

Höchstleistungsrechenzentrum
Universität Stuttgart
Prof. Dr.-Ing. Dr.h.c. Dr.h.c. M. Resch
Nobelstraße 19
70569 Stuttgart
Institut für Höchstleistungsrechnen

Service Level Agreements for Job Submission and Scheduling in High Performance Computing

Von der Fakultät Energie-, Verfahrens- und
Biotechnik der Universität Stuttgart
zur Erlangung der Würde eines Doktor-Ingenieurs (Dr.-Ing.)
genehmigte Abhandlung

vorgelegt von

Roland Kübert

aus Aschaffenburg

Hauptberichter: Prof. Dr.-Ing. Dr.h.c. Dr.h.c. Michael M. Resch
Mitberichter: Prof. Dr.-Ing. Stefan Wesner
Tag der Einreichung: 23. Januar 2013
Tag der mündlichen Prüfung: 04. August 2014

Höchstleistungsrechenzentrum Stuttgart

Universität Stuttgart

August 2014

Acknowledgements

No man is an island, entire of itself, as John Donne so pointedly stated in “Meditation XVII”. When writing a thesis, it sometimes feels as one has taken an endeavour where this is untrue, but in fact it is not. There are many people who provide help and support and I am very glad to acknowledge the help I received in the course of this work.

First and foremost, all the thanks in the world go to Lutz Schubert, a long-time colleague at HLRS without whom this thesis would not have been started in the first place. His continued interest in hosting the PhD seminar then helped to keep me going and to bring the thesis to a fruitful completion. Stefan Wesner, who was my superior during most of my time at HLRS and as well my second advisor, helped me a lot especially during the end of the thesis and contributed to a number of publications. Thanks to all participants at the regular HLRS PhD day, which offered the possibility to present my work to a wider audience than the small group of the PhD seminar, and to Michael Resch, the director of HLRS and my advisor.

Special thanks go to my parents, who have supported me throughout my life and whose support enabled me to study and pursue a PhD in the first place. Last but not least I also thank my friends for providing support and friendship – you know who you are.

Aschaffenburg, August 2014
Roland Kübert

Abstract

High performance computing (HPC) providers mostly offer their services on a best effort basis. The operational model for HPC resources is at its heart still the same as in its infancy when users placed sets of punch card at a desk for a job to be run. Nowadays, users can verify programs beforehand on their local computers before submitting online. They do not need to interact with human operators but can instead submit jobs themselves to various queues with different parameters. Still, though, jobs are scheduled in a batch system with the provider's goal of reaching a high utilisation. Scheduling is thus oriented only towards a global optimisation rather than an individual service offer.

Having only a single global optimisation point prevents providers from pursuing differing goals. There is no way to offer enhanced services with complex QoS parameters. Even if complex scenarios can be realised, they require manual intervention, which cannot scale. Urgent computing, for example, requires immediate execution of jobs, for example for weather predictions or a medical use case. The problem is therefore double sided, with different characteristics and consequences for clients and providers.

This work is based on an analysis of scenarios that cannot easily be performed today but are in demand. At first, requirements specific to these scenarios as well as generic requirements were elicited. Then, analysis of the state of the art in relevant areas was undertaken, namely in the fields of job submission and scheduling, simulation thereof, service level agreements (SLAs) and SLA management. The current models of SLA usage and SLA management, relying on a correspondence of one SLA per job to be computed, was identified as a hindrance to the uptake of service based offerings in the HPC domain. A different SLA model is proposed in this work: using SLAs as long-term contracts. In contrast to the usage of SLAs as describing quality of service requirements of a single job, long-term SLAs describes classes of service levels which can be referenced by individual jobs. Derived from the previously identified requirements, an SLA management framework for the HPC environment using long-term SLAs was proposed and its feasibility demonstrated by its implementation and ability to realise a scenario. In order to facilitate the simulation of service levels and scheduling, an existing simulation software tool was enhanced and modified to support SLAs in order to be able to simulate service levels offline without having to use actual physical infrastructure.

The work performed in this thesis introduced long-term SLAs as a tool for offering service level in HPC environments and confirmed their profitability. Implementing them can be done with a straightforward SLA management framework that can be easily integrated

into existing setups, from application layer to infrastructure layer. Scheduling according to service levels cannot be solved in a generic way. Nonetheless, for specific setups - for example, multiple service levels differing in priority relative to each other - some principles can be stated that generally hold true. In most cases, however, the scheduler implementation needs to be tailored to the offers an HPC provider chooses to tender. This is simplified by the simulation tool adapted to SLA based scheduling, which allows the simulation of different workloads, physical resources and scheduling algorithms.

In conclusion, the combination of long-term SLAs and an SLA management framework reduced to the bare necessities allows providers the implementation of service level based offerings. Work to be done includes among other things, the usage of a provider's real workload traces and infrastructures and the implementation of SLA management in the client domain.

Zusammenfassung

Dienste im Bereich Höchstleistungsrechnen (HPC) werden heutzutage hauptsächlich auf einer “best effort”-Basis angeboten. Das bedeutet, dass ein HPC-Anbieter sich bemüht, die Dienste bestmöglich zu erfüllen aber keinerlei Garantien hinsichtlich Ausführungszeitrahmen, Qualität oder ähnlichem gibt. Das Betriebsmodell für HPC-Ressourcen ist im Kern noch das gleiche wie zu Beginn des Aufkommens von HPC, als Benutzer Lochkarten persönlich abgaben um die darauf aufgeführten Berechnungen durchführen zu lassen. Heutzutage können Benutzer ihre Programme natürlich auf ihrem lokalen Rechner testen bevor sie diese übermitteln und anstatt einer Interaktion mit menschlichem Bedienpersonal werden die Jobs an verschiedene Warteschlangen mit unterschiedlichen Parametern übermittelt. Nach wie vor werden die Jobs allerdings in einer Stapelverarbeitung abgearbeitet und das Ziel des Anbieters ist es, eine hohe Auslastung des Systems zu gewährleisten. Das Scheduling zielt demnach auf eine globale Optimierung und nicht auf die Erfüllung individueller Dienstgüten.

Die Tatsache, dass nur ein einziger globaler Optimierungspunkt existiert verhindert dass ein Anbieter verschiedene Ziele verfolgt. Es gibt keine Möglichkeit, erweiterte Dienste mit komplexen Dienstgüteparametern anzubieten. Wenn doch komplexe Szenarien realisiert werden, bedarf es des manuellen Eingriffs in das System, was in keiner Weise skalierbar ist. Urgent computing verlangt zum Beispiel nach sofortiger Ausführung von Jobs, zum Beispiel für dringende Wettervorhersagen oder medizinische Anwendungsfälle. Das Problem ist daher zweiseitig, mit unterschiedlichen Ausprägungen und Konsequenzen für Kunden und Anbieter.

Diese Arbeit basiert auf der Analyse von Szenarien, die in der heutigen HPC-Landschaft nicht durchgeführt werden können, die aber von Kunden nachgefragt werden. Grundlage der Untersuchungen war die Analyse der Szenarien und die Identifikation von szenariospezifischen als auch von generellen Anforderungen. Danach wurde der Stand der Technik in den relevanten Feldern Job-Übermittlung und Job-Scheduling, Simulation desselben, Dienstgütevereinbarungen (SLAs) und der Verwaltung von SLAs untersucht. Das heute verwendete Modell der Nutzung und Verwaltung von SLAs, das eine eins-zu-eins-Entsprechung von Job zu SLA vorsieht, wurde als hinderlich für die Nutzung von SLAs im HPC-Bereich identifiziert. Diese Arbeit stellt deshalb ein anderes SLA-Modell vor: die Verwendung von SLAs als stabile Langzeitverträge. Im Unterschied zu SLAs wie sie momentan verwendet werden – zur Beschreibung von Qualitätsanforderungen eines einzelnen Jobs – können mit Langzeit-SLAs Klassen von Qualitätsmerkmalen abgebildet werden, die dann von einzelnen Jobs referenziert werden können. Basierend auf den ein-

gangs erwähnten Anforderungen wurde ein SLA-Verwaltungs-Framework für Langzeit-SLAs entworfen und implementiert und seine praktische Einsatzfähigkeit an Hand eines Szenarios demonstriert. Zusätzlich wurde, um die Analyse von SLA-basiertem Scheduling zu erleichtern, ein existierendes Simulationswerkzeug um Unterstützung für SLAs erweitert. Mit diesem Werkzeug wurden verschiedene Untersuchungen über den Einsatz von SLAs im HPC-Umfeld durchgeführt.

Diese Arbeit hat Langzeit-SLAs als ein Werkzeug für das Anbieten von Dienstgütern im HPC-Bereich eingeführt und deren Ertragswert bestätigt. Langzeit-SLAs können in einem existierenden HPC-Umfeld mit Hilfe eines unkomplizierten SLA-Verwaltungs-Frameworks eingeführt werden. Dieses erlaubt eine einfache Integration in die verschiedenen Ebenen, von der Infrastruktur bis zu den Anwendungen. Die Problematik des Scheduling von Langzeit-SLAs kann hingegen nicht im Allgemeinen gelöst werden. Nichtsdestoweniger können für bestimmte Situationen, zum Beispiel verschiedene Dienstgüter die sich nur relativ in ihrer Priorität unterscheiden, allgemeine Richtlinien empfohlen werden. Im Wesentlichen müssen Implementierungen aber auf die von einem Anbieter angebotenen Dienstgüter hin angepasst werden. Dies wird durch die Entwicklung des oben angesprochenen Simulationswerkzeugs erleichtert, das die Simulation von verschiedenen Workloads, Infrastrukturen und Scheduling-Algorithmen ermöglicht.

Zusammenfassend lässt sich sagen, dass eine Kombination aus Langzeit-SLAs und einem auf das nötigste reduzierten SLA-Verwaltungs-Framework HPC-Anbietern eine einfache Möglichkeit bietet, Dienstgüter anzubieten. Zukünftige Arbeiten können sich zum Beispiel mit der Verwendung von Anbieter-spezifischen Workloads und der Implementierung der SLA-Verwaltung auch auf Kundenseite beschäftigen.

Contents

List of Figures	xiii
List of Tables	xv
Abbreviations	xvii
1 Introduction	1
1.1 Objectives	2
1.2 Chosen Approach	3
1.3 Research Contribution	3
1.4 Background	4
2 A brief introduction to high performance computing and service level agreements	7
2.1 High performance computing	7
2.2 Job submission and scheduling	9
2.3 Service Level Agreements	11
2.4 Use Cases	12
2.4.1 Scientific use case	13
2.4.2 Virtual turbine use case	13
2.5 Requirements	14
2.5.1 Generic requirements	14
2.5.2 Scientific use case	16
2.5.3 Virtual turbine use case	16
3 State of the Art	17
3.1 High performance computing	17
3.2 Job scheduling	20
3.3 Service level agreements	21
3.3.1 Specifying SLAs for job submission	22
3.3.2 Using SLAs for job scheduling	22
3.3.3 Managing SLAs	24
3.4 Resource managers and job schedulers	33
3.4.1 Portable Batch System derivatives	33
3.4.2 Condor	35
3.4.3 Load Sharing Facility	36
3.4.4 LoadLeveler	36

3.4.5	Maui Cluster Scheduler and Moab Cluster Suite	36
3.4.6	Oracle Grid Engine	37
3.4.7	SLURM	37
3.4.8	Grid MP	38
3.4.9	Application Level Placement Scheduler	38
3.4.10	Summary	39
3.5	Simulation of HPC resources	39
3.5.1	SimJava	40
3.5.2	GridSim	41
3.5.3	MicroGrid	41
3.5.4	Bricks	41
3.5.5	SimGrid	42
3.5.6	Simbatch	43
3.5.7	MONARC	43
3.5.8	Grid Scheduling Simulator	44
3.5.9	Alea	45
3.5.10	Pyss - the Python Scheduler Simulator	46
3.5.11	Summary	46
3.6	Summary	47
4	Service level agreements for HPC job submission and scheduling	49
4.1	Service levels for high performance computing providers	49
4.1.1	Long-term SLAs for HPC service levels	49
4.1.2	Service levels for HPC	50
4.1.3	Service level provisioning benefits	54
4.2	HPC job submission and scheduling with SLAs	54
4.2.1	Submission of HPC jobs related to SLAs	55
4.2.2	Scheduling of jobs according to service levels	56
4.3	An SLA management framework for HPC providers	56
4.4	Evaluation	59
4.4.1	Generic requirements	59
4.4.2	Use case specific requirements	60
4.5	Summary	60
5	Implementation of SLA-enhanced job control	63
5.1	Scenario	63
5.2	Implementation architecture	65
5.2.1	Job submission and scheduling with the Globus Toolkit and TORQUE	65
5.2.2	Adding SLA-based job submission and scheduling	66
5.3	Prototypical implementation	68
5.3.1	Submitting a job related to an SLA	69
5.3.2	Receiving SLA information on the provider side	72
5.3.3	Securing the ManagedJobFactory service	72
5.3.4	Queuing jobs in reference to an SLA	75
5.3.5	Running the job	76

5.3.6	Accounting and billing	77
5.4	Evaluation	77
5.4.1	Generic requirements	77
5.4.2	Scenario specific requirements	78
5.5	Summary	78
6	Simulating SLA usage in an HPC environment	79
6.1	The Alea scheduling simulator	79
6.1.1	Features	80
6.1.2	Shortcomings	81
6.2	SLA-based scheduling with Alea	81
6.2.1	Service level distributions for workloads	82
6.2.2	Implementing scheduling with SLAs	85
6.3	Experimental results	86
6.3.1	Prioritised and best effort access	87
6.3.2	Cancellation and resubmission of best effort jobs	89
6.3.3	Influence of different cancellation policies	92
6.4	Summary	102
7	Conclusions and future work	105
7.1	Conclusions	105
7.1.1	Service level agreements for the HPC domain	105
7.1.2	Service level agreement management for HPC environments	106
7.1.3	Simulating SLA usage in an HPC environment	106
7.2	Future work	106
7.2.1	Refinement of simulation through feedback of real data	107
7.2.2	SLA management in the client domain	108
7.2.3	Analysing revenue changes for HPC providers	108

List of Figures

1.1	Layered architecture for HPC services	2
3.1	Brokered SLA architecture [49]	23
3.2	SLA management as seen by the BEinGRID project [2]	25
3.3	SLA management as seen by the BREIN project [54]	26
3.4	SLA management architecture proposed by the FinGrid project [55]	27
3.5	SLAs in the IRMOS architecture [56]	28
3.6	IRMOS service management [56]	29
3.7	The SLA layer developed by SLA4D-Grid in the D-Grid	30
3.8	SLA4D-Grid's SLA layer architecture	31
3.9	PBS/TORQUE architecture	34
3.10	The architecture of Bricks [66]	42
3.11	SimGrid component overview [67]	43
3.12	The architecture of GSSIM [71]	44
4.1	High-level SLA management components	58
5.1	WS-GRAM components [88]	66
5.2	WS-GRAM protocol sequence [88]	67
5.3	SLA selection and job submission using a GUI client	70
6.1	Architecture of Alea	80
6.2	The Alea architecture with the added SLA loader	83
6.3	Histogram showing the percentage of jobs for different job sizes	87
6.4	Average wait time increase for bronze jobs	88
6.5	Average wait time for silver jobs	89
6.6	Average wait time for bronze jobs	90
6.7	Average wait time for silver jobs	90
6.8	Wasted bronze core hours due to cancellations	91
6.9	Silver SLA price to make up for revenue loss due to cancellations	92
6.10	Wasted core hours versus job size for least percentage completed cancellation	94
6.11	Wasted core hours versus job size for shortest job by core hours cancellation	94
6.12	Wasted core hours versus job size for shortest job by time cancellation	95
6.13	Silver SLA price versus job size for least percentage completed cancellation	96
6.14	Silver SLA price versus job size for shortest job by core hours cancellation	97
6.15	Silver SLA price versus job size for shortest job by time cancellation	97

6.16	Bronze average wait time versus job size for least percentage completed cancellation	98
6.17	Bronze average wait time versus job size for shortest job by core hours cancellation	99
6.18	Bronze average wait time versus job size for shortest job by time cancellation	99
6.19	Silver average wait time versus job size for least percentage completed cancellation	100
6.20	Silver average wait time versus job size for shortest job by core hours cancellation	101
6.21	Silver average wait time versus job size for shortest job by time cancellation	101
7.1	Feedback between simulation, contractual offerings and analysis of workload traces.	107

List of Tables

2.1	Examples of current HPC computers as of November 2012	8
2.2	Fees and charges at the High Performance Computing Center Stuttgart [12]	9
2.3	Batch queues at HLRS' NEC SX-9 [15]	9
3.1	Batch queues at TU Dresden's cluster "Deimos" [26]	18
3.2	Batch queues at HLRS' Nehalem cluster [27]	18
3.3	Batch queues at FZJ's Blue Gene/L [28]	19
3.4	Comparison of SLA management approaches	32
3.5	Comparison of resource managers and job schedulers	39
3.6	Comparison of simulation software for HPC	46
6.1	Details for experiment 1	87
6.2	Details for experiment 2	89
6.3	Sample jobs showing the influence of different cancellation policies	93
6.4	Details for experiment 3	93

Abbreviations

ALPS	Application Level Placement Scheduler, page 38
API	Application programming interface, page 40
ASCI	Advanced Strategic Computing Initiative, page 19
BASIL	Batch and Application Scheduler Interface Layer, page 38
BEinGRID	Business Experiments in Grids, page 24
BLAST	Basic local alignment search tool, page 23
BMBF	German Federal Ministry of Education and Research, page 29
BMP	Bitmap file format, page 80
BOINC	Berkeley Open Infrastructure for Network Computing, page 36
BREIN	Business objective driven reliable and intelligent grids for real business, page 17
CAD	Computer-aided design, page 14
CAVE	Cave virtual automatic environment, page 14
CERN	European Organization for Nuclear Research, page 44
CIT	California Institute of Technology, page 44
CPU	Central processing unit, page 8
CSV	Comma-separated values, page 80
DES	Discrete event simulation, page 40
EDF	Earliest deadline first, page 20
EPR	Endpoint reference, page 66
EU	European Union, page 45
FCFS	First come, first served, page 20

FinGrid	Financial Business Grid, page 4
FLOPS	Floating point operations per second, page 8
FLOSS	Free/libre/open source software, page 106
FTP	File transfer protocol, page 65
FZJ	Forschungszentrum Jülich, page 18
GIF	Graphics Interchange Format, page 80
GPL	GNU General Public License, page 46
GPU	Graphics processing unit, page 8
GRAM	Grid Resource Allocation Manager, page 33
GSSIM	Grid Scheduling Simulator, page 44
GT	Globus Toolkit, page 18
GUI	Graphical user interface, page 44
GWF	Grid Workload Format, page 80
HLRS	High performance computing center Stuttgart, page 9
HPC	High performance computing, page 1
HPCWSAG	High performance computing WS-Agreement, page 22
HTC	High throughput computing, page 35
ICT	Information and communications technology, page 24
IRMOS	Interactive Realtime Multimedia Applications on Service Oriented Infrastructures, page 4
ISONI	Intelligent service-oriented network infrastructure, page 28
JPEG	Joint Photographic Experts Group, page 80
LAN	Local area network, page 44
LGPL	GNU Lesser General Public License, page 46
LHC	Large Hadron Collider, page 44
LSF	Load Sharing Facility, page 36
MONARC	Models of Networked Analysis at Regional Centers, page 44

MWF	MetaCentrum Workload Format, page 80
NextGRID	The Next Generation Grid, page 4
NQS	Network Queuing System, page 19
OGE	Oracle Grid Engine, page 37
PBS	Portable Batch System, page 33
PDP	Policy decision point, page 59
PEP	Policy enforcement point, page 59
PKI	Public key infrastructure, page 55
PNG	Portable Network Graphics, page 80
PWF	Pisa Workload Format, page 80
QoS	Quality of service, page 1
RAM	Random access memory, page 8
RMS	Resource management system, page 10
RR	Round-robin, page 20
RSL	Resource specification language, page 69
SDM	Service Domain Manager, page 37
SJF	Shortest job first, page 20
SLA	Service level agreement, page 1
SLA4D-Grid	Service Level Agreements For D-Grid, page 4
SLAMF	Service level agreement management framework, page 1
SLURM	Simple Linux Utility for Resource Management, page 37
SOA	Service-oriented architecture, page 65
SWF	Standard workload format, page 46
TMF	TeleManagement Forum, page 57
TORQUE	Terascale Open-Source Resource and Queue Manager, page 10
TU	Technical university, page 18
UNICORE	Uniform Access over the Internet to Computing Resources, page 35

Abbreviations

URL	Uniform resource locator, page 69
VO	Virtual organisation, page 23
VT	Virtual turbine, page 14
WAN	Wide area network, page 44
WS-Ag	WS-Agreement, page 22
WSLA	Web Service Level Agreements, page 22
WSRF	Web Services Resource Framework, page 65

Acknowledgements

No man is an island, entire of itself, as John Donne so pointedly stated in his “Meditation XVII”. When writing a thesis, it sometimes feels as one has taken an endeavour where this is untrue, but in fact it is not. There are many people that provide help and support and I am very glad to acknowledge the helped I received and that helped me to be able to achieve this work.

Chapter 1

Introduction

In high performance computing (HPC), computational jobs are still submitted to and scheduled through batch queues. Batch queues offer multiple advantages compared to direct execution of jobs, for example the availability of resources to a vast number of users, high utilisation of resources and the ability for the provider to schedule jobs according to specific properties like job priority, job deadline, estimated execution time etc.

The usage of batch queues, however, is not without its drawbacks. Efficient user submission of jobs requires high expertise of the user due to the fact that jobs need to be described through low-level properties. For providers, it is difficult to create a queue setup that provides a good amount of diversity but that can still be handled efficiently. Exceptional jobs, like jobs demanding a very large number of resources, are often run through manual interaction by administrators which beforehand have to stop the scheduling of other jobs in order to allow for enough free resources. Interventions like this are not scalable and there is no possibility to map them to business penalties.

HPC providers which want to offer their services as utilities face even more problems arising from the best effort based nature of batch systems. Quality of service (QoS) parameters, especially those that transcend simple prioritisation, can not be assured with batch systems. Examples of QoS parameters include maximum job waiting times, guaranteed compute environments, application versions, licenses, data treatment and, recently, even environmental factors [1]. This leads to a gap in user demand and provider offerings which is an issue especially for providers because they offer a utility that can be replaced by other providers.

Service level agreements (SLAs), which are bipartite contracts describing, among other things, quality of service parameters, are a promising tool for HPC providers to enable complex QoS support. SLA management frameworks (SLAMFs) allow the negotiation, agreement and management of SLAs but are not linked to the underlying infrastructure.

Even if integrated SLAMFs, linking high-level SLA management with infrastructure

management, were existing, another problem surfaces: providers may know what service levels or QoS requirements to offer through client requests but there is no possibility to readily evaluate the consequences of offering specific service levels on the infrastructure, other service levels or best effort based services.

1.1 Objectives

The objective of this thesis is to enable HPC providers to offer more than simple best effort based services by combining SLA concepts, SLA management, job submission and scheduling and simulation. This thesis delivers a result that works on different abstraction layers, from the technical infrastructure layer through the middleware to the application layer (see figure 1.1).

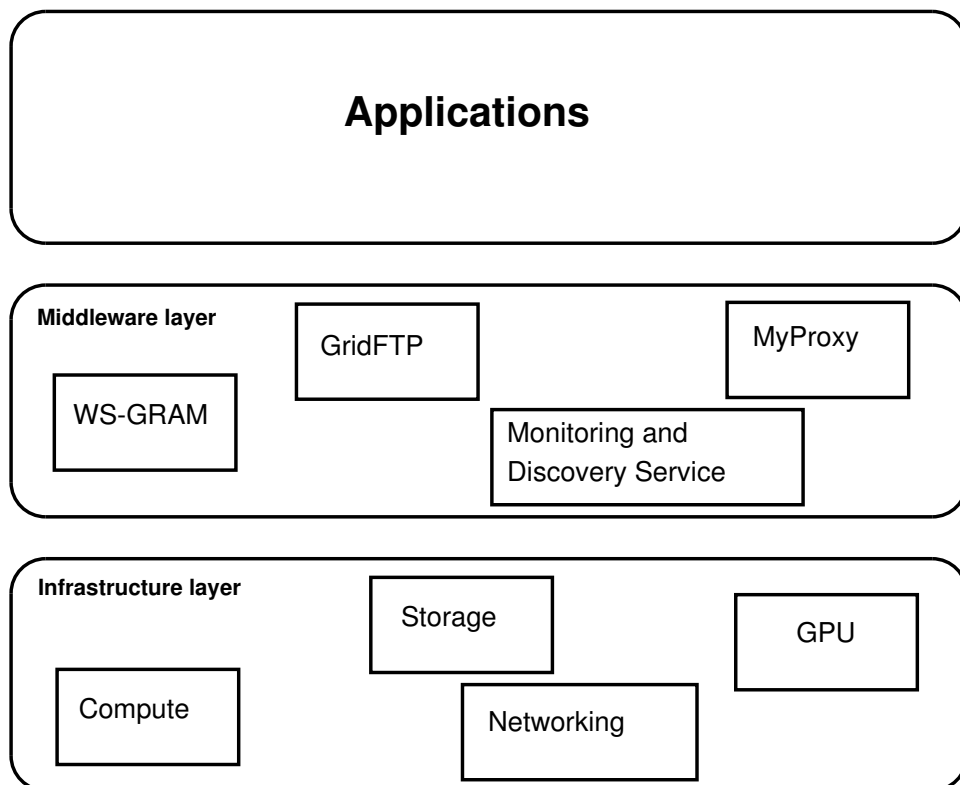


Figure 1.1: Layered architecture for HPC services

The possibility for HPC provider to offer service levels is beneficial for both clients and providers. Providers benefit by improving their image as being capable of providing specialised services, which results in a clear distinction compared to other providers, something which is not easily possible with only best effort based services. Additionally, this opens new revenue streams for HPC providers. Clients can make use of the enhanced

services and realise scenarios that have been formerly very difficult or even impossible to realise.

1.2 Chosen Approach

The starting point for this thesis was the observation that HPC providers mostly offer best effort services. Services transcending best effort can only be procured manually, which cannot scale. Furthermore, current SLA models and management frameworks are not suitable for immediate adoption in an HPC environment, as will be shown in detail in chapter 3. In short, this is due to the fact SLA models and management frameworks were developed for web service environments and not for HPC environments.

The first step taken was to find alternatives for the usage of a single SLA for each job. This resulted in the proposal to use long-term SLAs, which describe a service level that can be used by many jobs, thus allowing to focus on SLAs and their implications rather than on their negotiation and establishment. This is discussed in chapter 4.

With the contractual aspects settled, scenarios that are either currently not supported by HPC providers at all or only with tremendous difficulties were examined and requirements derived from these scenarios. The requirements were used as a basis for an SLA management framework suitable for HPC providers, whose feasibility was subsequently verified by implementation and execution of a selected scenario. Scenarios and requirements are discussed in chapter 4 while the implementation is presented in chapter 5.

In order to allow providers the theoretical evaluation of different SLA concepts and scheduling algorithms, a simulation tool was developed that permits these investigations. Different mixtures of service levels and scheduling algorithms have been examined using this tool. The findings are discussed in detail in chapter 6 and indicate how to provision service levels sensibly.

The thesis concludes with a summary of the results and proposals for future work in the theoretical, simulation and implementation areas in chapter 7.

1.3 Research Contribution

The research contribution of this thesis is a concept of how SLAs can be used in an HPC environment, enhancing HPC offers by service levels. The author has in the last years together with others published a number of conference papers, book chapters and journal papers about the concepts from these topics [2, 3, 4, 5, 6, 7] ranging from the theoretical

groundwork of using SLAs for job submission and scheduling in an HPC environment to simulation and implementation of these concepts.

In summary, the major contributions are:

1. A theoretical approach to the application of SLAs to the field of HPC computing.
2. An SLA management framework for HPC environments.
3. A prototypical implementation of the framework and theoretical concept, showing their feasibility for deployment in a production environment.
4. An adaptable, extensible simulation software ready to evaluate the theoretical concepts with different service levels and scheduling algorithms against different infrastructures

While the general concepts of SLA usage in an HPC environment and the simulation thereof are addressed in a generic way, the implementation of said concepts has been tailored to an existing HPC setup. This allows for an easier implementation in daily operations and reduces reluctance of implementation because of fear of incompatibility with existing components and procedures. Nonetheless, the presented implementation is generic enough to be easily transferable to other setups with exception of low-level functionality.

1.4 Background

The research performed by the author on the concept of SLA federation and the implementation of an SLA management framework in the NextGRID project [8] and subsequently the development and implementation of SLA management frameworks for the projects FinGrid [9], IRMOS [10] and SLA4D-Grid [11] was the foundation on which the idea to implement similar solutions for the support of SLAs in a HPC domain was developed. These projects used different approaches to achieve specific results but often there were overlaps in the SLA management layers. However, all of the solutions lacked in generality, which is understandable as it was not the aim of these projects to provide a generic SLA management framework or implementation. The NextGRID project, however, specified a basic architecture for SLA management and has often been a source of inspiration for other projects.

Over time it became obvious, that the implementation of an SLA management framework on a middleware layer is not sufficient to enable HPC providers to implement service levels. This is, firstly, due to the fact that the middleware layer is missing integration into the infrastructure, secondly because an ad-hoc change of current best effort based

services at an HPC provider to SLA-based service entails consequences that are not foreseeable. This extended the focus of the work towards the simulation of the usage of SLAs in an HPC environment. In order to be able to correctly decide the parameters for a simulation, a theoretical foundation of how SLAs can be used in an HPC environment was developed. The result is that this work builds on three main pillars, the theoretical background, an implementation of this concept and the simulation and evaluation of this concept.

Chapter 2

A brief introduction to high performance computing and service level agreements

This chapter introduces the relevant topics for this thesis on a sufficiently detailed, technical level. Firstly, the concept of HPC is explained in general, followed by a description of job submission and scheduling in HPC and where the boundaries between these two closely related topics are. Then, the concept of SLAs and its relation to HPC is explained and, finally, motivating use cases are introduced. From these use cases, requirements are extracted that will later be used to identify gaps in the state of the art and validate the proposed concept.

2.1 High performance computing

It is hard to find a generally accepted definition of the term “high performance computing” but the connotation is most often that of solving computationally huge problems with specialised computers. Previously, these specialised computer were often called supercomputers, likewise a term that is quite hard to define precisely, although it is common understanding that computers featured on the Top500 list of the world’s fastest computers can be called supercomputers. HPC is nowadays rarely performed by supercomputer in the classical sense of the word – for example vector computers – but mostly by clusters: more than 80% percent of the computers featured on the Top500 list are clusters as of November 2012 ¹.

Compute clusters are comparably homogeneous resources linked together with a high speed local area network, which distinguishes them from distributed systems or grid, which are more dispersed and heterogeneous. The recent advent of graphical processing

¹<http://www.top500.org/statistics/list/>

Name	Titan	Sequoia	Blue Gene/Q
Rank in Top 500	1	2	5
Performance (PetaFLOPS)	17.590	16.324	4.141
Cores	560,640	1,572,864	393,216
Rank in Green 500	3	29	5
Total Power (kW)	8,209	7,890	1,970
MFLOPS/W	2,142.77	2,069.04	2,102.12

Table 2.1: Examples of current HPC computers as of November 2012

units means that clusters feature heterogeneous resources as well but this is limited to having a high number of identical central processing units (CPUs) and a lower number of identical graphics processing units (GPUs). As the performance in CPUs has been constantly increasing over the past decades, the definition of what performance criteria actually make up a HPC system is constantly shifting as well.

HPC is used to solve problems that are very complex and therefore require so much processing time that it is infeasible to run these computations on single workstations. Memory or disk requirements can be so huge that trying to solve these problems on commercial off-the-shelf hardware is virtually impossible. Typical problems that are computed with HPC resources are thus intensive in or more of computation, memory, disk or input/output and include, among others, computational fluid dynamics, molecular dynamics, car crash simulations, climate prediction or nuclear weapons simulation. Performance increases in CPUs – for example through increased clock frequency, multi- and manycore CPUs and more functionality on chip through decreased transistor size –, RAM etc. have led to ever decreasing costs per floating point operation (FLOP); FLOPS are the common measurement for a computer’s performance.

Not only have the construction costs per FLOP decreased dramatically but also the operation costs. This allows to build more and more powerful clusters; the newest addition to the Top 500 list, Japan’s K computer, has a performance of 8 TeraFLOPS and is faster than the next five on systems on the list combined.

Access to HPC machines is restricted depending on the funding of the machine; machines which have been funded exclusively by research councils restrict access to research usage while machines co-funded by industrial enterprises provides access to both researchers and practitioners. Users apply for computational time and can then submit compute jobs. Clusters may be available for free to members of certain institutions; bwGrid, for example, is a decentralised cluster located at eight different universities in the German state of Baden-Württemberg which is available to all members of the participating universities. If the cluster is not free to use, users pay per amount of time used, for example in cores per hour. Table 2.1 lists some exemplary fees from the High Performance Computing Center Stuttgart. The fees are coarse-grained and only distinguish between members of the University of Stuttgart and members of other universities. Fees for foreign users are

Cluster	IA-64	NEC SX-8	NEC SX-9	Cray XT5
Euro per Hour (Univ. members)	0.09	0.28	0.63	0.02
Euro per Hour (other univ.)	0.85	2.78	6.22	0.16

Table 2.2: Fees and charges at the High Performance Computing Center Stuttgart [12]

charged by agreement according to market prices [12]. Contracting is thus done not in an ad-hoc, dynamic way but in a static, long-term fashion.

