Portable Tools for Interoperable Grids

Modular Architectures and Software for
Job and Workflow Management

Johan Tordsson

PhD Thesis, March 2009
Department of Computing Science
Umeå University
Sweden

Department of Computing Science
Umeå University
SE-901 87 Umeå, Sweden

tordsson@cs.umu.se

ISBN 978-91-7264-754-1
ISSN 0348-0542
UMINF 09.08

Printed by Print & Media, Umeå University, 2009

Abstract

The emergence of Grid computing infrastructures enables researchers to share resources and collaborate in more efficient ways than before, despite belonging to different organizations and being geographically distributed. While the Grid computing paradigm offers new opportunities, it also gives rise to new difficulties. This thesis investigates methods, architectures, and algorithms for a range of topics in the area of Grid resource management. One studied topic is how to automate and improve resource selection, despite heterogeneity in Grid hardware, software, availability, ownership, and usage policies. Algorithmical difficulties for this are, e.g., characterization of jobs and resources, prediction of resource performance, and data placement considerations. Investigated Quality of Service aspects of resource selection include how to guarantee job start and/or completion times as well as how to synchronize multiple resources for coordinated use through coallocation. Another explored research topic is architectural considerations for frameworks that simplify and automate submission, monitoring, and fault handling for large amounts of jobs. This thesis also investigates suitable Grid interaction patterns for scientific workflows, studies programming models that enable data parallelism for such workflows, as well as analyzes how workflow composition tools should be designed to increase flexibility and expressiveness.

We today have the somewhat paradoxical situation where Grids, originally aimed to federate resources and overcome interoperability problems between different computing platforms, themselves struggle with interoperability problems caused by the wide range of interfaces, protocols, and data formats that are used in different environments. This thesis demonstrates how proof-of-concept software tools for Grid resource management can, by using (proposed) standard formats and protocols as well as leveraging state-of-the-art principles from service-oriented architectures, be made independent of current Grid infrastructures. Further interoperability contributions include an in-depth study that surveys issues related to the use of Grid resources in scientific workflows. This study improves our understanding of interoperability among scientific workflow systems by viewing this topic from three different perspectives: model of computation, workflow language, and execution environment.

A final contribution in this thesis is the investigation of how the design of Grid middleware tools can adopt principles and concepts from software engineering in order to improve, e.g., adaptability and interoperability.

Sammanfattning

Dagens Grid-infrastruktur gör det möjligt för forskare att dela vetenskaplig utrustning såsom högpresterande datorer och dyrbara instrument och därmed samarbeta mer effektivt än tidigare, trots att de tillhör olika organisationer och är geografiskt åtskilda. Grid-tekniken erbjuder nya möjligheter, men ger även upphov till nya problemställningar. Denna avhandling studerar en rad frågeställningar inom Grid, med särskilt fokus på metoder, arkitekturer och algoritmer för resurshantering. Ett bidrag är en studie av hur matchning av jobb och resurser kan automatiseras och förbättras, trots heterogenitet i hårdvara, programvaror och användningspolicyer i de maskiner som finns tillgängliga i en Grid. Algoritmiska aspekter av detta problem inkluderar karaktärisering av jobb och maskiner, prestandaprediktion samt konsekvenser av placeringen av in- och utdatafiler. Vidare studeras kvalitetsgarantier (Quality of Service), i detta fall vilka mekanismer som krävs för att garantera start- och/eller sluttider för jobb. Ett relaterat problem är hur man bäst samallokerar flera resurser för koordinerad användning. Ett annat bidrag är en studie av lämpliga arkitekturer för automatiserad hantering av stora mängder jobb. Avhandlingen behandlar även hur man på bästa sätt integrerar Grid-resurser i vetenskapliga arbetsflöden (workflows), vilka programmeringsmodeller som lämper sig bäst för dataparallellism för workflows, samt hur verktyg för att definiera workflows bör konstrueras för ökad flexiblitet.

Vi har idag en något paradoxal situation där Grid-teknik, som delvis designats med målet att integrera heterogena plattformar och överbrygga kompatibilitetsproblem, i sig ger upphov till en ny nivå av kompabilitetsproblem, mellan olika Grid-plattformar. Dessa problem beror på stora skillnader i såväl de gränssnitt, protokoll och dataformat som används i dagens infrastrukturer som i deras övergripande arkitektur. Denna avhandling demonstrerar hur programvara för resurshantering kan göras oberoende av nuvarande Grid-plattformar genom att utnyttja (föreslagna) standardformat och protokoll såväl som principer från service-orienterade arkitekturer. Andra bidrag inkluderar en fördjupad studie om hur Grid-resurser bäst integreras i workflows. Denna studie analyserar skillnader mellan befintliga workflow-system och belyser interoperabilitet mellan dessa ur tre aspekter: beräkningsmodell, språk för att beskriva workflows samt exekveringsplattform.

Avslutningsvis studeras i denna avhandling hur resultat av forskning inom programvaruteknik (software engineering) kan användas för att förbättra designen av Grid-programvaror, bland annat för att öka interoperabilitet och anpassningsbarhet.

Preface

This thesis consists of an introduction and the following papers:

Paper I	E. Elmroth and J. Tordsson. Grid Resource Brokering Algorithms Enabling Advance Reservations and Resource Selection Based on Performance Prediction. *Future Generation Computer Systems. The International Journal of Grid Computing: Theory, Methods and Applications*, Elsevier B.V., Vol 24, No. 6, pp. 585-593, 2008.
Paper II	E. Elmroth and J. Tordsson. An Interoperable, Standards-based Grid Resource Broker and Job Submission Service. In H. Stockinger, R. Buyya, and R. Perrot (Eds.), *e-Science 2005. First IEEE Conference on e-Science and Grid Computing*, IEEE Computer Society Press, pp. 212-220, 2005.
Paper III	E. Elmroth and J. Tordsson. A Standards-based Grid Resource Brokering Service Supporting Advance Reservations, Coallocation and Cross-Grid Interoperability. *Concurrency and Computation: Practice and Experience*, accepted, 2009.
Paper IV	E. Elmroth, P. Gardfjäll, A. Norberg, J. Tordsson and P-O. Östberg. Designing General, Composable, and Middleware-independent Grid Infrastructure Tools for Multi-tiered Job Management. In T. Priol and M. Vaneschi (Eds.), *Towards Next Generation Grids*, Springer-Verlag, pp. 175 - 184, 2007.
Paper V	E. Elmroth, F. Hernández, and J. Tordsson. A Light-weight Grid Workflow Execution Service Enabling Client and Middleware Independence. In R. Wyrzykowski, J. Dongarra, K. Karczewski, and J. Wasniewsky (Eds.), *Parallel Processing and Applied Mathematics*, Lecture Notes in Computer Science vol. 4967, Springer-Verlag, pp. 754 - 761, 2008.
Paper VI	A-C. Berglund, E. Elmroth, F. Hernández, B. Sandman, and J. Tordsson. Combining Local and Grid Resources in Scientific Workflows (for Bioinformatics). In *9th International Workshop on State-of-the-Art in Scientific and Parallel Computing*, Lecture Notes in Computer Science, Springer-Verlag, accepted, 2009.
Paper VII	E. Elmroth, F. Hernández, and J. Tordsson. Three Fundamental Dimensions of Scientific Workflow Interoperability: Model of Computation,

Language, and Execution Environment. Technical report UMINF 09.05. Submitted for journal publication, 2009.

Paper VIII E. Elmroth, F. Hernández, J. Tordsson, and P-O. Östberg. Designing Service-based Resource Management Tools for a Healthy Grid Ecosystem. In R. Wyrzykowski, J. Dongarra, K. Karczewski, and J. Wasniewsky (Eds.), *Parallel Processing and Applied Mathematics*, Lecture Notes in Computer Science vol. 4967, Springer-Verlag, pp. 259 - 270, 2008.

This research was conducted using the resources of the High Performance Computing Center North (HPC2N). Financial support has been provided by The Swedish Research Council (VR) under contracts 343-2003-953 and 621-2005-3667, as well as by the European Community's Seventh Framework Programme ([FP7/2001-2013]) under grant agreement no. 215605.

Acknowledgments

First of all, I thank my supervisor Erik Elmroth for guiding, encouraging, and most of all inspiring this work. Your day-round rapid email response times however worry me :-). I also thank my associate supervisor Bo Kågström for enlightening discussions. A big *gracias* goes to Francisco Hernández, with whom I have had the pleasure to work closely together with for the last two years. I express my gratitude to my fellow Grid researchers Peter Gardfjäll, Arvid Norberg, and P-O Östberg. I also acknowledge Ann-Charlotte Berglund Sonnhammar and Björn Sandman for their contributions to the work in this thesis.

Åke Sandgren, Björn Torkelsson and Tomas Ögren provided valuable assistance in the never-ending task of (re)installing our local Grid testbed. I am grateful to Inger Sandgren, who numerous times patiently helped me navigate the maze of travel expenses reimbursement. I also thank the Grid computing group and all other collegues for providing a creative research environment.

On a more personal level, I thank my friends and family for their encouragement.

Thank you!

Umeå, March 2009
Johan Tordsson

Contents

Chapter 1

Introduction

1.1 Background

During the late 1980's and early 1990's, the rapid development of network capacity made it feasible to interconnect remotely located parallel computers, in order to tackle larger-than-supercomputer problems. The resulting infrastructures were referred to as meta-computers. The earliest meta-computing platforms were constructed with experiment-specific protocols. The I-WAY project [78] developed the first general-purpose toolkit and was successfully used by more than 50 applications, demonstrating the feasibility of interconnecting resources such as supercomputers, instruments and storage. The term Grid computing was coined to describe the interconnection of more general types of resources. The choice of name reflects the vision of computing made available as a utility, in analogy with how electricity is available from the power grid. In addition to the possibility to tackle larger-than-supercomputer problems, early motivating factors for the construction of Grids were improved collaboration and the possibility to remotely access scarce and expensive scientific instruments.

Perhaps the most well known Grid middleware is the Globus Toolkit [90], a further development of the I-WAY software, with greater focus on protocols for fundamental tasks such as resource discovery, job submission and data transfer. Later versions of the Globus Toolkit are used as building blocks in many of todays Grids. Another early general-purpose Grid toolkit is Legion [98] that models and controls remote resources in an object-oriented manner. Further examples of early Grid projects include the Storage Resource Broker [22] that enables uniform access to heterogeneous storage resources; AppLeS [35] and NetSolve [34], both application-level schedulers; and Cactus [11], a programming environment for parallel high performance computing on various platforms, including Grids. An in-depth description of early Grid computing projects is beyond the scope of this thesis, and can be found in overview books [77, 27].

1.2 Characteristics of Grid Computing

Early Grid computing efforts focused on uniform, efficient and secure access to computational resources, despite heterogeneity in ownership, security mechanisms, and policies. An early definition states that Grid computing is "Coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations" [82]. In Grid computing, the term Virtual Organization (VO) is used to describe the parties sharing resources. The formation of a VO is motivated by a scenario where "a number of mutually distrustful participants with varying degrees of prior knowledge (perhaps none at all) want to share resources in order to perform some task" [82].

One commonly used definition is the three point check list [74] by Foster that defines a Grid as a system that:

1. coordinates resources that are not subject to centralized control,
2. uses standard, open, general-purpose protocols and interfaces,
3. delivers nontrivial Qualities of Service.

A more recent definition is given in the Open Grid Services Architecture (OGSA) glossary of terms by the Open Grid Forum (OGF). Here, a Grid is defined as "A system that is concerned with the integration, virtualization, and management of services and resources in a distributed, heterogeneous environment that supports collections of users and resources (virtual organizations) across traditional administrative and organizational domains (real organizations)" [59]. Although this definition is in large in harmony with the one by Foster, a few things are worth noting. First, there is a clear focus on exposing the capabilities of the Grid as services. This is a result of the recent trend of usage of service-oriented architectures in Grid computing, a topic further discussed in Section 6.1. The use of virtualization in the OGF definition should not be confused with virtualization by use of virtual machines (further explored in Section 8.2), but is rather the process of hiding differences in properties and operations of a set of similar resources and making these available for view and/or manipulation through a set of common interfaces [59].

