvé, Fabrício A. B. Silva, Carla O. Barros, and Cirano Silveira. Running bag-of-tasks applications on computational grids: The mygrid approach. In *Proc. of the Intl. Conf. on Parallel Processing (ICPP)*, pages 407–, 2003.

[13] Daniel Paranhos da Silva, Walfredo Cirne, and Francisco Vilar Brasileiro. Trading cycles for information: Using replication to schedule bag-of-tasks applications on

computational grids. In Harald Kosch, László Böszörményi, and Hermann Hellwagner, editors, *Proc. of the 13th Int'l. Euro-Par Conference on European Conference on Parallel and Distributed Computing*, volume 2790 of *Lecture Notes in Comput. Sci.*, pages 169–180. Springer-Verlag, 2003.

[14] Distributed asci supercomputer 2. [Online] Available: `http://www.cs.vu.nl/das2/`.

[15] Delft grid simulator. [Online] Available: `http://www.st.ewi.tudelft.nl/~iosup/dgsim.php`.

[16] Allen B. Downey. A parallel workload model and its implications for processor allocation. In *Proc. of the ACM/IEEE Int'l. Symposium on High Performance Distributed Computing (HPDC)*, pages 112–140, 1997.

[17] Allen B. Downey and Dror G. Feitelson. The elusive goal of workload characterization. *Performance Evaluation Review*, 26(4):14–29, 1999.

[18] Catalin Dumitrescu and Ian T. Foster. Gangsim: a simulator for grid scheduling studies. In *IEEE/ACM Int'l. Symp. on Cluster Computing and the Grid (CCGrid)*, pages 1151–1158. IEEE Computer Society, 2005.

[19] D.H.J. Epema, M. Livny, R. Dantzig, X. Evers, and J. Pruyne. A worldwide flock of Condors: Load sharing among workstation clusters. *Future Gener. Comput. Syst.*, 12:53–65.

[20] Carsten Ernemann, Baiyi Song, and Ramin Yahyapour. Scaling of workload traces. In Dror G. Feitelson, Larry Rudolph, and Uwe Schwiegelshohn, editors, *Proc. of the Job Scheduling Strategies for Parallel Processing, Workshop (JSSPP)*, volume 2862 of *Lecture Notes in Computer Science*, pages 166–182. Springer, 2003.

[21] Carsten Ernemann, Baiyi Song, and Ramin Yahyapour. Scaling of workload traces. In Dror G. Feitelson, Larry Rudolph, and Uwe Schwiegelshohn, editors, *Proc. of the Job Scheduling Strategies for Parallel Processing, Workshop (JSSPP)*, volume 2862 of *Lecture Notes in Comput. Sci.*, pages 166–182. Springer-Verlag, 2003.

[22] Dror G. Feitelson. Packing schemes for gang scheduling. In Dror G. Feitelson and Larry Rudolph, editors, *Proc. of the Job Scheduling Strategies for Parallel Processing, Workshop (JSSPP)*, volume 1162 of *Lecture Notes in Computer Science*, pages 89–110. Springer, 1996.

[23] Dror G. Feitelson. Metric and workload effects on computer systems evaluation. *IEEE Computer*, 36(9):18–25, 2003.

[24] Dror G. Feitelson and Larry Rudolph. Metrics and benchmarking for parallel job scheduling. In *Proc. of the Job Scheduling Strategies for Parallel Processing, IPPS/SPDP'98 Workshop*, volume 1459 of *Lecture Notes in Comput. Sci.*, pages 1–24. Springer, 1998.

[25] Dror G. Feitelson and Larry Rudolph. Metrics and benchmarking for parallel job scheduling. In *IPPS/SPDP*, volume 1459 of *Lecture Notes in Comput. Sci.*, pages 1–24, 1998.

[26] Ian Foster and Carl Kesselman, editors. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Pub., 1999.

[27] Ian Foster, Carl Kesselman, and Steven Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *Int. J. High Perform. Comput. Appl.*, 15(3):200–222, 2001.

[28] Eitan Frachtenberg and Dror G. Feitelson. Pitfalls in parallel job scheduling evaluation. In Dror G. Feitelson, Eitan Frachtenberg, Larry Rudolph, and Uwe Schwiegelshohn, editors, *JSSPP*, volume 3834 of *Lecture Notes in Computer Science*, pages 257–282. Springer, 2005.