2.2 Job submission and scheduling

As soon as users are eligible to compute jobs on clusters they find out that execution of jobs is not as straightforward as on standard desktop computers or workstations. It is not possible to execute jobs directly on cluster nodes, users instead have to submit jobs to a scheduler that deals with the placement of jobs on the cluster.

In order to access the scheduler, users access dedicated nodes of the cluster, called “login” or “frontend” nodes. From a login node, a user can submit a job to be executed on the cluster. Running jobs on the login nodes is at least interdicted, as this leads to a slow response time of the login nodes and brings disadvantages for all users [13]. In some cases, this behaviour can even lead to account disablement [14].

The correct way to run jobs is for the user to submit a job description to a cluster’s job scheduler, also called batch system or resource manager. The job description includes the executable to run, a name for the job, specification for the redirection of input and output, the number of requested resources and the queue to which to submit the job. Job queues are data structures that job schedulers use to manage jobs that have been submitted but are not yet being executed. Clusters can have one or many job queues, it is common to have different queues with different parameters which govern the acceptance of jobs to a queue. Table 2.3 shows an exemplary queue setup at the High Performance Computing Center Stuttgart’s (HLRS) NEC SX-9 cluster, a vector supercomputer.

Class	Max. #CPUs per node	Max. #nodes	Max. memory	Max. elapsed time	Mode
test	4	1	16GB	00:05:00	shared
single	8	1	64GB	12:00:00	shared
multi	16	4	510 GB	12:00:00	dedicated

Table 2.3: Batch queues at HLRS’ NEC SX-9 [15]

The main factors that determine queue admittance are in this case run time and resource requirements: short jobs for testing with a runtime of only up to five minutes and low resource requirements can be submitted to the “test” queue, while medium-sized jobs and hugely-sized jobs both have separate queues. Providers are free in the way they setup their queues, however similar setups as the one shown in table 2.3 are popular.

Whereas a user can influence into which queue his job is queued – for example by adapting the run time or resource requirements – he has no control over when his job is actually run. Apart from the design of the queue system, that is number and characteristics of the queues, the question of which of the queued jobs to run is the second major task the provider has to perform. This is done by the job scheduler, which knows about available resources and queued jobs and selects jobs that are eligible to run. In order to be aware of which part of a cluster is currently idle, a resource management system is used (RMS). Some RMSs feature a built-in scheduler – for example the Terascale Open-Source Resource and QUEue Manager (TORQUE) [16] – but built-in schedulers tend to offer only basic scheduling functionality and sophisticated solutions purely for the scheduling problem exist.

Providers have two goals in scheduling: maximising resource usage and at the same time implementing some fairness in the scheduling between different users and job sizes. Maximum resource usage as a goal is quite straightforward, while the definition of fairness is questionable and can vary from provider to provider, machine to machine etc. Usually, a balance between users and job sizes is sought, for example like this [15]:

- Users with little computation time in the last weeks have high priority
- Small jobs can surpass large jobs and fill gaps
- Large jobs generally have priority
- Long waiting jobs gain priority
- Jobs are not held by the scheduler, only by users or administrators
- Jobs can be checkpointed and restarted

Once defined, both the queue setup and the scheduling are only changed infrequently. This means that users can rely on consistent scheduling behaviour, which on the flip side means the provider cannot adapt to changing situations.

Queuing today is performed in a similar manner as it was with the first mainframe computers. Users queued in line with a stack of punch cards which they wanted to be read into the machine and then executed. The punch cards themselves were queued if the card reader was already busy. Today, users do not use punch cards any more, but

the queuing solution has not evolved much.

Providers devise a queuing setup they deem suitable and are reluctant to change it without having an absolute necessity, for example long times where the machine is idle due to a faulty setup. For users, the setup can therefore be said to be carved in stone. As the numbers of queues is very limited (table 2.3 shows only two production queues and one testing queue), users are not able to differentiate job requirements in a fine-grained manner – the job either goes to queue *single* or *multi*, wherever its resource requirements make it fit. In a sense, this assumes that all jobs are in a basic sense equal, even though this is clearly not the case. This setup is not able to accommodate users with prioritisation; as described above, fairness rules are in place which can even deprioritise jobs. Exceptional cases therefore require human interaction, that is personal communication between a user and an administrator. This has a huge overhead and cannot scale and is only usable if prioritisation cases stay the exception.

Providers have no obvious gain when performing manual intervention (client satisfaction might be increased but is hard to measure). Provider infrastructures are not prepared to deal with prioritisation so there is no possibility for a provider to account for and bill prioritised jobs differently than regular jobs. Prioritisation can be provided by the means of an express queue, which is scheduled favourably. Access to an express queue can be granted to individual users or groups but it is cumbersome to define a consistent setup for a large machine and a big user pool. In addition, the queue might be accessible to different users and groups during different times. Even an express queue setup is inflexible in that it cannot foresee varying load situations which occur especially on heterogeneous clusters. Specialised nodes, for example with GPUs or large amounts of memory, are often underutilised while normal nodes are overbooked. Offering a special discounted access to these special nodes in times of high load on normal nodes is, however, difficult. If users' jobs are run on these nodes, the users have to pay for the resources actually used, not for the ones they wanted. Another shortcoming of a queuing solution is that elements describing quality of service (QoS) or quality of experience (QoE) cannot be mapped on queue properties and parameters. It is not possible to specify additional properties like the availability of a defined compute environment including application and library version, to define data treatment requirements or special configurations like a logically separate partition of a cluster in order to be isolated from other users.

2.3 Service Level Agreements

Section 2.1 has explained how contracts between HPC providers and users are formulated today: the contract defines formulas for walltimes and the amount of money to be paid for each hour of walltime used (see table 2.1). Contracts are defined statically and thus the possibility for providers to treat users differently is static as well. Nodes reserved for one user can hardly be provided to other users. With a contract in place, users can

submit jobs to queues defined by a provider, which has its own shortcomings as identified in section 2.2. A potential solution to both problem domains is the introduction of service level agreements (SLAs).

SLAs are bipartite contracts between a service provider and a client specifying details of a service offered from the provider to the client. The service's details can be both functional - what the service does - and non-functional - how the service does what it does. Furthermore, SLAs can be used to agree on responsibilities of both providers and clients and on penalties and rewards. SLAs are seen both as contracts in their own right as well as sub-contracts of greater framework contracts (see for example [17] and [18]); for HPC providers, the specification of SLAs as contracts in reference to framework contracts is sensible as these contracts already exist (see section 2.1).

As a result a significant amount of work has been spent on realising SLA frameworks allowing mutual agreement on terms of service between provider and client. These frameworks cover necessary steps to realise SLAs as legally binding agreements, however the concrete content of an SLA and more important how an SLA's terms can be guaranteed and provided from the service provider side are not adequately addressed.

The major drawback, however, is that SLA usage is assumed to be on a per-job basis. That means that from a single contract governing all submitted jobs and no QoS offered, there is now a single contract for each job with job-specific QoS requirements negotiated dynamically. This allows very fine-grained agreement on job parameters and their values but introduces additional problems: if each job requires a single contract, the overhead to negotiate these contracts becomes quite high. Furthermore, this scenario does not represent the way HPC users work and users cannot be expected to negotiate contracts for each single job, with the potential risk that an agreement cannot be reached and, therefore, job submission is not possible – even a simple queuing solution guarantees at least the acceptance of a job. Negotiation on a per-job basis does not scale if human intervention is necessary; automation of negotiation with high degrees of freedom not only poses technical problems but is as well legally challenging. Another question that poses itself regarding a 1-to-1 mapping of SLAs and jobs is how the SLAs parameters are mapped to the providers infrastructure: a provider is surely able to define some service levels but a high degree of freedom can hardly be mapped in a simple way to the infrastructure.

2.4 Use Cases

The previous sections have given an introduction to the topics addressed in this work: HPC in general and specifically the field of job submission and scheduling and the relation to SLAs. This section introduces two specific use cases that motivate the work and from which requirements are derived which are later evaluated in order to assess the presented

solution. The first use case has a scientist as a user, the second use case deals with the concept of a “virtual turbine”.

2.4.1 Scientific use case

Alice and Bob are scientists working in different departments and both are running simulations on HPC resources for publications they are working on. Sometimes their requests are competing, especially if both want to run large-scale simulations at the same time, however this is generally not a problem for most of the time they are preparing their respective publications.

As the deadline of Alice’s publication is approaching, she realises that she ran some of her large-scale simulations with wrong parameters. She checks the default queue and sees that many jobs are queued already, some of them are jobs queued by Bob, which she knows from experience to be long-running like hers. Her simulation is long-running and therefore needs to start right away, so queuing it at the end of the queue would mean that she would not get the results in the time needed but costs would still incur. Alice needs a way to submit jobs with a higher priority than other users.

Alice’s department has agreed on the usage of different service levels with the provider; by default, jobs are submitted to the default queue and receive no specific priority over other user’s jobs. For more urgent jobs, an SLA that grants prioritised usage of the resources has been agreed upon as well. This SLA incurs higher costs but referencing it on job submission results in the job bypassing the default queue. The higher price for prioritised jobs is a disincentive for users and ensures that the queue is not used permanently instead of the default queue. Alice submits her job referencing the prioritised SLA and her simulation is started promptly and finishes in time for her to use the generated results in her publication. Other jobs Alice submits for her daily work are submitted referencing the default SLA and do not compete with her urgent job.

2.4.2 Virtual turbine use case

The design process for hydraulic turbines is complex and iterative. It begins with a geometry definition for a single component which is created using a CAD tool. The geometry definition is imported into a meshing tool, which creates a computational mesh for the component. The mesh is used to perform simulations of the component and in a postprocessing step, assessment of the simulation results is performed. If the component quality is insufficient, the geometry has to be adapted and all steps have to be repeated. When all component’s designs are finished, a coupled simulation is used to resolve interactions with the single components and can lead to further refinements of

the individual components. This is, obviously, a lengthy process where each step is itself time consuming and computationally complex.

In order to allow the design process to be performed interactively, the concept of the virtual turbine (VT) has been developed [19]. The main aim of the VT is to simplify operations in order to reduce their latency and to allow interactive changes to be immediately visible. The simplification results in a trade-off between the loss in precision and the short latency. Even with simplification, however, operations are still computationally expensive and resources need to be available at specific points in time, otherwise the short latency in the computations themselves is lost or even backfires. This is especially true, if the visualisation is performed in conjunction with a Cave Virtual Automatic Environment (CAVE) [20]. Then, it needs to be ensured that the CAVE and the computational resources, as well as, for example, corresponding network resources, are available at the same time. If only one of the resources is not available at the planned time, the whole process fails.

Apart from the aspect that resources need to be available during a precise period in time, computations have different priorities: simulations which anyway have a high latency do not need to be processed urgently but rather in a best effort manner, thus requiring different prioritisation of computational jobs.

2.5 Requirements

The use cases in section 2.4 are used to elicit requirements of HPC providers and their clients. However, as each use case describes a specific scenario, general requirements apart from these use cases are elaborated as well.

The requirements will later be used to evaluate in how far they are covered by the solution presented in this work. In order to ease the identification of requirements, each requirement is denominated with a unique identifier.

2.5.1 Generic requirements

Apart from requirements targeted at the use cases in sections 2.4.1 and 2.4.2, requirements that are overarching different use cases can be found. This section therefore lists generic requirements which extend beyond the use cases described above and are applicable to most any use case.

Req.1 *Easy integration of SLA management into current infrastructure* The integration of the SLA management system into the current infrastructure should be as simple as

possible.

Req.2 *Visualisation tools for administrative tasks* Visualisation can help the provider to make decision during SLA management.

Req.3 *Exposition of contract-relevant data to users* Users need to be able to query the provider for contract-relevant data at all times, for example usage statistics.

Req.4 *Multiple service levels per provider* Providers want to provide multiple service levels, for example best effort and prioritised.

Req.5 *Specification and consideration of non-functional properties* Non-functional properties, for example availability, security, or data treatment policies, need to be specified and considered.

Req.6 *Express the need to access specific hardware* Certain jobs might need specific hardware, this needs to be expressed either on job submission time or fixed in an SLA.

Req.7 *Express the need to access specific software* Certain jobs might need specific software, this needs to be expressed either on job submission time or fixed in an SLA.

Req.8 *Clear pricing model* Pricing models should be transparent and easily understandable.

Req.9 *Ensure clients can always submit jobs* Clients need to have a contractual basis which allows them job submission. This is in contrast to dynamic negotiation, which may result in client and provider not reaching an agreement.

Req.10 *Service level selection at job submission* Users want to choose a job's service level during job submission.

Req.11 *Define and enforce access policies for service levels* As not all service levels will be contracted by all users, the provider needs to define access policies for the relevant users and needs to be able to enforce them in order to prevent misuse of service levels.

Req.12 *Accounting and billing against SLAs* Jobs need to be accounted against their service level, not only against the pure wall time as in a best effort case.

Req.13 *Constant evaluation of SLAs* Existing SLAs need to be evaluated constantly in order to minimise or even avoid violations.

Req.14 *Ability to estimate service level details* Providers should be able to estimate, possibly through simulation, the amount of non-conflicting service levels that they can offer. This can also include the ability to compute which job's SLA should be violated if this is unavoidable.

2.5.2 Scientific use case

The following requirements can be derived from the scientific use case, in addition to the generic requirements obtained from section 2.5.1:

Req.15 *Prioritised resource access* Computations are urgent and need to be started within a guaranteed time frame.

Req.16 *Ability to suspend / terminate running jobs* Jobs currently being executed in the system might need to be suspended or terminated and rescheduled in order to fulfil guarantees on execution time.

2.5.3 Virtual turbine use case

Apart from the requirements unique to the virtual turbine use case, which are listed below, the requirements R01, R02, R04, R05, R07 and R08 apply to this use case as well.

Req.17 *Timed access* Users want guaranteed resource access at a certain point in time.

Chapter 3

State of the Art

Two concurrent lines of research in the service level agreement area in cluster and high performance computing can be observed: on the one hand, most ongoing or recently finished research projects, like for example NextGRID [8], BREIN [21] or IRMOS [10], just to name a few, provide high-level SLA management frameworks and components, on the other hand, there is a growing interest to use service level agreements for quality of service-based job submission and scheduling.

Both these developments ameliorate cluster and HPC computing with concepts to provide not only service levels at all but diverse ones as well. The ongoing, intensive research effort in both areas shows the broad foundation on which European research projects are based.

The missing link, that can be seen as a gap between the previously mentioned developments, is the integration of high-level concepts on the management layer and the low-level implementation of similar concepts.

This chapter presents relevant state of the art in the areas of high performance computing, job scheduling, service level agreements, resource managers and job schedulers and simulation tools, closing with a summary and conclusions drawn out of the previously presented material.

3.1 High performance computing

An introduction to the current proceedings in HPC has already been given in section 2.1. This section elaborates the state of the art in job submission and scheduling in HPC in more detail.

Job submission can be performed in different ways. Submission directly to a cluster's job scheduler is only allowed from within a provider's domain, hence users need to log

Name	Users	CPUs	Max. elapsed time
interactive	all	[1; 32]	12:00:00
express	all	[1; 8]	04:00:00
small	all	[1; 8]	72:00:00
intermediate	all	[5; 128]	12:00:00
large	all	[128; 256]	12:00:00
benchmark	selected	[1; 2548]	∞
single_cpu	all	[1; 856]	24:00:00
nightexpress	selected	[1; ?]	04:00:00

Table 3.1: Batch queues at TU Dresden’s cluster “Deimos” [26]

Class	Max. #CPUs per node	Max. #nodes	Max. elapsed time
test	8	4	00:20:00
single	8	1	24:00:00
multi	8	4	04:00:00
	8	360	12:00:00
	8	320	24:00:00
hero	8	650	06:00:00

Table 3.2: Batch queues at HLRS’ Nehalem cluster [27]

onto a frontend node before being able to submit jobs. Additionally, the provider can offer access through a middleware which allows the submission of jobs from outside the provider’s domain. Middlewares provide tools for job submission as well as libraries which users can employ in order to build their own clients. Direct access to a job scheduler is interdicted because job schedulers do not provide sophisticated security features, in contrast to middlewares [22] [23].

In any case, users have to describe a job in a specific format. This format depends on the scheduler or middleware to which the job is submitted; the TORQUE resource manager defines this via command line arguments to its job submission executable “qsub” [24], the middleware Globus Toolkit (GT) uses an XML-based schema [25]. The job description is exclusively composed of functional aspects of the job, for example the executable’s name, the amount of requested nodes and RAM, input and output files etc. In addition to these parameters, the queue specification is a very important part of the job description. Providers define different queues which accept different jobs depending on their resource requirements.

There is no silver bullet regarding the quantity or exact definition of queues, however similarities between different resources and between different providers can be easily identified. Tables 3.1, 3.2 and 3.3 show different queues from different providers.

The level at which queue details are described varies; tabular information gives a quite

Class	Max. runtime	Max. jobs	Max. jobs per user
n8192	24:00:00	1	-
n4096	24:00:00	2	-
n2048	24:00:00	4	-
n1024	24:00:00	8	-
n0512	24:00:00	5	-
small	00:30:00	32	2
nocsmall	00:30:00	1	1

Table 3.3: Batch queues at FZJ's Blue Gene/L [28]

concise overview over a queue setup, sometimes, however, a more informal description is given, for example for Sandia National Laboratories [29] ASCI Red, which was the world's fastest supercomputer from June 1997 to June 2000:

The batch nodes of ASCI Red are allocated through the NQS (Network Queuing System). There are no quotas for use of the ASCI Red machine, but janus limits the number of jobs an individual user may have queued or running at any given time. This limit depends on whether janus is in big or small mode, and on the number of nodes and the computation time requested. In normal use, janus also limits the maximum CPU time that a job can run to 24 hours. Janus-s operates with fewer restrictions; users can submit a greater number of jobs for longer times than on janus. In addition, users with mission- and time-critical needs can request priority use of the machine by contacting janus-managers@sandia.gov. Approved priority requests are handled by giving the user access to special priority queues or by dedicating all or part of the machine to a specific project for a given time. [30]

The queuing setups differ in details as they have to cater to the specific infrastructure they serve. Nonetheless, some similarities can be identified. The most important parameters for job submission are run time and job size as some setups have nearly orthogonal queues regarding these parameters. Further parameters are the amount of RAM, accessibility of queues etc. If jobs do not fit the queuing setup, manual intervention is necessary.

Selection of the queue is the only possibility a user has regarding scheduling as the scheduling algorithm, that is the selection of which jobs from all queued jobs are eligible to run, is implemented by the provider.

3.2 Job scheduling

Job scheduling deals with the problem of which job to run from a number of jobs waiting to be executed. Two main approaches can be distinguished: queue-based and plan-based approaches [31]; plan-based approaches are also called schedule-based approaches. Queue-based approaches, which are in use in HPC environments, as has been shown in section 3.1, provide multiple queues which accept different jobs. Jobs are ordered in a queue according to a scheduling policy, for example “first come, first served” (FCFS) or “shortest job first” (SJF). If only the heads of the queues are eligible to run and not enough resources are available for any queue head, system are wasting time being idle. Backfilling strategies, which schedule jobs which are farther behind in a queue but have less resource requirements, can be used to reduce this problem. Popular backfilling strategies are conservative backfilling, which allows execution of later jobs only if this does not delay the start time of any other job in the queue, and EASY backfilling [32], which allows the execution of later jobs as long as this does not delay the start time of the first job. For backfilling to work, runtime estimates of jobs are necessary; without these, no backfilling is possible.

Popular queue-based scheduling algorithms are:

EDF Earliest Deadline First schedules processes according to their deadline, therefore requiring deadline information. EDF’s advantage is that if it is at all possible to schedule a collection of jobs so that no job violates its deadline, EDF schedules this collection so that no job violates its deadline [33].

FCFS First come, first served sorts jobs in the order they arrive in.

RR Round-robin scheduling, as its name implies, schedules process in a round-robin fashion if multiple queues exist; in HPC, the round-robin scheduling employed in operating systems, where jobs run only short time quanta at a time, is not practicable as the preemption of jobs is expensive.

SJF Shortest job first schedules jobs with shortest runtime first. SJF leads to minimal average waiting time but can lead to starvation of long-running jobs if short jobs are continuously queued.

All of the above can be combined with backfilling in order to increase the system’s utilisation.

Queuing-based systems have the drawback that they cannot provide information about the future, for example when a job will be started, how the load will be etc. Their advantages are low costs of scheduling and a simple way of selecting jobs which are eligible to run.

Systems employing queues for job scheduling can only provide very limited QoS as they do not provide any planning capabilities. Relative QoS, that is the prioritisation of jobs against other jobs, is easily possible and leads to “soft” guarantees on waiting times. Advanced QoS, which includes runtimes and deadlines, cannot be guaranteed with queue-based approaches. It even cannot be known if QoS requirements can be fulfilled.

In contrast, plan-based approaches do not use different queues but sort all jobs in a schedule plan. Incoming jobs are sorted in the schedule with distinct start times. As plan-based approaches have to recompute the schedule if jobs end before their expected finish time, the calculation of a new schedule can be computationally expensive. Advantages of plan-based scheduling approaches are the inherent possibility to provide reservations as each job is associated with timing information and the possibility to optimise the plan with advanced scheduling algorithms [34].

QoS is addressed by both approaches to scheduling. However, what is termed QoS are mostly trivial guarantees that cannot be compared with the requirements identified through the use cases in section 2.4. The number of jobs meeting their deadlines is the predominant factor considered as QoS; other timing-related parameters, for example the wait time, are considered as well [35]. Beyond that, system utilisation is often considered as well in order to not only consider user QoS but also provider QoS [34].

From the two existing scheduling approaches, queue-based and plan-based scheduling, queue-based scheduling is the approach that is used by HPC providers today. Queue-based scheduling is supported by resource managers and scheduling tools out of the box, in contrast to plan-based scheduling which is rarely supported by production-ready software and is rather a research topic. Furthermore, queue-based scheduling has been implemented at HPC providers for a long time and thus providers have good operating experience with it.

This operation experience stems also from the fact that providers often run multiple clusters which are operated through similar queuing setups. The scheduling is performed with simple algorithms, for example FCFS, but additionally a notion of “fairness” is implemented by collecting historical data and prioritising jobs of users with little runtime in the past [15] [36]. QoS is difficult to realise with queue-based scheduling, therefore it is not surprising that HPC providers offer at best very limited QoS; the current business model of only a single contract existing for all users contributes its share to this.

3.3 Service level agreements

This section analyses relevant topics for application of SLAs in an HPC environment: job submission, job scheduling and general SLA management.

3.3.1 Specifying SLAs for job submission

Three prominent specifications for structuring SLAs exist: WS-Agreement [37], WSLA [38] and SLAng [39]. WS-Agreement (WS-Ag) is a proposed recommendation from the Open Grid Forum (OGF) and is the specification with the most uptake; implementations which are maintained actively exist for WS-Ag [40]. WSLA and SLAng have not been widely used and no implementations exist. WS-Ag is an abstract specification that does not provide any domain-specific definition, WSLA is focused on the web services domain and SLAng is still under development and not usable in production [41]. A specialisation of WS-Ag for HPC purposes, HPCWSAG [42], has been recently proposed.

All of these specifications deal with SLAs on a high, contractual level. None of them detail how job submission in relation to SLAs would work. Little research has been performed in this direction. One exemption is an investigation of WS-Ag's suitability for job submission by Sakellariou and Yarmolenko [43]. The authors conclude WS-Ag's general suitability but propose extension, like the ability to describe guarantee terms with variables or functions for the reduction of overhead and more successful negotiations. Even this work, however, deals with SLAs as (sub-)contracts and does not detail the low-level technical details of how a concrete job would be submitted.

3.3.2 Using SLAs for job scheduling

Regardless of how job submission is performed with SLAs and how the corresponding information is handed over to a provider, when SLAs are used for scheduling, all approaches share the fact that they treat SLAs as agreements on a per-job basis. This means that for each job to be submitted, a unique SLA is established before the job can be submitted. It does not matter if QoS requirements of a job are taken as implicitly given per job or as described through an SLA - each job has potentially unique requirements. Therefore, even work that discusses QoS aspects of jobs without mentioning SLAs can be thought of as in the same vein [34] [44]. This section identifies and details work done in this field.

Seidel et al. analysed resource management systems and in how far they provide support for SLAs [45]. They assume that SLAs are negotiated dynamically but do not detail QoS parameters and rather focus on generic SLA support.

Yarmolenko and Sakellariou investigated the influence of different heuristics on the scheduling of parallel jobs, which are described through SLAs and are negotiated on a per-job basis. SLAs describe an individual job's properties using the parameters earliest job start time, latest job finish time, job execution time, number of required nodes and final price [46]. Similarly, Yeo and Buyya investigated SLAs that contain a job's deadline as a parameter and are used for deadline-constrained job admission control in a cluster

[47]; they identified that having solely a job’s deadline as parameter is problematic as job runtime estimates are often inaccurate. SLAs featuring more parameters are investigated by Dumitrescu et al. [48]; the so-called “usage SLAs” are characterised by four parameters: a user’s virtual organisation (VO) membership, group membership, required processor time and required disk space. However, they were only used for scheduling of grid-specific workloads using the BLAST tool from the domain of bioinformatics. Sakellariou and Yarmolenko later proposed a brokered solution enabling SLA-based scheduling on grid resources (see figure 3.1) [49]. They introduced “meta-SLAs”, which are used to describe properties of a workload of jobs; individual job requirements are then derived from this meta-SLA and described in a “sub-SLA”, therefore each job is described by a single SLA but jobs without SLA specification are possible as well and run on a best effort service level then. SLAs are negotiated dynamically on per-job or per-workflow basis.

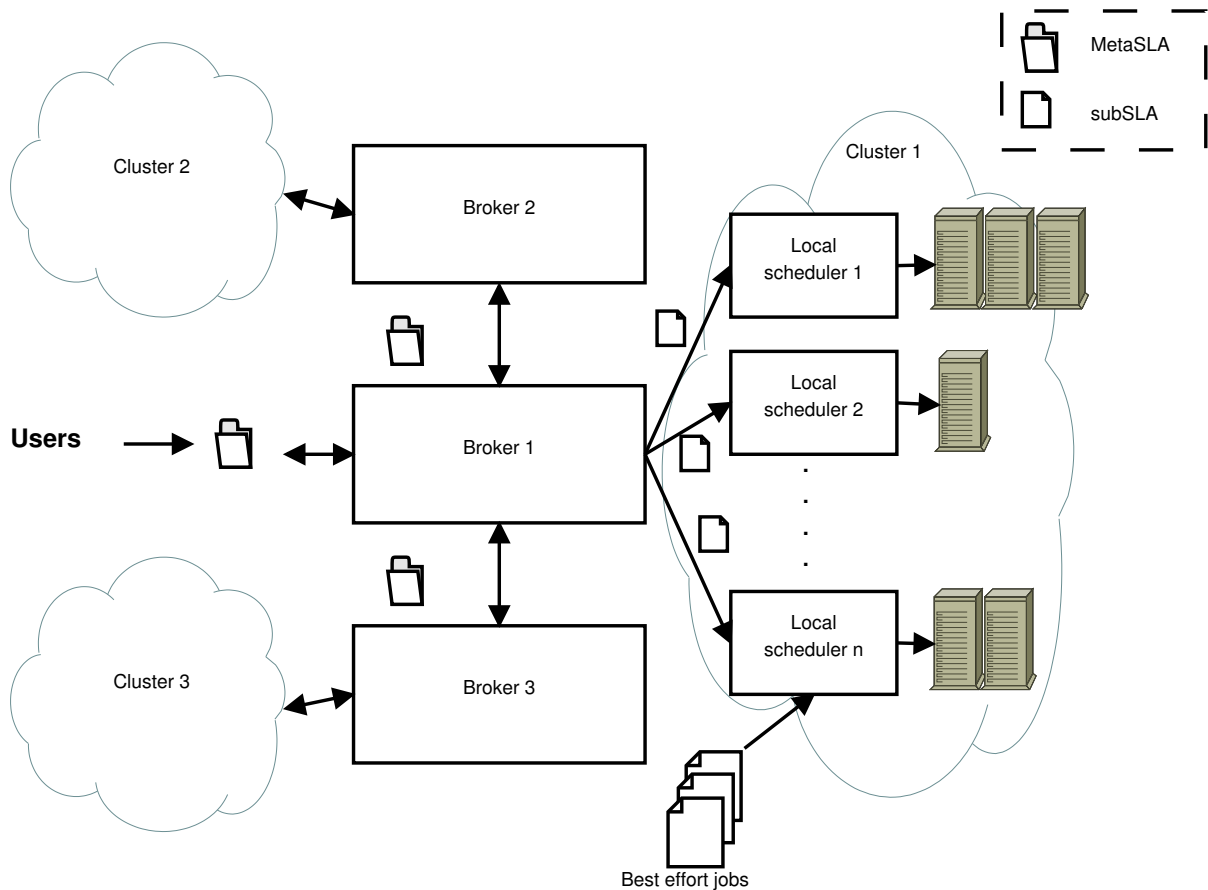


Figure 3.1: Brokered SLA architecture [49]

Djemame et al. discussed similar scenarios for job submission using a brokered architecture as well, but putting the focus on the risk that an SLA will be failing [50]. Risk management is integrated in the resource management system, thereby allowing the scheduling of jobs according to risk information. Ranjan, Harwood and Buyya employed SLAs for superscheduling, similar to the brokered solutions presented before [51]. SLAs

use budget and job deadline as QoS parameters and are negotiated on a per-job basis in a federated environment. If negotiations do not succeed, jobs are dropped out of the system.

Balakrishnan, Thamarai and Britto proposed the usage of SLAs in an environment with multiple resource providers where the SLAs are used to form ad-hoc virtual organisations [52]. SLAs cannot be specified by users in this approach and are only established between a meta-scheduler and individual resource providers. The focus is on selection of appropriate resources for satisfying an SLA, the actual scheduling of the jobs or SLAs is not elaborated.

3.3.3 Managing SLAs

Introducing SLAs in an HPC environment requires for the provider to introduce an SLA management layer into its infrastructure. This is necessary as the ability for each job to be governed by a separate SLA is not currently supported by provider's infrastructures; after all, there is only a singular contract that exists for all users. Thus, there is no need to obtain SLA-related information at job submission and to store this information and make it available to relevant components, e.g. the scheduler, the accounting or even towards the client.

Service level agreement management has been a popular research area in general. The topic has been broadly covered by various European research projects partially funded by the European Union. The research projects NextGRID [8], BEinGRID[53], BREIN [21] and IRMOS [10], just to name a few, have all at least partially investigated service level agreement management. The same holds for German-funded projects from the German grid initiative D-Grid, for example the FinGrid [9] and SLA4D-Grid [11] projects. Regarding SLA management specifically for HPC environments, an architecture has been proposed by Koller [42].

3.3.3.1 Business Experiments ingGrids

Business Experiments in Grids (BEinGRID) was a research project partially funded by the European Commission's Sixth Framework Programme. It started in June 2006 and ran until December 2009. BEinGRID's main aim was to promote the uptake of grid computing in commerce and industry. BEinGRID undertook two activities to this end, firstly to run business experiments targeting different issues, for example supply chain solutions, computational fluid dynamics, gaming etc., secondly to provide a repository of grid technology solutions in order to allow businesses a simplified uptake of grid computing. The solutions developed by BEinGRID are freely available and cover areas like

data management, license management, security, virtualization etc., and service level agreements as well.

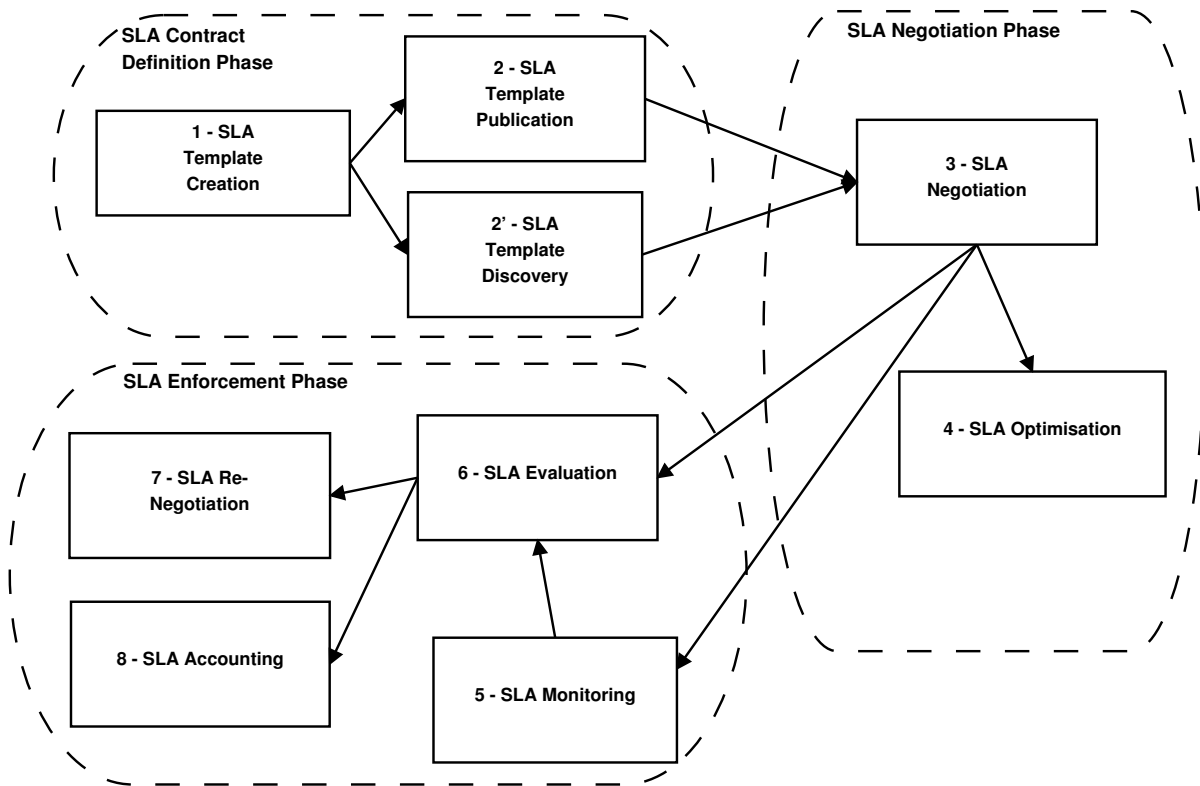


Figure 3.2: SLA management as seen by the BEinGRID project [2]

Figure 3.2 shows a high-level view of the different phases during the lifetime of an SLA that have been identified in BEinGRID. Different requirements were identified from this: SLA template specification, Publication & Discovery, Negotiation, Provider Resource Optimisation, Monitoring, Re-negotiation, Evaluation and Accounting. BEinGRID provides no ready-to-use SLA management framework addressing these requirements but rather composable components addressing specific use cases for SLA negotiation, monitoring and evaluation, violation notification and accounting.