Another recent Grid definition comes from the CoreGRID network of excellence that describes a Grid as a "fully distributed, dynamically reconfigurable, scalable and autonomous infrastructure to provide location independent, pervasive, reliable, secure and efficient access to a coordinated set of services encapsulating and virtualizing resources (computing power, storage, instruments, data, etc.) in order to generate knowledge" [47]. In the CoreGRID definition, the quality of service requirements (e.g., scalability, security, reliability) are made more explicit. Furthermore, the goal of the architecture has a clear focus on generation of knowledge, clearly inspired by the emergence of the e-Science discipline.

John Taylor defines e-Science as follows: "e-Science is about global collaboration in key areas of science and the next generation of infrastructure that will enable it" [199]. In e-Science, there is an inherent focus on scientific data. Large amounts of data are processed and analyzed in a collaborative fashion by distributed teams of scientists. This requires access to high-end computing resources, distributed data storage, and visualization equipment, as well as the software tools required by the scientists to cope with exponentially growing amounts of data. Grid computing can hence be seen as an enabling technology for e-Science [103].

The research presented in this thesis is well aligned with the above definitions of Grid computing. The Grid scenarios addressed stress that resource management tasks must be performed despite lack of global control. One main objective is to demonstrate how open, standard protocols and interfaces can be used to achieve interoperability between different Grids. The thesis also investigates various issues for Quality of Service (QoS), with particular focus on deadline constraints.

1.3 Fundamental Grid Capabilities

Many of the existing Grid infrastructures have fundamental differences in purpose, architecture, and type of resources that they interconnect. However, almost all Grids have: a model to describe resource capabilities, mechanisms to discover available resources and monitor known ones, resource allocation interfaces, data management functionality, and a security infrastructure. General discussions of Grid capabilities are found, e.g., in [84, 193]. The following descriptions focus on capabilities required by resource selection and job management systems. Most Grid infrastructures provide the discussed capabilities, although the used protocols, interfaces, and software components sometimes have fundamental differences.

1.3.1 Modeling, Discovery, and Monitoring of Resources

In all Grids, there is a need to describe the characteristics of the available resources using a general and extensible mechanism. Up to date information with sufficient level of detail is key to determine whether a given resource fulfills the requirements of a user. With a common information model in place, end user tools can focus on further information management issues such as to discover what resources are available and to monitor known resources. Discovery of available resources is normally performed by contacting an information index. Complete information about resources can either be aggregated into the index, or kept in a local information service on the resource itself. In the latter case, the index contains information about how to contact the local information service on the resource. In a typical resource discovery scenario, a client retrieves information about the set of currently available resources from an index, queries the most interesting resources for more detailed information, and selects which resource to use based on the retrieved information. A naive way to perform monitoring is to periodically query the monitored resources for new information. Mechanisms to asynchronously notify interested parties with information updates are however preferred. Such notification mechanisms can reduce the number of messages significantly, allowing clients to retrieve updates as these are generated without resorting to extensive polling. While discovery includes the retrieval of both static and dynamic information, monitoring is exclusively used for dynamic resource information. A typical use case for resource monitoring is a client that supervises the progress of a task performed by a resource. The potentially very large number of users and resources in a Grid makes scalability and responsiveness important criteria for information management systems. For discovery, scalability is the main issue, as an index may have to serve a large number of clients. Responsiveness is more important for monitoring than for discovery, as up-to-date information is vital in some monitoring scenarios, e.g., fault detection.

Information management in Grids is complicated by the fact that information is more or less always old as the state of a Grid resource may change rapidly. The asynchronous communication mode of the general purpose Internet typically used to interconnect Grid resources makes it impossible to maintain an updated view of state in remote resources. Resource availability also varies, as resources at any time may join, or due to policy reasons or unforeseen events, disappear from, the Grid. To avoid stale index entries from no longer available resources, information indices should be self-cleaning via soft-state registrations. In addition to always being old, information is often incomplete, as resource owners are free to choose what resource information they publish for public access in a Grid. Resource information is hence not to be trusted blindly, as the presence of information about a resource in an index neither guarantees the availability of the resource, nor guarantees that a particular user is authorized to use it.

The existence of several VOs in a Grid results in a variable grouping of information with multiple, overlapping indices instead of one well-structured, hierarchical index structure containing all available information. Most indices contain general information about a subset of the resources in a VO, although specialized indices can be used, e.g., for keeping track of available storage locations and the amount of free space on each. Specialized indices can help to reduce the load on general purpose ones. It is however hard to anticipate every type of client information request, and it is not feasible to construct a large number of special-purpose indices. For performance reasons, users and other information consumers should instead be able to specify and limit the information they are interested in retrieving, e.g, through a query language such as XPath [45] or SQL [112].

1.3.2 Resource Allocation

In order to use a given Grid resource efficiently, mechanisms are needed to allocate the resource, to control and monitor the usage of it, and once done, to release the allocated resource. As computational resources are the ones most commonly used in Grids, the rest of this section focuses on these, and the computational tasks (henceforth called *jobs*) that they execute. The described scenario focuses on the basic capability to execute a job on a predefined resource. Higher-level tasks, such as selection of which resource to run the job on, are discussed in Chapter 2.

There are many problems that need to be solved in order to provide high-quality job management mechanisms, both from a user and resource owner perspective. Users want a simple, secure, and efficient interface to initiate, monitor, and control jobs on a remote resource. Most existing Grid toolkits offer a set of basic functionalities that (partly) fulfills these requirements. A job description language allows users to express job configuration, e.g., the executable to run, job input and output files, as well as requirements on the resource executing the job, e.g., hardware architecture, amount of memory, and operating system. Job execution mechanisms typically provide an abstraction layer that hides the heterogeneity of the underlying execution platform. Batch system schedulers such as LoadLeveler [115], LSF [222], and PBS/Torque [110] are examples of such platforms. Other used execution backends include Condor [200], a loosely connected pool of machines available to Grid jobs when otherwise idle, and execution of the job on the frontend machine itself through the POSIX "Fork" command. Most job management mechanisms also define a state model for a computational job. Typical job states include "pending", "running", "finished", and "failed".

From a resource owner perspective, authentication and authorization mechanisms are needed to be able to control which users that may access the resources. Resource owners also want to control the environment a Grid job is executed in, e.g., via sandboxing techniques[1]. Other resource owner requirements include functionality for auditing and accounting, i.e., tracking who is using the resource, for what, at what time, and in what quantities.

More intricate aspects of resource (job) allocation such as interoperability of job submission systems and the associated standardization process, as well as negotiation of terms of use are discussed in Chapter 5 and Section 2.2.2, respectively.

1.3.3 Data Management

Many Grid applications require secure and efficient access to data that is stored distributed across a Grid. The GridFTP [61] protocol is the de-facto standard for transfer of data. GridFTP extends FTP, e.g., with authentication on both the data channel and the control channel, based either on the Grid Security Infrastructure [217] or Kerberos [152]. GridFTP contains several performance

[1]Sandboxing mechanisms provide a limited and secure execution environment for a not fully trusted application, in this case a Grid job. According to Thain et al. [200], sandboxing techniques must provide both the box, i.e., the protection mechanisms, and the sand, i.e., an execution environment as suitable as possible for the requirements of the application.

improvements over the original FTP protocol, including parallel transfers, i.e., the usage of multiple TCP data streams between two endpoints, and striped transfers, i.e., usage of data streams from different endpoints, both on the sender and receiver side. Other functionality improvements are restartable transfers and transfers of partial files. Third-party transfers (server-to-server transfers) allow a client to transfer files between two servers without acting as an intermediate proxy for the data channel. GridFTP also specifies both an option for automatic TCP buffer size negotiation and protocol messages for explicitly setting the buffer size.

In addition to the actual transfer of data, current Grid infrastructures typically contain higher-level data management capabilities. Reliable transfer of files is achieved by tracking progress of transfers and restarting these upon failure, as done e.g., by Globus RFT [8] and Stork [121]. The motive behind *data virtualization* is to decouple the file identity (the logical file name) from the location(s) of a file (the physical file name(s)). This is typically achieved by the usage of a catalogue, e.g., CASTOR [38] or Globus RLS [42], where mappings between logical and physical file names are stored. Virtualization of data enables replication [39, 64, 124], i.e., distribution of multiple physical copies of the same logical file across the Grid. Replication can improve both performance [120, 172] and fault tolerance [125, 130].

1.3.4 Security

As the resources used in a Grid typically are valuable and the data transferred between the resources may be confidential, security is an important aspect of Grid computing. New challenges arise in Grid security as the interactions between user tools and resources are more complex than the traditional client-server model. Grid security is further complicated by the fact that resources that belong to different administrative domains (trust domains) interact, each domain having different security policies and using different mechanisms to implement the respective policies.

Most Grid security scenarios are covered by three fundamental computer security concepts, often abbreviated AAA [212]: *Authentication* establishes the identity of other entity, in the Grid case, typically a user or a resource; *Authorization* concerns the privileges (access rights) to a particular entity (resource); and *Accounting* includes the control, monitoring, and metering (potentially including billing) of resource consumption.

From a user perspective, ease of use is key for a Grid security toolkit. A *single sign-on* mechanism allows users to authenticate themselves only once, instead of having to manually repeat the authentication procedure for each resource they interact with. The requirement for delegation of access rights arises from the often complex interaction pattern between users and resources. Through delegation, a user can grant a resource the permission to perform operations on behalf of the user. For resource owners it is vital that the Grid security mechanisms are easy to integrate with the local security infrastructure used within the administrative domain. Resource owners also need mechanisms to track access to the resource, i.e., metering and accounting. Flexibility is the key issue for Grid application developers. A versatile API for authentication, delegation and similar tasks enables developers to cope with the complexity of the interactions between applications and Grid resources.

A commonly used security mechanism is the Grid Security Infrastructure (GSI) [81, 217]. GSI uses a public key infrastructure with X.509 certificates [105] and supports TLS (SSL) [57] for secure communication. Delegation and single sign-on are handled through proxy certificates [206], often simply referred to as proxies. A proxy certificate is valid for a short period of time only, typically a few hours.

1.3.5 A Basic Job Submission Scenario

A typical interaction between a user client and a set of Grid resources is illustrated in Figure 1. In this scenario, the user initially sends a query to the index to discover what resources are available. As described in Section 1.3.1, the configuration of the index determines whether the user retrieves all information available about the registered resources, or only references to other information sources that describe the resources in more detail. In the latter case, the user has to perform subsequent queries to find out more detailed information about the resources. After retrieving resource information, the user selects which resource to use and then submits the job to the selected resource by one of the job execution mechanisms described in Section 1.3.2. Finally, the user ensures that the job input files (including the executable) are transferred to the selected resource. This last task is either performed from the user client host via direct transfer, or, as illustrated in Figure 1, by the resource through third-party transfer. The presence of higher-level data management capabilities such as reliable file transfers and replication can improve both the performance and the fault tolerance of this part of the job submission process. Security mechanisms can be involved in all of the these tasks, including authentication of the user by the resource, authorization of the user's permission to execute the job, and delegation of the user's credentials to enable the resource to download job input files.