[29] Eitan Frachtenberg and Dror G. Feitelson. Pitfalls in parallel job scheduling evaluation. In Dror G. Feitelson, Eitan Frachtenberg, Larry Rudolph, and Uwe Schwiegelshohn, editors, *JSSPP*, volume 3834 of *Lecture Notes in Comput. Sci.*, pages 257–282. Springer-Verlag, 2005.

[30] Noriyuki Fujimoto and Kenichi Hagihara. Near-optimal dynamic task scheduling of independent coarse-grained tasks onto a computational grid. In *Proc. of the Intl. Conf. on Parallel Processing (ICPP)*, pages 391–398. IEEE CS, 2003.

[31] Grid 5000. [Online] Available: `http://www.grid5000.fr/`.

[32] Grid workloads archive. [Online] Available: `http://gwa.ewi.tudelft.nl/`.

[33] Alexandru Iosup, Dick Epema, Todd Tannenbaum, Matthew Farrellee, and Miron Livny. Inter-operating grids through delegated matchmaking. In *ACM/IEEE Conference on High Performance Networking and Computing (SuperComputing)*. ACM Press, 2007.

[34] Alexandru Iosup, Dick H.J. Epema, C. Dumitrescu, Hui Li, and Lex Wolters. How are real grids used? the analysis of four grid traces and its implications. In *Grid2006: Proceedings of the Grid Computing. 2006 IEEE International Conference on*. IEEE Computer Society Press., 2006.

[35] Alexandru Iosup and Dick H. J. Epema. Grenchmark: A framework for analyzing, testing, and comparing grids. In *IEEE/ACM Int'l. Symp. on Cluster Computing and the Grid (CCGrid)*, pages 313–320, 2006.

[36] Alexandru Iosup, Dick H. J. Epema, Carsten Franke, Alexander Papaspyrou, Lars Schley, Baiyi Song, and Ramin Yahyapour. On grid performance evaluation using synthetic workloads. In Eitan Frachtenberg and Uwe Schwiegelshohn, editors, *JSSPP*, volume 4376 of *Lecture Notes in Computer Science*, pages 232–255. Springer, 2006.

[37] Alexandru Iosup, Mathieu Jan, Ozan Sonmez, and Dick H.J. Epema. On the dynamic resource availability in grids. In *IEEE/ACM Int'l. Conf. on Grid Computing (GRID)*, pages 26–33. IEEE CS, 2007.

[38] Alexandru Iosup, Mathieu Jan, Omer Ozan Sonmez, and Dick H. J. Epema. The characteristics and performance of groups of jobs in grids. In Anne-Marie Kermarrec, Luc Bougé, and Thierry Priol, editors, *Euro-Par*, volume 4641 of *Lecture Notes in Computer Science*, pages 382–393. Springer, 2007.

[39] A. Iosup, H. Li, M. Jan, S. Anoep, C. Dumitrescu, L. Wolters, and D.H.J. Epema. The grid workloads archive. *Future Generation Comp. Syst.*, 24(7), Feb. 2008.

[40] Alexandru Iosup, Omer Ozan Sonmez, Shanny Anoep, and Dick H. J. Epema. The performance of bags-of-tasks in large-scale distributed systems. In Manish Parashar, Karsten Schwan, Jon B. Weissman, and Domenico Laforenza, editors, *IEEE Int'l. Symp. on High Performance Distributed Computing (HPDC)*, pages 97–108. ACM Press, 2008.

[41] Alexandru Iosup, Omer Ozan Sonmez, and Dick H. J. Epema. Dgsim: Comparing grid resource management architectures through trace-based simulation. In Emilio Luque, Tomàs Margalef, and Domingo Benitez, editors, *Euro-Par*, volume 5168 of *Lecture Notes in Computer Science*, pages 13–25. Springer, 2008.

[42] Joefon Jann, Pratap Pattnaik, Hubertus Franke, Fang Wang, Joseph Skovira, and Joseph Riordan. Modeling of workload in mpps. In Dror G. Feitelson and Larry Rudolph, editors, *Proc. of the Job Scheduling Strategies for Parallel Processing, Workshop (JSSPP)*, volume 1291 of *Lecture Notes in Computer Science*, pages 95–116. Springer, 1997.

[43] Derrick Kondo, Gilles Fedak, Franck Cappello, Andrew A. Chien, and Henri Casanova. Characterizing resource availability in enterprise desktop grids. *Future Gener. Comput. Syst.*, 23(7):888–903, 2007.