3.3.3.2 Business Objective Driven Reliable And Intelligent Grids For Real Business

Business Objective Driven Reliable And Intelligent Grids For Real Business (BREIN) was a research project partially funded by the European Union's Sixth Framework Programme that started in September 2006 and ran until January 2010. BREIN identified that the focus of developments for the grid is not on business needs of organisations or individuals, which hinders its uptake. BREIN's main goal was the development of a

framework that allows easier access to the grid for business entities. Along with this came the enhancement of web services with regard to intelligent and autonomous decisions.

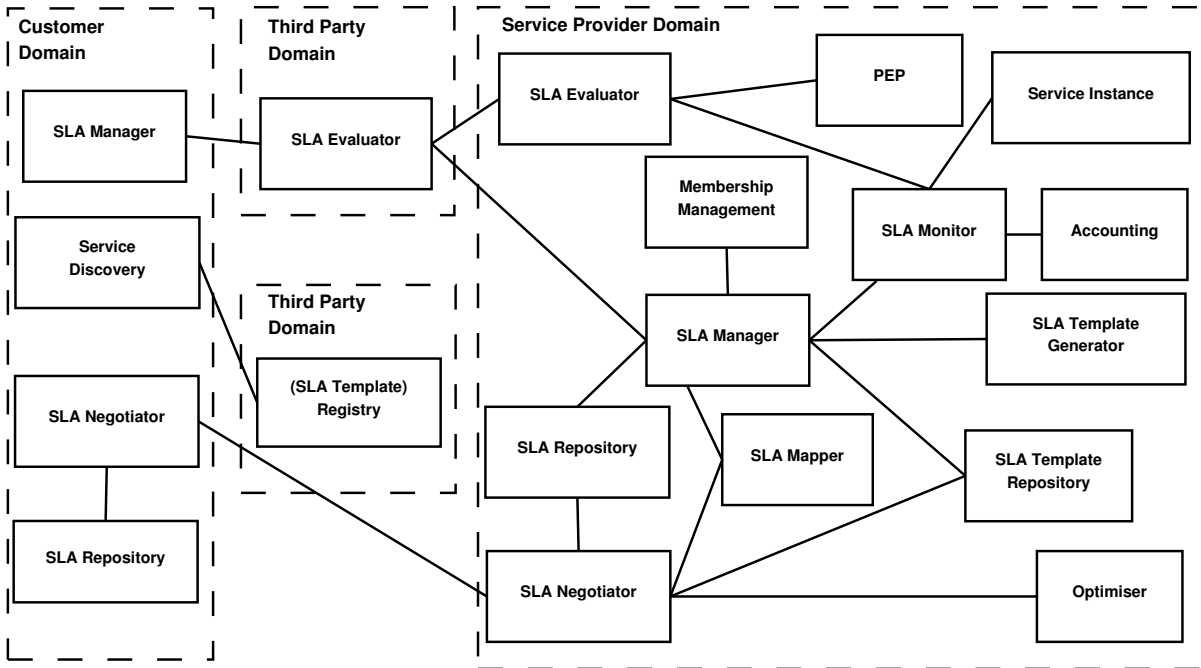


Figure 3.3: SLA management as seen by the BREIN project [54]

Figure 3.3 shows BREIN’s approach to SLA management. Out of the components shown, the project has developed the following components: SLA Manager, SLA Monitor, SLA Evaluator, SLA Negotiator, SLA Repository, a template repository and an SLA translator. BREIN’s SLA management framework is tailored at automating the life cycle of an SLA between clients and service providers. The framework has been developed for both the .NET platform as well as for GT4.

3.3.3.3 Financial Business Grid

Financial Business Grid (FinGrid) was a research project partially funded by the German Federal Ministry of Education and Research (BMBF), which was started in May 2007 and completed in April 2010. FinGrid focused on grids for the financial industry; the main aim of FinGrid’s SLA management framework was to support the negotiation, management and enforcement of SLAs. The architectural definition and implementation was performed by the author, among others. FinGrid implemented a rapid prototype of the architecture, which is shown in full in figure 3.4, which included a subset of the components. The simplified rapid prototype scenario deals with job submission to a grid middleware.

A great set of the components is only needed due to FinGrid’s focus on the establishment

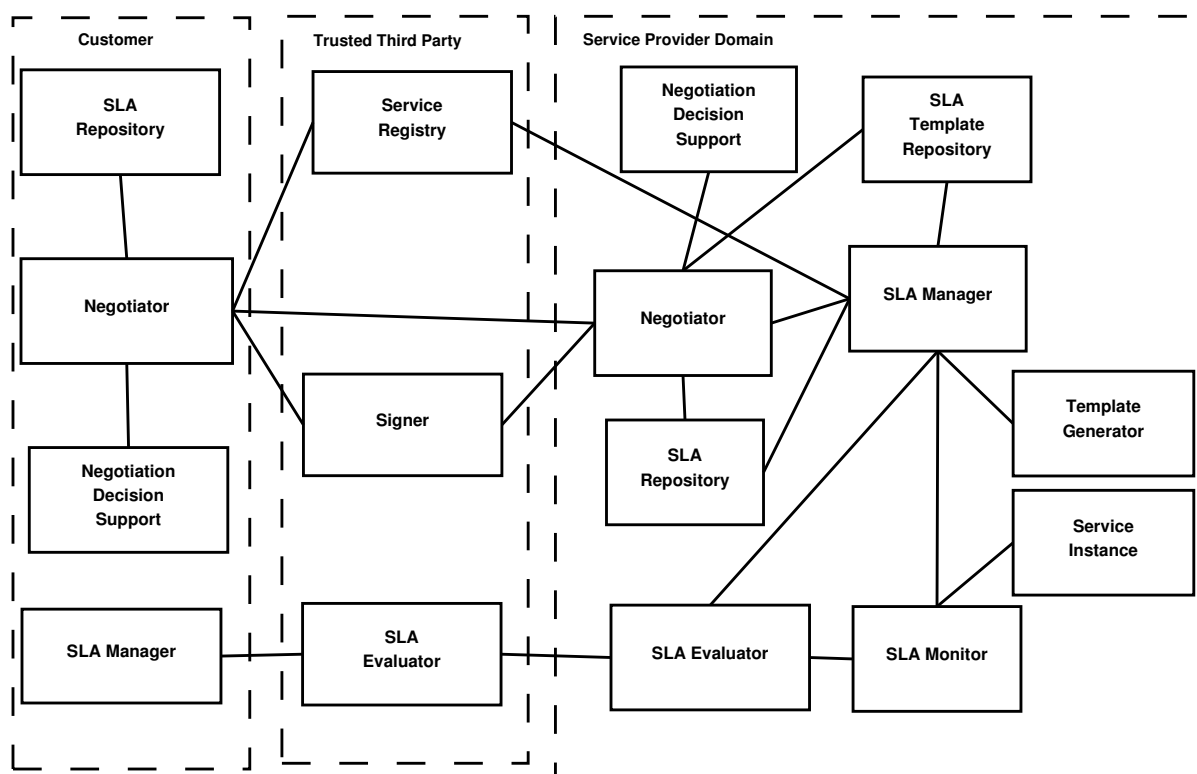


Figure 3.4: SLA management architecture proposed by the FinGrid project [55]

of SLAs: Negotiator, Negotiation Decision Support, Signer, Template Generator and SLA Template Repository. The remaining subset of components is centred around the SLA Manager and is quite lean. The components are developed in the Globus Toolkit 4 middleware.

3.3.3.4 HPCWSAG

HPCWSAG has been already mentioned in section 3.3.1 because it proposes a structure for SLAs in the HPC domain [42]. In addition to this structure, HPCWSAG also proposes an SLA management framework. The SLA management framework intends to cover six phases of an SLA's lifecycle: development, creation, provisioning, execution, assessment and termination. Covering so many different phases of an SLA's lifecycle at once leads to the resulting framework being very complex, which is already apparent through the 36 different components which make up the whole framework.

3.3.3.5 IRMOS

Interactive Realtime Multimedia Applications on Service Oriented Infrastructures (IRMOS) was a research project partially funded by the European Union’s seventh framework programme and was running from February 2008 to January 2011. IRMOS aimed at supporting real-time aware applications provisioned over service-oriented infrastructures. QoS was ensured through SLAs; IRMOS’s architecture was divided along the lines of cloud computing service provisioning with service providers at the infrastructure, platform and application layer. As each layer contains potentially independent service providers, IRMOS requires SLAs between each layer; as shown in figure 3.5, the SLAs between Infrastructure as a Service provider (called “ISONI provider”) and Platform as a Service provider (called “IRMOS provider”) and Software as a Service provider (called “Application provider”) and client are dynamically negotiated. The SLA between Platform as a Service provider and Software as a Service provider is, in contrast, a static SLA.

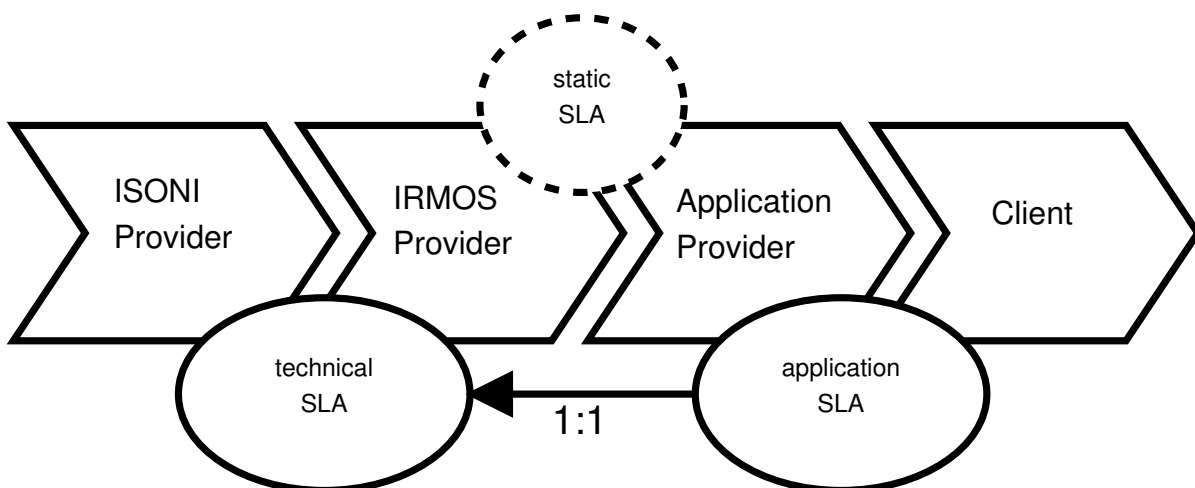


Figure 3.5: SLAs in the IRMOS architecture [56]

Each of the SLAs needs to be managed on its own; IRMOS provided an elaborate SLA management framework for the PaaS provider layer through its main software development, the so-called Framework Services.

Figure 3.6 shows a high-level view of IRMOS’ service management. The main responsibility of the service management was the on-demand negotiation of SLAs, resource reservation, service instantiation, service execution and QoS monitoring for deployed applications.

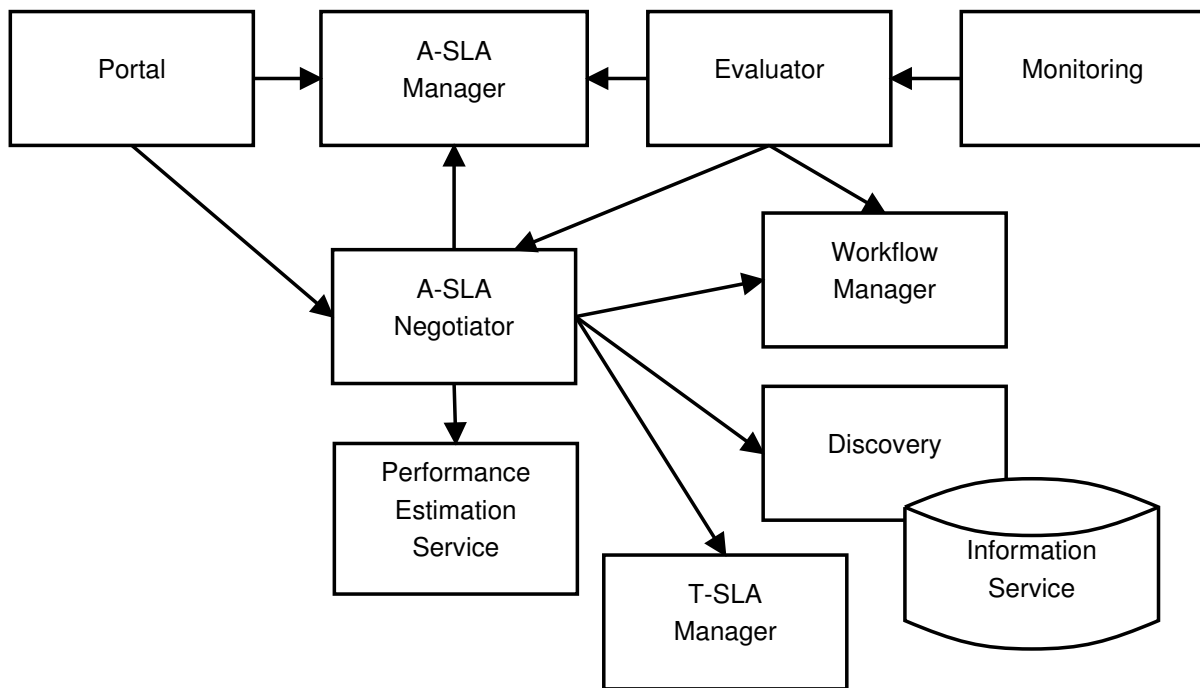


Figure 3.6: IRMOS service management [56]

3.3.3.6 NextGRID

The NextGRID research project, partially funded by the European Union's 6th Framework Programme, started in September 2004 and ended in February 2008. NextGRID's goal was to develop architectural components for next-generation grids. SLAs were part of NextGRID's core architecture as each interaction in NextGRID was governed by an SLA.

NextGRID developed its own SLA schema which is in parts similar to the schema proposed by WS-Agreement, but it is not compatible. NextGRID did not provide a full SLA management framework but focused on SLA negotiation and related aspects like template provisioning, template evaluation and service provisioning.

3.3.3.7 Service Level Agreements for D-Grid

Service Level Agreements for D-Grid (SLA4D-Grid) is a research project partially funded by the German Federal Ministry of Education and Research (BMBF), which started in June 2009 and was running until May 2012. The project's aim was to develop an SLA layer for D-Grid, the German Grid Initiative. The SLA layer is situated above the middleware and can be integrated transparently into applications or even not considered at all if applications are not compatible with it, as shown in figure 3.7.

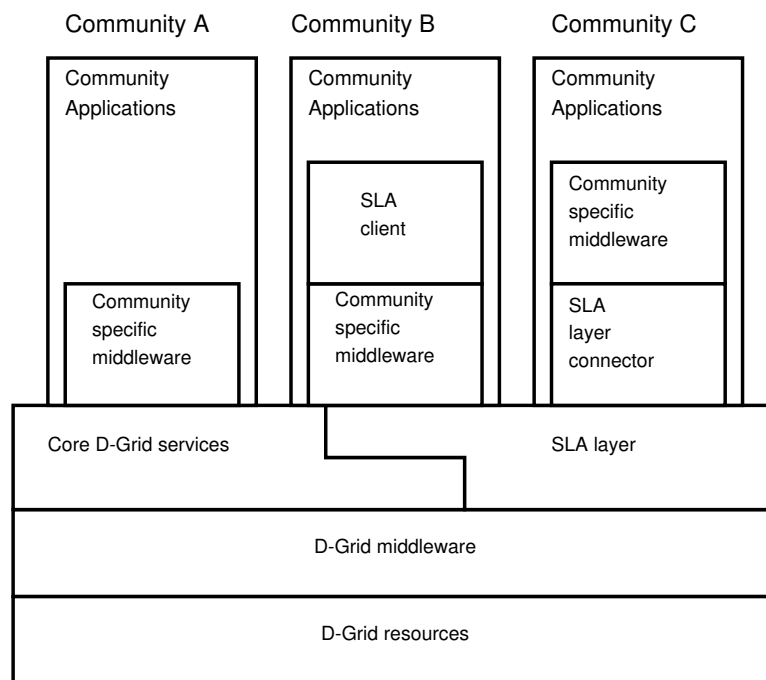


Figure 3.7: The SLA layer developed by SLA4D-Grid in the D-Grid

The SLA layer allows the automatic creation, negotiation and observance of SLAs in order to use the D-Grid infrastructure in a more efficient way. Figure 3.8 shows a simplified architecture of the SLA layer based on individual components.

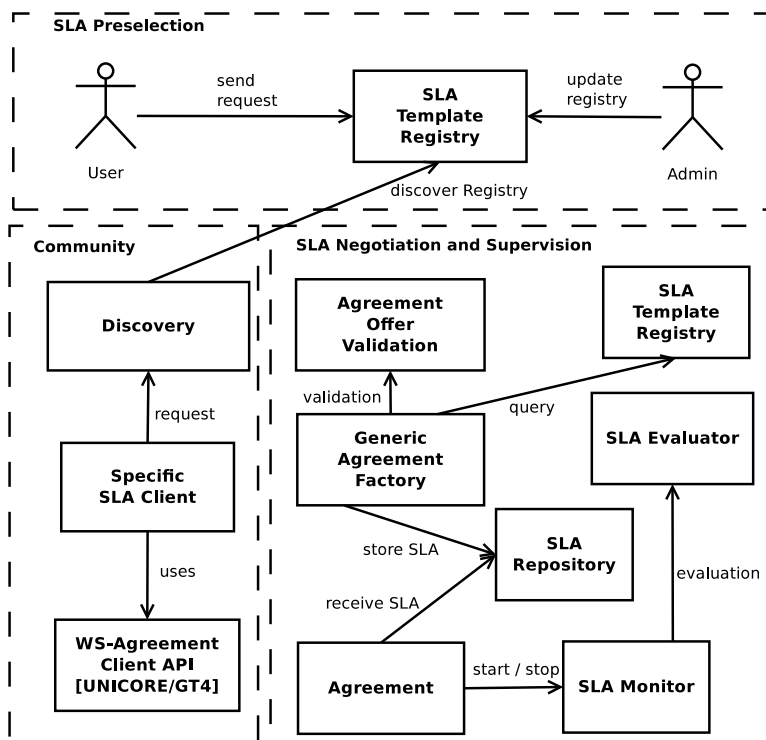


Figure 3.8: SLA4D-Grid's SLA layer architecture

3.3.3.8 Summary

This section has investigated different approaches to SLA management. The approaches differ muchly but some common points could be observed. A comparison of these points is given in table 3.4. The first important point is support for HPC. Some approaches support HPC out of the box, others do not offer specific support for HPC. In general, even if HPC support is given, this does not mean that a provider can easily employ an SLA management framework as the provider's infrastructure likely uses a differing implementation. Secondly, a point that all frameworks have in common is the strict 1-to-1 mapping between SLAs and jobs. This results in provider and client having to negotiate and individual SLA for each job that is to be submitted. Except for HPCWSAG, all approaches that were investigated provide at least a partial implementation of their proposed components. The complexity varies but is, in general, quite high - NextGRID, for example, propose more than 40 different components and this does not even include HPC specific components. If only a part of the SLA lifecycle is addressed, the focus is usually on negotiation. Even then, however, the number of components is high.

Name	HPC	1 SLA per job	SLA focus	Components	Implementation
BEinGRID	Yes	Yes	Negotiation	9	Yes
BREIN	Yes	Yes	Whole Lifecycle	19	Yes
FinGRID	Yes	Yes	Negotiation	16	Yes
HPCWSAG	Yes	Yes	Whole Lifecycle	36	No
IRMOS	No	Yes	Whole Lifecycle	20+	Yes
NextGRID	No	Yes	Whole Lifecycle	40+	Yes
SLA4D-Grid	Yes	Yes	Negotiation	11	Yes

Table 3.4: Comparison of SLA management approaches

3.4 Resource managers and job schedulers

The distinction between resource managers and job schedulers is often not clear and the terms are sometimes used synonymously, sometimes with a different meaning. There is no exact definition for both terms and products offering functionality discussed here often claim to offer resource management and job scheduling functionality. This is logical since job scheduling in this case means the scheduling of jobs on distributed compute nodes, in contrast to the scheduling of processes onto central processing units performed by the operating systems. In order to schedule jobs on distributed resources, a centralised scheduling software needs to know about the resources themselves, which means that basic resource management capabilities are needed anyway. TORQUE, for example, which is often credited as being a resource manager, is described by the developers as "an open-source resource manager providing control over batch jobs and distributed compute nodes", therefore uniting the resource management and scheduling functionality.

This section investigates the capabilities of current resource management and job scheduling software and in how far they support service level agreements.

3.4.1 Portable Batch System derivatives

The *Portable Batch System* (PBS) is a software used for job scheduling, allocating tasks to available compute resources. PBS's interface is supported by various meta schedulers, notably *Moab* and the *Grid Resource Allocation Manager* (GRAM), a component of the Globus Toolkit.

PBS, which was developed in the beginning of the 1990s, has evolved into different software products:

OpenPBS

the now unsupported original open source version.

TORQUE

an advancement of OpenPBS that is actively developed by Cluster Resources Inc.¹

PBS Professional

a commercially available version sold by Altair Engineering.²

The following sections describe the different versions.

¹<http://www.clusterresources.com/>

²<http://www.altair.com/>

3.4.1.1 OpenPBS

OpenPBS has been taken over by PBSWorks and is not offered by the company anymore; instead, the commercially available *PBS Works* is the successor of OpenPBS. The Math & Computer Science Division of the Argonne National Laboratory, however, maintains a collection of patches for OpenPBS.³ It is, however, unclear as to if and where the original code for OpenPBS can be obtained, as it is not available through the PBSWorks web site nor through any other website; various installation guides and how-tos offer defunct links which only redirect to the PBSWorks home page's root. TORQUE is the software which is then used instead of OpenPBS and since TORQUE is an evolution of OpenPBS, it can be assumed to offer a superset of OpenPBS's functionality.

3.4.1.2 TORQUE

TORQUE, short for *Terascale Open-Source Resource and QUEue Manager*, has been developed based on PBS. TORQUE offers various features related to job scheduling and resource management, most notably the submission and management of jobs, the management of nodes and the setting of server policies.

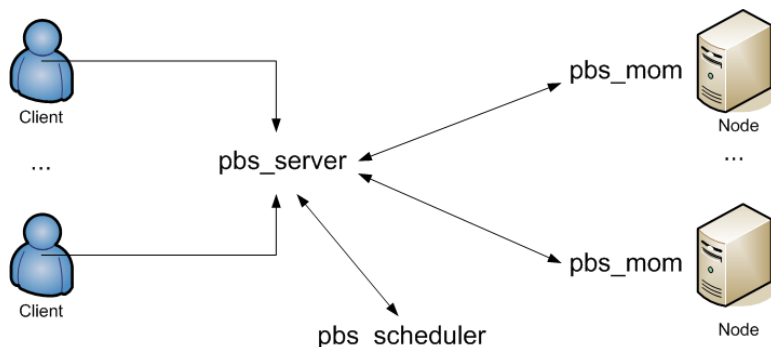


Figure 3.9: PBS/TORQUE architecture

TORQUE itself, even though offering job scheduling functionality, considers itself only as a basic scheduler, “because the *default* TORQUE scheduler, **pbs_sched**, is very basic and will provide poor utilisation of [a] cluster’s resources”⁴. The default scheduler used by TORQUE is named *fifo* and offers the following functionality:

The default PBS scheduler implementing a variety of common scheduling algorithms, including: round-robin, by-queue, strict-fifo, fair-share, and load-balancing. Each of these can be further configured to use a variety of job sorting methods to indicate what class of job should be run first: no-sort, shortest,

³<http://www.mcs.anl.gov/research/projects/openpbs/>

⁴<http://docs.adaptivecomputing.com/torque/4-0-2/Content/topics/5-scheduler/integratingSchedulers.htm>

longest, smallest-memory, largest-memory, high-priority, low-priority, large-walltime, ascending-walltime, and combinations of these. ⁵

TORQUE has no notion of service levels but supports policies through the usage of queues. Therefore, service level like behaviour can be emulated by using queues with different priorities where jobs from a higher-priority queue are scheduled before jobs of a lower-priority queue. There is no support for service level agreements, but through an SLA-aware scheduler this functionality can be added and a TORQUE-installation can be enhanced to use SLAs. This requires the implementation of a corresponding scheduler and a recompilation of TORQUE with this scheduler.

3.4.1.3 PBS Professional

PBS Professional is a commercial product sold by PBS Works, a division of Altair Engineering. It is as yet not clear if PBS Professional offers support for SLAs, but the lack of information about this indicates that this is not the case.

3.4.2 Condor

Condor is an open source job scheduling software for high throughput computing (HTC) developed by the University of Wisconsin-Madison ⁶. Condor's approach differs from the approach of "classical" job schedulers, like PBS and its derivatives mentioned above, in that it is primarily used to pool workstations and run jobs once these workstation are idle for a certain time. If user activity is detected, jobs are removed from the workstation and are added back into the queue. Condor is supported by the Globus Toolkit's Grid Resource Allocation Manager (GRAM). Condor-G enhances Condor in order to use grid and cloud resources, for example from Globus, UNICORE and Amazon EC2, or batch resources managed by PBS and Load Sharing Facility.

The concept of high-throughput computing is different from the high performance computing case regarded in this work in the sense that "HPC deals with floating-point operations per second" and that "HTC [deals with] floating-point operations per year" [57]. In other words, while HPC deals with tightly coupled jobs that have high memory and specific architecture requirements - for example, vector processors -, HTC processes massive amounts of independent jobs. Condor's approach is similar to the well-known Berkeley Open Infrastructure for Network Computing (BOINC) which uses idle time to run various projects, for example SETI@home and LHC@home. The fact that idle time is more or less randomly provided to Condor and therefore the available compute power

⁵<http://www.clusterresources.com/torquedocs21/5.1schedulers.shtml>

⁶<http://www.cs.wisc.edu/condor/>

may fluctuate greatly means that service levels as they are needed for traditional HPC and distributed computing applications are not applicable. Quite reasonably, Condor therefore has no notion of service levels nor support for SLAs.

3.4.3 Load Sharing Facility

Load Sharing Facility (LSF) is a commercial job scheduler developed by Platform Computing, an initially privately owned company that has been acquired by IBM in January 2012. LSF supports “goal-oriented SLA-driven scheduling”⁷. An SLA is “a “just-in-time” scheduling policy that defines an agreement between LSF administrators and LSF users”. Goals for LSF are specified in service classes which consist of a service class name, one or more goals, a time window when the goal is active and a service class priority. Goals for service classes can be either deadline goals (a specified number of jobs should be completed in a defined time frame), velocity goals (a specified number of jobs should be running concurrently in a defined time frame) or throughput goals (a specified number of jobs should be finished in a defined time frame). Service classes can be thought of as high level queues as this is the way they appear to users. LSF then uses lower level policies like queues and host partitions to satisfy a service class’s service level goals.

3.4.4 LoadLeveler

LoadLeveler (full name Tivoli Workload Scheduler LoadLeveler) is a job scheduler developed by IBM. LoadLeveler matches processing requirement of jobs and their priority to available resources in order to maximise resource utilisation [58]. In addition to the pure scheduling, LoadLeveler also supports workload management and accounting.

LoadLeveler does not seem to know the notion of complex service levels but offers the priority to schedule jobs based on their priority. Priorities can be derived from different features of a job, for example job run-time (short or long-running) or job owner (user or group). Resources can be restricted to only be available at a certain time, for specific users or specific job classes.

3.4.5 Maui Cluster Scheduler and Moab Cluster Suite

The Maui Scheduler is an open source job scheduler with extended management capabilities, supporting among other things multiple scheduling policies, dynamic priorities, reservations and fairshare capabilities. A commercial and enhanced derivative of Maui,

⁷http://www.cisl.ucar.edu/docs/LSF/7.0.3/admin/sch_sla_aware.html

Moab Cluster Suite, is sold by Cluster Resources, Inc. There exists another job scheduler named Maui Scheduler that is available through SourceForge ⁸. This latter application seems to have been planned as an extension to the original Maui Scheduler but has seen no file releases in the last five years and seems to be dead.

3.4.6 Oracle Grid Engine

The Oracle Grid Engine (OGE), previously known as the Sun Grid Engine, is an open source batch queuing system. After Oracle's purchase of Sun in 2010, the project has been discontinued by Oracle, which prompted the Grid Engine community to fork the Open Grid Scheduler project⁹ in order to maintain a free version of Grid Engine. Another, commercial, fork has been created by Univa Corporation¹⁰ [59].

OGE did not support service levels out of the box, but an add-on, Service Domain Manager (SDM), enables support for Service Level Objectives (SLO). This is realized by the SDM knowing about a pool of resources and being responsible for multiple clusters. If a cluster notices it is violating one or more SLOs, for example by having too many pending jobs, it asks the SDM for more resources, which are assigned to the requesting cluster if available. It is unclear if the SDM is supported by the Open Grid Scheduler project.

3.4.7 SLURM

SLURM is an open-source resource manager developed by the Lawrence Livermore National Laboratory and other contributors [60]. It has been primarily developed for Linux, but should be easily portable to other UNIX-like operating systems. SLURM itself provides three key functions: allocation of resource access, a framework for work execution and monitoring and the management of a queue of pending work.

SLURM has a built-in FIFO scheduler but can be enhanced with plugins for other schedulers. There are readily available plugins for, among others, Load Sharing Facility, Maui Scheduler and Moab Cluster Suite.

SLURM supports the specification of QoS for jobs. QoS specifications will influence a job's scheduling priority, preemption and limits. QoS influences a job's priority if a specific plugin, the *multi-factor plugin*, is loaded and is one factor, apart from age (the time the job has been waiting in the queue), fair-share (difference between promised and consumed resources), job size (number of allocated nodes) and partition (a factor

⁸<http://sourceforge.net/projects/mauischeduler/>

⁹<http://sourceforge.net/projects/gridscheduler/>

¹⁰<http://www.univa.com/>

associated with each node partition). Factors are weighted so that a job's priority is computed with the following formula:

$$\begin{aligned} \textit{Job_priority} = & \\ & (\textit{PriorityWeightAge}) * (\textit{age_factor}) + \\ & (\textit{PriorityWeightFairshare}) * (\textit{fair} - \textit{share_factor}) + \\ & (\textit{PriorityWeightJobSize}) * (\textit{job_size_factor}) + \\ & (\textit{PriorityWeightPartition}) * (\textit{partition_factor}) + \\ & (\textit{PriorityWeightQOS}) * (\textit{QOS_factor}) \end{aligned}$$

The preemption QoS determines which jobs are allowed to preempt which other jobs. As for QoS limits, various options exist to limit a job's resource consumption, for example the maximum number of CPU minutes consumable (per job and per QoS class), maximum number of jobs per user, maximum number of nodes for a job, maximum number of jobs allowed in the system etc.

3.4.8 Grid MP

Grid MP is a commercial product by Univa UD for the building of distributed computing environments using non-dedicated resources [61]. In that sense, it is similar to Condor, which has been discussed in section 3.4.2.

According to Univa UD's statements, Grid MP provides some form of job prioritisation, but no further public information can be found regarding this statement.

3.4.9 Application Level Placement Scheduler

The Application Level Placement Scheduler (ALPS) is a software suite for addressing resource and workload management on Cray computers. ALPS is built with a modular architecture with specialised components for specific tasks. Different components run on different types of nodes (service nodes, login nodes and compute nodes).

ALPS provides functionality to reserving nodes, placing and launching applications, management and monitoring. ALPS's scheduling component, *apsched*, manages memory and processor resources and guarantees the correctness of application placement, providing a list of resources for a reservation or placement request. It does, however, not enforce any policy, leaving policy enforcement to workload managers or other components. ALPS's client, for example, can be used as a gateway between ALPS and third-party batch systems. Communication with batch systems is enabled through the Batch and Application Scheduler Interface Layer (BASIL), an XML-RPC protocol.