1.4 Outline

As described in Section 1.3.5 and illustrated in Figure 1, the basic Grid capabilities for management of information, jobs, data, and security, all provided by contemporary toolkits, can be used to perform most tasks in a basic job submission scenario. However, real-life Grid usage scenarios require capabilities beyond the fundamental ones described in Section 1.3. This thesis investigates topics that extend on the basic job submission scenario. The outline of the rest of this introduction is as follows. Chapter 2 describes architectural models

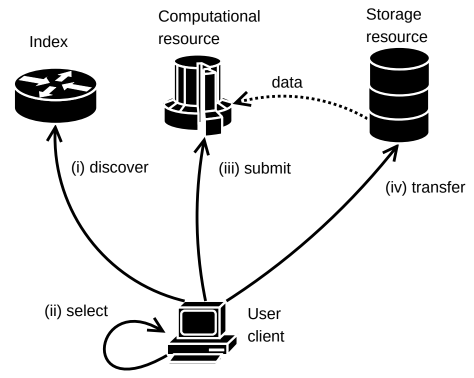


Figure 1: Component interactions in a basic job submission scenario.

and algorithmic considerations for resource selection, topics studied in papers I, II, and III. In Chapter 3, issues related to submission, monitoring, and fault handling of large numbers of independent jobs are studied. Frameworks for such job management capabilities is the topic of Paper IV. Chapter 4 describes workflows, i.e., sets of jobs with internal execution order dependencies. Chapter 4 also discusses issues related to composition of workflows, resource selection for jobs in a workflow, and workflow execution. Papers V, VI, and VII study a wide range of topics related to the composition, scheduling, and execution of Grid workflows. Chapter 5 describes interoperability problems in current Grid infrastructures and the related standardization efforts. This thesis addresses interoperability problems at several levels, e.g., at job submission level in papers I and II, job management level in paper IV, workflow execution level in paper V, and also from a more conceptual perspective for workflows in paper VII. Some design considerations for Grid software are given in Chapter 6, including discussions of service-oriented architectures, Web services, and design heuristics. A more in-depth analysis of these topics is found in Paper VIII. The final part of this introduction consists of a summary of the papers in the thesis (Chapter 7), an outline of potential future work (Chapter 8), and a bibliography, respectively.

Chapter 2

Grid Resource Brokering

The selection of which Grid resource(s) to use for a specific application is unfeasible even for the educated user due to the large number of resources, their fluctuating availability, differences in resource hardware, software, access policies, etc. A resource broker is a tool that automates and improves resource selection, or more commonly, the whole job submission process, for the user. A Grid resource broker is sometimes referred to as a meta-scheduler, as it selects which local scheduler (which local resource) to interact with.

2.1 Brokering Scenarios

There are two main Grid resource brokering scenarios. In the centralized scenario, all access to Grid resources is controlled by one broker. A centralized broker has good knowledge of, and control over, all submitted jobs and can, via load balancing techniques, produce good schedules. One obvious drawback of this type of broker is the potential performance and scalability bottleneck and that a centralized broker is a single point of failure. Another shortcoming of the centralized brokering approach is that it is hard to introduce dynamic policies, e.g., user-specified resource selection algorithms, in a centralized system. Furthermore, despite having complete knowledge about all the Grid jobs, a centralized broker cannot produce completely reliable schedules as the ultimate control over the resources remains in the hands of their respective owners.

The alternative approach is a decentralized (distributed) brokering architecture, where individual users have their own resource brokers. This type of broker typically manages only a fraction of the total number of jobs submitted to the Grid, and can hence not (alone) perform load balancing. Advantages of the distributed brokering approach include scalability and fault tolerance. A decentralized brokering architecture also enables customization, as each individual broker can be tailored to the specific requirements of an certain user or application. Centralized and distributed brokers, as well as hybrid brokering approaches with a hierarchies of brokers, are further discussed in [122].

A centralized broker is typically used to schedule jobs to a specific set of resources, often belonging to the same VO. A decentralized broker on the other hand, is most often not tied to certain resources, but allows the user to specify for each job request which resources the broker should consider. There is however no fixed hierarchy of resources as some resources may be accessible to users from different VOs, possible even through different Grid middlewares. Such resources can be seen as belonging to multiple Grids.

The scheduling policy used by a resource broker can be either system-oriented or user-oriented [122]. The goal expressed in a system-oriented scheduling policy can be to maximize resource utilization, load balance, fairness, or a combination of these. System-oriented scheduling policies are commonly used by centralized brokers. A distributed broker most often strives to maximize a user-oriented scheduling policy, typically by improving the throughput or response time for jobs submitted by the individual user, regardless of the impact on the overall Grid performance and at the expense of competing users.

This thesis investigates the decentralized Grid brokering problem. The varying characteristics of different applications make resource selection a problem that must be solved on a per application basis. The problem is further complicated by heterogeneity in resource hardware, software and usage policies. Since a decentralized broker operates without global control, it must base all its decisions on information about, and negotiation with, the resources, and not on control over them. As discussed in Section 1.3.1, information gathered about the state of the available resources is often incomplete, as resources may limit the published information due to misconfiguration or policy reasons. Information is typically also outdated, as the current load of a resource may change at any time, making Grid resource brokering an online scheduling [166] problem. A decentralized broker may serve more than one user, but typically handles every user in isolation in such a scenario, in effect giving each user a personal broker. Users of a decentralized broker have to compete for resources with other users of the same broker, with users of other Grid brokers, and, as most resources are not dedicated to Grid use exclusively, with users accessing the resources through local, non-Grid interfaces.

All attempts to perform Grid resource brokering should consider the degree of transparency. Users want transparent access to the resources, but some issues, such as the resource selection criteria, typically requires some user involvement. A flexible mechanism that enables users to express their different resource requirements is of particular importance for a decentralized broker with a user-oriented scheduling policy, as user satisfaction is the main goal for this type of broker. A user's typical resource requirements for a computational job include hardware architecture and the number of CPUs of the resource, as well as the amount of main memory and secondary storage available to the job. Further resource requirements include operating system; installed software, including availability of licences for commercial software toolkits; capacity of the network connecting the resource to the Grid; and available QoS guarantees, such as job start and/or completion time.

2.1.1 Approaches to Brokering

Figure 2 illustrates a decentralized brokering scenario with a set of computational Grid resources. Each resource uses a local batch system scheduler to plan and manage the execution of jobs submitted to that resource. The local schedulers control the backends that execute the jobs, see Section 1.3.2 for more details. Also shown in Figure 2 are two indices that store Grid resource information. The dotted arrows in the figure illustrate how resources register information about themselves in the indices. Resource information queries from brokers to the indices are shown as dashed arrows. For clarity, the two information indices aggregates all available resource information and no information queries are hence sent from the brokers to the resources. The two rectangularly shaped brokers in Figure 2 each serves a small set of users. The other type of broker, illustrated as Broker/Client in the right-hand side of the figure, is integrated into an application and is hence used by the user(s) of that application only. The solid arrows in Figure 2 illustrate job requests, either sent from the users to the brokers, or from a broker to a resource. Data transfers are omitted from the figure for clarity.

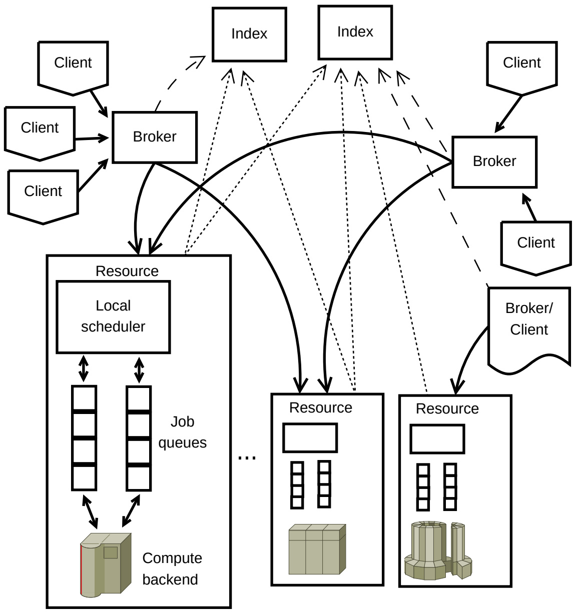


Figure 2: Overview of components and interactions in a decentralized brokering scenario.

General brokering discussions in the literature include "Ten actions when

Grid scheduling" [181], where Schopf discusses the eleven (!) steps in the job submission process. Pugliese et al. [168] analyze the requirements of Grid resource management and classify schedulers according to scheduling policy as well as the type and scope of resources they manage. Work by Vázquez-Poletti et al. [209] compares the centralized scheduling paradigm of EGEE WMS [63] with the decentralized one of GridWay [109].

Examples of resource broker software include EMPEROR [4], a meta-scheduler with a framework for implementing performance (prediction) based scheduling criteria. EMPEROR utilizes time series analysis to predict, e.g., job execution time and resource utilization. The composable ICENI [228] Grid scheduling architecture supports multiple scheduling algorithms, including random, simulated annealing, best of n random, and a game theoretic approach. The eNANOS Grid resource broker [176] supports submission and monitoring of Grid jobs and enables users to customize resource selection by weighting the importance of resource attributes such as CPU frequency and main memory size. Chapman et al. [40] present a Grid scheduling framework that uses prediction theory, in particular Kalman filters, to minimize response time for Grid jobs. A theoretical study of the Grid scheduling problem and a comparison to methods applied in multiprocessor systems is given by Schwiegelshohn et al. [182].

An alternative approach to resource selection is taken in market-based Grids, where participants trade resource shares in artificial markets. Claimed advantages of the marked-based approach are increased resource utilization and better load balancing, both a result of the supply and demand equilibrium that is predicted to occur in an economic system [221]. Users can furthermore assign priorities to their jobs by adjusting the budget allocated to each job. Examples of market-based mechanisms for Grid computing include preallocation of artificial Grid credits [87] and systems where resource consumption is based on real economic compensation [12, 107]. The pricing mechanism in a market-based Grid is often implemented either as a commodity market or as an auction. Additional models for market-based resource allocation exist and a taxonomy of these as well as a discussion of market-based systems can be found in the work by Yeo et al. [225].

Although the market-based approach may seem fundamentally different from resource selection algorithms used in other Grid systems, the task of a Grid resource broker remains much the same. The broker still has to identify, characterize, negotiate, select, etc. resources to solve the decentralized Grid brokering problem. The additional constraint in a market-based Grid is that the resource selection algorithm also has to consider price as a parameter. Instead of requesting the "best" resource, a user may prefer to use the best resource that fulfills certain budget constraints, i.e., resource selection becomes a tradeoff between cost and performance. This issue and other resource selection considerations are further discussed by Yahyapour [223].

2.2 Tasks of a Grid Resource Broker

As outlined in Section 1.3.5, the tasks performed by a Grid resource broker are discovery of available resources, selection of which resource to use, invocation of the job request operation of the chosen resource, and transfer of the job input files to the resource. Whereas most of these tasks are straight-forward, resource selection is a complicated process. In overview, resource selection consists of three phases. First, the list of discovered resources is filtered. In this step, resources are removed if they do not fulfill the user's requirements for hardware, software, etc, or if the user is not allowed to use them. After the initial filtering follows two dependent and often interleaved tasks, prediction of resource performance and negotiation of terms of use. Performance prediction includes estimating the characteristics of both the application and the resources considered. Resource heterogeneity results not only in performance differences between Grid resources, but also means that the relative performance characteristics may vary for different applications. This complicates predictions of job performance. Nevertheless, good performance estimates are of great value, e.g., during negotiation of terms of use. Such a negotiation of QoS terms result, when successful, in the creation of a Service Level Agreement (SLA) between the broker and the selected resource. Accurate performance predictions are also useful in cases when no performance guarantees are available. Various perspectives of performance prediction and SLA negotiation are discussed in the following sections.

Optional tasks for a broker include to monitor the job during execution and job clean up tasks. Jobs may also be migrated, e.g., for performance reasons, and resubmitted to alternative resources upon failure. Other commonly used features include management of sets of individual jobs and coordination of jobs with internal dependencies, the latter also known as workflows. Job management and workflows are further discussed in chapters 3 and 4, respectively.

2.2.1 Performance Predictions

An accurate estimate of job performance serves multiple purposes. During SLA negotiations, knowledge of application performance helps avoiding requesting more resource capacity than required, and can hence reduce response time, cost, and/or other scheduling criteria. Performance prediction techniques are also useful when no SLA negotiation occurs between a resource broker and Grid resources. In this case, job placement decisions are based exclusively on the (predicted) performance of jobs on the considered resources.

Research on performance prediction often focus on models for estimating various parts of the job lifecycle, including execution time and time spent waiting in a batch queue for access to the resource. This is a well studied problem for traditional batch system scheduling [187, 204]. In Grid environments, performance prediction is complicated by the existence of multiple heterogeneous machines, and the fact that the considered job need not have executed on all of these before. One way of classifying performance prediction methods is to divide them into statistical methods that uses data gathered from previous executions [6, 116, 127, 187, 204] and heuristics-based methods that take into account job and resource characteristics [99, 128, 216].