[44] Lhc computing grid. [Online] Available: `http://lcg.web.cern.ch/`.

[45] Young Choon Lee and Albert Y. Zomaya. Practical scheduling of bag-of-tasks applications on grids with dynamic resilience. *IEEE Trans. Computers*, 56(6):815–825, 2007.

[46] Arnaud Legrand, Loris Marchal, and Henri Casanova. Scheduling distributed applications: the simgrid simulation framework. In *IEEE/ACM Int'l. Symp. on Cluster Computing and the Grid (CCGrid)*, pages 138–145. IEEE Computer Society, 2003.

[47] Hui Li, David L. Groep, and Lex Wolters. Workload characteristics of a multi-cluster supercomputer. In Dror G. Feitelson, Larry Rudolph, and Uwe Schwiegelshohn, editors, *Proc. of the Job Scheduling Strategies for Parallel Processing, Workshop (JSSPP)*, volume 3277 of *Lecture Notes in Computer Science*, pages 176–193. Springer, 2004.

[48] Hui Li, Michael Muskulus, and Lex Wolters. Modeling job arrivals in a data-intensive grid. In Eitan Frachtenberg and Uwe Schwiegelshohn, editors, *Proc. of the Job Scheduling Strategies for Parallel Processing, Workshop (JSSPP)*, volume 4376 of *Lecture Notes in Comput. Sci.*, pages 210–231. Springer-Verlag, 2006.

[49] Uri Lublin and Dror G. Feitelson. The workload on parallel supercomputers: modeling the characteristics of rigid jobs. *J. Parallel Distrib. Comput.*, 63(11):1105–1122, 2003.

[50] Muthucumaru Maheswaran, Shoukat Ali, Howard Jay Siegel, Debra A. Hensgen, and Richard F. Freund. Dynamic matching and scheduling of a class of independent tasks onto heterogeneous computing systems. In *Heterogeneous Computing Workshop*, pages 30–, 1999.

[51] Daniel A. Menascé, Debanjan Saha, Stella C. S. Porto, Virgilio Almeida, and Satish K. Tripathi. Static and dynamic processor scheduling disciplines in heterogeneous parallel architectures. *J. Parallel Distrib. Comput.*, 28(1):1–18, 1995.

[52] National grid service uk. [Online] Available: `http://www.grid-support.ac.uk/`.

[53] Nordugrid. [Online] Available: `http://www.nordugrid.org/`.

[54] Takayuki Osogami and Mor Harchol-Balter. Closed form solutions for mapping general distributions to quasi-minimal ph distributions. *Perform. Eval.*, 63(6):524–552, 2006.

[55] Abhay K. Parekh and Robert G. Gallager. A generalized processor sharing approach to flow control in integrated services networks: the multiple node case. *IEEE/ACM Trans. Netw.*, 2(2):137–150, 1994.

[56] Parallel workloads archive. [Online] Available: `http://www.cs.huji.ac.il/labs/parallel/workload/`.

[57] Bianca Schroeder and Garth A. Gibson. A large-scale study of failures in high-performance computing systems. In *DSN*, pages 249–258. IEEE Computer Society, 2006.

[58] Sharcnet. [Online] Available: `http://www.sharcnet.ca/`.

[59] Baiyi Song, Carsten Ernemann, and Ramin Yahyapour. User group-based workload analysis and modelling. In *IEEE/ACM Intl. Symp. on Cluster Computing and the Grid (CCGrid)*, pages 953–961. IEEE Computer Society, 2005.

[60] SPECCPU Team. SPEC CPU2006. Standard Performance Evaluation Corporation, Mar 2008. [Online] Available: `http://www.spec.org/cpu2006/`.

[61] Douglas Thain, Todd Tannenbaum, and Miron Livny. Distributed computing in practice: the condor experience. *Elsevier Concurrency - Practice and Experience*, 17(2-4):323–356, 2005.

[62] Richard Uhlig and Trevor N. Mudge. Trace-driven memory simulation: A survey. In Günter Haring, Christoph Lindemann, and Martin Reiser, editors, *Performance Evaluation*, volume 1769 of *Lecture Notes in Computer Science*, pages 97–139. Springer, 2000.

[63] Songnian Zhou. A trace-driven simulation study of dynamic load balancing. *IEEE Trans. Software Eng.*, 14(9):1327–1341, 1988.