3.4.10 Summary

Table 3.5 gives an overview over the resource managers and job schedulers presented in this section and their features regarding QoS and SLA support. None of the analysed products supports SLAs as presented previously in section 3.3, although some offer support for QoS. QoS in this sense relates mostly to prioritisation only. About half of the products offer support for advance reservation, which allows to give guarantees on precise execution times at the cost of a fragmented workload.

Name	QoS support	Advance Reservation	Commercial	SLA support
OpenPBS	No	Yes	No	No
TORQUE	No	No	No	No
PBSPPro	No	Yes	Yes	No
Condor	No	No	No	No
Load Sharing Facility	No	Yes	Yes	No
LoadLeveler	Yes	No	Yes	No
Maui	Yes	Yes	No	No
Oracle Grid Engine	No	No	Yes	No
SLURM	Yes	Yes	No	No
GridMP	No	No	Yes	No
ALPS	No	No	Yes	No

Table 3.5: Comparison of resource managers and job schedulers

3.5 Simulation of HPC resources

Computer simulations are programs that are used to simulate abstract models of particular systems. There are many incentives for using computer simulations instead of studying a particular system's behaviour in the real world: sometimes it is impossible to study a system directly, for example for climate prediction, sometimes this is possible but dangerous, for example handling of hazardous materials and sometimes the simulation is cheaper, not only in financial terms but also in terms of setup time etc., for example in car crash simulations.

Simulations are classified according to different dichotomies, among others this are:

- Stochastic or deterministic
- Steady-state or dynamic
- Continuous or discrete

- Local or distributed

The most important attribute pair is the third one, continuous versus discrete, which are different ways of treating state changes in a system. In continuous simulation, state changes appear continuously in time and the system is thus described through differential equation systems. Continuous time simulations are, for example, used for the simulation of fluid dynamics. In discrete time simulations, events appear only at certain moments in time. Simulation time jumps from one event to another. This approach is used to describe systems like communication networks and computing systems.

Discrete time simulation models can be further classified into time oriented and event oriented models. Time oriented systems keep track of the simulation time and advance it in constant intervals. If events appear irregularly or there are long intervals without state changes, this model is inefficient. Then, event oriented models are advantageous. In their case, the system is only observed when events occur. The simulation time is advanced to the time of the occurring event and the system state is updated.

This simulation model, called discrete event simulation (DES) is suited well for the simulation of scheduling in HPC resources. Events are, for example, the arrival of a job, a job's completion, resource failures, etc. The system reacts on events, for example the arrival of a new job, and changes its state, for example queuing a job or executing it immediately. The system's behaviour between events is of no importance if all relevant events are kept and a DES offers much better performance compared to a time oriented simulation.

This section examines simulation tools that can be used to model the usage of service level agreements for scheduling. Even though the direct simulation of SLAs is not supported by any tool, various approaches to scheduling have been realised and will be investigated in this section. A plethora of simulations tools and languages exist; [62], for example features a long but most likely not exhaustive list of software libraries, visual simulation tools, simulation languages, etc. which consists of 25 software libraries alone. This work analyses a relevant selection of existing tools.

3.5.1 SimJava

SimJava is a discrete event simulation package provided as a Java API [63]. Simulations are built out of entities, running in their own thread, connected together by ports for communication purposes. Entities can send and receive events via ports. A central class coordinates the threads, advances simulation time and delivers events to threads. Simulation progress is recorded through messages produced by the entities and is saved in a file. SimJava consists of the three packages `simjava`, `simanim` and `simdiag`; `simjava` is the main, text-based simulation package, `simanim` provides animations and `simdiag` can graphically display results.

SimJava's concept is very generic and thus it can be used to simulate various systems, as long as these can be modelled according to SimJava's architecture of ports, entities and events. It has no special support for scheduling algorithms, job scheduling or service level agreements. SimJava's current version 2.0 has been released in December 2004.

3.5.2 GridSim

GridSim is built on a modified version of SimJava 2. GridSim is a toolkit for modelling and simulating parallel and distributed computing, with a focus on design and evaluation of scheduling algorithms [64]. GridSim is therefore less generic than SimJava but provides various interesting features for the analysis of scheduling algorithms, especially the ability to read failure and workload traces, support for the modelling of resource failures, support for advance reservation, etc. GridSim includes visualisation support, but only for debugging, not for production simulation runs. GridSim's current version is 5.2 beta, which has been released in November 2010.

3.5.3 MicroGrid

MicroGrid is a set of simulation tools that enable the analysis of middlewares, applications and network services for computational grids [65]. This is achieved by simulating application on a virtual grid environment. Applications run on virtual hosts which are mapped to actual physical hosts; library calls as *gethostname*, *bind*, *send*, *receive* and process creation need to be intercepted and mapped to the corresponding host id. Inside the simulation, virtual time is used in order to allow correct behaviour of programs that monitor system time. Physical resources, such as computation and networking, need to be virtualized as well.

MicroGrid focuses on providing a virtual environment for simulating applications running on a computational grid. To this end, it virtualizes all necessary library calls and resources. The middleware itself is only modified as far as it is necessary for it run on virtualized resources. MicroGrid provides a simulation environment that behaves just like a real environment and can be used to analyse how applications behave on real computational grids. Thus, it is more an emulator than a simulator. MicroGrid is available as free software, the latest version 2.4.6 has been released in December 2004.

3.5.4 Bricks

Bricks is a Java-based performance evaluation system for scheduling algorithms and scheduling framework components [66]. It allows the simulation of resource schedul-

ing algorithms, programming modules for scheduling, network topologies and processing schemes for networks and servers.

Bricks consists of two parts, a scheduling unit and a grid computing environment (see figure 3.10). The scheduling unit models a canonical scheduling framework and its associated components while the grid computing environment models physical resources (clients, servers and networks). Bricks is, however, not available as software, neither free nor proprietary.

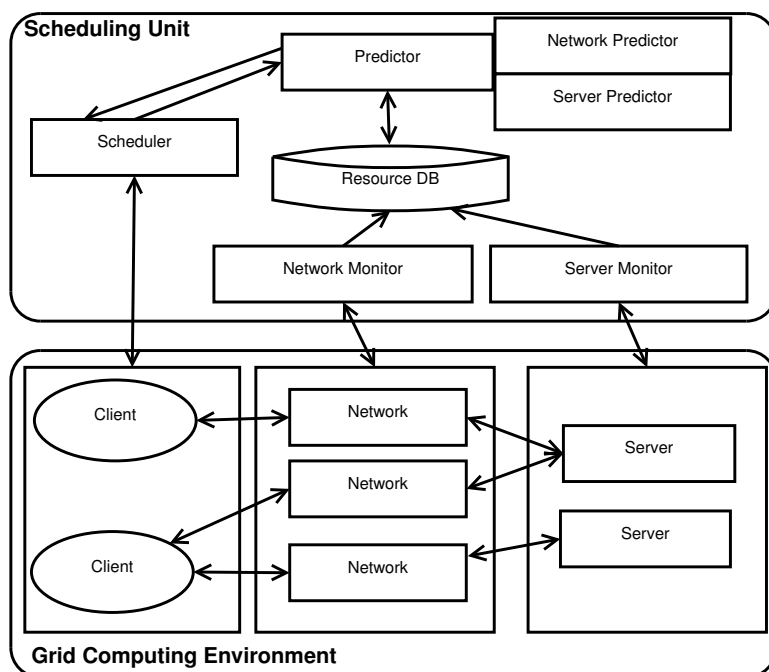


Figure 3.10: The architecture of Bricks [66]

3.5.5 SimGrid

SimGrid is a toolkit for the simulation of distributed applications in heterogeneous distributed environments [67]. SimGrid aims at facilitating research in distributed systems like grids, peer to peer systems and clouds. It allows the simulation of computation with different models and supports the simulation of varying network topologies and compute resources. Besides the simulation engines provided by SimGrid, it provides a high-level interface for the prototyping of simulations (C or Java) and APIs for developers of distributed computing applications which can either simulate an application or run it in a “real-world mode”.

SimGrid was initially developed for the study of scheduling algorithms for heterogeneous platforms, allowing the prototyping of scheduling heuristics and their test on a variety

of applications and platforms [68]. Later versions have improved the simulation engine, added support for dynamic resource availabilities and failures, etc.

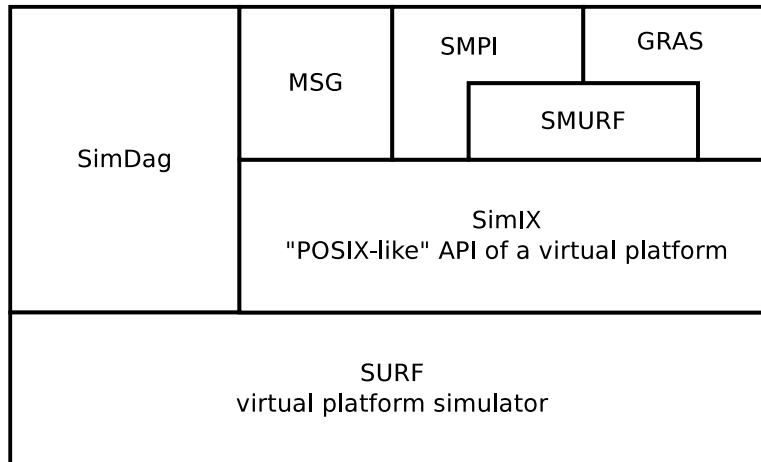


Figure 3.11: SimGrid component overview [67]

The *SimDag* component (see figure 3.11) is responsible for the simulation of scheduling heuristics. SimGrid uses directed acyclic graphs to describe tasks and their dependencies.

SimGrid is freely available under the GNU Public License; the latest version has been published in May 2012 and is available through the SimGrid website¹¹.

3.5.6 Simbatch

Simbatch is a C API for the development and evaluation of grid and cluster scheduling algorithms [69]. It relies on SimGrid (see section 3.5.5) for the simulation of clusters and their schedulers. Simbatch results can be visualised with the Pajé visualisation tool¹².

Simbatch is freely available under the GNU Public License, the latest version has been published in May 2007 and is available through the Simbatch web site¹³.

3.5.7 MONARC

MONARC (MOdels of Networked Analysis at Regional Centers) is a discrete event simulation framework that can be used to design and model tools for large scale distributed

¹¹<http://simgrid.gforge.inria.fr/>

¹²<http://paje.sourceforge.net/>

¹³<http://graal.ens-lyon.fr/simbatch/>

systems [70]. It was developed for application to high energy physics experiments with focus on the Large Hadron Collider (LHC) experiments at the European Organization for Nuclear Research (CERN).

MONARC is structured into three packages called *engine*, *network* and *monarc*. The *engine* package contains the simulation core, the *network* package simulates data traffic on LANs and WANs and the *monarc* package simulated CPUs, jobs, job schedulers and provides GUIs, parsers, random number generators etc. User modifications are foreseen to take place in the *monarc* package. MONARC's focus on distributed systems becomes obvious in its architecture as *regional centers* are a core concept. The infrastructure modelled by MONARC consists of interconnected regional centers which in turns possess clusters, database servers and mass storage units etc. Each regional center can have a scheduler of its own.

MONARC is available under a proprietary license by the California Institute of Technology (CIT). The latest release has been made available in August 2007¹⁴.

3.5.8 Grid Scheduling Simulator

Grid Scheduling Simulator (GSSIM) is a framework for the simulation of scheduling algorithms in heterogeneous infrastructures built on SimJava 2 and GridSim [71]. GSSIM can use real data or generate synthetic data for workloads, resources, users and events; furthermore, GSSIM distinguishes between resource brokers, which are used to schedule jobs between different providers, and local schedulers, which are used to schedule jobs at a provider into appropriate queues (see figure 3.12).

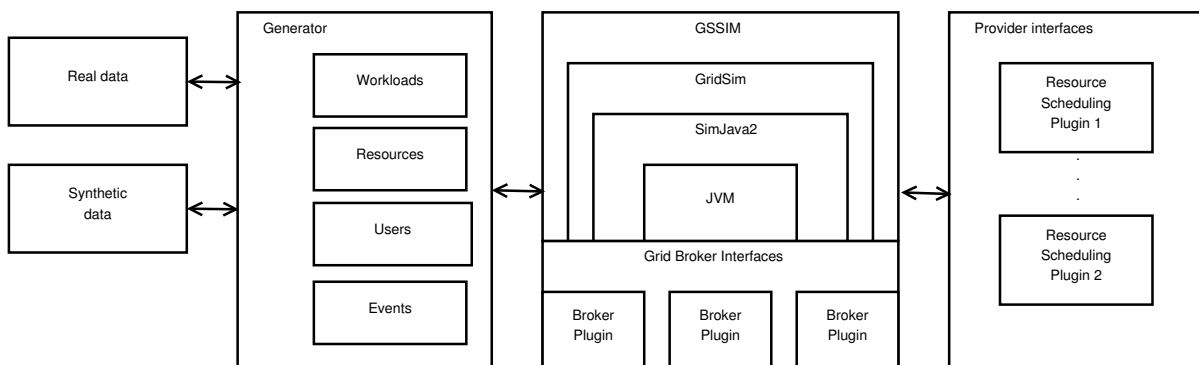


Figure 3.12: The architecture of GSSIM [71]

GSSIM is actively used, for example in the project CoolEmAll¹⁵, which started in October 2011 and is partially funded by the EU's Seventh Framework Programme. The latest

¹⁴<http://monarc.cacr.caltech.edu/>

¹⁵<http://coolemll.eu/>

version has been released in August 2011. It is available freely for download from the GSSIM homepage¹⁶, but the license conditions are unclear. The current version 1.0 has been released in August 2011.

3.5.9 Alea

Alea, “Grid Scheduling Simulation Environment” [72], is an extension of GridSim [64] (see subsection 3.5.2). It is built in a modular architecture where different entities, which communicate via message passing, are composed to form the whole system. The entities are

- the scheduler
- the job submission system and
- grid resources.

Alea is normally used to read in workload traces, but a job generator, that can generate a synthetic workload trace is also supplied; by default it supports uniform and normal distribution of job inter-arrival times. For either a generated synthetic workload or an existing, real-world workload trace, the job submission system is responsible for the handling of jobs and communicates with the scheduler. The scheduler is the main component of the application and - also following a modular architecture - consists of three parts: the first part for communication with the job submission system, the second part for the management of data regarding grid resources and the third part, the scheduling algorithms. Grid resources are executing jobs, communicating with the scheduler for receiving jobs and reporting finished jobs to the job submission system.

Alea has been further developed into Alea 2[73]. The new version features an improved design with extended functionality, providing better scalability and higher simulation speed. Consequently, Alea 2 allows the usage of much bigger workload traces and much faster simulation runs. The main new feature, except from internal implementation changes, is the capability to provide visualisations of various characteristics of a simulation: average system usage per hour and per day, cluster usage per hour, average cluster utilisation per day, number of waiting and running jobs, number of requested available and used CPUs and average up and down time per day. Alea 2.1 has been released in October 2009 and has been followed up by Alea 3.0, released in July 2011.

¹⁶<http://www.gssim.org/>

3.5.10 Pyss - the Python Scheduler Simulator

Pyss is an event-based scheduling simulator programmed in the Python programming language [74]. It has been developed at the Hebrew University of Jerusalem, which also maintains the Parallel Workloads Archive. Pyss' focus lies on the simulation itself, therefore the implemented scheduling algorithms centre around backfilling, apart from a simple FCFS algorithm. Pyss can read workloads in standard workload format (SWF) and accepts machine specification as number of processors. The project is still in prototype state and there has not been any release. The last activity in the project's source code repository has taken place in November 2008.

3.5.11 Summary

This section analysed tools for the simulations of HPC resources. A plethora of simulation tools exist and they vary greatly in level of abstraction, support for different scheduling algorithms, visualisation support etc., as shown in table 3.6. If we do not take into account proprietary software and software that has never been released, still a number of tools remain. In part, the selection of one of the tools is surely motivated by individual taste, for example by taking personal skill in programming languages into account. Apart from that, the above-mentioned features help to arrive at a decision.

Name	Programming Language	License	Workload Trace Support	Scheduling Algorithms	Results visualisation	Latest release
SimJava	Java	Proprietary	No	No	No	09/2002
GridSim	Java	GPL	Yes	No	No	11/2010
MicroGrid	C	Proprietary	No	No	No	12/2004
Bricks	?	?	?	?	?	Never
SimGrid	C/Java	LGPL	No	No	External tool	05/2012
Simbatch	C	LGPL	Yes	Few	External tool	05/2007
MONARC	Java	Proprietary	No	Priority-based	Basic	08/2007
GSSIM	Java	?	Yes	Few	Basic	08/2011
Alea	Java	GPL	Yes	Many	Extensive	07/2011
pyss	Python	GPL	Yes	Many	No	Never

Table 3.6: Comparison of simulation software for HPC

3.6 Summary

The state of the art that has been presented in this chapter has a number of deficits if it is contrasted with the requirements presented in section 2.5. HPC as it is performed today offers very limited possibility to offer QoS to clients, which makes providers easily exchangeable. Offerings enhanced with QoS features would be a unique selling point for providers. Job scheduling, even though researched intensively, does not offer out-of-the-box solutions for scheduling QoS-enhanced jobs; HPC providers therefore need to align their scheduling with the concrete QoS they provide in order to adhere to their QoS.

SLAs, which are often mentioned in the same breath as the term QoS, seem like a promising tool for HPC providers. The current status, however, is that SLAs have two major drawbacks: their focus is on a high, contractual level and SLAs are regarded on a usage of one SLA per job. Even though the expression of QoS terms in SLAs is relevant for HPC providers, SLAs in this sense offer little support for concrete implementation of QoS-enhanced job submission and scheduling. Furthermore, per-job SLAs are inconsistent with the way HPC providers contract with their clients.

On a higher level than the actual job submission and scheduling, HPC providers need to implement SLA management as well. Various SLA management architectures have been proposed, some generic, some tailored to specific use cases. All of them share the feature of being overly complex, which makes them not only difficult to set up and manage but also leads to a reluctance in their adoption. But even if a simple SLA management toolkit would exist, providers would be reluctant to adopt this as they have no possibility to simulate the influence of offering SLAs on their infrastructure. To this end, several simulation tools have been examined but none readily provide support for simulation of SLAs in an HPC environment.

Chapter 4

Service level agreements for HPC job submission and scheduling

This chapter proposes a concept of how service level agreements can be used for job submission and scheduling in high performance computing, transcending beyond the state of the art as presented in chapter 3. Furthermore, it introduces an SLA management framework reduced to its bare necessities that realises the concept in practice.

4.1 Service levels for high performance computing providers

This sections proposes the usage of SLAs in an HPC environment and suggests service levels that can be offered by HPC providers. This section focuses on the conceptual solution, technical details will be analysed in subsequent sections.

Chapters 2 and 3 have examined HPC today and investigated related work in the field of SLAs. It has been seen that SLAs do not fit well with HPC: whereas SLAs are assumed to be of dynamic, short-lived, per-job nature, HPC providers contract with clients¹ in a static, long-term fashion.

4.1.1 Long-term SLAs for HPC service levels

Employing SLAs for HPC in a way that is appealing to both providers and clients therefore cannot be done with SLAs as they are treated today. However, SLAs which are long-term can be easily aligned with provider and client needs. Long-term SLA are

¹The term “client” refers to an entity that contracts with an HPC provider; “users” are individuals working for a client and making use of the contracts established between client and provider.

SLAs that are not dynamically contracted per job. Instead, the provider prepares the SLAs in advance and clients can contract each SLA individually. The client can then select an appropriate SLA out of all contracted ones when submitting a job. Long-term SLAs are, in a sense, a continuation of the user agreement in place today. However, each long-term SLA is basically a user agreement in its own right and specifies not only usage fees but also details about the offered service.

As the provider is preparing the SLAs it will offer at will, it can brace itself for implications that different usage mixes of the offered SLAs have on its infrastructure. This is much easier than in the case of dynamically negotiated SLAs, where even a few negotiable parameters can lead to plethora of possible outcomes, whose impact on the infrastructure is hardly predictable. Providers can simulate different usage distributions for the selected service levels and can thus identify issues, fine-tune the parameters and refine the SLAs.

The client gains, in comparison to HPC provider offerings today, the ability to choose the desired service level, where today only best effort service is provided. This leads to the client being able to better predict how his jobs will be treated by the provider, according to the selected service levels. The advantage of using long-term SLAs against dynamically negotiated SLAs for provisioning of service levels is that at all times a client can be sure that he can use the provider's infrastructure. With dynamically negotiated SLAs, the client could arrive at a situation where a negotiation does not lead to an agreement, resulting in the client being unable to run his computations. Even if negotiations were to be successful, the negotiation itself poses an overhead not only on the client but also on the provider which can be obviated through the usage of a small number of statically contracted SLAs.

4.1.2 Service levels for HPC

The previous section explained the foundation of this work: long-term SLAs as contracts between providers and clients. This section presents different suggestions that show how long-term SLAs can be offered in a sensible way by HPC providers.

4.1.2.1 Guaranteed environment

Providing a stable environment is especially important for simulations performed as part of an overall design cycle for a complex product such as a car or aeroplane [4] [75]. A software environment, consisting of an operating system, specific versions of numerical libraries and even application codes, is frozen for a full development cycle in order to ensure reproducible simulation results. The desire for a stable environment, however does not stop at the software. Hardware as well as other procedural constraints such as data

handling, security policies or environmental properties need to be taken into account as well.

To address this requirement, a dedicated computing resource is provided for each SLA. An SLA is agreed upon for a product design cycle period and the computing resource is configured accordingly. Advances in virtualization technologies as well as the possibility to apply different boot images in diskless cluster environments allow a more flexible treatment related to software configurations. Using such technologies a potentially unlimited number of predefined images, or even user-defined images, might be provided. Even then, however, specific hardware needs to be available if this is specified in the SLA.

The increased flexibility would allow to offer customised environments not only to large clients asking for resources for a long time period but also for users looking to meet their peak demands with outsourcing avoiding tedious customisation activities of the environment reducing the entry gap.

4.1.2.2 Exclusive access

For some users, exclusive access to resources is important. There are different characteristics of this requirement:

- Assured usage of special hardware, for example a rendering pipeline [76] or specific nodes featuring accelerators or pre- and postprocessing capabilities [77]
- Repeatable performance of computing, networking and storage resources, for example for application benchmarks
- Privacy reasons

Depending on the kind of required access, the solution can range from exclusive access to a part of node through exclusive access to a defined part of a cluster to exclusive access to a complete cluster. Therefore, offering exclusive access can have wide ranging consequences on a provider's infrastructure. It is likely that SLAs offering some form of exclusive access do not contain other complex requirements, for example priorities.

4.1.2.3 Soft prioritisation

Prioritisation, however, is an important field where HPC providers can distinguish themselves from the current best effort based service provisioning. Soft prioritisation, as it

is called, defines priorities for jobs relative to each other and thus does not guarantee a certain time for execution or completion of a job.

A typical system for prioritised access is the division in three service levels named *Bronze*, *Silver* and *Gold*:

Bronze can be realised as either best effort based, as today, or with minimal QoS.

Silver is a service level for clients who want a minimal level of prioritisation compared to the Bronze service level.

Gold service level jobs are prioritised against all other jobs.

The advantage of soft prioritisation is that it can be realised in a straightforward way by mapping the classes to different queues. Thus, it only needs to be assured that clients can only submit jobs for service levels they have contracted and that jobs go to the correct queue. As schedulers support the setup of different queues and can already handle different priorities for these, they can remain unchanged.

Even if implementation of soft prioritisation is simple in this respect, the provider is faced with the task of deciding, firstly, how many clients may contract each class and, secondly, and how many jobs per class can be submitted, both by a single client and in total. Offering the gold service level cheaply to too many clients will mean that its advantage, the quicker execution of jobs compared to the other service levels, is watered down. The highest service level should therefore be contracted rarely. Additionally, the provider needs to ensure that the waiting time for the lowest service level does not increase without limit as this could lead to the loss of clients. In general, the provider should ensure that the amount of clients having contracts for a service level strongly decreases from bronze to silver to gold. Chapter 6 will examine in detail how different distributions of jobs to service levels influence each service levels average waiting times.

4.1.2.4 Timed access

Extending prioritisation further, a provider could offer SLAs that guarantee resource access at a specific point in time. This allows the realisation of scenarios like application steering or interactive access. The latter is closely related to the use case presented in section 2.4.2, the virtual turbine, where geometry changes in a turbine are performed interactively and can be visualised in real time, for example in a CAVE [20].

Providing timed access has a huge impact on the provider's infrastructure and implementation is not as straightforward as with soft prioritisation. While soft prioritisation can be realised with a mapping of service levels to queues, this hardly works for timed access as the impact on other service levels can be tremendous in a queue based solution:

at the worst case, if jobs do not support checkpointing, they have to be rescheduled and all previous progress is lost; this has to be factored into accounting and billing as well as a client won't pay for computing time that is lost due to another client's job. Schedule-based approaches, in contrast, have the advantage that the reservations can already be factored into a schedule during its creation. In any case, support from the underlying queuing/scheduling system is definitely required.

Additionally, the provider has to ask himself which service levels he will offer exactly. Two important parameters are the smallest amount of time between a job submission and the start of the reservation and the precision of the requested start time. For the first, a provider will require some time to set up its infrastructure when a timed access job is submitted; different service levels can offer different amounts of time between submission and earliest reservation time, with a higher price the shorter the time span. A similar solution can be employed for the precision of the reservation. Depending on the client's requirements, for a job that should start at point A in time, the provider can offer service levels that allow it to move the reservation time $\pm Xh$ (bronze), $\pm Yh$ (silver) or $\pm Zh$ (gold), where $X > Y > Z$. Higher precision is then associated with a higher price.

4.1.2.5 Guaranteed maximum waiting time

A service level that is situated between soft prioritisation and timed access is one that guarantees a job a maximum waiting time before job start. It borrows from soft prioritisation that no specific point in time is defined and from timed access the fact that jobs cannot be postponed beyond a certain deadline. This service level distinguishes itself further from timed access through the fact that with timed access the resource reservation itself is important; the resources need only be available during a certain time, they might not be used constantly. In the case of guaranteed waiting time, a job needs to run to its completion, which is, depending on the job, hard to impossible to predict, therefore the only guarantee, besides a job running to completion, is that of a guaranteed maximum waiting time. If the client would be forced to specify an end time, he would either specify a too short deadline, potentially losing all computational progress, or a too long deadline, resulting in overprovisioning that would need to be paid by the client.

With soft prioritisation, a job could be postponed over and over again as higher-class jobs are queued. It is important that a service level exists which guarantees a client that a job will not wait longer than time span X before being executed. This is closely related to the scenario described in the scientific use case in section 2.4.1 as well as to the evolving field of urgent computing [78] [79].

From the level of complexity, guaranteed waiting time is situated between soft prioritisation and timed access as well. The fact that no reservation is made leads, in comparison, to simplified implementability – a queue based solution is sufficient. Compared to soft prioritisation, the logic to decide which job to schedule when is, however, more com-

plicated. This is especially true if other service levels are also in place and need to be respected at the same time.

4.1.3 Service level provisioning benefits

The previous sections made it clear that there are many possibilities for HPC providers to offer higher-quality services instead of the currently prevalent best effort based services. This will lead to a much higher diversity than today, even though the use of long-term SLAs reduces the number of services to a select few compared to dynamically negotiated SLAs. The service levels mentioned exemplarily above showed the underlying assumption that service levels in an HPC environment are advantageous for both clients and providers. Clients can choose specific service levels as needed, enabling new usage models for high-end computing resources. Providers can differentiate themselves from competitors by offering specific products rather than aiming at cost leadership. Consequently, there is a clear need from the client side and a clear motivation from the provider side to deliver SLA-based service in the HPC domain.

The result is a change of the current model, where clients need to fully adapt to the provided environment, to a model where the provider is offering choices and the client elects the most becoming service level according to his needs. In order for the provider to be able to manage the offered services effectively, the space of services is discrete and limited.

Moving from best effort based services to service level-based service poses different requirements on the provider's infrastructure. Providers can no longer solely aim at high utilisation. Instead, they have to focus on the adherence to the service levels, only then being able to take utilisation as a second factor into account. As best effort based service levels are still part of the providers service offering, the benefit of these (high utilisation) and of special services (high value and price) can be combined. Ignoring the aspect of agreeing on specific service conditions – with long-term SLAs this is much more a question of contract law – the main task of the provider is to manage his resources in a way that agreed SLAs are met. Impact of failures and incidents on the resource level need to be mitigated in order to avoid repercussions on the service levels.

4.2 HPC job submission and scheduling with SLAs

The previous sections explained the concept of long-term SLAs and laid the theoretical foundation, describing motivation for providers to implement QoS support and giving examples of what long-term SLAs could encompass. This section addresses the practical

aspects of QoS support for the submission and scheduling of HPC jobs, assuming that multiple SLAs are provided and have been agreed upon ².

4.2.1 Submission of HPC jobs related to SLAs

In contrast to the dynamic negotiation of SLAs per job to be submitted, the usage of long-term SLAs is much simpler as the negotiation is performed offline and well in advance of any job submission. As the negotiation is decoupled from the actual job submission, the question is how does the client link a job to an SLA and how is this communicated to the provider?

For a provider, it is essential to have a mapping of clients to SLAs in order to keep track of the amount of clients having contracted each service level. HPC providers usually have an authentication and authorisation scheme already in place, thus it makes sense to use this in order to identify clients (authentication) and subsequently check if they are authorised to use an SLA (authorisation). HPC providers offering middleware-based access have a public key infrastructure (PKI) in place and this can be leveraged here³. The PKI uses certificates to identify entities, enable digital signatures, secure communication and more [80].

During each communication the client is identified through a certificate that is validated by the provider. On job submission, he adds a reference to an SLA he has contracted to his job submission message. The provider firstly identifies the client through his certificate and also checks if the client is allowed to submit jobs in general. Then, the provider makes use of an internal storage mapping identities via their certificates to SLA – if the client is allowed to use the SLA he has referenced, the job can be accepted. This easily integrates with an existing infrastructure. A further advantage is that different users of a client organisation can be easily allowed access to all SLAs a client has contracted, for example if the organisation specifier is used in the mapping instead of the distinguished name.

Jobs without a referenced SLA can be mapped to a default SLA. This has the advantage that the provider can configure his system to treat these jobs in a predefined way, not visible to the client. Furthermore, allowing such jobs to be submitted allows the provider to be backwards compatible to clients which have not yet contracted an SLA. This enables a seamless transition to the usage of service levels, keeping the best effort service in place.

²A client that has contracted only one single SLA could be mapped automatically by the provider.

³If no PKI is in place, the way the client will implement SLA support depends on the authentication and authorisation schemes in place.

4.2.2 Scheduling of jobs according to service levels

Following on the job submission as described in the previous section, we assume that all jobs are identified to pertain to a specific SLA, either one specified by the client or a default one to which the provider has mapped it. In contrast to the job submission, where it is trivial to add SLA-related information, the subsequent job scheduling is of much higher complexity.

Section 4.1 has presented a number of service levels potentially offered by HPC providers. It could be seen that the service levels are quite different from each other in respect to what they offer to clients and the requirements they pose on the provider. Thus, no generic solution to the question of how to schedule jobs can be given.

Looking at the service levels presented in section 4.1, soft prioritisation is seemingly easy to realise – jobs only need to be prioritised against each other, which allows a queue based solution where the queue priorities reflect the priorities of the different service levels. Guaranteed weighting time, for example, is much harder to realise, especially if the provider has offered guarantees (“X% of jobs are started in less than Y minutes”).

Even for soft prioritisation, however, a queue based setup might not be satisfactory, both for provider and clients. Thus, a decision process needs to be implemented that, preferably automatically, performs certain corrective actions. As an example, the client – and hence the provider – will not be satisfied if low priority jobs have to wait potentially an indefinite time. Therefore, the provider needs to schedule even low priority jobs in such a way that the low service level stays attractive.

Adding penalties and rewards to the equation makes the scheduling decision even harder as the provider is forced to weigh not only penalties and rewards against each other but just as likely penalties of one job failing against penalties of another job failing, likewise with rewards. The provider’s goals therefore move from the current goal, achieving high utilisation, to the fulfilment of SLAs.

4.3 An SLA management framework for HPC providers

This section integrates the main findings of this chapter and proposes an SLA management framework for HPC providers.

A number of SLA management frameworks have been analysed in section 3.3. An immediate question poses itself: why “yet another SLA management framework”? There are a number of answers to this question.

Firstly, looking at HPC providers today, there is no support for service levels yet, even though service levels have been widely used in other fields which are closely related to HPC. A reason for this can be found in the overly complex SLAMFs that have been proposed, for example in [8] [10] [21]. The reason for this complexity lies in the fact that these solutions are generic in nature. Therefore, it is difficult to reduce an SLAMF to a core of specialised components. Overly high complexity is, however, even found in specialised SLAMFs, for example in [42], which proposes an SLAMF for the HPC domain, which uses 36 different components to compose the SLAMF of, divided into third party, client and provider components (see also section 3.3.3.4). Implementation, setup and interdependent configuration of such a number of different components is a gargantuan task. The same goes for other SLAMFs investigated in section 3.3.