Krauter et al. [122], divide performance prediction methods into predictive and non-predictive ones. The predictive methods, which include pricing models and machine learning, take historical use in consideration. Non-predictive methods include probability distributions that do not make use of historical data. In the taxonomy by Krauter et al., heuristical methods can be either predictive or non-predictive. Alternative classifications of performance prediction methods include work by Smith [185] that discusses statistical and analytical models. The statistical models are based on analysis of previously completed applications and equations are used to model execution time. The analytical ones are either derived by hand or by automatic code analysis or instrumentation [185]. Smith further describes two statistical run time prediction methods, investigates how to predict batch queue waiting time, and discusses scheduling methods that utilize run time predictions.

Ali et al. [7] analyze how to model execution time on heterogeneous computing systems and discuss the minimization of performance metrics such as job start time and job completion time. They present a model for expected execution time that takes into account heterogeneity of machines and tasks, as well as consistency, the latter defined as whether a given machine is faster than another one for all types of tasks. In [91], Goyeneche et al. evaluate the accuracy of current data mining and statistical methods for performance prediction based on application similarity classifications. Based on their findings, they propose a prediction mechanism with weighted templates that, after initialization, prediction, and incorporation of historical information, gives run time predictions with corresponding confidence intervals [91].

Smith [185] describes two techniques for batch queue wait time prediction. The first is based on run time predictions and simulates commonly used batch scheduling algorithms to transform the run time prediction problem into a queue time prediction one. The second method by Smith is history-based and uses application similarity categorization techniques to classify the scheduler and the application. Li et al. [127] apply local learning techniques [18] to queue time predictions. Evaluations using workflow traces from large clusters demonstrate that the local training techniques by Li et al. are more efficient than global and adaptive ones.

A prediction method that takes into account file transfer times, batch queue waiting time, and application execution time is suggested and evaluated by Smith [186]. The proposed method uses instance-based learning and is based on historical information.

2.2.2 Negotiation of QoS Terms

The term QoS refers to guarantees (beyond best effort) for the performance of a service. This covers virtually all aspects of service delivery (in this case job execution), including availability of the service, amount of resources allocated to the service request, compensation for the consumed resources, agreed levels of security, risk of failure, response time, throughput, etc. The essential mechanism to guarantee QoS terms is a priority mechanism that enables differentiation between consumers of a service. To support QoS in a general infrastructure, the following components are needed: a language to describe QoS terms, a protocol that consumers can use to negotiate a SLA with a provider, a mechanism to model the agreed upon SLA, and functionality to monitor the SLA for violations.

A language for describing QoS terms is by necessity domain dependent, e.g., bandwidth for networks, as defined by the Resource ReSerVation Protocol (RSVP) [31]. For computational jobs, typical QoS terms include amount of memory allocated to the job and number of CPUs, as well as job start and/or completion time. It is also useful to be able to express under what restrictions the terms apply, e.g., a resource will promise to complete a job before a certain deadline only if the user submits the job within a given time.

Various types of negotiation protocols have been studied within the area of agent-based computing. A framework for auction protocols is suggested by Chard and Bubendorfer [41]. Bai et al. [21] propose a three actor model with providers, brokers, and consumers. They evaluate an economic brokering algorithm with respect to resource utilization, consumer satisfaction and provider revenue. A game-theoretic approach is suggested by Khan et al. [207]. This work studies utilization, fairness, completion time, and request rejection rate for various cooperative and non-cooperative resource allocation methods. Venugapol et al. [210] propose a bilateral negotiation mechanism that, in addition to accept and reject messages, also allows the negotiating parties to express counter offers. Simulations confirm that this approach is beneficial for brokers that accept reservation start time delays. In context of the NextGRID project [46], Hasselmeyer et al. [101] discuss how to negotiate SLAs based on business-level objectives instead of details of the hardware used to deliver the service. For example, a user can select, e.g., from the “gold”, “silver”, and “bronze” service levels instead of specifying detailed XML descriptions of the requested resources.

A proposed standard that has received attention the last few years is WS-Agreement [14] that defines mechanisms to model and monitor agreements between a (Web service based) agreement provider and an agreement initiator. Seidel et al. [183] survey Grid resource management projects that utilizes WS-Agreement to model SLAs, typically to implement start time guarantees. A WS-Agreement-based negotiation protocol is proposed by Siddiqui et al. [184]. Here, resource allocation is modelled as a strip packing problem. The negotiation protocol is based on a three-layered approach, with allocators for single requests, coallocators that coordinate requests from the same application, and coordinators that resolve conflicts between coallocators.

Advance reservations. Of particular interest for this thesis is response time related QoS aspects. In the batch system schedulers used in current Grid infrastructures, advance reservations is the only mechanism available to guarantee job start and/or completion times. An advance reservation is an assurance for the consumption of a certain amount of resources at a specified time in the future. However, such a guarantee may be cancelled due to resource failure or policy reasons, such as when a higher priority activity (typically another reservation) causes the reservation to be preempted. As advance reservations most often are given very high priority, the latter case is unlikely to occur. There are multiple uses for advance reservations, including time-critical tasks that must meet a deadline, which would be impossible without a start time guarantee. Further examples include reducing job start time uncertainty [141], demonstration purposes, debugging, and other interactive use, when access to the resource at a known time is critical.

Advance reservations also enable the job to be synchronized with other activities, which is essential for coallocation and workflows. The usage of advance reservations for computational resources however reduces resource utilization [138, 188, 189]. This reduction can be explained by increased fragmentation of batch queues that in turn reduces the efficiency of the backfilling scheduling algorithms used in current batch systems. Castillo et al. [37] use methods from computational geometry to tackle fragmentation. Another possibility to achieve good performance despite the use of reservations is to allow a certain degree of laxity in reservation start times [71]. However, current batch system schedulers lack the mechanisms required to implement such reservation rearrangements [134]. Qu describes an architecture [169] where advance reservations can be offered to Grid jobs regardless of whether they are supported at the local batch system level. This is achieved by adding a reservation management layer in the Grid scheduler. However, this solution assumes that no jobs are submitted to resources through non-Grid interfaces or by other resource brokers.

Coallocation. Another QoS related topic studied in this thesis is coallocation, i.e., the coordinated allocation of a set of resources to be used together for solving a problem. A typical use case for coallocation is when multiple parallel computers are used to execute a job that communicates not only between the CPUs in one cluster, but also across different machines, using e.g., the Message Passing Interface (MPI) [145]. A more complex scenario is the concurrent usage of instruments and computers to in real-time gather, analyze, and possibly visualize, scientific data from an experiment.

In order to provide a coallocation mechanism, a resource broker must solve two problems. The first problem is the selection of which set of resources to use. This is a non-trivial selection procedure as the set of resources suitable for the subjobs forming the coallocated job may overlap, as happens, e.g., in the MPI-scenario where multiple instances of an application must execute on a set of (binary compatible) machines. The second problem is to ensure that the selected resources have a coordinated (most often common) start time. This is typically guaranteed through the use of advance reservations. In addition to the two problems related to the selection and allocation of the resources, a coallocation scenario also requires initialization and synchronization of the coallocated subjobs. For many applications, e.g., jobs that use cross-cluster MPI, each subjob must be aware of, and able to communicate with, the other subjobs. Preferably, this functionality should be as transparent as possible to the application. For the MPI scenario, Coti et al. [48] suggest a solution that uses connectivity services to hide the complexity of communication.

The work by Czajkowski et al. [50] addresses the application initialization problem by defining a library for initiating and controlling coallocation requests and another library for application synchronization. By compiling an application that requires coallocation with the application library, the subjob instances can be instructed to wait for eachother at a barrier prior to commencing execution. A similar project is the Globus Architecture for Reservation and Allocation (GARA) [80] that provides a programming interface to simplify the construction of application-level coallocators. GARA can perform both immediate reservations (allocations) and advance reservations of networks, computers, and storage.

Attempts to solve the second coallocation problem, start time coordination, include the KOALA system [147] that uses a mechanism for implementing coallocation without using advance reservations. This is achieved by requesting longer execution times than required by the jobs, and delaying the start of each job until all allocated jobs are ready to start executing. Another contribution where coallocation is achieved without the use of advance reservations is [19], where Azzedin et al. propose an algorithm based on synchronous queueing. MacLaren [135] treats coallocation as a transaction problem and uses the PAXOS commit protocol to ensure consistency. This protocol is based on messages that create, modify, and cancel reservations. Mateescu [139] defines an architecture for coallocation based on Globus Toolkit 2. The suggested coallocation algorithm uses a window of acceptable job start times and tries to reserve all required resources at predefined positions in this start time window. Decker et al. [51] describe another window-based coallocation algorithm that by considering the execution and communication times of a set of dependent tasks tries to minimize the overall completion time. The coallocation algorithm suggested Wäldrich et al. [215] models reservations using the WS-Agreement framework and uses a concept of coallocation iterations. In each iteration of the algorithm, a list of free time slots is requested from each local scheduler. Next, an off-line matching of the time slots with the resource requests is performed. If the complete coallocation request can be mapped onto some set of resources, reservations are requested for the selected slots. Röblitz et al. [175] describe a coreservation architecture where resources are matched to coreservation requests in a off-line style similar to the algorithm by Wäldrich et al. [215]. The coreservation algorithm by Röblitz et al. can handle both temporal and spatial dependencies between requested resources. Netto et al. [151] discuss a coallocation algorithm based on malleable (flexible) reservations and processor remapping. In this work, coallocation requests can be moldable, i.e., the number of requested resources can change. Individual resource requests can also change in terms of request size and/or start time, all in order to achieve the earliest possible start time. The actual assignment of resources to requests is performed with an off-line matching algorithm. A coscheduling (coallocation) mechanism inspired by the coordinating tasks performed by a travel agent to guarantee the availability of a multi-resource itinerary is suggested by Yoshimoto et al. [227]. In ASKALON [184], coallocation is modelled as a constraint-satisfaction-problem. An on-line coallocation algorithm described by Castillo in her PhD thesis [36] makes use of tree structures to handle free time slots at the resources and decides which resources to reserve by traversing the trees.

2.3 Non-goals of a Decentralized Broker

In the discussion of the tasks of a decentralized broker, it is useful to explicitly state some related tasks not performed by this type of broker, and discuss why these tasks are best handled by other components.

A good distribution of the load over the resources in a Grid is important for both achieving good performance and utilizing the resources efficiently. A decentralized resource broker that only handles a small fraction of the total number of submitted jobs cannot alone achieve good load balance. However, as decentralized brokers typically seek to minimize job response time, each broker contributes to the overall load balance of the Grid by avoiding the most heavily loaded resources.

One capacity allocation mechanism commonly used in scientific collaborative Grids is policy-driven preallocation of resource shares based on scientific impact. Fairness in this type of Grid can for a given user be described as the difference between the user's historical, and possible also current, resource usage and the preallocated amount of resources the user is entitled to [66]. A decentralized resource broker does not necessarily try to enforce fairness. On the contrary, the broker could be considered successful if it manages to allocate more resources than the user is entitled to. Fairness should hence be enforced on the resource side [66], not by throttling mechanisms in the broker. A topic related to fairness is accounting [87], i.e., book keeping the amount of resources consumed by a user. As fairness, possible payment, etc. typically are determined from accounting information, accounting should be the responsibility of the resource provider, not the consumer (i.e., the broker). A user trying to circumvent accounting (and hence also payment and fairness) could otherwise use an alternative broker that does not report resource consumption truthfully.

2.4 Contributions to Grid Resource Brokering

The results related to resource brokering in this thesis is the definition of the total time to delivery for a Grid job and an associated unified model for resource selection that can utilize, but do not depend on, a wide range of prediction techniques as well as SLA negotiation mechanisms. This work also demonstrates how advance reservations can be used both to meet hard deadline constraints and reduce uncertainty in resource selection. Algorithmical contributions in the thesis include advances to the state-of-the-art in coallocation.

Chapter 3

Grid Job Management

Although a Grid resource broker handles the perhaps most cumbersome operation - resource selection, it is not a complete turnkey solution for Grid enabling an application. Other capabilities typically required by Grid application developers include monitoring of submitted jobs, resubmission of failed jobs, and potentially also migration of jobs from slowly or non-responding resources. Complex applications such as computational steering and interactive visualization need, in addition to job monitoring, also the ability to interactively steer the job during its execution.

One common problem type that is particular suitable to Grid environments is the parameter sweep study, where large numbers of jobs without internal dependencies are executed, typically to study a problem for different input parameters. Parameter sweep applications require tools that simplify the management, and execution (as well as reexecution) of large groups of jobs. Performance critical applications may need performance-aware job monitoring, and also a framework that automatically detects performance anomalies and takes appropriate actions, e.g., submits the job to an alternative resource, or, if possible, migrates the running job. Performance aware job migration requires both a checkpointing mechanism and a method to communicate application-specific performance metrics from the application to the Grid middleware. As both checkpointing and performance metric reporting necessitate application modification, some users are reluctant to use these features.

Some resource-demanding users use the Grid as their daily production environment for running very large numbers of computationally intensive applications. These users would benefit from having their own personal job queues, where jobs can be added for later submission to the Grid. To avoid overloading the Grid resources, user job queues typically include some sort of backoff functionality. Ideally, such functionality should not be required. However, the job submission pattern of some users shows a temporal and spatial burstiness [129], and jobs failure due to resource overload is hence not uncommon [129]. Personal job queues also enable inter-job priorities for jobs that belong to the same user. This allows users, to some extent, to control the order in which their jobs are executed. The exact job execution order is however determined by batch system scheduling algorithms and the user can hence not control the internal execution order of jobs after they are submitted to batch queues.

3.1 The Role of Job Management Frameworks

Many Grid application developers implement their own set of tools for the above described functionalities. This approach may at a glance seem attractive as it potentially can save time if only trivial job management functionality is needed, but typically results in non-reusable software due to too strong coupling to the application. Furthermore, the built-in heterogeneity of Grid infrastructures and their error-prone nature necessitate middleware-independent job management tools and robust fault tolerance mechanisms that are non-trivial to design and implement. Preferably, resource brokering and job management capabilities as those described here should be gathered in well defined APIs, or as argued in Section 6.1, exposed as services.

3.2 Contemporary Job Management Tools

The construction of general toolkits for Grid job management is a large area of active research. Casanova et al. [35] describe a user-level middleware with an integrated scheduler (resource broker) that simplifies the execution of parameter sweeps on a Grid. The scheduling algorithm used combines Gant charts with various heuristics for estimating and minimizing completion times of jobs as well as of file transfers. The Nimrod-G [2] resource broker consists of a task farming engine, a scheduler (for resource discovery, trading, and scheduling), dispatchers and actuators (for interfacing different Grid middlewares), and agents for managing job execution, e.g., setting up the job environment on resources. Nimrod-G supports parameter sweeps and master-worker style applications. The goal of the GridWay [109] framework is simplified and more efficient job execution in a "submit and forget" fashion. The GridWay resource selection algorithms strive to minimize the job completion time. In GridWay, a Grid-aware application contains both a performance profile and a restart file, the latter used for user-level checkpointing. GridWay supports job resubmission, which can be initiated by failures, user requests, or performance declines. The last two scenarios are enabled by the checkpointing mechanism. Users of GridWay have their own personal submission agents (job queues).

The GridLab Grid Application Toolkit (GAT) [9] aims to provide a simple and robust environment for developing applications that exploit the Grid. GAT provides a layered view of the available functionality, ranging (bottom up) from a core layer, a service layer, a GAT API layer, and finally an application layer. The GridLab Resource Management System (GRMS) resides at the service layer. It provides services for job management, and management of infrastructure elements such as information, data, networks, and accounting. Job management capabilities include a Job receiver (job queue) and services for resource discovery, job requirement prediction, resource performance estimation, and brokering, as well as mechanisms for negotiating QoS terms and performing advance reservations. The GridSAM [144] job submission pipeline is based on the principle of a staged event-driven architecture [218]. Instead of treating each job submission in isolation, the event-driven job submission pipeline is divided into stages, allowing the processing load of each stage to be controlled by adjusting the size of a thread pool serving it, thus increasing fairness under high load. Steps in the GridSAM pipeline include file stage-in, job description generation, job submission, job monitoring, file stage-out, and cleanup.

One main focus of the P-GRADE project [114] is to run parallel applications, i.e., jobs using MPI or PVM, on the Grid. P-Grade provides a graphical environment to design, execute, and monitor applications. The design of message passing applications is facilitated through a graphical language that enables also non-expert users to define the communication pattern between the processes. In P-Grade, checkpointing (and hence also migration) is supported for PVM applications. The focus of the Falkon system [170] is to improve performance for large numbers of submitted jobs. In Falkon, a provisioner creates and destroys executors. These executors are placed on available Grid resources and can accept jobs submitted by the user. This model provides for good performance and is particularly well suited for fine-grained jobs as the non-neglectable overhead for submitting a task to a Grid resource is done only once for each executor, instead of once for each job. Moltó et al. [148] discuss how to model and implement a resource brokering architecture using the Web Services Resource Framework (further discussed in Section 6.1). Their architecture is divided into three layers, information management, job submission, and metascheduling of multiple tasks.

Closely related to Grid job management is the process of enabling a certain application to execute on the Grid. By wrapping legacy code applications as Grid services, GEMLCA [56] provides submission, monitoring and result retrieval of jobs that execute legacy codes. The GEMLCA approach to wrapping the applications is non-invasive, as it does not require access to, or modification of, the application source code. Users can deploy new applications into the GEMLCA hosting environment through a Web portal and also manage jobs executing these applications. Mateos et al. [140] discuss various approaches to gridify, i.e., Grid-enable, applications. In their work, a set of existing frameworks are compared with respect to job granularity and ease of gridification, the latter including aspects such as whether the application source code needs to be recompiled.

3.3 Contributions to Grid Job Management

The main contribution to Grid job management in this thesis is the investigation of suitable architectures for job management tools. Our study proposes a multi-layered architecture in order to improve abstraction and flexibility. A proof-of-concept implementation demonstrates the feasibility of this model from a performance point of view.

Chapter 4

Grid Workflows

A *workflow* is a set of coordinated activities that combined solve a complex problem. The tasks that constitute a workflow are typically arranged as nodes in a directed graph where the links represent dependencies between tasks. These graphs are typically acyclical (so-called DAGs), although some workflow systems use graphs with cycles in order to model iterations.

The edges in the workflow graph can denote a *control flow* that explicitly dictates the execution order of the workflow tasks. This model of computation shares many characteristics with sequential programming languages and can hence easily be understood by programmers. Alternatively, the workflow graph edges can specify a *data flow* that implicitly defines a partial execution order of the workflow tasks. In the data flow model, a workflow can be thought of as a flow of data, where the nodes (tasks) merely are functions that transform the data as it flows through the graph. Although perhaps less straight-forward to most programmers, the data flow model has proven useful for non-programmers and application scientists [162]. Other Grid workflow approaches [104] use *Petri nets* as model of computation. Petri nets are a Turing complete formalism commonly used for modeling and analyzing systems that are concurrent, asynchronous, and non-deterministic.

Workflows are not exclusive to Grid computing, research on these started long before Grids existed. The emergence of Grid computing enables computational workflows to leverage powerful Grid resources for computationally intensive tasks. Such Grid workflows typically consist of a combination of computational jobs and file transfers between the machines that execute the jobs. A broad introduction to Grid workflows as well as a in-depth description of contemporary workflow systems is found in the book by Taylor et al. [196]. Van der Aalst et al. [208] discuss commonly used patterns in workflows. Taxonomies of workflow capabilities include [53, 230]. For a wide overview of current research in scientific (Grid) workflows, see e.g., [25, 55, 85, 133, 196].

4.1 Workflow Management Capabilities

The typical tasks in the scientific workflow process include composition of a new workflow, scheduling of the workflow, workflow enactment (execution), and analysis of the results, the latter including recording of enactment metadata for reproducibility purposes.

4.1.1 Workflow Composition

Important aspects when composing a new workflow include the language used to encode the workflow, the tools available to design a particular workflow, and to what extend reuse of existing workflows are possible.

The composition of a scientific workflow is simplified if the workflow language has expressive power equivalent to that of a (Turing complete) programming language. However, there is disagreement within the workflow community about whether workflow languages should be simpler and more restricted than programming languages [52], or whether constructs such as iterations and conditions are essential to encode complex scientific problems [20].

As most workflow systems use a textual, often XML-based, representation of the workflow tasks and their dependencies, design of a new workflow can be performed by encoding the workflow description manually. This can however be error-prone even for expert users, and the usage of a drag-and-drop style GUI client for defining the workflow graph is common.

Workflow composition is greatly simplified if existing workflows can be reused in parts or in full. This is exemplified by the Taverna [158] tool, where more than 3000 bioinformatics services can be utilized in new workflows. The $^{\text{my}}$Experiment project [179] allows scientists to find, share, modify, and reuse workflows, as well as build communities and collaborate with colleagues in a manner inspired by social networking Web sites.

4.1.2 Workflow Scheduling

Workflow scheduling is the transformation of an *abstract workflow* with only tasks specified, to a *concrete workflow*, where resources have been assigned to each task. As a Grid workflow consists of a combination of computational tasks and file transfers, all considerations that apply for the brokering and management of individual jobs (as discussed in chapters 2 and 3) are also valid for workflow scheduling. The main difference between workflow scheduling and management of independent jobs, is to what extent workflow scheduling system should take dependency constraints among the tasks in the workflow into account. One method is full-ahead planning, where resources are assigned to each task at the start of the workflow execution. The completely opposite approach is just-in-time workflow scheduling, where each task is scheduled as its preceding task(s) complete. Hybrid solutions also exist, where the workflow is divided into subgraphs that are planned as execution of previous subgraphs complete.

Full-ahead planning techniques typically involves determining the *critical path* of the workflow, that is, the path of subsequent tasks in the workflow graph with the longest estimated total execution time. By giving higher priority to the tasks along this path, critical path workflow scheduling algorithms aims to minimize the workflow completion time, the so called *makespan*. However, just-in-time scheduling may also reduce the makespan, as this technique can leverage the dynamic nature of the Grid, and benefit from appearance of new resources and/or changed load on known ones. To minimize execution time, the workflow scheduling system can also rewrite the workflow graph, e.g., to group tasks for increased data reuse.

The Pegasus workflow management system [54], is an example of a workflow scheduler that reorganizes the DAG during the mapping process. In Pegasus, the final concrete DAG is produced in a format interpretable by an enactment engine such as Condor DAGMan [200]. Pegasus supports the use of placeholder jobs that in advance are submitted to Grid resources for execution. Once running, a placeholder job can execute one or more tasks in the workflow. When combined with task clustering, placeholder jobs can improve the workflow performance considerably, especially for fine-grained tasks [54]. Huang et al. [108] propose a simplified workflow scheduling algorithm that takes into account neither resource information nor task information beyond intertask dependencies. In this work, resources are grouped by heterogeneity and connectivity. A performance evaluation confirms that their proposed method gives shorter makespans than critical path algorithms.

Before the emergence of Grid workflows, the problem of scheduling of a set of dependent tasks, typically expressed as a DAG, on a set of heterogeneous processors, has been studied extensively. Examples of algorithms include Heterogeneous Earlist-Finish-Time (HEFT) [203], that also has been evaluated in the context of Grid workflow scheduling [219]. Rotithor's taxonomy [178] of dynamic task scheduling schemes is based on a separation of state estimation (i.e., resource load and/or performance prediction) and decision making. Alhusaini et al. [5] describe a DAG scheduling algorithm to minimize the makespan that operates on one layer of the DAG at the time. In this work, various heuristics for scheduling of tasks from a given layer are evaluated, including min-finish-time and max-finish-time. In the work by Canon et al. [33], 20 DAG scheduling heuristics are evaluated, focusing on makespan and robustness of the heuristics, the latter considering to what extent they are affected by unpredictable run-time changes.