The complexity even of targeted solutions can be explained through the complex SLA lifecycle proposed by the TeleManagement Forum (TMF) [81]. Most SLAMFs are based on this lifecycle, which divides an SLA's life time into six phases: Development, Creation, Provisioning, Execution, Accounting, Assessment and Termination. Even though there are other, slightly simpler lifecycles, for example [82], the TMF's lifecycle seems to be a de facto standard. SLAMFs generally cover the whole lifecycle and are, therefore, highly complex.

The SLAMF proposed here mainly supports the submission and scheduling of SLAs; regarding the TMF lifecycle, this can be mapped to the phases Execution, Accounting and Assessment. The development and creation phase have been mentioned above in section 4.1; with long-term SLAs, these phases can be regarded as antedating the submission and scheduling phases, therefore they do not need to be treated in detail; on job submission, SLAs should already be in place and contracted between service provider and client. Provisioning, however, needs to be taken into account as the providers needs to set up and configure his resources according to the SLAs offered to clients. As SLAs are long-term and only a certain, small number of SLAs will be offered by a provider, the preparation of the infrastructure is only briefly mentioned. It is anyway dependent on the exact nature of the SLAs offered by the provider as well as the existing infrastructure. Likewise, termination is a rarely occurring event for long-term SLAs and thus is not relevant for the core implementation.

Having identified the two main reasons for the complexity of SLAMFs – generic usability and full addressing of a complicated SLA lifecycle –, this thesis proposes a minimal SLAMF for the HPC environment. The high-level architecture is shown in figure 4.1.

As can be seen, the number of components as well as the number of interactions is kept low. This results in a SLAMF that is simple to implement, deploy and configure. The aim of this SLAMF is to arrive at a minimal yet sufficient number of components: “Il semble que la perfection soit atteinte non quand il n’y a plus rien à ajouter, mais quand il n’y a plus rien à retrancher [83].”

The components serve the following tasks:

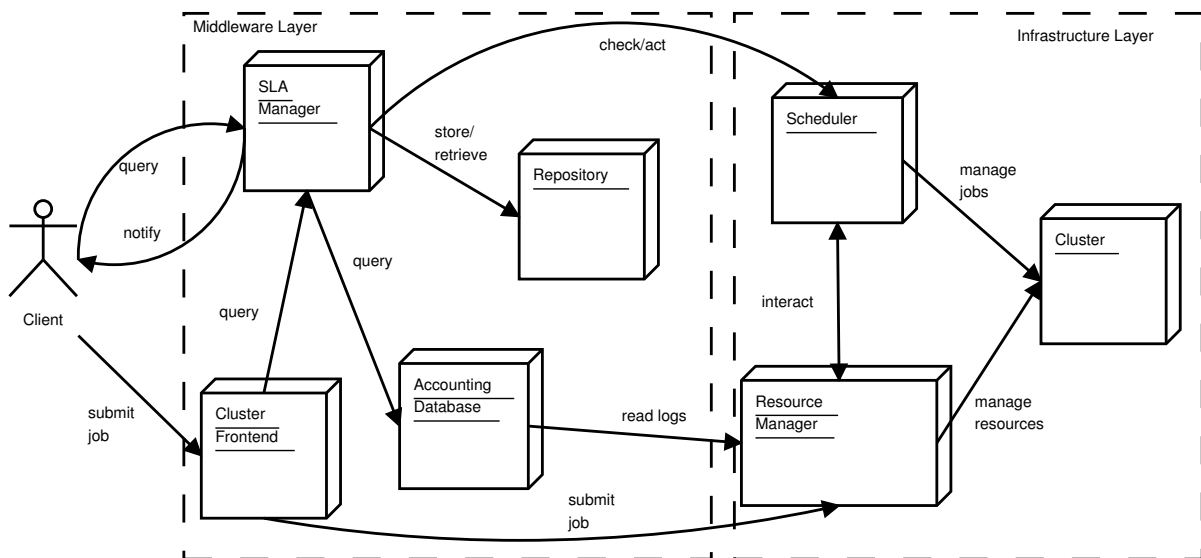


Figure 4.1: High-level SLA management components

SLA Manager The main component for SLA related functionality. It offers an interface that can be queried both externally by clients and internally by other components. During job submission, the SLA Manager acts as a policy decision point (PDP) responsible for deciding if a job submission can be accepted. The SLA Manager can take various parameters into account for this (authentication, authorisation, non-functional parameters).

Repository SLA related information is stored persistently in a special repository. This can be either dynamic information, for example for jobs currently in the system, or static information, for example regarding the long-term SLAs currently in place.

Cluster Frontend The Cluster Frontend acts as a policy enforcement point (PEP) on job submission. Thus, it enforces the decisions taken by the SLA Manager and other PDPs. Accepted jobs are submitted to the Resource Manager. The Cluster Frontend is usually an already existing component that needs to be adapted to the proposed solution.

Accounting Database Accounting information is extracted from the Resource Manager's logs and processed by the Accounting Database. SLA related information is factored in through the SLA Manager. It can be expected that the Accounting Database is an already existing component and only minor adaptations are necessary.

Scheduler The Scheduler distributes and manages jobs. It interacts with the Resource Manager. For the purpose of considering SLA related information, it should as well communicate with the SLA Manager. The Scheduler is an existing component that only needs to be adapted. The actual implementation of the Scheduler needs

to take the service levels to be provided into account and has to be tailor-made.

Resource Manager The Resource Manager is responsible for the management of the cluster. The Resource Manager is an existing component that, depending on its actual realization, might need to be adapted.

Cluster The cluster is the collection of the physical resources. The resources are managed by the Resource Manager while jobs are distributed and managed by the Scheduler. Basically, the physical resources do not need to be modified, but more detailed or specific monitoring details, for example, can make this necessary.

4.4 Evaluation

The previously introduced concept is now evaluated against the requirements collected in section 2.4. To address the requirements in full, they need to be supported by both the concept and the implementation. Consequently, the implementation's suitability to fulfil the requirements will be analysed in chapter 5.

4.4.1 Generic requirements

Requirement 1 is fulfilled through the simplicity of the SLA management framework itself. It can easily be configured and has only little connection points with the existing infrastructure. This makes the integration into the infrastructure straightforward and simple. While no visualisation tools have been explicitly mentioned above, they can be easily added and connected to the framework, for example through the SLA Manager (requirement 2). The SLA Manager as a central service for SLA related information can also be queried by clients in order to get data relevant to their contracts (requirement 3).

Requirements 4, 5, 6, and 7 are addressed more by the approach of how SLAs are used. The provision of multiple service levels (requirement 4) is explicitly foreseen. SLAs can, of course be used to capture non-functional requirements (requirement 5) or guarantees on hardware (requirement 6) and software (requirement 7). The latter two requirements may be addressed differently by the client requesting specific hardware and software at job submission time as well, leaving the SLAs more generic. The concentration on only a few number of service levels which are carefully formulated like the user agreements in place today will lead to clear and comprehensible pricing (requirement 8). The similarity to user agreements ensures that clients can always submit jobs if they have at least one service level contracted (requirement 9) and it is foreseen that users specify a service level at job submission time (requirement 10). Authentication and authorisation are

controlled during job submission and unauthorised use of service levels not contracted by users are prevented (requirement 11). Accounting for usage will in general be already in place in an existing infrastructure, as will be billing. However, tracking job ids and the service level used allows for accounting and billing against service levels (requirement 12).

The remaining requirements 13 and 14 cannot be realised through the design of the framework alone but rather have to be realised through implementation. The framework of course supports the evaluation of SLAs by providing relevant information through the SLA Manager (requirement 13). Estimation of parameters for service levels (requirement 14) is a process that can only happen through simulation and is discussed at length in chapter 6.

4.4.2 Use case specific requirements

The scientific use case has two requirements in addition to the generic requirements mentioned above. Ensuring certain jobs can have prioritised access (requirement 15) has, however, to be realised through the implementation and correct configuration of the existing infrastructure, thus it is an implementation issue, as is the ability to suspend or cancel and reschedule running jobs (requirement 16). The same is true for the use case specific requirement of the virtual turbine use case, timed access (requirement 17), which can at best be supported by the framework by providing relevant information and not inhibiting this behaviour of the infrastructure.

4.5 Summary

This chapter has presented a novel approach to the usage of SLAs in an HPC environment. The core of the concept is the usage of only a small number of SLAs which are offered by the provider as long-term contracts. This is in contrast to related work, where SLAs are negotiated on a per-job basis and the amount of potential SLAs is broodingnagian, leading to unpredictability of the influence of newly negotiated SLAs on a provider's infrastructure. Furthermore, treating SLAs as long-term contracts is in line with the way HPC providers contract with their users today.

Treating SLAs more as contracts and less as specifications of individual job requirements means that it is less important for the provider to employ a purely technical SLA description as provided by several competing specifications (for example WS-Agreement [37], WSLA [38], and HPC WS-Agreement [42]). The provider needs, of course, to translate the service levels offered through the SLA contracts in order to prepare and manage his infrastructure accordingly.

The analysis of related work has resulted in the identification of a gap which obstructs the usage of SLAs in a HPC environment, namely the missing part between the concept of SLAs and the actual implementation. Section 4.3 has addressed this gap by identifying problems with existing SLAMFs and has proposed a minimal SLA management framework. Chapter 5 will prove that this concept can be implemented, deployed and configured easily, thereby closing the gap. The important task left for a provider is, of course, the configuration of his scheduling system and the adaptation to the service levels he has chosen to provide. This is a part that can not be solved in a generic fashion, but, due to the striking simplicity of the SLAMF presented in this work, this is exactly what a provider can focus on.

Chapter 5

Implementation of SLA-enhanced job control

Chapter 4 has presented a concept for the usage of SLAs in a HPC environment and proposed an architecture for an SLA management framework. This chapter demonstrates how this SLA management framework can be implemented and integrated with a typical HPC setup.

It does not need to be emphasised that in the scope of this thesis a full-fledged implementation of the architecture is not possible. The implementation will, however be a proof of the applicability of the previously postulated concept. This chapter begins with the description of a scenario, simplified compared to the scenarios presented in chapter 2, before presenting details on the implementation's architecture and further details on the actual implementation.

5.1 Scenario

The scenario used for validating the implementation is similar to the scientific use case presented in section 2.4.1. Thus, the implementation will be validated against the requirements presented in section 2.5.2. Details of this scenario will be examined through simulation in section 6.3.1.

HPC provider FooHPC offers, apart from the usual best effort service level, named *bronze*, an additional service level with higher priority, named *silver*. This has been introduced as *soft prioritisation* in section 4.1.2.3. Jobs submitted in reference to the silver service level are queued in the front of the job queue, before bronze jobs. Therefore, the price per core hour is higher for silver jobs than for bronze jobs, which is an incentive for users not to use the silver service level if they do not require it. For this scenario, we assume that the service levels are, in detail, as follows:

- Best effort (bronze)
 - No limit on job size
 - Maximum execution time of 24 hours
 - No limit on jobs in the queue at the same time
 - No limit on how many jobs are executed simultaneously
 - No guarantees on waiting time

- Guaranteed waiting time (silver)
 - Maximum job size 10 % of the used cluster
 - Maximum execution time of 4 hours
 - At most 1 job in the queue at the same time per user
 - At most 1 job per user is executed simultaneously
 - Waiting time is guaranteed to be less than 1 hour

Alice is a scientist working for Acme Corporation, which is a client of FooHPC and has contracted both the bronze and the silver service level. Alice as a user commonly makes use of the bronze service level as her computations are, in general, not time critical. However, while preparing a research paper for a conference, she noticed that the simulations she ran give only inconclusive results and that she needs urgently to run more simulations in order to finish her paper in time for the submission deadline. Checking the queue of FooHPC, she notices that many bronze jobs are queued already which makes it questionable if her jobs will be finished in time for her to analyse the data and use the obtained results in her paper. However, no silver jobs are currently queued. Thus, Alice decides to submit her simulation run in relation to the silver service level, bypassing all currently waiting bronze jobs. Alice is, however, limited to having at most two silver job at a time in FooHPC's infrastructure, one running job and one job still queued. A subsequent submission of a job in relation to the silver service level will thus fail if Alice already has one job queued and one job running. Alice's jobs run once enough resources have been freed by completed bronze jobs and she obtains her results in time to use them for her paper.

5.2 Implementation architecture

Section 4.3 has presented a high-level architecture for an SLAMF for an HPC environment. The main components can be implemented independent from an actual infrastructure as services following the service-oriented architecture (SOA) principle [84]. Thus, the general architecture can be implemented as previously described. There needs to be, however, an adaptation to the provider's infrastructure, depending on how job submission and scheduling are currently realised. Furthermore, requirements from the scenario that will be addressed need to be taken into account as well.

The high-level components are based on the Globus Toolkit 4's WSRF core, a platform for the development and deployment of web services. The realisation as web services means that these components can be accessed from different platforms as long as the client can communicate with web services.

For the adaptation to the provider's infrastructure, we assume that the provider employs the Globus Toolkit 4 with TORQUE as a resource manager. This is a quite common infrastructure setup [85] [86] [87].

The following subsections firstly explain how job submission and scheduling work in an infrastructure set up with the Globus Toolkit and TORQUE before detailing the adaptation of the generic part of the SLAMF to this infrastructure.

5.2.1 Job submission and scheduling with the Globus Toolkit and TORQUE

GT4 is a grid middleware that is built in a modular fashion. Individual components include, among others, an FTP-like data transfer service (GridFTP), authorisation services, a Java and a C web service core, a reliable file transfer service and the Grid Resource and Allocation Management (GRAM) job submission service.

Figure 5.1 shows the architecture of GRAM, specifically the web service realisation WS-GRAM. There are two web services that make up the heart of WS-GRAM, *ManagedJobFactory* and *ManagedJob*. The *ManagedJobFactory* services exposes an interface for the creation of jobs for a local scheduler. Submitted jobs are exposed as resources of the *ManagedJob* service. This service can be queried in order to monitor the status of a job, terminate a job etc.

A simple job submission scenario using WS-GRAM follows the sequence shown in figure 5.2: the client invokes the *ManagedJobFactory*'s *createManagedJob* operation, which results in a *ManagedJob* resource being created and its endpoint reference (EPR) being returned to the client. GRAM schedules the job with the local scheduler and notifies

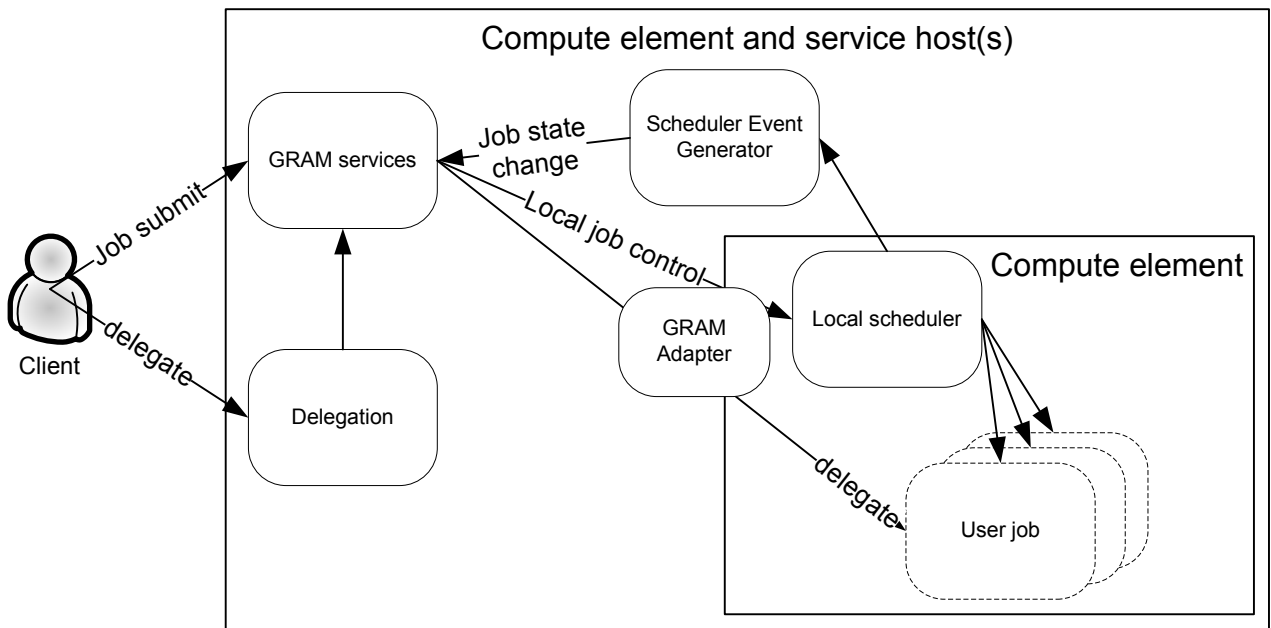


Figure 5.1: WS-GRAM components [88]

the client that the job is pending. On subsequent events - scheduling of the job, job start, job end etc. - GRAM notifies the client. Explicit termination of resources is not necessary.

Access to a GT4 container or, more specifically, the GRAM services, are usually secured via X.509 certificates. Clients present certificates to the container and are allowed to invoke operations if they can be positively identified.

5.2.2 Adding SLA-based job submission and scheduling

The architectural requirements for realising SLA-based job submission are quite straightforward if we adhere to a service-oriented architecture (SOA) approach. Section 2.5 stated a number of requirements which we can translate quite easily to services and operations.

The high-level architecture is shown in figure 4.1. The main component is the *SLA Manager*, which is responsible for the management of SLA-related data; it uses a *Repository* where the complete SLAs and relevant information are stored. Furthermore, the *SLA Manager* acts as a policy decision point (PDP): when a job is submitted by a client to the *ManagedJobFactory*, the factory relays SLA-related data to the *SLA Manager*. The *SLA Manager* analyses the data and takes a decision that is communicated back to the factory. Multiple PDPs can be activated at the same and their decision can result in various actions to be enforced (specifically for the Globus Toolkit, the algorithm “Al-

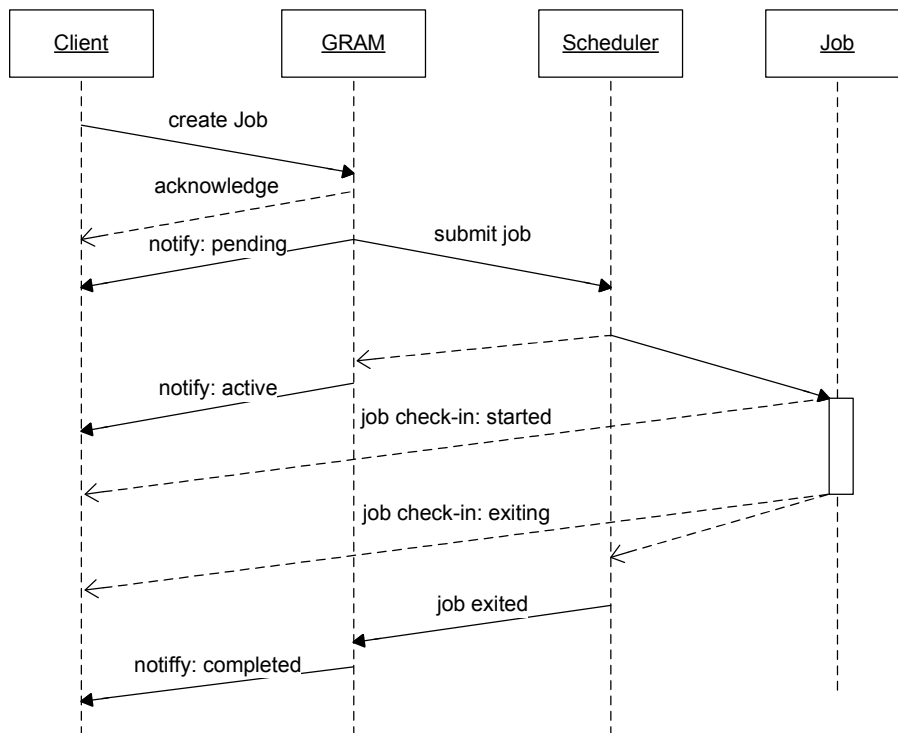


Figure 5.2: WS-GRAM protocol sequence [88]

lowOverride”, for example, results in one allow received from multiple PDPs to override one or more denies, while “DenyOverride” works the other way round). The *SLA Manager* can take arbitrary decisions in order to arrive at a conclusion if a job submission should be accepted or not, for example:

1. Is the submitter allowed to use this SLA?
2. Is the SLA still valid?
3. Is a job submission related to this SLA allowed at this point in time?
4. Are the parameters specified for the job submission in line with the referenced SLA?

All the points mentioned can be addressed in the following way:

1. The job is submitted with the user identifying himself towards the provider. This is done to prevent fraudulent use of SLAs and to ensure correct accounting.
2. The SLA has an expiry time element in the context element and it is checked if the expiry time is still not reached and maybe even if current time plus wall time are before the expiry time [37, p. 8]. Expiry could as well have been reached if

the SLA has been breached a certain number of times, either by the provider, the client or even both.

3. The corresponding parameters specified in the SLA need to be checked against the current system status; this could include the total number of running jobs for a client, the total number of running jobs for each service level per client, the amount of jobs scheduled during a certain time frame, etc.
4. It is checked if the referenced SLA allows the amount of resources requested by the job.

If a job is accepted, the *ManagedJobFactory* creates a *ManagedJob* web service resource and subsequently submits the job description to the scheduling system. Depending on the actual realisation of the scheduling, the *Scheduler* might communicate with the *SLA Manager* in order to obtain relevant data from an SLA associated with a specific job. The *SLA Manager* is the central point regarding communication of SLA related information to the user. Information about which SLAs are in place, for example, can thus be queried through the *SLA Manager* which can obtain information from other internal components on the provider side.

5.3 Prototypical implementation

Section 5.2 has described the overall architecture of the Globus Toolkit, the middleware used to submit jobs, which is in place today at HPC provider sites, and has presented a solution to integrate SLA-based job scheduling into this setup. This section now will describe a prototypical implementation of the solution presented in section 5.2, describing in more detail the components that have been implemented, their communication, between themselves and already installed components, issues that have arisen during the implementation and their solution.

In order to cover all relevant steps, we analyse a whole run-through of a basic use case - SLA-based job submission and scheduling - without explicitly addressing the use case specific requirements up front, as it will later be analysed in how far they are addressed. Therefore, we

- start with a user submitting a job in relation to an SLA;
- show how the provider receives the request with the corresponding SLA information and validates it;
- explain security measures on the provider's side for SLA-based access;

- describe what happens on the provider side once a job is accepted and
- what happens during the job run time and after it has finished.

5.3.1 Submitting a job related to an SLA

For the submission of a job related to an SLA, a client needs to be able to have an overview of SLAs which are in place for an HPC provider. The client can either implement its own SLA management stack or can use services provided by the provider. For the sake of simplicity, we assume that SLA-related information is not stored on the client side but is queried on demand from the provider's *SLA Manager* component.

The client is provided with a GUI tool that can be used to submit jobs in reference to contracted SLAs as shown in figure 5.3. For each of the providers listed in the first text box, SLAs that can be used will be listed in the text box below. The client can then select a prepared job description file that shall be submitted for the given SLA and can finally submit the job.

In the screenshot shown the client is connected to a single HPC provider, FooHPC. It is sufficient for the client to, for example, configure the application with the URLs of the *SLA Manager* component of additional providers that shall be used. SLA information can then be queried dynamically from each providers *SLA Manager*.

Figure 5.3 shows two SLAs in place, namely “Silver” and “Bronze”, which the client application has queried from the provider's *SLA Manager* service. The user can select any one of these when submitting a job.

The addition of SLA-related information to the job submission is performed by adding a reference - in our case, the unique id of the SLA - to the *MessageContext* when communicating with the *ManagedJobFactoryService*, as shown in Listing 5.1. This is certainly not the only way to communicate the SLA id between client and service provider, but it is a simple and straightforward solution. Both parties of course need to use the same way of conferring and expecting the SLA id in a job submission message.

ReferenceParameters are an optional element of an Endpoint Reference and can easily be extracted from the SOAP header on the recipient side [89].

The specification of the job description to be submitted to the provider is taken from a file which the user can select and whose contents viewed or edited in the submission client as well, as shown in figure 5.3. Globus uses Resource Specification Language (RSL), a self-developed XML schema, to allow users to specify complex jobs [25]. An example that runs the executable */bin/sleep* with the argument *60* and specifies custom files for standard output and standard error is given in Listing 5.2.

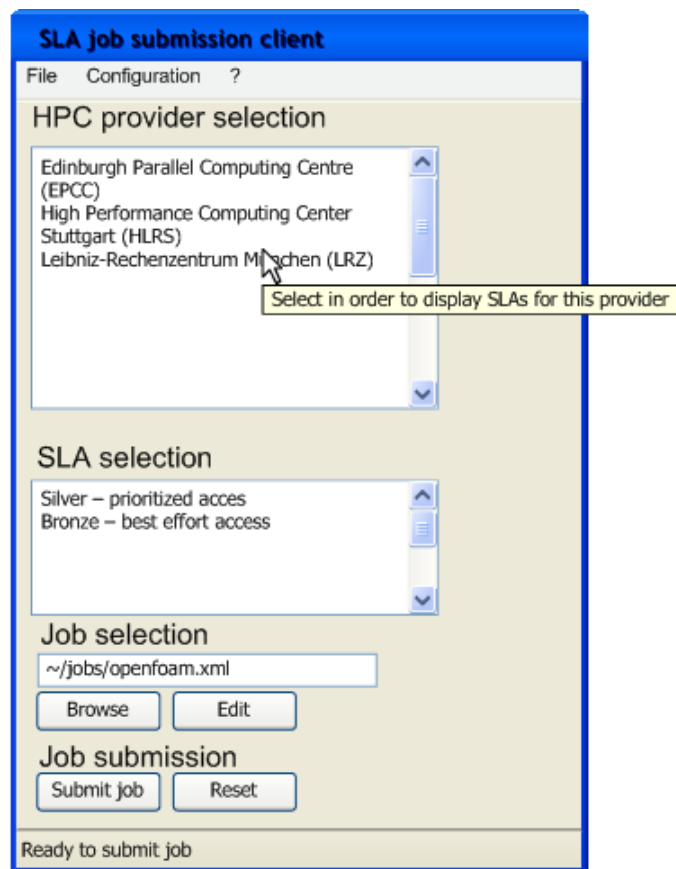


Figure 5.3: SLA selection and job submission using a GUI client

```
1 private EndpointReferenceType getFactoryEpr(String contact,
2     String factoryType) throws Exception {
3     URL factoryUrl = ManagedJobFactoryClientHelper.getServiceURL(contact)
4         .getURL();
5
6     EndpointReferenceType epr = ManagedJobFactoryClientHelper.
7         getFactoryEndpoint(factoryUrl,
8             factoryType);
9
10    // SLA id start
11    ReferenceParametersType parameters = epr.getParameters();
12    MessageElement slaId = new MessageElement("http://www.hlrs.de/
13        namespaces/services/SLA", "slaId");
14    slaId.setValue("6C4E20EA7F000101B05FCB518079B0");
15    parameters.add(slaId);
16    // SLA id end
17
18    return epr;
19 }
```

Listing 5.1: Adding an SLA id to the factory's message context

```
1 <job>
2   <executable>/bin/sleep</executable>
3   <argument>60</argument>
4   <stdout>${GLOBUS_USER_HOME}/stdout</stdout>
5   <stderr>${GLOBUS_USER_HOME}/stderr</stderr>
6 </job>
```

Listing 5.2: A simple sleep job in RSL

If the user has selected a service level and a job description file, the corresponding job will be submitted to the provider once the user activates the *Submit job* button shown in the bottom left corner in figure 5.3.

5.3.2 Receiving SLA information on the provider side

As described in the previous section, the SLA id specified by the client is contained in the SOAP header. This reference is extracted on the provider side and the following checks are performed:

- The certificate identifying the user is checked against one stored with the SLA in order to confirm if the user is allowed to use this certificate; without this check, any user who would somehow obtain an SLA id could use it to run jobs against this service level.
- The SLA is checked for timely validity, that is if the run-time of the SLA has expired or not.
- A client is not allowed to have more than one job running and one job queued at the same time for the silver service level.
- The silver service level allows a maximum wall time of 4 hours, the bronze service level a maximum of 24 hours. If *maxWallTime* property in the job description exceeds this time, the job is not allowed to run.

Both the selection of the parameters that are checked and the values of these parameters are arbitrary and illustrate the possibility of how easily checks of a job submission's properties can be validated against an SLA.

5.3.3 Securing the ManagedJobFactory service

Calls to the *ManagedJobFactory* service need to pass the SLA PDP¹, additional to any other PDPs that are in place as well. GT4 uses so-called “security descriptors” for configuring different security properties, for example credentials, authentication and authorisation mechanisms etc. [90] Therefore, the SLA PDP has to be added to the security descriptor for the *ManagedJobFactory* as shown in Listing 5.3. ²

¹Previously, it has been stated that the *SLA Manager* is the PDP; this is correct in the general notion of the term PDP; what is, in the notion of the word, a policy enforcement point (PEP) is termed PDP in GT4.

²The default location is `$GLOBUS_LOCATION/etc/globus_wsrp_gram/managed-job-factory-security-config.xml`.

```

1 <serviceSecurityConfig>
2 <methodAuthentication>
3   <method name="createManagedJob">
4     <auth-method>
5       <GSISecureConversation/>
6       <GSISecureMessage/>
7       <GSISecureTransport/>
8     </auth-method>
9   </method>
10 </methodAuthentication>
11 <authzChain combiningAlg="DenyOverride">
12   <pdps>
13     <interceptor
14       name="slapdp:de.hlrs.gt4.pdp.sla.Gt4SlaPdp" />
15     <interceptor
16       name="gridmap"/>
17   </pdps>
18 </authzChain>
19 <reject-limited-proxy value="true"/>
20 </serviceSecurityConfig>

```

Listing 5.3: Security descriptor for the *ManagedJobFactory*

PDPs are configured in an “authorisation chain” and are evaluated in turn to finally arrive at a permit or deny decision. If the PDPs are combined with deny override, as is shown in Listing 5.3, all PDP have to arrive at a permit decision in order for the complete chain to authorise a request. The SLA PDP is, as first PDP, evaluating if the request specified a correct SLA id which is valid for the user presenting it and checking that the referenced SLA allows for a submission of a job at the current time. A user may, for example, be only allowed to have a certain number of jobs in the system and further job submission would therefore be denied. If the SLA PDP denies the request, no further evaluation is necessary. However, if the SLA PDP permits the request, the Gridmap PDP, a default Globus PDP, is invoked, to see if the user calling the service can be mapped to a local user. Only if this PDP arrives at a permit, the job can finally be queued.

The PDP extends the interface *org.globus.wsrp.security.authorization.PDP* and has one important operation, *isPermitted*. Listing 5.4 shows how the SLA PDP is implemented in a high-level pseudo code.

The caller is passed to this operation by GT4, as well as the message context and the name of the operation called. The SLA id, which has been embedded by the user in the message context, is extracted and the SLA id and caller id are passed on to the SLA Manager in order for it to assess if the SLA id is valid and if a job can be submitted

```
1 public boolean isPermitted(Subject peerSubject, MessageContext context,
2     QName operation) throws AuthorizationException {
3
4     boolean isPermitted = true;
5
6     String caller = SecurityManager.getManager(
7         (SOAPMessageContext) context).getCaller();
8
9     String operationName = operation.getLocalPart();
10
11     // Only execute authorisation check for the createManagedJob method
12     if (operationName.equals("createManagedJob")) {
13         String slaId = getSlaIdFromMessageContext();
14         /*
15          * Actual checks are implemented in the method
16          * jobSubmissionPermitted
17          */
18         isPermitted = jobSubmissionPermitted(slaId, caller);
19     }
20
21     return isPermitted;
22 }
```

Listing 5.4: Realization of the SLA PDP

at the current time for the specified contract. The PDP then relays the SLA Manager's decision back to the caller.

5.3.4 Queuing jobs in reference to an SLA

The connection between the Globus GRAM web services (*ManagedJobFactory*), to which the users submits its job, and the underlying scheduler is realised with a "Scheduler Interface" programmed in the Perl programming language [91].

Globus provides some interfaces for the schedulers Condor, LSF and PBS and a basic fork scheduler which is mainly for testing purposes; only the fork and PBS scheduler interfaces are installed by default ³.

Every scheduler interface has to be provided as a Perl module that subclasses the *Globus::GRAM::JobManager* module which provides an implementation of the base behaviour of a job manager; the scheduler interface provides the connection tailored to a specific scheduler. The interface's name needs to match the scheduler type string, but in lower case; for our case of the PBS scheduler, this results in *Globus::GRAM::JobManager::pbs*. The parent module, *Globus::GRAM::JobManager*, provides several methods of which only the *submit* and *cancel* methods need to be implemented. The PBS script, which has been adapted to this purpose, provides four different methods:

cancel Cancels a job with a given job id by calling PBS' *qdel* command.

myceil Used for rounding when computing the number of nodes to use in a cluster.

poll Polls the status of a job with a given id by calling PBS' *qstat* command.

submit Constructs a job description file out of the parameters given to the script and submits it by calling PBS' *qsub* command.

The scheduler script is passed a job description that is contained in the *ManagedJob* resource. In the implementation at hand, the SLA id is mapped to the name of the queue to which the job is to be submitted; to which queue a job is submitted could of course be determined in other ways. The job description is finally submitted via the *qsub* command to PBS. In this implementation, the service levels are *Silver* and *Bronze* and the queue setup, obtained by running the command *qstat -Q -f*, is shown in Listing 5.5. If necessary, jobs for the Bronze queue are cancelled and re-queued.