4.1.3 Workflow Enactment

Grid workflow enactment can be performed using different levels of abstractions. For job-based workflows, some systems interact directly with batch system schedulers, whereas others use a Grid middleware, or even a toolkit that abstracts over multiple middlewares, to manage the workflow tasks. Alternatively, if the workflow is composed of a set of orchestrated services, workflow enactment is the process of invoking the services in the specified order. As failures are frequent in Grid environments, fault tolerance is an important aspect of workflow enactment. This can be achieved on a per-task basis, e.g., by re-executing a failed task on an alternative resource or restarting an interrupted file transfer with a replica of the transferred file. These task-level fault recovery mechanisms are in the literature described as implicit fault management [104]. Workflow-level fault tolerance is another possibility, where an alternative path of enactment in the workflow graph is taken upon failure. However, the latter approach requires that the workflow language supports conditions and/or exceptions. A comparison of fault tolerance techniques in contemporary workflow systems is provided by Plankensteiner et al. [167]. As only concrete workflows can be enacted, there is a strong coupling between tools for workflow scheduling and workflow enactment, especially if workflow scheduling is performed in a just-in-time fashion (as opposed to full-ahead planning).

One commonly used workflow engine (workflow enactment system) is the control flow based Condor directed acyclic graph manager (DAGMan) [200]. In DAGMan, a workflow graph consists of computational tasks (Condors jobs) along with pre and post scripts, that can be used e.g., for data transfers. DAGMan is used as enactment engine, e.g., by Pegasus [54] and P-GRADE [89, 114]. The goal of the MOTEUR workflow engine [88] is to combine ease of use through simplified iterations (parameter sweeps) with flexible data composition mechanisms and efficient parallel execution. The data composition strategies in MOTEUR enables users, through the Scufl language of Taverna, to express nested dot and cross products over data sets.

4.1.4 Workflow Reproducibility Concerns

Reproducibility is key in the scientific process. In order to verify and repeat experiments, scientific workflow systems need to store information about input data sets, used Grid resources, versions of softwares and libraries, etc. This *provenance* metadata enables, when properly recorded, the workflow execution to be reproduced. Ideally, the workflow provenance information should not only capture the computational steps of the Grid workflow, but rather be a complete description of the greater, scientific process of generating new knowledge. Simmhan et al. [226] discuss various provenance issues as well as survey provenance systems and workflow management tools that support provenance.

4.2 Contemporary Grid Workflow Systems

Well-known scientific workflow systems include the ASKALON Grid application development and computing environment [69, 70]. The ASKALON architecture includes a Scheduler for placement decisions, a Resource Manager that performs resource discovery and related tasks, and an Enactment Engine for reliable and fault tolerant execution of tasks. The UML-based AGWL language used to define workflows supports both control and data flow, as well as conditions and iterations. In ASKALON, placement decisions are based on a combination of advance reservations and performance predictions. The ASKALON scheduling process is divided into refinement (workflow rewriting), mapping (resource to task matching), and, upon failures or similar events, rescheduling. The used scheduling algorithms include HEFT.

Taverna [157] is an extensive workflow environment for the life science community. Workflows in Taverna are expressed in the Scufl language that supports both data flow and control flow. Taverna users benefit from being able to reuse a large set of predefined workflow services. Notably, as users only select among services hosted at predefined locations, no workflow scheduling is performed in Taverna.

The Kepler [132] workflow management system defines a data flow model where *actors* modify the data. Data transformation actors can fine-tune the data streams using, e.g. XSLT [58] or XQueury [29]. Kepler supports a wide range of control and data flow enactment styles, including synchronous communication, publish-subscribe notifications, and continuous time feedback loops. In Kepler, parameter sweeps are implemented using higher-order functions.

Triana [44, 197] is fundamentally a data flow system, but supports control flow through trigger tokens. Triana uses a component-based execution model, similar to that of the Common Component Architecture (CCA) [73]. Components can either be job-based or service-based. The job-based workflows execute on the Grid through the GridLab Grid Application Toolkit (GAT) [10], with bindings to the Globus GRAM, GridFTP, etc. Such workflows can be scheduled using the GRMS [123] described in Chapter 3. Triana supports service-based workflows by orchestrating services using the GAP toolkit [198]. There are no iterations or condition constructs in the Triana language, these are instead embedded in custom components. Triana decouples the GUI for workflow design from the workflow enactment system.

The P-GRADE [114, 89] Web portal allows users to define and manage workflows. Similar to Triana, the P-Grade portal supports (through DAGMan [200]) enactment of workflows composed of tasks, or of services, the latter through MOTEUR [88]. Nodes in a P-GRADE workflow graph can either be jobs or GEMLCA legacy code services. P-GRADE allows the use of resources from multiple Grids, and uses MyProxy [23] to enable the use of multiple certificates for the same user. Parameter sweeps are supported through parameter spaces that are mapped into data segments and further on to tasks (DAGMan jobs or MOTEUR service invocations) in the workflow manager. This mapping can either be dot or cross product.

Karajan [213], the workflow component of the Globus Java CoG kit, is based on earlier work on GridAnt, a Grid extension to the ant build system [17]. The Karajan system supports job execution and file transfers, as well as monitoring and checkpointing of workflows. In Karajan, a declarative control flow language is used, with constructs for expressing sequential and parallel tasks as well as conditions and iterations, where the latter can be either sequential or parallel.

Construction of large workflows is simplified by the support for user-defined procedures that enable custom extensions to the Karajan language.

The Imperial College e-Science Networked Environment (ICENI) [141, 142] workflow pipeline consists of the three phases: specification, realization, and execution. ICENI provides a pluggable scheduling framework that easily can incorporate new algorithms. Rapid completion of time critical tasks is ensured either by advance reservations or by placeholder jobs, the latter a mechanism also used by Pegasus [54] and Falkon [170].

4.3 Contributions to Grid Workflows

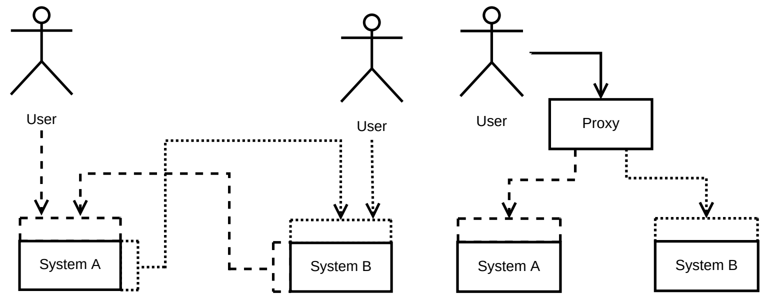
This thesis includes studies of suitable abstractions, techniques, and tools for integrating local and Grid resources in scientific workflows. One result is the investigation of suitable programming models for parameter sweeps and tools that enable the binding of workflow task parameters to be delayed until enactment time. We also survey workflow interoperability, albeit with a different approach than the standards-adoption one taken for resources brokering and job management. Our interoperability study has a theoretical perspective with particular focus on differences between Grid workflows and locally executing ones. This study builds on research within the areas of theory of computation, compiler optimization, and visual programming languages.

Chapter 5

Grid Interoperability and Standardization

There is today a somewhat paradoxical situation where Grids, partly being developed to increase interoperability between different computing platforms, themselves to high extent have interoperability problems. Although the reasons are obvious, expected, and almost impossible to circumvent (as the task of defining appropriate standards, models, and best practices must be preceded by basic research and real-world experiments), it makes development of portable Grid applications hard. The current situation with isolated islands of Grid infrastructures unable to interoperate with eachother is partly due to the many Grid projects that focus on infrastructures tailored for specific application areas despite only slightly different use cases. These projects have resulted in the creation of a number of Grid middlewares with similar functionality but without the ability to interoperate. The problem of achieving interoperability is however not only a technical one, but also a policy (political) issue. Lack of coordination among Grid projects and the related research communities have resulted in fragmented infrastructures without a clear structure. There is furthermore large overlaps in existing virtual organizations. A user can belong to multiple VOs, and be identified by different certificates in these VOs.

Although interoperability intuitively can be understood from a high-level perspective, the concept has no single agreed upon definition. IEEE defines interoperability as “The ability of two or more systems or components to exchange information and to use the information that has been exchanged” [111]. This definition captures the intuitive understanding that the interoperable systems must be able to communicate in a meaningful way. Another definition, by ISO, describes interoperability as “The capability to communicate, execute programs, or transfer data among various functional units in a manner that requires the user to have little or no knowledge of the unique characteristics of those units” [72]. One important aspect of this definition is that the use



(a) Gateway approach to interoperability. (b) Proxy approach to interoperability.

Figure 3: Different approaches to interoperability through user transparency.

(i.e., communication, execution, data transfer etc.) of two interoperable systems should be transparent to the client, be it a human operator or another system. For this second definition, there are (at least) two possible approaches to achieve the envisioned transparency. In the gateway approach, illustrated in Figure 3(a), one or both systems are modified to communicate with the other. The end user invokes the original system through its native interface, and requests are transparently forwarded to the second system. This transparency increases usability, but can on the other hand be seen as intrusive, as the gateway approach necessitates modification of both systems. The second option, shown in Figure 3(b), is to add an additional layer (a proxy) on top of the two systems that hides the differences in operation. With this approach, the end user interacts with the proxy layer, instead of with the individual systems. The benefits of this is that no modifications are required to the existing systems, which both simplifies implementation and reduces intrusiveness. However, the wrapping layer may appear different to the user than the native system interfaces, and can hence be more difficult to use.

In achieving interoperability for Grid middleware, the proxy approach is more feasible as it can be implemented on the client side. The gateway approach necessitates modification to software running on the resource side. This is impractical, and potentially also problematic from a policy point of view, as resource owners not necessarily agree that their machines need modifications and/or reconfigurations.

5.1 A Layered View on Interoperability

One well-proven method to achieve interoperability is to agree on standardized protocols, interfaces, and data formats. Standardization of lower-level protocols and data formats enables the definition of other ones with a higher level of abstraction. This can be exemplified by the successful Internet network stack, where higher-level protocols such as HTTP and FTP are constructed on top of lower-level ones such as Ethernet and IP. A data format example of this principle is SOAP that leverages XML and XML schema. This idea of hierarchical reuse can be observed also in Grid computing. Within the data management area, the GridFTP protocol, further discussed in Section 1.3.3, greatly simplifies the construction of higher-level data services such as Stork [121] and RFT [136].

Unlike the GridFTP case, where a well-proven standard (the FTP protocol) existed, early Grid job management functionality was developed as a distributed extension to non-standardized batch system interfaces. These batch system interfaces were much later standardized through OGF DRMAA [171]. One commonly used early job management protocol is the Globus GRAM [76], which later was redesigned with Web service tools and renamed WS-GRAM. Although adopted by many projects as foundation for higher-level job management components, the GRAM interface never became a standard, and alternative implementations of the offered functionality exists, as illustrated by the NorduGrid/ARC middleware [65] that performs job management through GridFTP. The OGF has recently promoted two specifications that are important for standardization of job management functionality, JSDL [16], a language to describe job requests, and an interface to the job submission system, OGSA-BES [79]. Although rather new, both of these have gained widespread support [143].

In both the data management and job management cases, one can observe that standardization only occurs up to the point when there no longer exists a broad agreement on the required functionality and/or semantics. For data management, agreement exist on the transport level (GridFTP), but consensus has yet to be established, e.g., for replication and fault-tolerant transfers. Participants within the job management community agree on the language (JSDL) and the basic submission mechanism (OGSA-BES), but still struggle to define higher level services such as negotiation and resource selection. Some conceptual ideas are specified in the OGSA Resource Selection Services RSS [83], where the Candidate Set Generator (CSG) identifies resources where a job can execute, whereas the higher level Execution Planning Service (EPS) decides where a job should execute. The complete interfaces of these services, as well as their exact semantics remain to be defined.