For example, a request reaching the scheduler interface where the SLA id references the silver service level would be submitted to the queue named *silver* with *qsub -q silver*

³The modules can be found in *\$GLOBUS_LOCATION/lib/perl/Globus/GRAM/JobManager/*

```
1 Queue: silver
2   queue_type = Execution
3   Priority = 75
4   total_jobs = 0
5   state_count = Transit:0 Queued:0 Held:0 Waiting:0 Running:0 Exiting:0
6   mtime = 1225212274
7   enabled = True
8   started = True
9
10 Queue: bronze
11  queue_type = Execution
12  Priority = 50
13  total_jobs = 2
14  state_count = Transit:0 Queued:0 Held:0 Waiting:0 Running:0 Exiting:0
15  mtime = 1225212230
16  enabled = True
17  started = True
```

Listing 5.5: Queue status obtained by `qstat -Q -f`

.... The mapping of service levels to queues can be implemented at different levels; for the sake of simplicity it has been implemented as a 1:1 mapping of service level to queue in the *ManagedJobFactory*. The scheduler interface could as well only be passed the SLA id and could itself query the *SLA Manager* for the corresponding service level; as the job submission parameters are already checked against the SLA in the higher level middleware layer, it suggests itself to perform the mapping directly at this stage as well.

5.3.5 Running the job

The queues shown in the previous section are set up with different priorities and jobs from higher-priority queues run before lower-priority queues do. It can be seen that even with the simple queue setup a basic support for service levels is possible. More complex cases, maybe even with the implementation of a custom scheduler, can of course support different service levels in a more fine-grained way but require much more implementation. It will be shown in chapter 6 how different parameters might be taken into account and how that influences job scheduling; taking this data and transferring it to a practical setup is therefore only a question of implementation.

5.3.6 Accounting and billing

Even though the focus of this implementation is on the submission and scheduling of jobs, it would not be complete without giving at least pointers on how to implement accounting and billing. This is due to the fact that accounting is not as simple anymore as different jobs might have to be accounted differently. Accounting allows an organisation not only to charge users but also analyses users and their usage.

For accounting purposes, a provider can either use a self-developed solution or rely on software developed by others, like *pbsacct* or *Grid-SAFE* [92] [93] [94]. Regardless of the solution that is employed and the practical adaptation thereof, a simple way of accounting jobs with varying SLAs is to just save job ids and related SLAs when the job is actually scheduled. This way, the accounting software just needs to be made aware of this mapping file and can then account jobs in the right way.

5.4 Evaluation

Even though the implementation presented in this chapter is tailored at a specific scenario, it should nonetheless fulfil the generic requirements (see section 2.5.1) as well as the scenario specific ones (see section 2.5.2).

5.4.1 Generic requirements

The implementation has shown that integration of the management framework into an existing infrastructure is simple and straightforward (requirement 1). Visualisation tools or query tools for clients have only been developed on a very basic level for this prototype, but they can be easily implemented using the SLA Manager as interaction point, which then fulfills requirements 2 and 3.

Because the provider is offering multiple SLAs which also contain non-functional parameters, requirements 4 and 5 are fulfilled. This scenario assumes that no specific hardware or software is required, thus requirements 6 and 7 are ignored. Pricing is not discussed in this scenario as well, leading to requirement 8 being ignored as well. Information on how to implement accounting and billing that take in service levels has been given but not implemented (requirement 12).

The scenario assumes that clients have at least contracted one service level. Indeed, this is necessary to be considered a client of the HPC provider and leads to the client being able to always submit jobs (requirement 9). Users select the service level for a job during submission time (requirement 10) and the provider authenticates and authorises the user

during this time in order to ensure legitimate use of SLAs (requirement 11). The provider ensures that jobs are not waiting longer than promised in the SLA (requirement 13). As mentioned in section 4.4.1, simulation is discussed at length in chapter 6.

5.4.2 Scenario specific requirements

There are two scenario specific requirements (see section 2.5.2). Requirement 15 demands prioritised resource access, which is available through the corresponding service level. Basic support for termination and rescheduling of already running jobs has been implemented (requirement 16).

5.5 Summary

This chapter has shown how high-level SLA management can be integrated into a common grid middleware, the Globus Toolkit, and, subsequently, into the scheduler. This allows the realisation of an integrated solution for the offering, usage and accounting of service level agreements in a grid environment. The scope of this work only allows a prototypical implementation of the, in total, quite complex solution. However, most of the requirements are already addressed by this implementation. For requirements that were not fulfilled, pointers have been given as to how these requirements can be addressed.

In terms of the scheduling of service levels, we have presented a queue-based approach. This decision was due to the proof of concept nature of this work. One can, however, only go so far with queue-based approach; more complex operations, for example advance reservation, are not or not easily possible with queue-based approaches but require a schedule-based approach. Realising more complex scheduling components is a complex yet interesting work and needs to be done with respect to the desired service levels and scheduling features.

Chapter 6

Simulating SLA usage in an HPC environment

In the previous chapters a new concept for using service level agreements for job control in a high performance computing environment has been presented. It is a necessary precondition for a provider to evaluate such a concept before realising it in production. Apart from the general evaluation of the concept, each provider might have different parameters he wants the system to take into account. This chapter therefore presents a solution to simulate the concept without actually influencing a production system.

Section 3.5 concluded with the fact that there are no specific simulation tools for the simulation of SLA usage, neither in general nor in a HPC environment. The usage of SLAs in this work is focused on job submission and scheduling and therefore tools for the simulation of job scheduling were investigated. In principle, a lot of tools can be used to simulate the usage of SLAs for job control.

Based on the analysis of different tools performed in section 3.5 the Alea scheduling simulator has been chosen (see section 3.5.9 as well) [73]. This chapter provides further details on Alea beyond section 3.5 before discussing how SLA-based scheduling can be implemented in Alea. Different experiments that have been performed using the adapted implementation of Alea are presented before the implementation and the simulation results are summarised.

6.1 The Alea scheduling simulator

Alea is a grid scheduling simulation tool that is based on the GridSim toolkit [72]. It extended GridSim in order to enable the simulation of different grid scheduling algorithms. Alea has been initially published in February 2007 as version 1.0; the work performed in this thesis is based on version 2.1, released in October 2009.

6.1.1 Features

Alea is built in a modular fashion. Its main high-level components are loaders that are able to read in different data files: *JobLoaders* for different workload trace formats, *MachineLoaders* for hardware resources and *FailureLoaders* for machine failures (see figure 6.1).

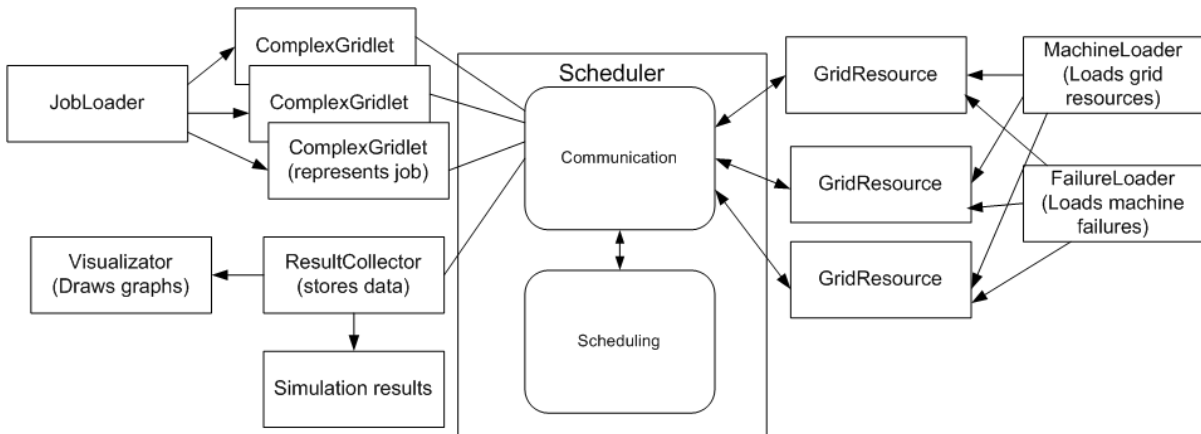


Figure 6.1: Architecture of Alea

Jobs are read in and stored as *ComplexGridlets*, a class that contains all job characteristics while hardware resources are read in and stored as *GridResources*. The main *Scheduler* consists of a component that is responsible for communication with both *ComplexGridlets* and *GridResources* and another component for scheduling of jobs. Additionally, the *Communicator* also collects results, which are stored in a *ResultCollector* in order to save or visualise them.

Alea supports a number of schedule-based and queue-based scheduling approaches – for example First Come, First Served, Earliest Deadline First and Earliest Suitable Gap – and can visualise different system properties, among others average system utilisation per day, number of waiting and running jobs per day, number of requested, available and used CPUs etc. Generated graphs can be exported in BMP, GIF, JPEG or PNG format and data generated during a simulation run is saved in text files with comma-separated values (CSV) [95].

Alea supports the usage of different workload traces; workload traces are logs that have been created on real machines by real users. Alea supports the Grid Workload Format (GWF), MetaCentrum Workload Format (MWF), Pisa Workload Format (PWF) and Standard Workload Format (SWF).

Summing up, Alea has a good support for different scheduling policies and algorithms, can read basically every existing workload trace format and features basic visualisation support.

6.1.2 Shortcomings

Alea looks like a promising tool to use for the simulation of SLAs for scheduling of jobs as it already provides support of quite a number of scheduling algorithms and policies. In contrast to many other tools Alea provides the features that are most closely related to the proposed SLA-based scheduling (see also the analysis of the state of the art in simulation of SLAs in section 3.5).

Alea can, however, not be used out of the box to simulate SLA-based scheduling: there is no extension mechanism foreseen in Alea to specify additional scheduling algorithms, let alone the possibility to add algorithms without changing Alea's own source code. Furthermore, the binary distribution of Alea does not even allow a user to specify the selection of an algorithm or a workload trace; instead, the only main class that can be called by users is iterating through all implemented algorithms and all existing workload traces¹.

In addition, workload traces do not support the specification of service levels; this is, however, not a shortcoming of Alea itself. Nonetheless, an extension simulating service levels needs to take this into account and needs to define how service levels can be specified for existing workload traces.

Apart from the shortcomings in the binary version as mentioned, the source code of Alea is hard to comprehend, let alone extend. Most values are hard-coded as magic numbers, as is the behaviour of the main class. The following section details with the extension of Alea in order to implement SLA-based scheduling and will specify in some detail the code-level shortcomings.

6.2 SLA-based scheduling with Alea

Alea does not support service level agreements or service levels nor does it provide an extension mechanism to implement support for these features. Even if Alea would provide support for SLAs in one way or another, another problem surfaces: workload traces that are available through various sources do not provide service level data. This is quite logical, if one considers that HPC providers do not offer service levels. Just extending Alea with the capability to simulate scheduling based on SLAs therefore does not help: workload traces need to be extended with SLA data. This section describes solutions to both problems by describing how distributions of service levels can be generated for existing workload traces and by describing how Alea has been extended to be able to

¹That is actually only the intention of the implementation; there is a bug in the loops controlling algorithm and data set selection so that each data set is read but only two of the eight algorithms are simulated

take account of these data.

6.2.1 Service level distributions for workloads

Workload traces do not provide any SLA-related data in the sense of service classes or similar data. This fact needs to be addressed before service levels can be included in Alea’s simulation capabilities.

The fact that HPC providers as of yet do not implement service levels makes this a twofold problem: firstly, service provider do not use service levels, so their workload traces do not even foresee the specification of service levels. Secondly, how can distributions be generated for existing, real-life workloads if this not done in the real world?

The first problem, that workload formats do not foresee the specification of service level-related parameters can be tackled in multiple ways. The specification of the Standard Workload Format makes a compelling argument to not mix in service level-related parameters into the specification [96]:

“The format is completely defined, with no scope for user extensibility. Thus you are guaranteed to be able to parse any file that adheres to the standard, and multiple competing and incompatible extensions are avoided. If experience shows that important attributes have been left out, they will be included in the future by creating an updated version of the standard.”

Adhering to this, it was decided not to extend the specification with additional, domain-specific and custom data fields. Instead, a separate mapping file specifying the service class for each job, identified by its unique id, has been created; listing 6.1 shows the first 13 lines of such a file.

```
1 1 SILVER
2 2 BRONZE
3 3 BRONZE
4 4 BRONZE
5 5 BRONZE
6 6 BRONZE
7 7 BRONZE
8 8 BRONZE
9 9 BRONZE
10 10 BRONZE
11 11 BRONZE
12 12 SILVER
13 13 BRONZE
```


14 . . .

Listing 6.1: Example of an SLA id to job id mapping file

Using a separate data file allows to keep the specification in its original format and it fits well with Alea’s architecture, as Alea uses different loader classes to load different data related to a simulation. It therefore comes naturally to implement an *SlaLoader* class that loads SLA-related data for jobs. Figure 6.2 shows the *SlaLoader* integrated into Alea’s architecture. The loader class is only used if an SLA algorithm is used for scheduling in order not to unnecessarily influence other simulation runs. If an SLA algorithm is used, the *JobLoader* class, which is responsible for reading workload traces, queries the *SlaLoader* and sets the corresponding service level information to an instance of class *ComplexGridlet*, which represents the corresponding job.

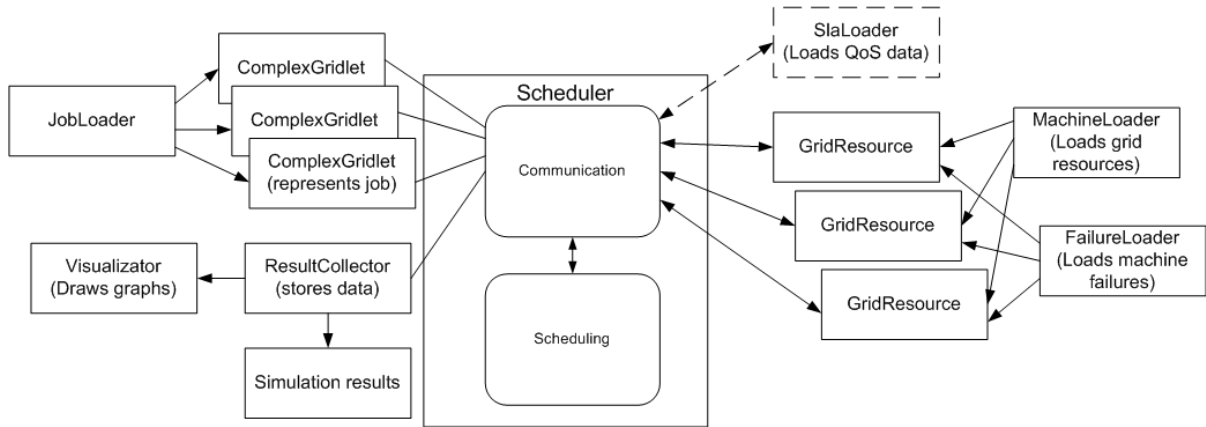


Figure 6.2: The Alea architecture with the added SLA loader

Having solved the problem of how to specify service level distributions, we now can tackle the problem of how service level distributions can be generated for existing, real-life workloads.

The advantage of using a real workload trace is that, even though the service level distribution is generated artificially, the workload consists of real jobs that have been submitted by real users. By generating different distributions of service levels that users can potentially specify, a provider can see how the implementation of service levels may influence his real work load.

There are two aspects to the generation of SLA distributions: firstly, there is the question of which SLAs to offer (see chapter 4). Secondly, once it has been decided which SLAs to offer, a distribution for the selected service levels needs to be performed. Assuming a number of n service levels $S = s_1, \dots, s_n$ has been chosen, and a work load J consists of m jobs, $m = |J| = |j_1, \dots, j_m|$, the distribution is a surjective function $f : J \rightarrow S$, that results in each job being mapped to a service levels and multiple jobs being potentially mapped to the same service level.

An important goal of a provider is to not violate contracts. This does not mean that non-violation of contracts can be achieved at all times, but if this can not be achieved, then the impact of violations should be minimal. After all, clients should profit from a provider offering complex service levels and should not experience “worse” conditions than with the previous best effort based offering. Of paramount interest is therefore the investigation of how different usage of service levels can be handled by an infrastructure.

Algorithm 1 Pseudocode for generating service level distributions

```
Input values: percentage of jobs to be prioritised ( $p$ ), minimum prioritised job size ( $m$ ), maximum prioritised job size ( $n$ )
Read all jobs from a workload trace into set  $J$ 
for all  $j \in J$  do
    Determine size of job  $j$  as  $i$ 
    if  $i \geq m$  and  $i \leq n$  then
        store job  $j$  in list of jobs  $X$  that are eligible to be prioritised
    else
        store job  $j$  in list of jobs  $Y$  that are not eligible to be prioritised
    end if
end for
Determine amount  $P$  of jobs to prioritise:  $P = p * (|X| + |Y|)$ 
if  $|X| < P$  then
    Not enough jobs are eligible to be prioritised with the given parameters
else
    Randomly draw  $|P|$  jobs from  $X$  into set  $Z$ 
    return job distribution that results from the sets  $X$ ,  $Y$  and  $Z$ 
end if
```

Algorithm 1 describes how service level distributions were generated. This algorithm allows to constrain the selection of jobs that can potentially be prioritised by specifying an allowed minimum or maximum size. By setting the parameters accordingly, the following options are possible:

1. Allow any job to be prioritised – if the minimum size is set to 1 and the maximum size to the total amount of available processors.
2. Allow only an exact job size to be prioritised – if the minimum and the maximum size are set to the same value.
3. Allow a range of job sizes to be prioritised – if the minimum is set to a value over 1 and below the maximum, which is set below the total amount of available processors.

Each of these options will subsequently be used to generate service level distributions for the simulations performed in 6.3. The third option, however, can be considered the

most suitable for generating service level distributions in touch with reality as it results in one being able to prioritise jobs from specific “job classes”, similar to current limits on express queues, for example.

6.2.2 Implementing scheduling with SLAs

In order to simulate SLA-based job scheduling, Alea has been modified and extended. This section describes the performed modifications and extensions.

In order to simplify the addition of other scheduling algorithms - and, specifically, SLA-based scheduling - firstly, cleanup and refactoring has been performed on the source code. As quite many schedule-related aspects were hard-coded with magic numbers, Java enumerations have been introduced to list all available algorithms and policies. Even though this still requires changing the source code when adding new algorithms, it facilitates the process and is much simpler than before. Furthermore, a new main class, reading arguments from the command line, has been created, thus allowing for script-based execution of various scheduling runs with different data; originally, scheduling runs were hard coded.

The SLA scheduling algorithm itself has been implemented in the *Scheduler* class. The current preliminary implementation is queue-based and sorts newly incoming jobs into the queue according to their priority. The decision, which job’s priority is higher, is taken in the class *ServiceLevelComparator*; this is in line with implementation of the other queue-based algorithms, which use *Comparators* to compare two jobs and return which job has the higher priority.

In a first implementation, two service levels have been implemented, as has been described above, *bronze* and *silver*. This is a high-level implementation in the sense that each job can only have one of the two service levels or, if the service levels are directly mapped, priorities. It is questionable if job deadlines, which are contained in some workload traces, are realistic (Klusáček et al. use in [97], for example, job deadlines, but acknowledge that these are in reality seldom reliable), but they can be of course be used as secondary scheduling parameters. In general, users are accustomed to specifying run time estimates. These might not be reliable as well, but if the specification of more precise requirements is an incentive for users, they might increase in reliability. Incentives can easily be provided by scheduling shorter jobs earlier, which is then pretty similar to the scheduling according to deadlines as shown in [97]; jobs that transcend the specified run time for a certain quota might be terminated.

6.3 Experimental results

The modified and extended Alea has been used to simulate a different number of service levels, with varying distributions and using different workload traces. The settings of these simulations and their results are discussed in this section.

In an ideal case, providers have workload traces of their infrastructure at their disposal. Introducing service levels will most probably influence the future workload, but an original workload trace matching the infrastructure gives indications about different users, job sizes, job length, etc. If no workload trace exists, as in the case at hand, one can turn to either historic logs of other infrastructures or to synthetic workloads. Synthetic workloads have the advantage of being configurable while historic workloads are not. Historic workloads, however, have at least been run in real life, so a point can be made for the usage of both. The Parallel Workloads Archive² offers both historical workloads³ and models that can be used to generate synthetic workloads⁴. Of the 30 historical workloads that are provided by the Parallel Workloads Archive, 5 are not suitable for our needs as the logs are from sets of clusters, not from single clusters, thus 25 logs remain. We have chosen to use the log of the San Diego Supercomputer Center's Blue Horizon log⁵. The Blue Horizon is a 144-node IBM SP, with 8 processors per node, providing 1152 processors in total. The original trace spans about two years and contains 250,000 jobs. For this initial investigation, a subset of about 12,000 jobs which runs 930,457 core hours has been investigated in order to first of all ensure the feasibility of the implementation.

Figure 6.3 shows a distribution of jobs with respect to their job sizes. The x axis shows the number of processors requested for jobs and the y axis the percentage of jobs with this size relative to all jobs. Each tick on the axis x_i stands for jobs that require more processors than the number at x_{i-1} and a number of processors up to and including x_i . For example, circa four percent of all jobs request a number of processors between 129 and 256. For the first bin, specified by 8 processors, this is the exact number as there are no jobs requesting less than 8 processors.

Several observations can be made from figure 6.3. Firstly, the majority of jobs is quite small with around 41 % of all jobs requesting only 8 processors. Secondly, with the exception of jobs requesting between 9 and 16 processors, whose amount is only third in classes, the classes' frequency is monotonically decreasing. Thirdly, even though the class including jobs requesting between 513 and 1152 processors is the most "wide", covering the greatest amount of potential jobs, it accounts for only about 0.53 % of all jobs. Due to the small amount of jobs requesting more than 128 processors, the simulations used prioritised jobs of up to 128 processors and, if limits on the maximum size of prioritised jobs were used, the limits were investigated for the classes displayed in figure 6.3.

²<http://www.cs.huji.ac.il/labs/parallel/workload/>

³<http://www.cs.huji.ac.il/labs/parallel/workload/logs.html>

⁴<http://www.cs.huji.ac.il/labs/parallel/workload/models.html>

⁵http://www.cs.huji.ac.il/labs/parallel/workload/l_sdsc_blue/index.html

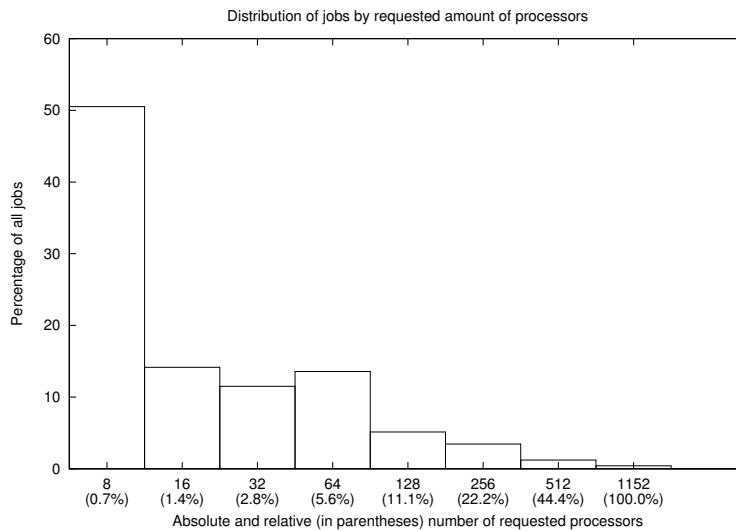


Figure 6.3: Histogram showing the percentage of jobs for different job sizes

6.3.1 Prioritised and best effort access

This first experiment investigated the influence of offering a simple service level with prioritised access in addition to the best effort based access that is provided by default; in the following, these service levels are called silver (prioritised) and bronze (best effort). Prioritisation in this case meant the preferential queuing and execution of silver jobs against bronze jobs. However, no running jobs were cancelled. The job queue was sorted “First Come, First Served” (FCFS) in each service level, silver jobs were always scheduled before bronze jobs. The SLA distribution varied from no silver jobs in order to determine the average wait time with no prioritisation and from 1 % up to 4 % silver jobs in 1 % increments, the rest of the jobs then being bronze jobs.

# of service levels	2
Best effort service level	Bronze
Prioritised service level	Silver
Service level distributions	99 % / 1 % to 96 % / 4 % Bronze/Silver
Prioritisation	Soft (no cancellation)
Prioritised job sizes	0.7 %, 1.4 %, 2.8 %, 5.6 % and 11.1 % (8, 16, 32, 64 and 128 processors)
Queue policy	FCFS with Silver always preceding Bronze

Table 6.1: Details for experiment 1

Figure 6.4 shows the increase in average wait time for unprioritised jobs relative to the average wait time for a job in the original workload. If the size of prioritised jobs is not bigger than 2.8 % (32 processors) of the machine, the influence is negligible. Increasing the prioritised job size up to 5.6 % (64 processors) of the machine, works well if 2 %

or less of the jobs are prioritised. If more jobs are prioritised or the allowed size of prioritised jobs increases further, the average waiting time for unprioritised jobs gets higher quickly.

In contrast, figure 6.5 shows the average wait times for silver jobs in seconds. Even without assuring timely execution of silver jobs by cancellation of bronze jobs, the average waiting time is, in all cases, below one hour (3600 s). For only small sizes of prioritised jobs, that is 0.7 %, 1.4 % and 2.8 % of the machine’s capacity, the average waiting time is even less than 10 minutes (600 s), regardless of the number of prioritised jobs. If the silver job size increases, however, the average waiting time gets larger quickly. This is due to the fact that larger silver jobs have to wait longer until enough bronze jobs are finished and enough resources are free.

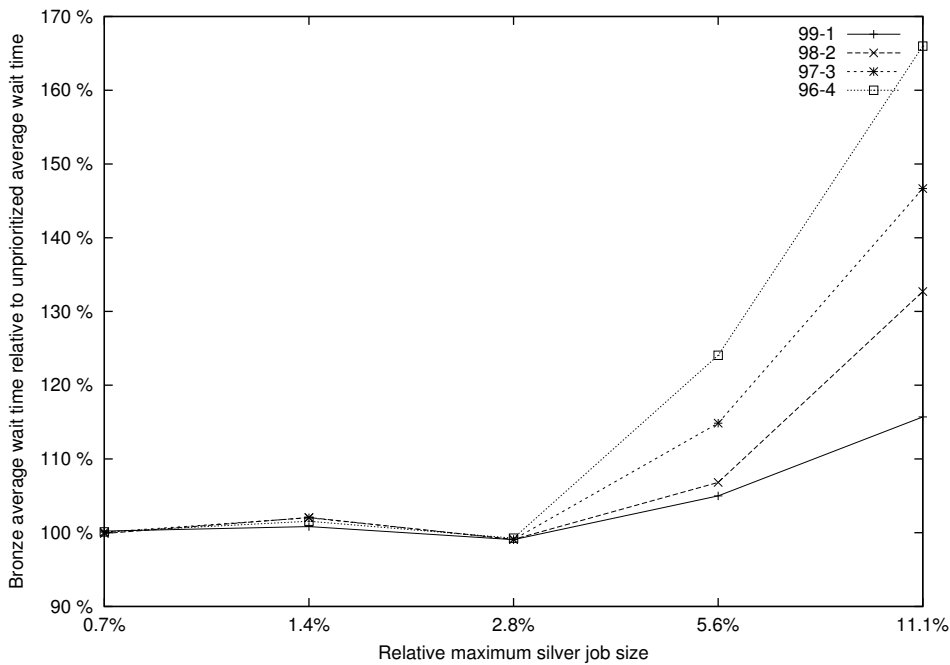


Figure 6.4: Average wait time increase for bronze jobs

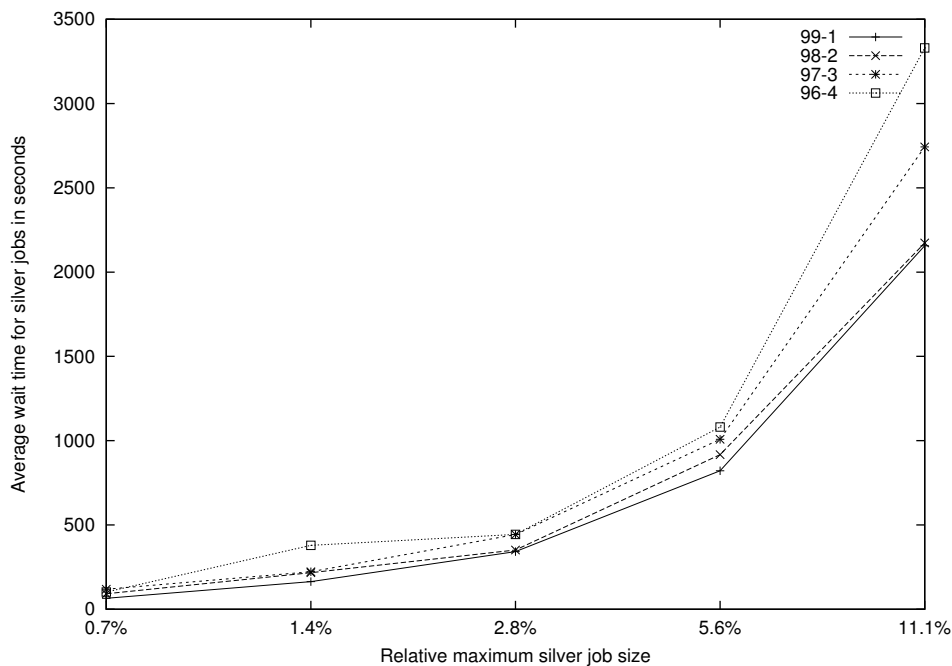


Figure 6.5: Average wait time for silver jobs

6.3.2 Cancellation and resubmission of best effort jobs

The previous experiment showed us that for increasing prioritised job size, just waiting for best effort jobs to run to their end can lead to a large increase in waiting time. In this experiment, we therefore investigate a more “urgent computing style” approach: if a silver job is waiting for execution, as many bronze jobs as necessary to free enough resources for the silver job to run are cancelled and inserted again into the queue.