The interoperability problems within contemporary Grid environments are the motivation behind substantial efforts from both research and infrastructure projects. These efforts can be divided into two classes, short-term solutions to provide interoperation between existing infrastructures and more sustainable approaches based on adoption of standard data formats and interfaces. The focus of the Grid Interoperability Now (GIN) project (later renamed Grid Interoperation Now) [96] is to provide increased interoperation among existing Grid infrastructures. Examples of software tools that interoperate with multiple middlewares include GridWay [109], which interfaces resources running one of GT2, GT4, and LCG. Venugopal et al. [211] share their experiences in designing a broker that interoperates with various batch systems as well as multiple Grid middlewares. In an approach suggested by Kertész et al. [118], interoperability is achieved on a resource broker level instead of a Grid middleware level. Rather than interfacing resources directly, their software interacts with the respective resource brokers of the used middlewares. Kertész et al. also define a language for communicating broker capabilities [119]. A similar approach is suggested by Rodero et al. [177] that argue in favor of a meta-broker based on their experiences from the HPCEuropa project. The proposed meta-broker would have unified mechanisms for all tasks in the job management process and interact with Grid-specific brokers. Rodero et al. also survey to what extent existing and proposed standards fulfill the requirements of the meta-broker. Bobroff et al. [30] discuss different architectures for interoperable metascheduling and present a hybrid approach that combines hierarchical and peer-to-peer architectures. The contribution by Peirantoni et al. [165] utilizes WSRF-based Metagrid services as a bridge between users and different Grids. Set theory is used to formally describe the Metagrid services and their interactions.

Notable examples of projects that focus on standardization include the Genesis II [150] Grid system. Genesis II supports a wide range or OGSA standards, including JSDL, OGSA-BES, OGSA Resource Namespace Service (RNS) [164], and OGSA-ByteIO [149]. A strong focus on usability allows users to interact with Genesis II resources through a set of common shell commands such as, e.g., cat, cp, and ls. An OGF report [143] shares implementation and interoperability experiences of JSDL and surveys projects that adopt this standard. An OGF report [153] by Newhouse et al. discuss five scenarios related to the execution of applications on a cluster from an interoperability perspective. This report further illustrates how these scenarios can be realized using OGSA standards such as JSDL (with proposed extensions), OGSA-BES, the GLUE 2 information model [13], RNS, and DRMAA.

The purpose of an ongoing collaboration between the European Telecommunications Standards Institute (ETSI) [68] and the OGF is to ensure that the standards for communication within the Grid are based on requirements from both industry and research. The Open Middleware Infrastructure Institute (OMII) Europe [159] targets to make components for job management (OGSA-BES), data integration, and accounting available for multiple platforms, e.g., gLite [62], Globus [75], and UNICORE [195]. The UniGrids project [174] focuses on developing an OGSA-compliant infrastructure on top of the UNICORE software. Grimme et al. [97] analyze the teikoku scheduling framework in the light of standards-compliance, and investigates the gaps between this framework and the proposed OGF Grid scheduling architecture.

5.2 Contributions to Interoperability

This thesis demonstrates how standard protocols and data formats contribute to middleware independent architectures for resource brokering, job management, and workflow enactment. A related contribution is the evaluation of proposed standards through an investigation of how these can be applied in practice. We describe some of the first results related to the usage of WSRF, JSDL, WS-Agreement, and the GLUE information model in software for Grid resource management. Furthermore, our proof-of-concept software tools provide easy integration with multiple middlewares.

Chapter 6

Design Considerations for Grid Software

This chapter discusses selected topics related to the development of Grid software, including the adoption of technologies associated with Service Oriented Architectures (SOAs). Also included is a description of the rationale behind the design choices taken during development of the software described in this thesis.

6.1 Service-Oriented Architectures

In a Grid middleware, it should be easy to add support for new types of resources such as sensors, instruments and alternative data sources. The construction of client tools is simplified if basic tasks such as allocation, invocation, notifications, and termination can be performed in a uniform manner, independent of the type of resource the client interacts with. The first generations of Grid software failed to fulfill these requirements, as illustrated by the early versions of the Globus toolkit that used resource type specific protocols, e.g., LDAP [60] for resource discovery, GridFTP [61] for data transfer, and GRAM [76] for job management. New resource types could only be integrated by the introduction of additional protocols, further adding to the protocol heterogeneity of the toolkit. More recently, many Grid projects have adopted principles from SOAs to overcome these limitations.

A SOA is “a paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains. It provides a uniform means to offer, discover, interact with and use capabilities to produce desired effects consistent with measurable preconditions and expectations” [156]. A key observation in SOAs is that a capability required by one entity (a person or an organization) often is offered by other entities. Needs and capabilities in a SOA are brought together by services. A service is described as the capability to perform work for another, the specification of the capability, and the offer to perform the work [156]. All services in a SOA are described in a uniform manner through service interfaces. A service interface is a visibility mechanism that enables potential service consumers to decide if a particular service provider fulfills their requirements. Invocation of a service is transparent to both service location and the respective platforms of the service provider and the service consumer. With this transparency, services have the potential to increase reuse beyond that offered by software libraries as the latter are restricted to specific programming languages and/or platforms.

Figure 4 gives an overview of a SOA and illustrates the commonly applied publish-find-bind pattern. In the figure, a service provider publishes information about the capabilities offered by its service(s) in an index. A service consumer that requires some capability queries the index and locates a suitable service. In the last step, the service consumer binds to the service provider, and requests the capability.

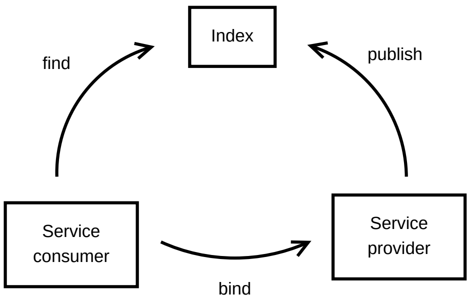


Figure 4: Publish-find-bind in a Service Oriented Architecture.

6.2 Web Services

Web services is a technology that can be used for realizing a SOA. The World Wide Web Consortium describes a Web service as a software system designed to support interoperable machine-to machine interaction over a network [214]. Web services are described using the Web Services Description Language (WSDL) [43] and communicate using SOAP [100], typically with HTTP as transport protocol. For Web services, an Universal Description, Discovery, and Integration (UDDI) [106] server can have the role of the index in the publish-find-bind scenario. Despite being a general architecture, Web services have some shortcomings when applied to Grid computing. Web services are stateless, i.e., there is no standard mechanism to store state information in a service between invocations and session management is thus complicated. Web services are also persistent and cannot be created and destroyed dynamically. Furthermore, the UDDI is designed for persistent services and can hence not handle [26] the volatile nature of the Grid, where services may be available for limited periods of time only. The infeasibility of using pure Web services for Grid computing was the prime motivation for the design of the by now obsolete Open Grid Services Infrastructure (OGSI) [205]. The OGSI defines Grid services as an extension to the Web service concept. Grid services do, as opposed to Web services, have state and a limited lifetime. Lifetime management mechanisms allow Grid Services to be created and destroyed, the latter occurring either immediately or at a specified time. Furthermore, the OGSI specifies a mechanism for asynchronously notifying clients about changes in service state. An OGSI ServiceGroup uses soft-state registration of entries, making it more suitable than an UDDI server as an index for the Grid.

The OGSI specification did not gain widespread support, and was furthermore criticized by the Web services community, e.g., for being too tightly coupled to object orientation, for having poor support for existing Web service tools, and for being monolithic. To overcome these issues, OGSI was refactored into the Web Services Resource Framework (WSRF) [155] family of specifications that does not have the monolithic structure and many of the other shortcomings of OGSI. In WSRF, Web services are stateless and persistent. A stateless Web service can however have one or more stateful WS-Resources [94] associated with it. A client that invokes a WSRF-enabled service can specify, as part of the service address, which WS-Resource it wants to interact with. The WS-ResourceProperties [93] specification defines how state is stored in WS-Resources. WS-ResourceLifetime [192] defines mechanisms for lifecycle management of WS-Resources, but not of Web services as these are persistent in WSRF. The basic fault messages defined in the WS-Basefault specification [131] are used to increase consistency. In a WS-ServiceGroup [137], Web services and WS-Resources can be grouped together. Information registered in a ServiceGroup is, to avoid stale entries, removed unless it is renewed within a given time. The WS-BaseNotification [92] framework is not part of the WSRF family of specifications, but uses WSRF mechanisms to deliver asynchronous updates to interested entities and is often used together with the WSRF. An in-depth introduction to the areas of SOA and Web Services is beyond the scope of this thesis, but can be found in books [160, 190]. Joseph et al. [113] discuss the specifications for stateful Web Services and their impact on architectural principles of Grid computing.

6.3 Design Heuristics

It is of paramount importance for Grid system developers to study and understand the specific characteristics of Grid environments. These include the lack of total control over the considered Grid resources as well as the incomplete and typically outdated information that can be collected from them. Furthermore, failures occur at a regular basis, not only under rare conditions. One common misconception is that the Grid gives unlimited access to resources that can be used freely if locally available ones become scarce. On the contrary, the demand for resources typically far exceeds the supply. Fundamental Grid middleware capabilities (as discussed in Section 1.3) help address heterogeneity in resource hardware and software. What must not be forgotten however is the more cumbersome heterogeneity in ownership and usage policies that is left to the developer of Grid software to handle. Resource availability is bound to vary over time, due both to faults and policy reasons. Failures in understanding the characteristics of Grid environments result in architectures and software tools that are unsuitable for Grid environments. Examples of this includes centralized software of omnipotent character that fail to function as expected due to competition from other, competing software tools.

The software tools constructed as part of this thesis are designed with great care to comply with the here discussed characteristics. Software is also built as we envision that it will work, without restricting ourselves to limitations of current environments. This includes support for functionalities that we foresee as inevitable for future systems. As an example, the advance reservation management system supports malleable reservations as this is foreseen as a critical functionality that most likely will be available in the future, although no contemporary batch schedulers support them. However, at the same time, care is taken to ensure that the tools are easily integrateable into existing infrastructures, and are fully operational also in environments that do not support the anticipated extensions. The software is not limited to use within a particular domain or application, but is rather indented as building blocks to simplify the construction of higher-level, domain-specific services.

Although there is large number of researchers worldwide that study Grid technologies and tools, surprisingly few have shared their ideas on topics such as design processes and software engineering for Grids. There is furthermore a focus on the application side, few contributions discuss the engineering of Grid middleware and other fundamental software. Abeti et al. [1] study how model driven architectures can be exploited in the design of SOAs. Hernández et al. [102] propose the use of domain-specific modeling to increase reuse when designing Grid applications. Experiences in using the CCA to build Grid applications are described by Gannon et al. [86]. In their discussion of the EveryWare system, Wolski et al. [220] describe a programming model and methodology for writing Grid programs and discuss some quality attributes. They also describe, from an application development perspective, how to use a *lingua franca* (mutually understood third language) to interface with the EveryWare system. Byun et al. [32] discuss adaptability, scalability, and reliability as well as demonstrate a WSRF-based resource provisioning framework with these characteristics.

With the increasing adoption of principles from SOAs by Grid projects, developers of Grid software would benefit from leveraging the software engineering work performed within the fields of SOAs [154, 194] and service-oriented computing [126, 161].

6.4 Contributions

This thesis demonstrates how principles of SOAs can be applied to Grid computing. This includes an investigation of how SOAs can be used to achieve loose coupling as demonstrated by the job management architecture in Paper IV, as well as improve reuse of workflow tools, the latter illustrated by papers V and VI. We investigate the characteristics of the future landscape of service-oriented Grid infrastructures in Paper VIII. This study also gives some of the first results regarding software engineering aspects of the design process for Grid middleware tools.

Chapter 7

Summary of the Papers

7.1 Paper I

Extending on our earlier work [67] in the area, Paper I focuses on algorithms and principles for the decentralized Grid brokering problem. A prediction of the Total Time to Delivery (TTD) for a Grid application is used as the main criteria in resource selection. The TTD includes the time required to perform the following operations: (i) transfer of the input files and executable to the selected Grid resource, (ii) wait for resource access, (iii) execute the application, and (iv) transfer of job output files to their requested location(s).

Paper I also discusses algorithms to estimate each part of the TTD, including the usage of bandwidth prediction tools for tasks (i) and (iv), the prediction of batch queue waiting times using either advance reservations or an estimate based on current resource load, and a benchmark-based mechanism to predict the application execution time. Paper I describes the implementation of resource selection algorithms based on prediction of the TTD in a proof-of-concept job submission tool for the NorduGrid/ARC middleware. Two approaches for advance reservations are also discussed and compared, one closely integrated with existing NorduGrid/ARC job submission mechanisms, the other a general service-based framework for reservation management.