# of service levels	2
Best effort service level	Bronze
Prioritised service level	Silver
Service level distributions	99%/1% to 96%/4% Bronze/Silver
Prioritisation	Hard (random cancellation of bronze jobs)
Prioritised job sizes	0.7 %, 1.4 %, 2.8 %, 5.6 % and 11.1 % (8, 16, 32, 64 and 128 processors)
Queue policy	FCFS with Silver always preceding Bronze Cancellation of Bronze jobs until first silver job can be run

Table 6.2: Details for experiment 2

It can be seen that for big prioritised job sizes, the wasted core hours due to job cancellation quickly get out of hand, showing the importance of limiting the job size for a scenarios like this. The dashed line parallel to the x axis shows the total amount of core

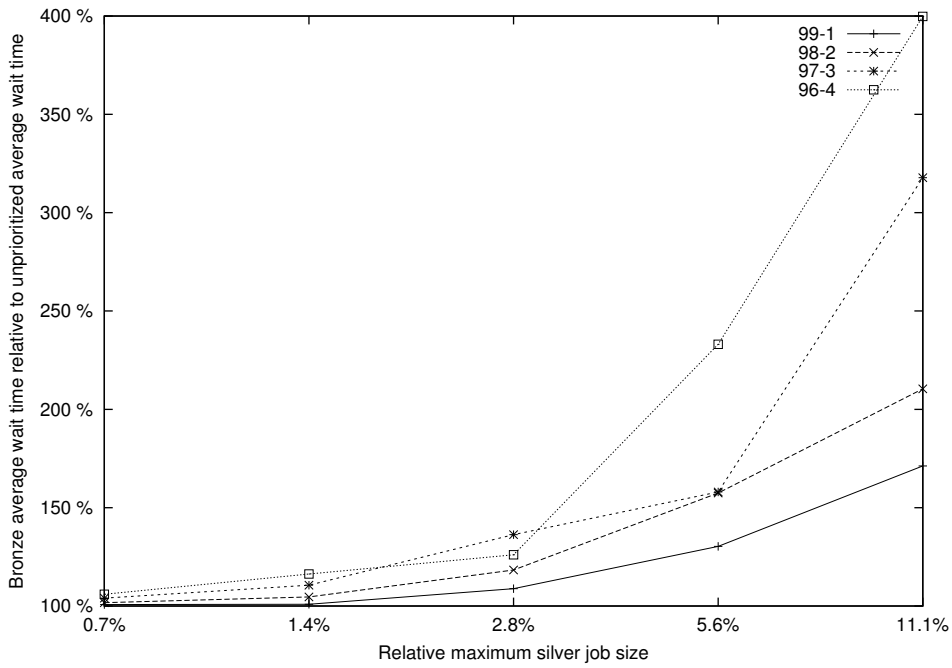


Figure 6.6: Average wait time for bronze jobs

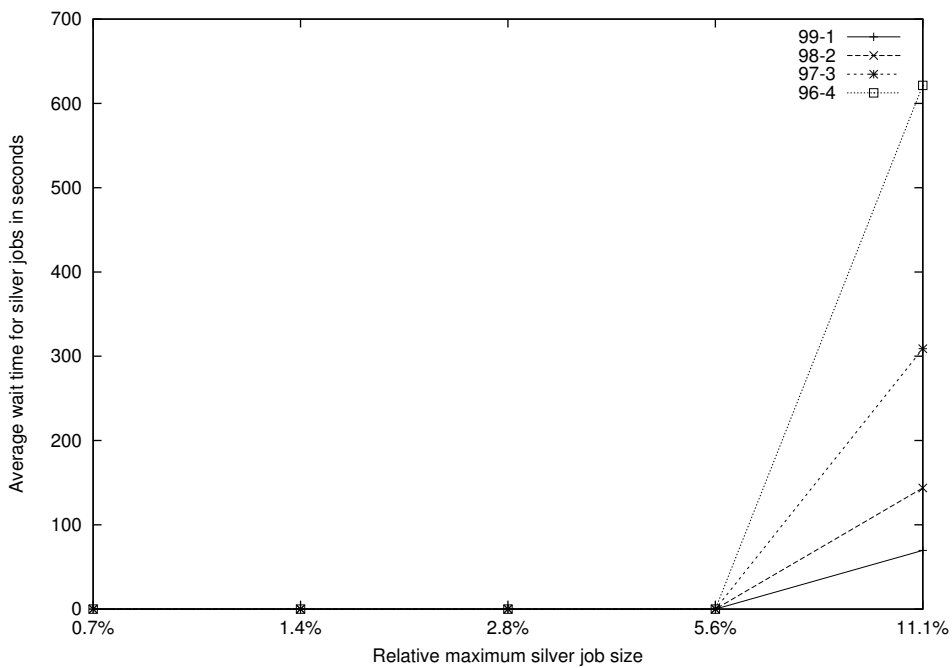


Figure 6.7: Average wait time for silver jobs

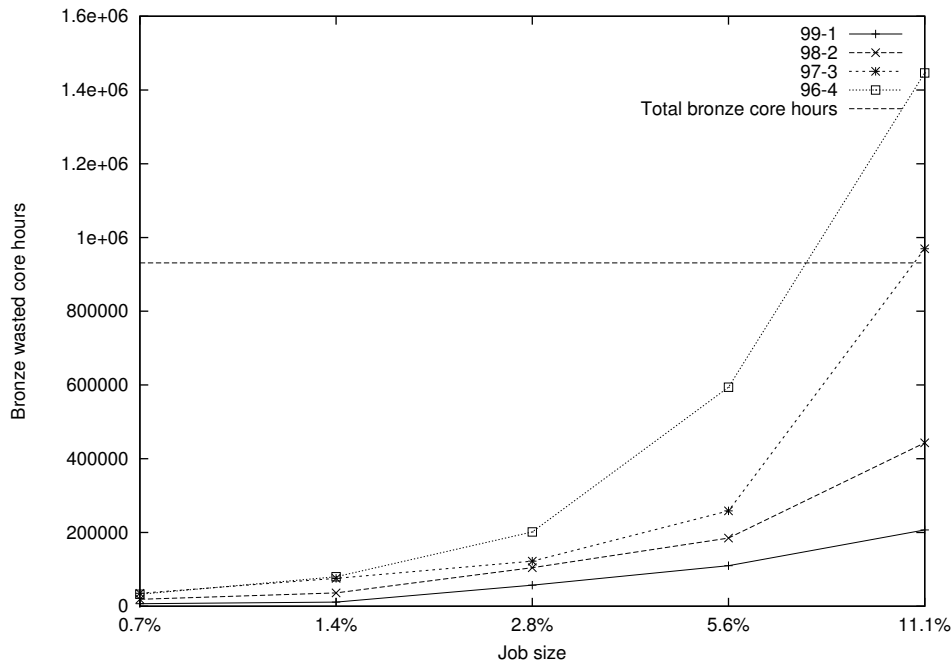


Figure 6.8: Wasted bronze core hours due to cancellations

hours in the complete workload. If the maximum job size of prioritised jobs is not limited enough, already at only 3 % silver jobs more core hours are wasted than the workload contains as a whole.

Figure 6.9 shows the price a silver core hour needs to cost to at least make up for the revenue lost due to cancelled jobs: the core hours that have been computed but cannot be billed due to cancellation are summed up and divided by the total amount of silver core hours. In reality, a more complex calculation would probably be used, but this number gives an indication about the cost that is incurred by the silver jobs. It can be seen that for small prioritised jobs of less than 32 CPUs (2.8 %), the price needs to be quite high because the occurring cancellations have to be apportioned to a very small number of short jobs. The optimum price is achievable when jobs up to 64 CPUs (5.6 %) can be prioritized, for larger job sizes the price rises again.

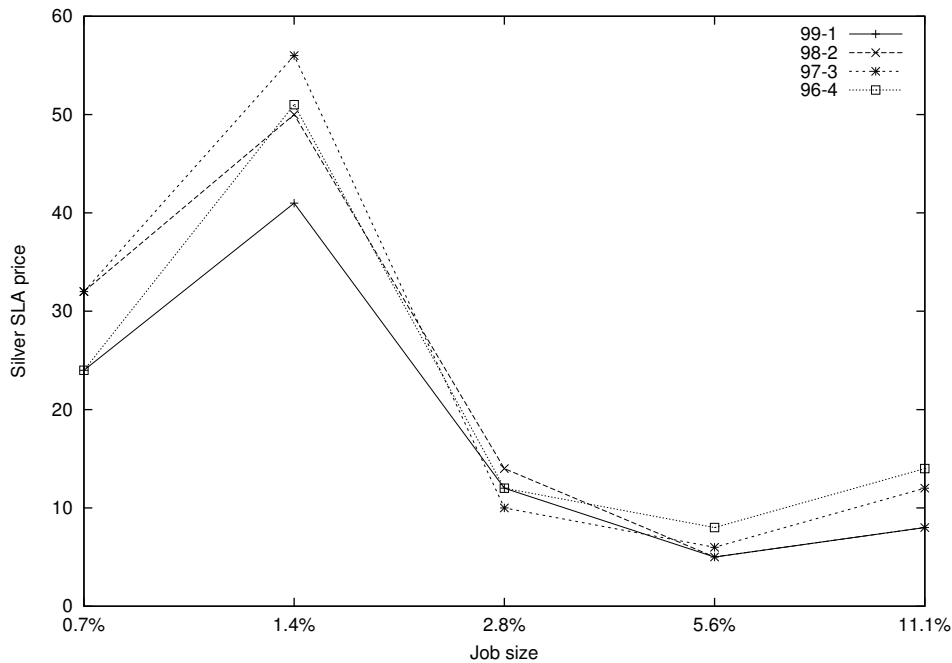


Figure 6.9: Silver SLA price to make up for revenue loss due to cancellations

6.3.3 Influence of different cancellation policies

The previous experiment in section 6.3.2 showed that there are combinations of maximum prioritised job size and allowed percentage of prioritised jobs against the whole workload that are disadvantageous for both providers and clients. Up to a certain boundary, however, introducing prioritised service levels are beneficial for all parties. This section continues the examination by investigating the influence of different cancellation policies on certain distributions and job sizes. The previous experiment used a policy cancelling jobs randomly until enough CPUs were available to start a prioritised job. The following additional cancellation policies are investigated now:

Least percentage already completed Jobs which have the least completion percentage (based on the run time) are cancelled first.

Shortest total job length (time) Cancel jobs which have shorter runtime first.

Shortest total job length (core hours) Cancel jobs which have shorter core hours first.

While the latter two policies are static in that they do not take into account how much of the jobs has already been executed, the former policy does take this into account. Even if all the policies seem quite similar at first glance, they are actually quite different, as can be seen by the jobs that are shown in table 6.3. Each of the policies mentioned above would cancel a different job as first job: least percentage already completed would select

job 1, shortest total job length (time) would cancel job 3 and shortest total job length (core hours) would cancel job 2.

Id	CPUs	Runtime (hours)	Runtime (core hours)	Time ran (hours)	% completed	Core hours completed
1	8	20	160	2	10	16
2	6	8	48	1	12.5	6
3	32	2	64	1	50	32

Table 6.3: Sample jobs showing the influence of different cancellation policies

Following the results performed in section 6.3.2, each cancellation policy was used for simulation runs with priority job sizes limited to 8, 16, 32, 64 and 128 CPUs, respectively. For each of these job sizes, distributions from 99 % bronze and 1 % silver to 96 % bronze and 4 % silver were run.

# of service levels	2
Best effort service level	Bronze
Prioritised service level	Silver
Service level distributions	99%/1% to 96%/4% Bronze/Silver
Prioritisation	Hard (different cancellation policies)
Prioritised job sizes	0.7 %, 1.4 %, 2.8 %, 5.6 % and 11.1 % (8, 16, 32, 64 and 128 processors)
Queue policy	FCFS with Silver always preceding Bronze Cancellation of Bronze jobs until first silver job can be run

Table 6.4: Details for experiment 3

Firstly, we are interested in the effect of the different policies on the amount of wasted core hours due to job cancellation. Figures 6.10, 6.11 and 6.12 thus show the amount of wasted core hours for each distribution and job size for each of the different cancellation policies; for random cancellation, please refer to figure 6.8.

The random cancellation policy performs worst, having the highest amount of wasted core hours. This is expected, as the randomness can result in cancellation of jobs that are nearly finished. From the remaining three policies, the cancellation of the shortest job by core hours performs best. The reason that it performs better than cancellation of the shortest job by time is that it takes both the time and the number of required CPUs into account. Cancellation according to the percentage already finished performs slightly better than according to shortest job by time.

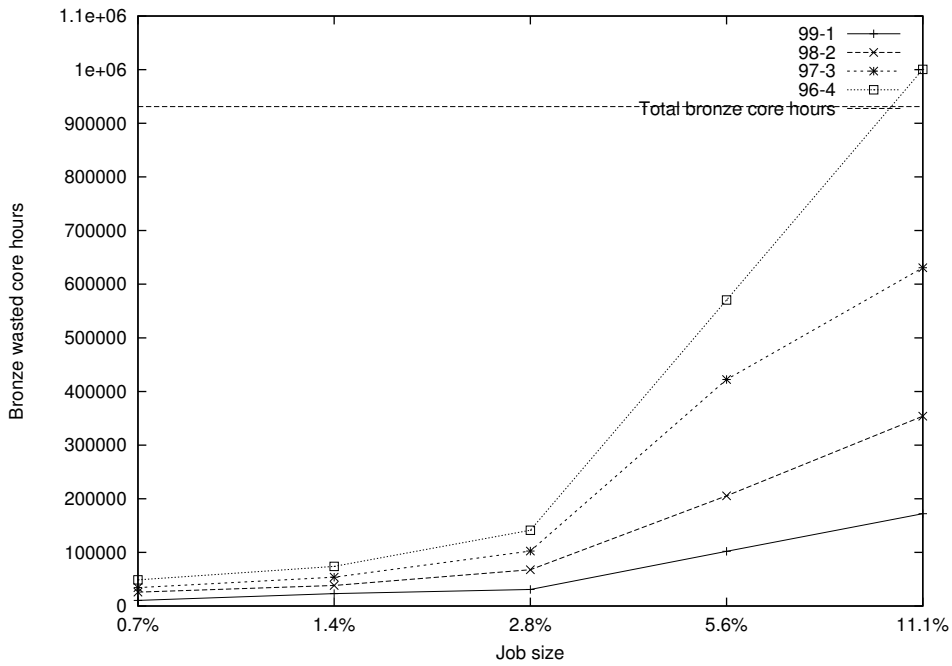


Figure 6.10: Wasted core hours versus job size for least percentage completed cancellation

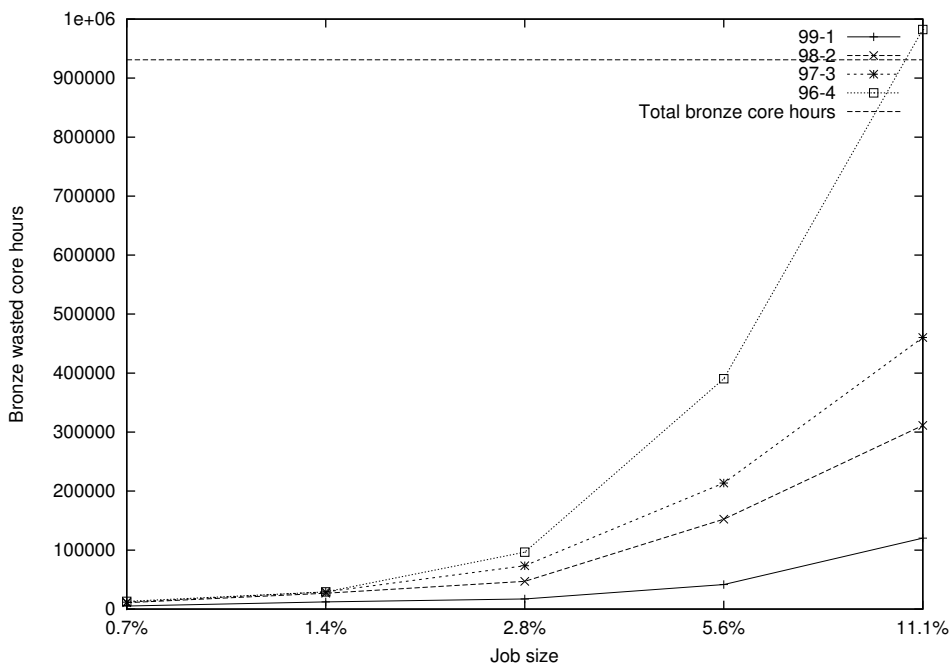


Figure 6.11: Wasted core hours versus job size for shortest job by core hours cancellation

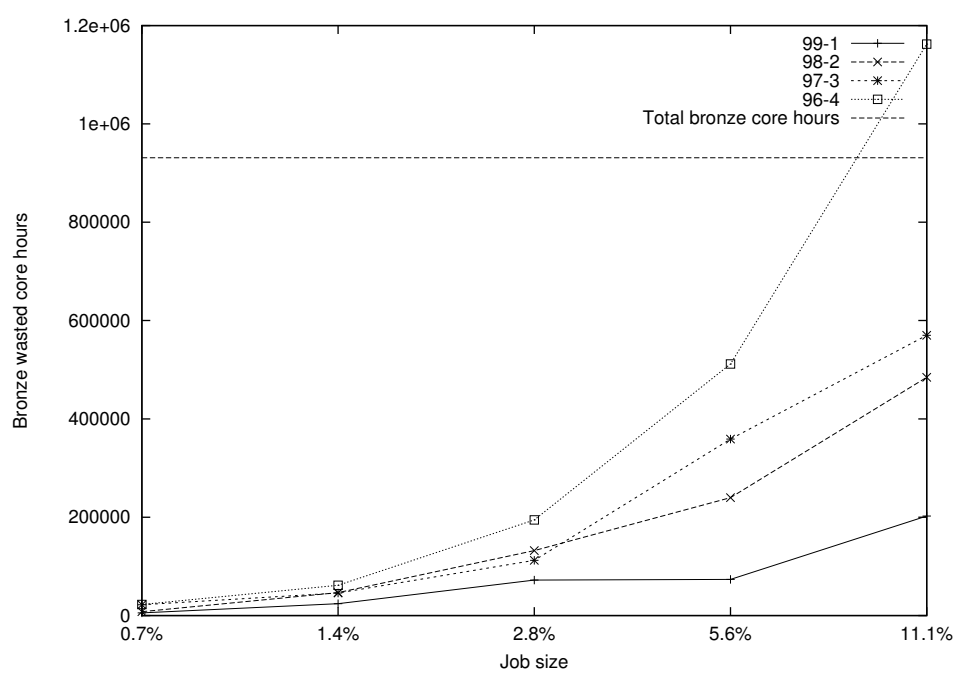


Figure 6.12: Wasted core hours versus job size for shortest job by time cancellation

Section 6.3.2 examined the price a silver core hour needs to cost in order to at least make up for the lost revenue due to job cancellations. This is shown in figures 6.13, 6.14 and 6.15 for each of the cancellation policies investigated in this chapter and in figure 6.9 for the random cancellation policy. In principle, the findings from section 6.3.2 are confirmed: allowing only small jobs up to 32 CPUs (2.8 %) to be prioritised leads to quite high prices for these jobs. For higher job sizes, the required price drops significantly. However, an exception to this is the policy cancelling jobs with less amount of core hours first. The price for prioritised jobs can be quite low, compared to the other cancellation policies, even for only small jobs being prioritised. The graphs show as well that there is an amount of randomness in the price for the silver core hours regardless of maximum silver job size or maximum percentage of silver jobs. This is due to the fact that, apart from the fixed parameters, the actual state of the system when a cancellation has to be performed is still not totally determined. Still more complex cancellation policies taking more parameters into account could yield more predictable results.

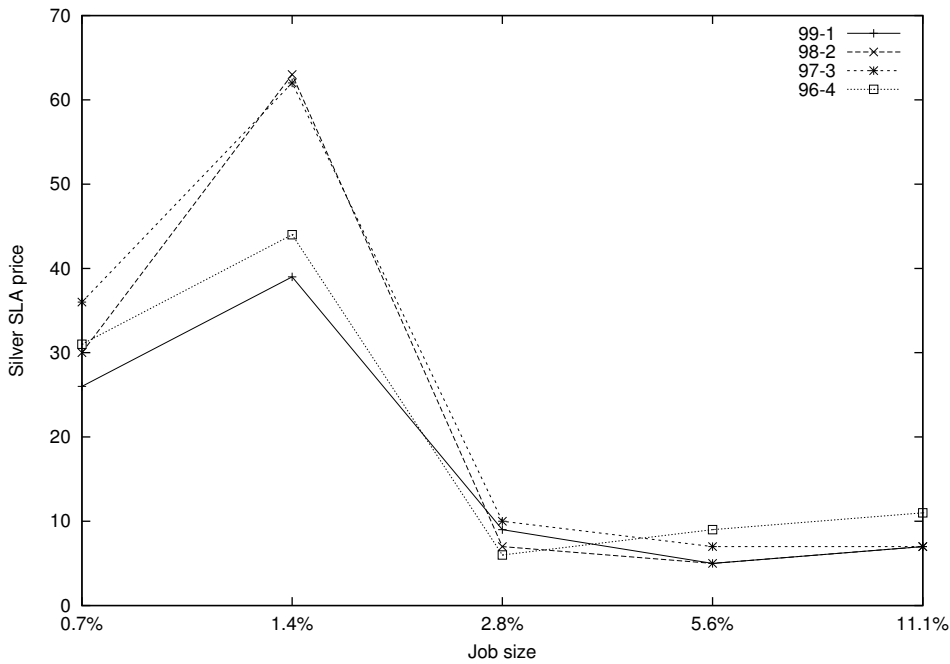


Figure 6.13: Silver SLA price versus job size for least percentage completed cancellation

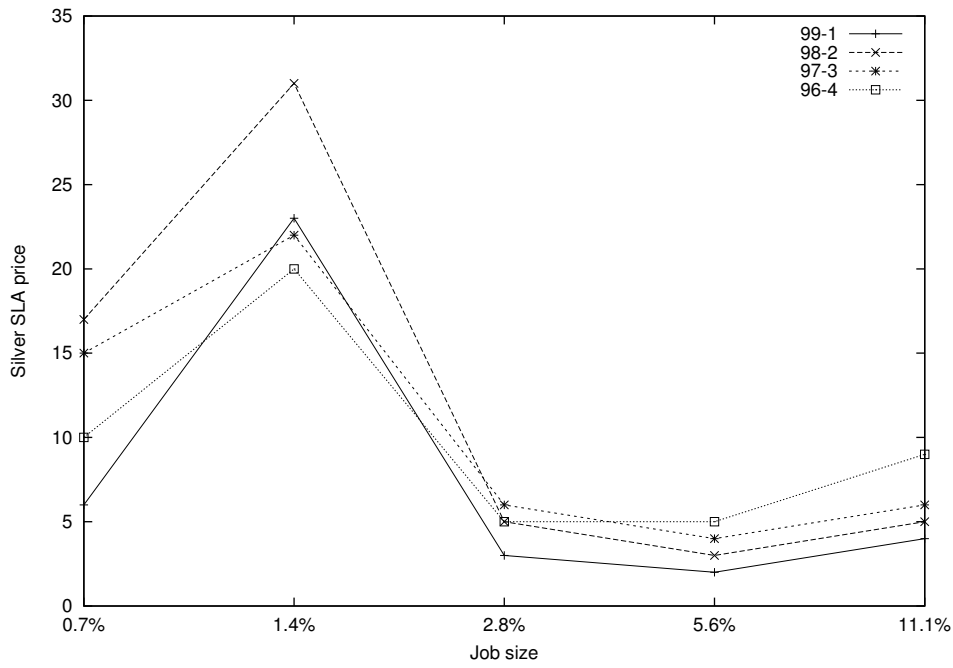


Figure 6.14: Silver SLA price versus job size for shortest job by core hours cancellation

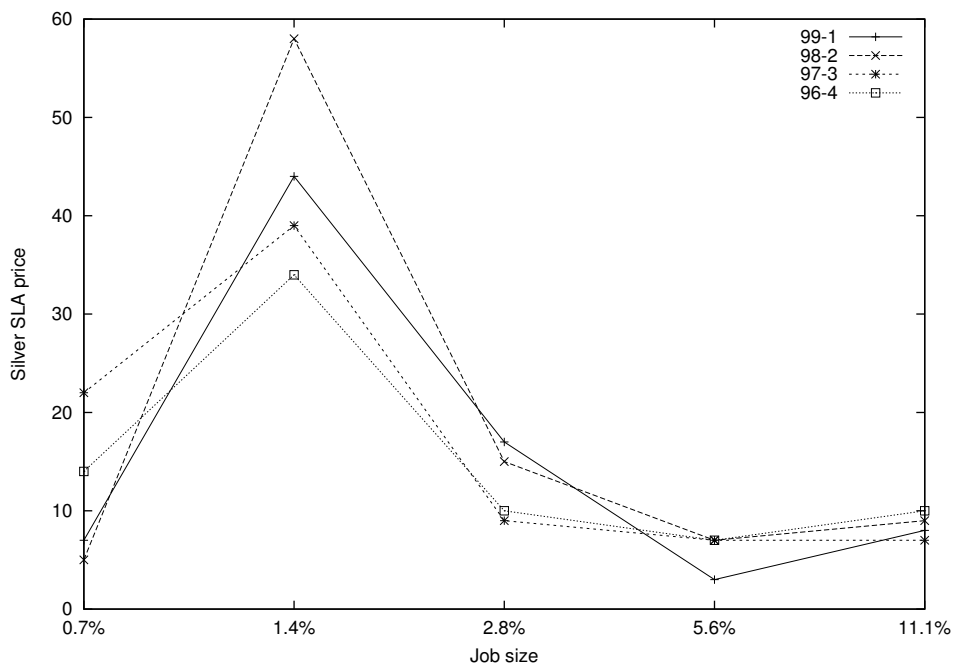


Figure 6.15: Silver SLA price versus job size for shortest job by time cancellation

Figures 6.6, 6.16, 6.17 and 6.18 show the average waiting time for bronze jobs. The random cancellation policy again performs worst, doubling the waiting time from jobs up to 32 CPUs (2.8 %) already at jobs with 128 CPUs (11.1 %) for higher percentages of prioritised jobs while the other distributions do not reach the double waiting time at all. In general, as the previous results have indicated, the cancellation policy taking core hours into account performs best again. Taking only the runtime into account, either dynamically (least percentage completed) or statically (shortest job by time) gives comparable results.

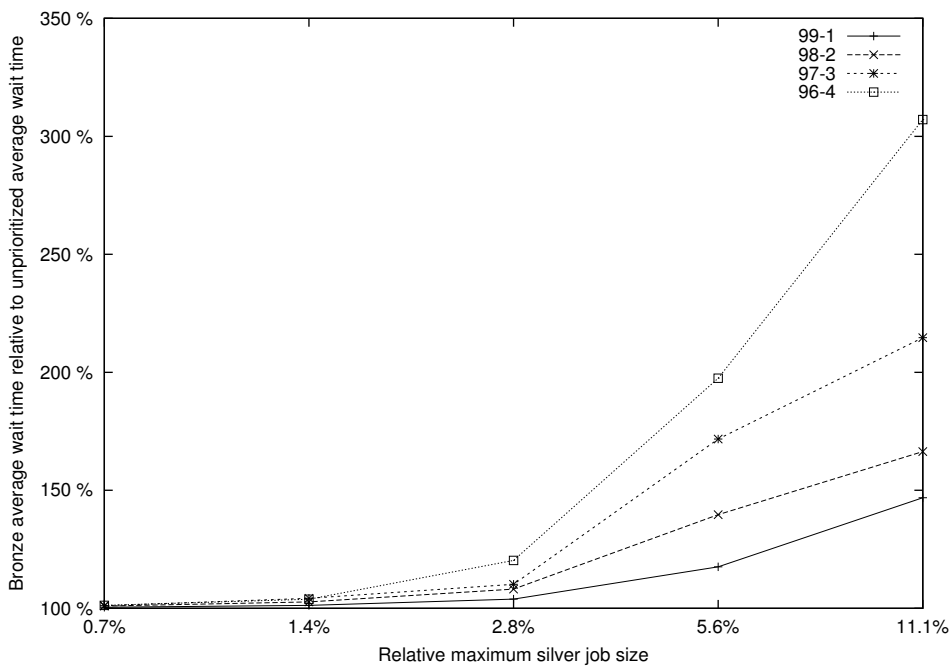


Figure 6.16: Bronze average wait time versus job size for least percentage completed cancellation

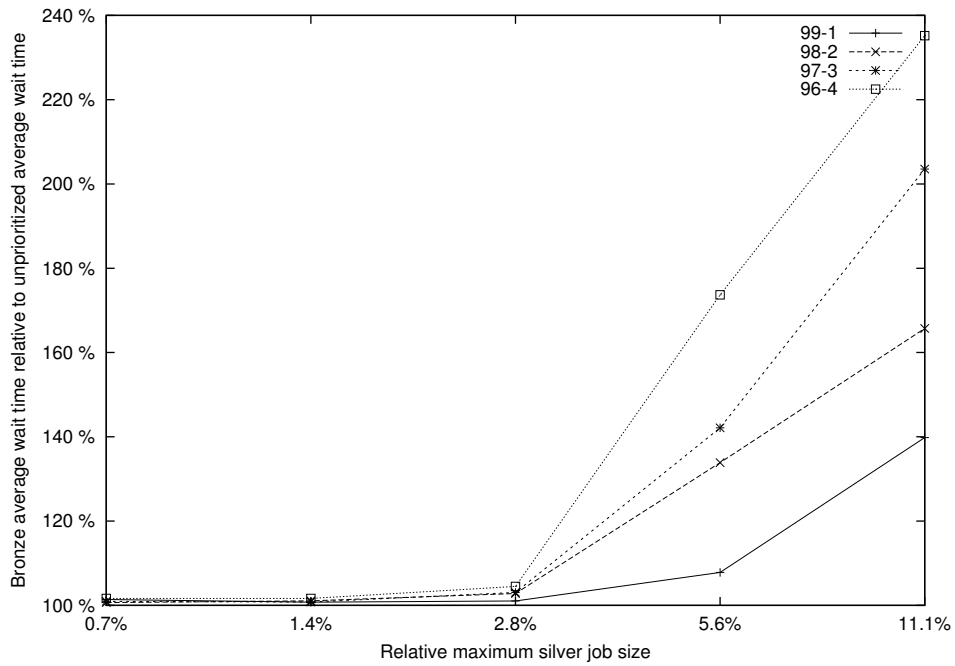


Figure 6.17: Bronze average wait time versus job size for shortest job by core hours cancellation

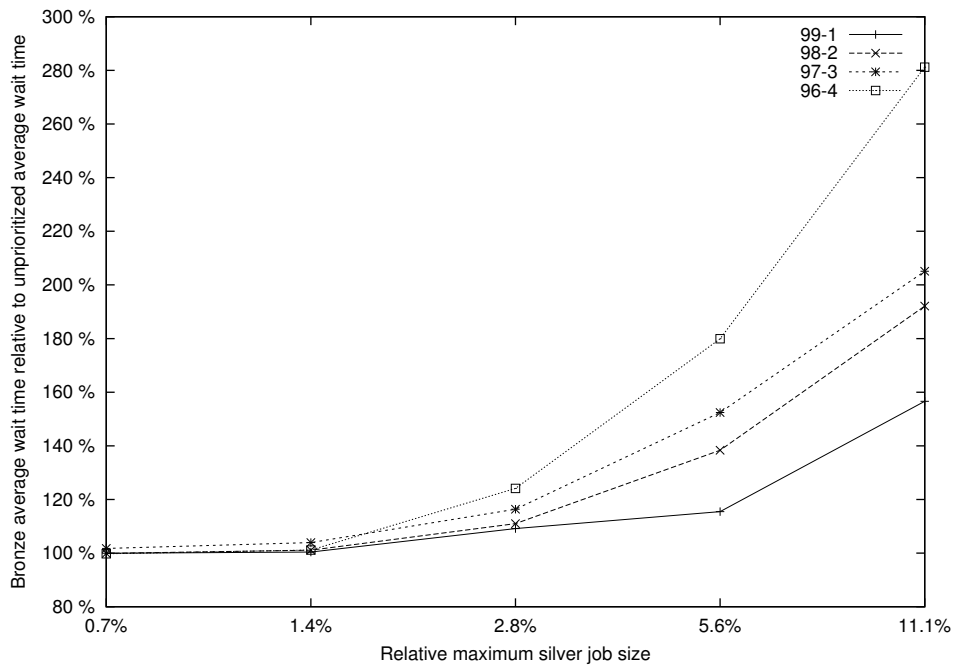


Figure 6.18: Bronze average wait time versus job size for shortest job by time cancellation

Figures 6.7, 6.19, 6.20 and 6.21 show the average wait time for silver jobs. It can be easily seen that for silver job sizes of up to including 64 CPUs (5.6 %) of the total CPU number the cancellation of bronze jobs leads to immediate execution of silver jobs. For higher percentages of silver jobs the silver jobs are starting to compete with each other and waiting time quickly increases, the more the higher the percentage of prioritised jobs is. 64 CPUs (5.6 %) is thus a threshold which still allows the offering of immediate execution by the provider.

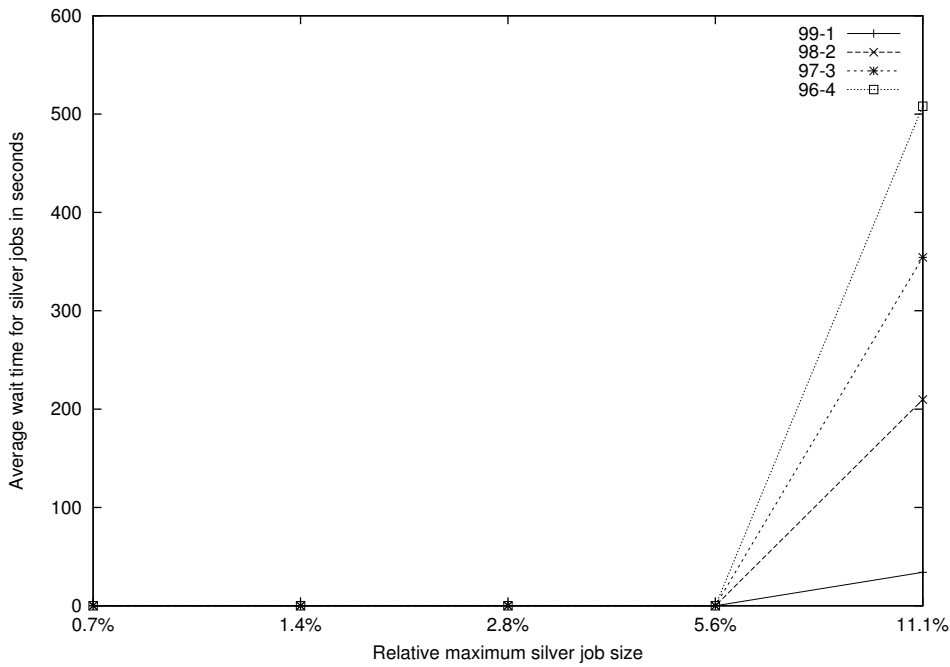


Figure 6.19: Silver average wait time versus job size for least percentage completed cancellation

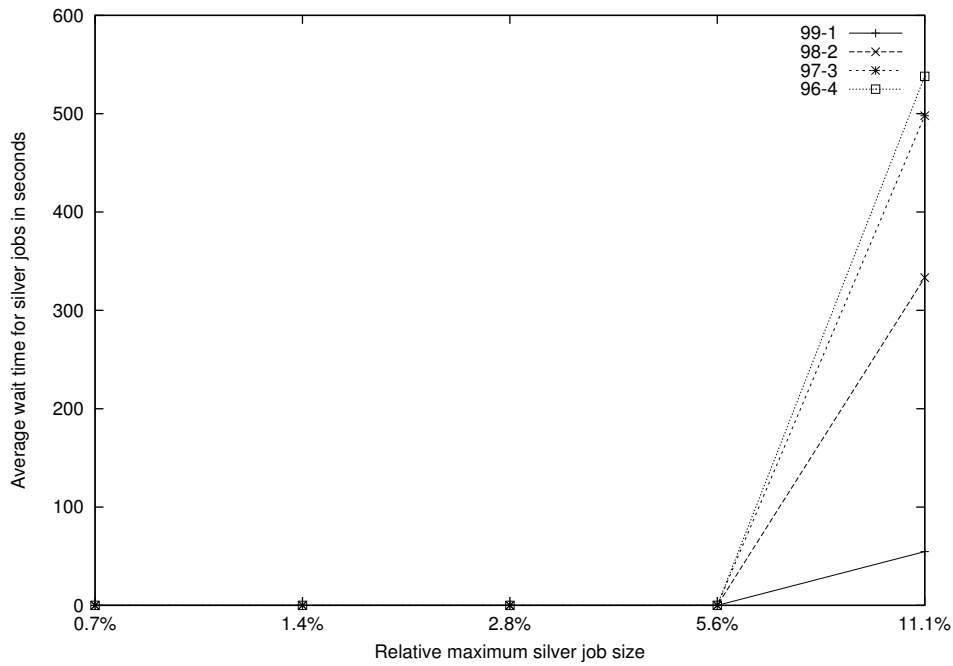


Figure 6.20: Silver average wait time versus job size for shortest job by core hours cancellation

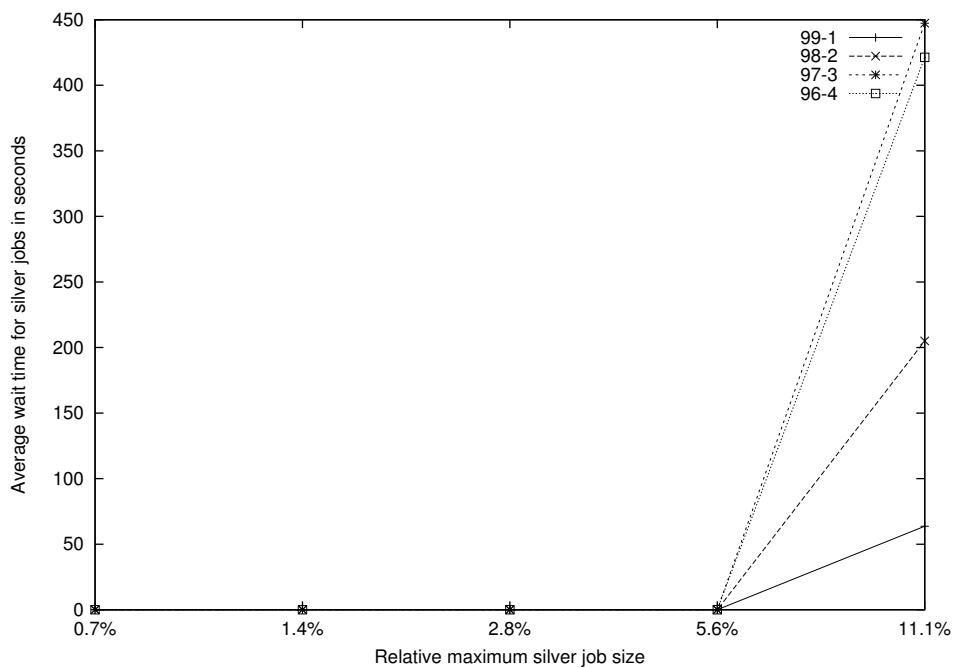


Figure 6.21: Silver average wait time versus job size for shortest job by time cancellation

6.4 Summary

This chapter has presented a program for the simulation of SLA-based scheduling which has been developed out of a discrete event based scheduling simulation tool. Modifications have been performed in order to allow for the simulation of different distributions of service levels, the limitation of job sizes for service levels, different scheduling policies and different cancellation policies. The productivity of this tool has been demonstrated by performing simulations for different scenarios.

Firstly, a scenario with a best effort service level and a prioritised service level was investigated. In this case, jobs were sorted in a queue relative to each other and prioritised jobs were always scheduled before best effort jobs. It could be seen that even a simple scheduling policy like this can be effective if either the total amount of prioritised jobs or the maximum size of prioritised jobs is limited. Then, waiting times are quite predictable even though no definite guarantees are given in this case.

Secondly, extending from the first scenario, it was investigated how the situation changes if running best effort jobs are cancelled and rescheduled once prioritised jobs are in the queue. Naturally, the waiting time for prioritised jobs is only limited by the time it takes to cancel running jobs in order to free resources. However, this leads to waste, as computing time is lost. The questions posing itself from this simulation are how much more expensive a prioritised job can be and how much waiting time increase due to cancellation users of the best effort service level are willing to accept.

Thirdly, the same scenario as in the second simulation was used as a basis. Instead of a policy cancelling best effort jobs at random, however, three different cancellation policies that take different aspects of a job into account were investigated. In general, regardless of cancellation policies, the factor with the most influence has shown to be the maximum size of prioritised jobs. The number of jobs allowed to be prioritised comes only second to this. For the workload at hand, allowing jobs of up to 2.8 % (32 processors) of the infrastructure's size is feasible. Larger jobs quickly lead to huge increases in wasted computing time. It is questionable if a best effort service level with the resulting large waiting time would be attractive to clients. Limiting the allowed prioritised job size to smaller numbers is likewise not sensible, albeit for different reasons: cancellations still occur but as small jobs in our case only resulted in a few core hours, silver SLAs would have to be priced very high in order to make up for the cancellation. It comes as no surprise that taking information provided by the system into account when cancelling jobs gives better results than randomly cancelling jobs. Cancelling jobs which require in total the least amount of core hours, regardless of their current state, proved to be the most effective policy out of the ones investigated. Pairing this cancellation policy with maximum prioritised job size of 2.8 % (32 processors) allows the provider a good silver service level offering with little influence on the bronze service level. By allowing a different number of prioritised jobs – between 1 % and 4 % - – the provider can fine-tune the service offering.

The simulations in this chapter have been performed against a realistic workload trace and they are specific to this trace and its infrastructure. Providers wishing to perform simulations like this should at best obtain a workload trace for their own infrastructure. If service levels are introduced into a production infrastructure, the resulting workload traces can be used as a new input for simulation which can in turn be used to fine-tune the service levels.

Chapter 7

Conclusions and future work

This chapter analyses the work performed in this thesis, whose main part has been described in chapters 4, 5 and 6 and proposes topics for future work that transcend the scope of this thesis.

7.1 Conclusions

Starting from the observation that HPC provider offerings are still limited to best effort based services, this thesis proposed a concept for the usage of SLAs in HPC environments, along with an SLA management framework, which was subsequently implemented. Simulation was performed in order to investigate influence of different service levels on each other and the provider's infrastructure with various scheduling policies.

7.1.1 Service level agreements for the HPC domain

SLAs are a tool for defining QoS requirements in many situations. The predominant use of SLAs related to computational jobs assumes each SLA corresponds to a single compute job. This is fine for investigation of job scheduling but is inconsistent with the contracting behaviour between HPC providers and their clients. Negotiation of and agreement on SLAs for individual jobs is out of the question.

The approach which has been developed in this work is to regard an SLA as a contract that describes a service level that can be used to execute many jobs. This is a way to enhance the current offerings with service levels without having to individually negotiate SLAs for each job. When SLAs are used in that way, we can refer to them as long-term SLAs, because they are a contract (or part of a contract) and not just a description of QoS requirements for an individual job. Not having to cater for an enormous amount of

possible SLAs but only for a small, select number of service levels makes implementation at an HPC provider reasonable in the first place.

7.1.2 Service level agreement management for HPC environments

In chapter 4, the architecture of an SLA management framework for HPC environments was proposed. This framework was developed out of the requirements identified in chapter 2 and was influenced by the analysis of other frameworks (see section 3.3.3). The focus, however, was put on implementing a framework that is, at its core, as lightweight as possible. This was achieved partly through the observation that not the whole SLA lifecycle needs to be supported (see section 4.3).

The framework has been implemented for a specific scenario but has been developed with extensibility for other scenarios in mind. For similar scenarios, the framework can be used with minimal adaptations. The framework is available as free software licensed under the GNU General Public License (GPL) v3 [98].

7.1.3 Simulating SLA usage in an HPC environment

Simulation allows the analysis of service levels and scheduling algorithms using defined physical infrastructures and workloads. This helps to develop and investigate scheduling algorithms before using them on real, physical infrastructures, as demonstrated in chapter 6.

Many simulation tools exist and were analysed (see section 3.5), but none readily provided support for simulation of service levels. A suitable candidate for extension was Alea, due to its simple architecture. Alea was enhanced in order to allow the creation and usage of SLA information for existing workloads. An HPC provider can now generate different service level distributions and examine, for example, how scheduling of the same workload is influenced with different SLA distributions, the impact of different scheduling algorithms or even the consequences of offering different service levels. The extended software is available as free software licensed under the GNU General Public License (GPL) v3 [98].

7.2 Future work

Future work is proposed in all of the three fields that make up the core of this work: the general usage of SLAs in an HPC environment, SLA management in an HPC environment

and simulation of SLA usage in HPC.

7.2.1 Refinement of simulation through feedback of real data

Adoption of the solution proposed in this work immediately suggests an area to be investigated: how does the mapping of service levels to job workloads correspond to users' behaviour when a provider offers service levels? A provider that implements service levels will quickly see patterns of how users react to the service levels. This is true both on a high level, regarding which contracts are requested by the user, and on a lower level, regarding which jobs users submit according to which of the offered service levels, how often and in which situations each service level is used and how this impacts the provider's service offering.

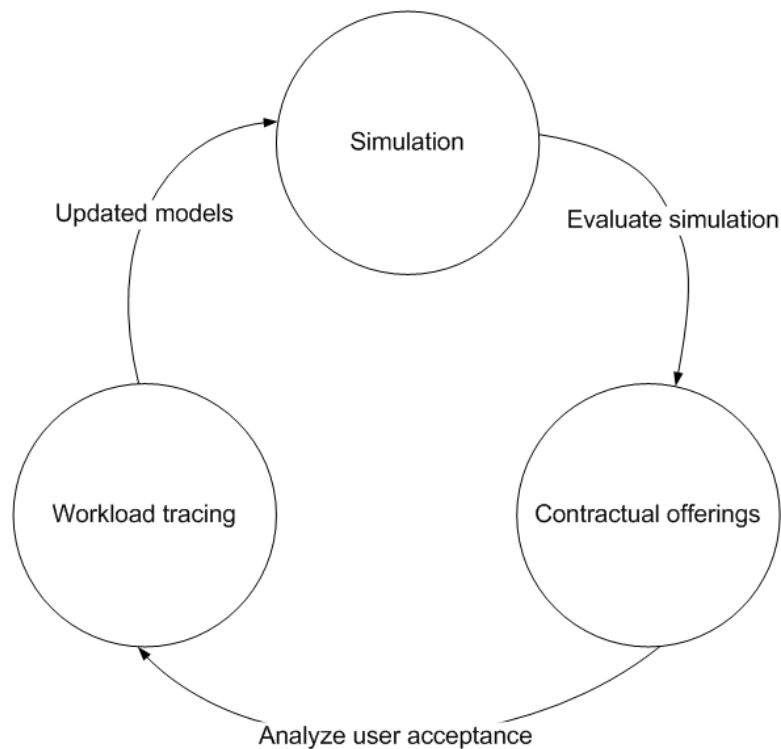


Figure 7.1: Feedback between simulation, contractual offerings and analysis of workload traces.

The data gathered through analysis of user behaviour therefore can be used to provide a more complex model of how users assign service levels to jobs, allowing the provider to simulate behaviour that may be closer to the real world than the pragmatic model that has been used in this work. This will be an ongoing process that allows the provider to align the simulation, contractual offerings and scheduler implementations on a continual basis.

7.2.2 SLA management in the client domain

SLA management in the client domain is, in current research, mostly neglected and this work is only partly an exception to this rule. The implementation presented in Chapter 5 has already raised the issue of SLA management in the client domain and has dealt with it in a pragmatic way (see especially section 5.3.1): a single application is used to submit jobs in reference to SLAs and any user on the client side working with this client would have identical rights regarding job submission - that is, the right to submit any job in relation to any SLA the client has in place with a provider.

The assumption in section 5.3.1 in this regard was of course very simple, but it does not matter if individual users in the client domain are identified via a shared account or individually but belong to a group - the provider would not account this usage differently, after all the request are coming from the same origin. Therefore, it is desirable for a client to have its own set of SLA management tools in place. These could allow different users rights to use different SLAs - an intern would most probably not have access to high-priced SLAs for urgent computing but would be limited to best effort jobs in general. Higher ranking employees would have more freedom in what contract they select but the usage of individual employees would be accounted for. This would as well enable the client to validate invoices from the provider.

There are many more features that would be desirable in an SLA management system in the client domain and a provider being able to support its client by placing a free/open source (FLOSS) client side SLA management solution at their disposal would have, beside its service levels, a compelling reason for usage of its services. Without offering the client any support, it is anyway not likely that service level offerings will be accepted easily, especially for more complex usage scenarios.

7.2.3 Analysing revenue changes for HPC providers

The simulations performed in chapter 6 examined, among other things, which price would be required for a prioritised job to make up for revenue loss due to job cancellation. This was a simple calculation not taking any additional overhead into account but, nonetheless, it showed the importance of pricing when introducing service levels.

In short, it can be said that providers naturally do not want to be worse off in terms of revenue when offering service level apart from best effort. A revenue calculation is more or less straightforward if only one service level exists but can get complicated quite quickly.

For a simple best effort based service, the gain $Gain_{total}$ is the difference between earnings $Earnings_{total}$ and $Expenditures_{total}$ and gain, earnings and expenditures are at the same

time total and for the single service level.

If more service levels are provided, the equation gets more complicated:

$$Gain_{total} = Earnings_1 - Expenditures_1 + \dots + Earnings_n - Expenditures_n$$

or

$$Gain_{total} = \sum_{i=1}^n Earnings_i - Expenditures_i$$

for n service levels where the expenditures are in this case made up of the sum of expenditures due to operation of the infrastructure itself and waste incurred through, for example, cancellation of lower priority jobs, deliberate idle time of the machine in order to keep resources available etc. Implementing penalties and rewards for achieving or missing certain goals is certainly interesting as well. Care needs to be taken, however, to ensure beforehand that the provider cannot be exploited and will at least be covering its expenditures.

Bibliography

- [1] Gregor Laszewski and Lizhe Wang. GreenIT Service Level Agreements. In Philipp Wieder, Ramin Yahyapour, and Wolfgang Ziegler, editors, *Grids and Service-Oriented Architectures for Service Level Agreements*, pages 77–88. Springer US, 2010. 10.1007/978-1-4419-7320-7_8.
- [2] Igor Rosenberg, Antonio Conguista, and Roland Kübert. Management for Service Level Agreements. In Theo Dimitrakos, Josep Martrat, and Stefan Wesner, editors, *Service Oriented Infrastructures and Cloud Service Platforms for the Enterprise*, pages 103–124. Springer Berlin Heidelberg, 2010. 10.1007/978-3-642-04086-3_5.
- [3] Roland Kübert. Service Level Agreements for Job Control in Grid and High-Performance Computing Grid Computing. In Nikolaos P. Preve, editor, *Grid Computing: Towards a Global Interconnected Infrastructure*, Computer Communications and Networks, chapter 8, pages 205–221. Springer London, London, 2011.
- [4] Roland Kübert and Stefan Wesner. Service Level Agreements For Job Control in High-Performance Computing. In *IMCSIT*, pages 655–661, 2010.
- [5] Roland Kübert. Providing Quality of Service through Service Level Agreements in a High-Performance Computing Environment. In P. Ivnyi and B.H.V. Topping, editors, *PARENG 2011, 2-nd Int. Conf. on Parallel, Distributed, Grid and Cloud Computing for Engineering*, Ajaccio, France, April 2011. Civil-Comp Press. Paper 52.
- [6] Pierre Gilet, Axel Tenschert, and Roland Kübert. Use of Graphical Modelling, Semantics and Service Level Agreements for High Performance Computing. In *eChallenges e-2011 Conference, Florence, Italy*, pages **–**, 2011.
- [7] Axel Tenschert and Roland Kübert. SLA-BASED JOB SUBMISSION AND SCHEDULING WITH THE GLOBUS TOOLKIT 4. *Computer Science*, 13(4), 2012.
- [8] NextGRID Consortium. NextGRID project home page. <http://nextgrid.org/>, 2008.
- [9] FinGrid Consortium. FinGrid project home page. <http://141.2.67.69/>, 2008.

- [10] IRMOS Consortium. IRMOS project home page. <http://irmos-project.eu/>, 2008.
- [11] SLA4D-Grid Consortium. SLA4D-Grid project home page. <http://www.sla4d-grid.de/>, 2008.
- [12] Michael Resch. *Entgeltordnung für die Nutzung der Rechenanlagen und peripheren Geräte des Höchstleistungsrechenzentrums Stuttgart (HLRS) an der Universität Stuttgart*, 2010.
- [13] eResearch SA. Hydra user guide. http://www.eresearchsa.edu.au/hydra_guide#usage.
- [14] Lancaster University. Running jobs on the hpc. http://www.lancs.ac.uk/iss/hpc/running_jobs.html.
- [15] High Performance Computing Center Stuttgart. Batch System - HLRS SX-9. https://wickie.hlrs.de/platforms/index.php/Batch_System.
- [16] Cluster Resources Inc. TORQUE Resource Manager. <http://www.clusterresources.com/products/torque-resource-manager.php>.
- [17] M. J. Boniface, S. C. Phillips, and M. SurrIDGE. Grid-based business partnerships using service level agreements. *Proc. of Cracow Grid Workshop 2006*, 2006.
- [18] Tobias Mahler, Alvaro Arenas, and Lutz Schubert. Contractual frameworks for enterprise networks and virtual organisations in e-learning. In P. Cunningham and M. Cunningham, editors, *eChallenges e-2006 Conference, Exploiting the Knowledge Economy: Issues, Applications, Case Studies*, pages 271–278. IOS Press, 2006.
- [19] H. Hérenger, S. Roller, and E Göde. Interactive design of hydraulic turbomachinery. In B.H.V. Topping, L.F. Costa Neves, and R.C. Barros, editors, *Proceedings of the Twelfth International Conference on Civil, Structural and Environmental Engineering Computing*, Stirlingshire, UK, 2009. Civil-Comp Press.
- [20] Carolina Cruz-Neira, Daniel J. Sandin, Thomas A. DeFanti, Robert V. Kenyon, and John C. Hart. The cave: audio visual experience automatic virtual environment. *Commun. ACM*, 35:64–72, June 1992.
- [21] BREIN Consortium. BREIN project home page. <http://www.eu-brein.com/>, 2008.
- [22] Ian Foster, Carl Kesselman, Gene Tsudik, and Steven Tuecke. A security architecture for computational grids. In *Proceedings of the 5th ACM conference on Computer*

-
- and communications security*, CCS '98, pages 83–92, New York, NY, USA, 1998. ACM.
- [23] T. Goss-Walter, R. Letz, T. Kentemich, H.-C. Hoppe, and Ph. Wieder. An analysis of the uncore security model. <http://www.gridforum.org/documents/GFD.18.pdf>, 2003.
- [24] Inc. Adaptive Computing Enterprises. qsub. <http://www.clusterresources.com/torquedocs21/commands/qsub.shtml>, 2010.
- [25] Globus Alliance. GT 4.0 WS GRAM: Job Description Schema Doc, 2004.
- [26] Technische Universität Dresden. TUD - Hochleistungsrechnen (HPC) - Informationen für Nutzer der PC Farm Deimos. http://tu-dresden.de/die_tu_dresden/zentrale_einrichtungen/zih/hpc/hochleistungsrechner/deimos/nutzerinfo.
- [27] High Performance Computing Center Stuttgart. Nec nehalem cluster queue policies. https://wickie.hlr.de/platforms/index.php/NEC_Nehalem_Cluster_QueuePolicies.
- [28] Forschungszentrum Jülich. FZJ-ZAM Blue Gene/L User Information - Scheduling Policy. <http://www2.fz-juelich.de/jsc/ibm-bgl/usage/reservation/>.
- [29] Sandia National Laboratories. Sandia National Laboratories: Securing a peaceful and free world through technology. <http://www.sandia.gov>.
- [30] Allen G. Sault, Suzanne M. Kelly, Paula L. McAlliser, Joel D. Miller, and Gerald F. Quinland. *ASCI Red for Dummies: A Recipe Book for Easy Use of the ASCI Red Platform*, 2006.
- [31] Matthias Hovestadt, Odej Kao, Axel Keller, and Achim Streit. Scheduling in hpc resource management systems: Queuing vs. planning. In *Proc. of the 9th Workshop on Job Scheduling Strategies for Parallel Processing*, pages 1–20. Springer, 2003.
- [32] Joseph Skovira, Waiman Chan, Honbo Zhou, and David A. Lifka. The easy - loadleveler api project. In *Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing*, pages 41–47, London, UK, 1996. Springer-Verlag.
- [33] C.L. Liu and James Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment, 1973.
- [34] Dalibor Klusáček, Hana Rudová, Ranieri Baraglia, Marco Pasquali, and Gabriele Capannini. Comparison of Multi-Criteria Scheduling Techniques. In *Integrated Research in Grid Computing*. Springer, 2008.

- [35] Mohammad Islam, Pavan Balaji, P. Sadayappan, and Dhabaleswar K. Panda. Qops: A qos based scheme for parallel job scheduling. In Dror G. Feitelson, Larry Rudolph, and Uwe Schwiegelshohn, editors, *JSSPP*, volume 2862 of *Lecture Notes in Computer Science*, pages 252–268. Springer, 2003.
- [36] Saeed Iqbal, Rinku Gupta, and Yung-Chin Fang. Planning considerations for job scheduling in hpc clusters. *Dell Power Solution*, pages 133–136, 2005.
- [37] Alain Andrieux, Karl Czajkowski, Asit Dan, Kate Keahey, Heiko Ludwig, Toshiyuki Nakata, Jim Pruyne, John Rofrano, Steve Tuecke, and Ming Xu. Web Services Agreement Specification (WS-Agreement). Technical report, Global Grid Forum, Grid Resource Allocation Agreement Protocol (GRAAP) WG, September 2005.
- [38] Alexander Keller and Heiko Ludwig. The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services. *J. Netw. Syst. Manage.*, 11(1):57–81, 2003.
- [39] James Skene, D. Davide Lamanna, and Wolfgang Emmerich. Precise service level agreements. In *Proceedings of the 26th International Conference on Software Engineering*, ICSE '04, pages 179–188, Washington, DC, USA, 2004. IEEE Computer Society.
- [40] Open Grid Forum GRAAP Working Group. Ws-agreement implementations. <https://forge.gridforum.org/sf/wiki/do/viewPage/projects.graap-wg/wiki/Implementations>.
- [41] University College London. SLAng FAQ: Why should I not use SLAng for real SLAs yet? <http://uclslang.sourceforge.net/faq.php#q13>.
- [42] Bastian Koller. *Enhanced SLA mangement in the high performance computing domain*. Dissertation, Universität Stuttgart, Stuttgart, Germany, 2011.
- [43] Rizos Sakellariou and Viktor Yarmolenko. On the Flexibility of WS-Agreement for Job Submission. In *MGC '05: Proceedings of the 3rd international workshop on Middleware for grid computing*, pages 1–6, New York, NY, USA, 2005. ACM.
- [44] Dalibor Klusáček and Hana Rudová. Improving QoS in Computational Grids through Schedule-based Approach. In *Scheduling and Planning Applications Workshop at the Eighteenth International Conference on Automated Planning and Scheduling (ICAPS08)*, Sydney, Australia, 2008.
- [45] Jan Seidel, Oliver Wäldrich, Philipp Wieder, Ramin Yahyapour, and Wolfgang Ziegler. Using SLA for Resource Management and Scheduling - A Survey. In Domenico Talia, Ramin Yahyapour, and Wolfgang Ziegler, editors, *Grid Middle-*

- ware and Services - Challenges and Solutions*, CoreGRID Series. Springer, 2008. Also published as CoreGRID Technical Report TR-0096.
- [46] V. Yarmolenko and R. Sakellariou. An evaluation of heuristics for sla based parallel job scheduling. In *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*, pages 8 pp.–, April 2006.
- [47] Chee Shin Yeo and Rajkumar Buyya. Managing Risk of Inaccurate Runtime Estimates for Deadline Constrained Job Admission Control in Clusters. In *ICPP '06: Proceedings of the 2006 International Conference on Parallel Processing*, pages 451–458, Washington, DC, USA, 2006. IEEE Computer Society.
- [48] Catalin L. Dumitrescu, Ioan Raicu, and Ian Foster. Usage SLA-based scheduling in Grids. *Concurrency and Computation: Practice and Experience*, 19(7):945–963, 2007.
- [49] Rizos Sakellariou and Viktor Yarmolenko. Job Scheduling on the Grid: Towards SLA-Based Scheduling. In Lucio Grandinetti, editor, *High Performance Computing and Grids in Action*. IOS Press, 2008.
- [50] Karim Djemame, Iain Gourlay, James Padgett, Georg Birkenheuer, Matthias Hovestadt, Odej Kao, and Kerstin Voß. Introducing risk management into the grid. In *e-Science*, page 28. IEEE Computer Society, 2006.
- [51] Rajiv Ranjan, Aaron Harwood, and Rajkumar Buyya. A SLA-based coordinated superscheduling scheme and performance for computational grids. Technical report, In *Proceedings of the 8th IEEE International Conference on Cluster Computing (Cluster 2006)*, 2006.
- [52] P. Balakrishnan, S. Thamarai Selvi, and G. Rajesh Britto. Service Level Agreement Based Grid Scheduling. In *Proceedings of the 2008 IEEE International Conference on Web Services*, pages 203–210, Washington, DC, USA, 2008. IEEE Computer Society.
- [53] BEinGRID Consortium. BEinGRID project home page. <http://www.it-tude.com/projects/beingrid>, 2008.
- [54] Roland Kübert and Tinghe Wang. Implementing Service Level Agreement Management. 2010. http://www.eu-brein.com/index.php?option=com_docman&task=doc_download&gid=74.
- [55] Axel Tenschert, Roland Kübert, Bastian Koller, Tobias Pontz, Kay Dörnemann, Niels Fallenbeck, and Roland Schwarzkopf. Deliverable 8.4.1: Testen gemäß dem beschriebenen Verfahren zur Validierung, 2010. FinGrid project deliverable.

- [56] Andreas et al. Menychtas. D3.1.4 Final version of IRMOS Overall Architecture, 2011. IRMOS project deliverable.
- [57] Alan Beck. High Throughput Computing: An Interview With Miron Livny. <http://www.cs.wisc.edu/condor/HPCwire.1>, 1997.
- [58] IBM Corporation. IBM Tivoli Workload Scheduler LoadLeveler. <http://www-03.ibm.com/systems/software/loadleveler/>. [Online; accessed 11-November-2010].
- [59] The Register. Univa forks Oracle's Sun Grid Engine. http://www.theregister.co.uk/2011/01/18/univa_forks_oracle_grid_engine/.
- [60] Lawrence Livermore National Laboratory. Simple Linux Utility for Resource Management. <https://computing.llnl.gov/linux/slurm/slurm.html>. [Online; accessed 11-November-2010].
- [61] UnivaUD. Grid MP v5.6 Data Sheet. <http://www.univaud.com/about/resources/files/ds-grid-mp.pdf>. [Online; accessed 12-November-2010].
- [62] Andrea Emilio Rizzoli. A Collection of Modelling and Simulation Resources on the Internet. <http://www.idsia.ch/~andrea/sim/simtools.html>. [Online; accessed 22-February-2011].
- [63] R. McNab and F. W. Howell. Using Java for Discrete Event Simulation. In *Proc. Twelfth UK Computer and Telecommunications Performance Engineering Workshop (UKPEW)*, Univ. of Edinburgh, pages 219–228, 1996.
- [64] Rajkumar Buyya and 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 (CCPE)*, 14(13):1175–1220, 2002.
- [65] H. J. Song, X. Liu, D. Jakobsen, R. Bhagwan, X. Zhang, K. Taura, and A. Chien. The microgrid: a scientific tool for modeling computational grids. In *Proceedings of IEEE Supercomputing (SC 2000)*, pages 4–10, 2000.
- [66] Atsuko Takefusa. Bricks: A performance evaluation system for scheduling algorithms on the grids. *JSPS Workshop on Applied Information Technology for Science (JWAITS 2001)*, 2001.
- [67] Henri Casanova, Arnaud Legrand, and Martin Quinson. SimGrid: a Generic Framework for Large-Scale Distributed Experiments. In *10th IEEE International Conference on Computer Modeling and Simulation*, March 2008.
- [68] Henri Casanova. Simgrid: a toolkit for the simulation of application scheduling. In

-
- Proceedings of the First IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2001)*, pages 430–437, 2001.
- [69] Jean-Sébastien Gay and Yves Caniou. Simbatch : une api pour la simulation et la prédiction de performances de systèmes batch. In *17ieme Rencontres Francophones du Parallélisme, des Architectures et des Systèmes*, pages 180–187, Perpignan, October 2006.
- [70] Ciprian Dobre and Corina Stratan. Monarc simulation framework. *CoRR*, abs/1106.5158, 2011.
- [71] Krzysztof Kurowski, Jarek Nabrzyski, Ariel Oleksiak, and Jan Weglarz. Grid scheduling simulations with gssim. In *Proceedings of the 13th International Conference on Parallel and Distributed Systems - Volume 02*, pages 1–8, Washington, DC, USA, 2007. IEEE Computer Society.
- [72] Dalibor Klusáček, Luděk Matyska, and Hana Rudová. Alea - Grid Scheduling Simulation Environment. In Roman Wyrzykowski, Jack Dongarra, Konrad Karczewski, and Jerzy Wasniewski, editors, *Parallel Processing and Applied Mathematics*, volume 4967 of *Lecture Notes in Computer Science*, pages 1029–1038. Springer Berlin / Heidelberg, 2008.
- [73] Dalibor Klusáček and Hana Rudová. Alea 2 – Job Scheduling Simulator. In *Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques (SIMUTools 2010)*. ICST, 2010.
- [74] The pyss project. pyss - the Python Scheduler Simulator. <http://code.google.com/p/pyss/>.
- [75] Stefan Wesner. *Integrated Management Framework for Dynamic Virtual Organizations*. Dissertation, Universität Stuttgart, Stuttgart, Germany, 2008.
- [76] Joanna Leng and Nigel W. John. Scientific application of opengl vizserver within high performance computing. http://www.ukhec.ac.uk/publications/reports/vizserver_casestudy_dec02.pdf.
- [77] High Performance Computing Center Stuttgart. Nec nehalem cluster — hrs.de. [Online; accessed 23 November 2011].
- [78] Pete Beckman. Urgent computing: Exploring supercomputings new role. *CTWatch Quarterly*, 4(1), 2008.
- [79] Pete Beckman, Ivan Beschastnikh, Suman Nadella, and Nick Trebon. Building an infrastructure for urgent computing. In Lucio Grandinetti, editor, *High Performance*
-

- Computing and Grids in Action*, volume 16 of *Advances in Parallel Computing*, pages 75–95, 2008.
- [80] Carlisle Adams and Steve Lloyd. *Understanding PKI: Concepts, Standards, and Deployment Considerations*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd edition, 2002.
- [81] TeleManagement Forum. *SLA Management Handbook - Vol. 2 - Concepts and Principles*, 2005.
- [82] I.B.M. Redbooks. *Service Lifecycle Governance With IBM Websphere Service Registry and Repository*. Vervante, 2009.
- [83] Antoine de Saint-Exupery. *Terre des hommes / Antoine de Saint-Exupery*. Gallimard, Paris :, 21st ed. edition, 1939.
- [84] Thomas Erl. *SOA Principles of Service Design (The Prentice Hall Service-Oriented Computing Series from Thomas Erl)*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2007.
- [85] University of Southern California. USC - High Performance Computing & Communications. <http://www.usc.edu/hpcc/systems/globus.php>. [Online; accessed 12-Dec-2011].
- [86] J. Laitinen, H. Heller, and G. Pringle, editors. *Globus Job Submission in DEISA*. Distributed European Infrastructure for Supercomputing Architectures, 2010.
- [87] Thomas Röblitz, editor. *Resource Management for Grid-Jobs*. AstroGrid-D, 2009.
- [88] The Globus Alliance. GT 4.0 WS GRAM Approach. http://www.globus.org/toolkit/docs/4.0/execution/wsgram/WS_GRAM_Approach.html.
- [89] Don Box, Erik Christensen, Francisco Curbera, Donald Ferguson, Jeffrey Frey, Marc Hadley, Chris Kaler, David Langworthy, Frank Leymann, Brad Lovering, Steve Lucco, Steve Millet, Nirmal Mukhi, Mark Nottingham, David Orchard, John Shewchuk, Eugène Sindambiwe, Tony Storey, Sanjiva Weerawarana, and Steve Winkler. *Web Services Addressing (WS-Addressing)*. Technical report, W3C, August 2004.
- [90] The Globus Alliance. Security Descriptors. http://www.globus.org/toolkit/docs/4.0/security/authzframe/security_descriptor.html. [Online; accessed 06-May-2011].
- [91] Globus Alliance. *WS-GRAM Scheduler Interface Tutorial (Perl Module)*, 2005.

- [92] Ole Holm Nielsen. Torque accounting scripts, 2009.
- [93] Albino Aveleda. pbsacct project home page, 2006.
- [94] University of Edinburgh. Grid-SAFE project home page, 2010.
- [95] Y. Shafranovich. RFC 4180: Common Format and MIME Type for Comma-Separated Values (CSV) Files, 2005.
- [96] Dror Feitelson. The Standard Workload Format. <http://www.cs.huji.ac.il/labs/parallel/workload/swf.html>. [Online, accessed 28-February-2011].
- [97] Dalibor Klusáček. Dealing with Uncertainties in Grids through the Event-based Scheduling Approach. In *4th Doctoral Workshop on Mathematical and Engineering Methods in Computer Science (MEMICS 2008)*, pages 91–98, 2008.
- [98] Free Software Foundation, Inc. GNU GENERAL PUBLIC LICENSE Version 3. <http://www.gnu.org/licenses/gpl-3.0.txt>.