7.2 Paper II

In Paper II, we investigate how principles of SOAs combined with standardization can be used to achieve adaptability in resource selection as well as Grid middleware independence. As a proof-of-concept, a general, service-based Grid resource brokering architecture is designed and implemented. This job submission service consists of a set of replaceable modules that each performs a well-defined task in the job submission process. The concept of replaceability is used also within some components, e.g., the resource selection algorithms can be exchanged for alternative implementations.

The brokering architecture is designed to be as independent as possible of the Grid middleware used on the resources, and also to support any job description format on the client side. This interoperable design is achieved through the use of (proposed) standard formats and protocols, including WSRF, JSDL, GLUE, and WS-Agreement. These are used both internally in the job submission service and, if possible, in communication with resources. Interaction with a specific Grid middleware that uses non-standard data formats or protocols is handled through a set of well-defined integration points. The feasibility of this approach is demonstrated by the integration of the job submission service with the NorduGrid/ARC middleware.

Algorithmical contributions in Paper II include further development of the resource selection algorithms from Paper I, with focus on greater flexibility. Users may choose to select resources with the objective to achieve the earliest possible job completion, or the earliest possible job start. Furthermore, users may fine-tune the resource selection process by including a document with job preferences when invoking the broker. This document can be used to express job start time requirements and can also give helpful hints to the broker about the characteristics of the application.

7.3 Paper III

Paper III provides advances to the state-of-the-art in Grid resource coallocation, including the design, implementation, and analysis of an algorithm for arbitrarily coordinated allocations of resources. By viewing coallocation as a bipartite matching problem, we leverage well-known principles from graph theory to improve the likelihood of successfully allocating a set of resources. Paper III also contains a in-depth survey of existing coallocation approaches. By classifying contemporary coallocation algorithms and comparing these with methods from transaction management, we contribute to the general understanding of the coallocation problem.

The feasibility of the approach proposed in Paper II with a service-based architecture that enables interoperability through standardization is further demonstrated by the integration of the job submission service with Globus Toolkit 4 middleware in addition to the already supported NorduGrid/ARC. The implementation of integration plugins for a middleware is typically less than ten percent of the middleware-neutral code. This suggests that a feature-rich job submission tool for a Grid middleware can, with comparatively little effort, be obtained by implementing job submission service plugins for that middleware.

In Paper III, the performance of the job submission service is evaluated in depth, with focus on service response time and throughput for various Grid configurations. The coallocation algorithm is studied in a series of tests that both illustrate the performance dependencies of the different parts of the algorithm and provide insight in the intrinsic complexity of the coallocation problem.

7.4 Paper IV

Paper IV investigates suitable architectural decompositions for job management software. The proof-of-concept Grid Job Management Framework (GJMF) further elaborates on ideas from papers II and III by extending the resource brokering software to also include job monitoring and control tools, as well as mechanisms that simplifies management of large sets of independent jobs. Another contribution is the study of fault tolerance techniques, through a multi-layer mechanism for automatic resubmission of failed jobs.

In the GJMF, the job management functionality is decomposed into a set of services that forms the following layers: the middleware abstraction layer that performs basic job management regardless of the Grid middleware at the resource side; the brokered job submission layer that offers resource discovery, resource selection, and job submission; and the reliable job submission layer that adds fault tolerance for single jobs or sets of jobs. Just as its predecessor, the GJMF makes use of the Web services, WSRF, and JSDL to promote standardization and increase interoperability. These aspects are further emphasized by implementing the draft resource selection services from OGSA [83] in the brokered job submission layer.

A performance evaluation demonstrates that the extra functionality offered by the GJMF services adds little overhead (down to 0.2 seconds per job). This evaluation suggests that for Grid application developers, multi-tiered job management frameworks such as the GJMF is an attractive alternative to interacting with the Grid middlewares directly.

7.5 Paper V

Paper V studies how Grid workflow capabilities can be decomposed into a loosely coupled architecture where capabilities are exposed as a set of workflow services with clear separation of concerns. By orchestrating these services, environments tailored to the needs of a certain group of users can be constructed with much less effort compared to (re)implementation of full-featured workflow systems. The feasibility of this approach is demonstrated by the design and implementation of the Grid Workflow Enactment Engine (GWEE). GWEE is not proposed as a replacement to existing workflow systems, but rather as a core component for developing new end-user tools and problem solving environments.

Paper V also demonstrates how workflow enactment can be completely decoupled from the processing of individual tasks, such as computational jobs or file transfers. It is further shown how Grid workflows can combine a data flow model of computation in which data dependencies decides the task execution order, with a control flow approach where control tokens are communicated to initiate subsequent tasks. A contribution to workflow interoperability is the demonstration of how the GWEE tool can enact both data driven workflows expressed in the GWEE language and control driven workflows in the Karajan format.

7.6 Paper VI

Paper VI investigates how to define data flows that transparently integrate local and Grid workflows. In addition, the benefits of parameter sweep workflows are examined and a means for describing this type of workflow in an abstract and concise manner is introduced. Our parameter sweep mechanism extends the state-of-the-art in the area as it is not restricted to a predefined data distribution pattern. Both the problem of decoupling data flow from task specification and that of dynamically configuring workflow tasks during runtime are addressed by the introduction of a task template mechanism that delays the binding of task input parameters until workflow execution.

Paper VI further demonstrates the feasibility of the concept of a composable set of services with fundamental workflow capabilities by illustrating how a proof-of-concept client tool can be built on top of GWEE. With this client GUI, users can design, execute, and monitor workflows. The final contribution is a use case demonstrating how the developed mechanisms are employed to reduce the complexity of a particular bioinformatics problem - orthology detection analysis.

7.7 Paper VII

Paper VII is a comprehensive study of topics related to interoperability among scientific workflow systems. Rather than discussing if these workflow systems are completely interoperable or not at all, we argue that interoperability must be considered from three distinct dimensions: model of computation, workflow language, and workflow execution environment. Paper V illustrates workflow execution environment interoperability by showing how a workflow enactment engine can interoperate with multiple Grid middleware. Extending on this effort, the two most commonly used models of computation, Petri nets and dataflow networks, are studied with particular focus on how these have been adapted to fit the execution environment imposed by currently used Grid middleware.

Paper VII also investigates the minimum language constructs required for a language to be expressive enough to support scientific workflows. The fundamental differences in the respective execution environments of end-user desktop machines and the Grid suggest that whereas complex (Turing complete) languages could be used to describe locally executing workflows, simpler task coordination languages are preferred for the Grid. Leveraging on earlier results in the areas of theory of computation, compiler optimization, and visual programming languages, Paper VII studies, from a workflow perspective, fundamental language constructs such as iterations, conditions, exceptions, and the implications of having a type system.

7.8 Paper VIII

With a starting point in the characteristics of Grid environments, Paper VIII describes the vision of a SOA-based ecosystem of Grid software components and outlines competitive factors for tools to “survive evolution” in such an environment. From these factors, the paper investigates design heuristics, design patterns, and quality attributes that are central to building software well adapted to the Grid ecosystem. These are divided into the following five groups. *Co-existence* includes aspects relating to decentralization, non-intrusiveness, and avoidance of resource over-consumption. Concerns for *composability* include well-defined interfaces, single-purpose components, and other characteristics that improve reuse. Ease of installation, configuration and use, as well as portability related concerns are all aspects of *Adoptability*. *Adaptability and changeability* affect ease of adaption into new or changed environments, and benefit from separation of policies from mechanisms. The last group is *interoperability* and the corresponding techniques to simplify ease of interaction with other software systems as well as the related standardization efforts. With a starting point in these five groups, the characteristics of software developed by the GIRD [201] team, including JSS (papers II and III), GJMF (Paper IV), GWEE (papers V, VI, and VII), as well as the SweGrid Accounting System (SGAS) [87], and FSGrid [66] are reviewed.

Chapter 8

Future Work

There are several potential directions of future work with a starting point in this thesis. These are grouped into three broad categories, the first one describing Grid brokering, job management, workflows, and SLA related topics. The last two sections of this chapter discuss anticipated extensions into the areas of virtualization and cloud computing.

8.1 Grid Resource Management

The current architecture for managing advance reservations is based on the WS-Agreement specification that defines an agreement request protocol message, with acceptance or rejection as the only possible answers. This makes multi-phase negotiation difficult, as experienced in the work with coallocation. A more general negotiation framework that includes offers and counter-offers, is described in WS-AgreementNegotiation [15], but this specification is not yet ready to be used. A full-featured negotiation protocol like WS-AgreementNegotiation would enable a resource broker to efficiently negotiate sophisticated agreements and hence meet a wider range of QoS requirements.

The coallocation algorithm introduced in Paper III can be improved, including development of better techniques to avoid (or resolve) conflicts that arise due to the competition for resources between the subjobs in a coallocated job. The construction of such algorithms would benefit from the development of theoretical models for the resource selection and coallocation processes. Further studies of the respective advantages and shortcomings of off-line and on-line coallocation algorithms should build on the large body of work on locking and optimistic concurrency control within the area of transactions. In addition to these algorithmic aspects, support for coordination of subjobs within a coallocated job is required. This should include mechanisms for subjobs to synchronize themselves prior to execution at their respective resources, and should leverage previous work in the area [28, 48, 50, 95].

Papers II and III demonstrate the feasibility of integrating the job submission service with the GT4 and NorduGrid/ARC Grid middlewares. A topic of current interest is the creation of a scheduling hierarchy, where higher-level Grid schedulers (brokers) negotiate with lower-level ones [118, 202].

If market-based Grids become increasingly more popular to the extent that they become the standard mechanism for Grid resource management, the resource selection algorithms described in this thesis can be adapted to select resources taking into account also Grid-economic parameters. This could e.g., include design and implementations of mechanisms that allow the user to specify the acceptable cost-performance tradeoff, as discussed in Section 2.1.

Future work in the workflow area includes investigating the interaction between workflow enactment and workflow scheduling. More specifically, the benefits and drawbacks of the respective approaches of preplanning and just-in-time scheduling, as well as the consequences of these for the design of enactment tools could be studied. Other future directions include the study of how streaming workflows [163] could extend the previous work on coallocation. Additional topics include further efforts in applications areas such as life sciences, e.g., by extending on the results in Paper VI.

8.2 Virtualization

The concept of logically dividing a physical computer into multiple, virtual machines has been used since the 1960's [3, 146]. The last few years, techniques such as paravirtualization [49] have improved the performance [229] of virtualization technologies and hence made adoption of these more feasible [180] for performance critical areas such as Grid computing. One motivating scenario for virtualization in Grids is the application-driven demand for tailored environments, with certain preinstalled software packages. Currently, one major obstacle in many production Grids is binary compatibility. It is cumbersome to use non-trivial (i.e., dynamically linked) applications that also may depend on external software, and be available for certain platforms only. One approach to overcome these problems is dynamic deployment, where software environments are installed and applications compiled as part of the job submission process. Although studied extensively [24] by, e.g., the OGF, the dynamic deployment approach has not been adopted to any larger extent. With virtualization techniques, dynamic deployment is greatly simplified as it can be performed by booting an already installed and configured virtual machine [117]. Another advantage with virtualization in Grid environments is the enhanced sandboxing functionality achieved through separation of (virtual) hardware.

Virtualization can also be used to provide checkpointing [191] and to enable migration of running jobs. Ubiquitous availability of an efficient checkpointing mechanism would fundamentally change Grid infrastructures and hence also affect the topics studied in this thesis. For example, resource brokering algorithms need not consider preemption of jobs upon runtime expiration, as jobs can be restarted from a checkpoint. Furthermore, local batch scheduling algorithms can use aggressive backfilling and still avoid starvation of longer jobs, resulting in high utilization. Job migration mechanisms would enable job management tools to continuously monitor the performance of submitted jobs, and initiate migration upon performance declines. Another use case for checkpointing is improved fault tolerance, as failed jobs only need to restart from the last checkpoint. Checkpointing and migration mechanisms can also be used to improve Grid workflow management, e.g., by improving fault tolerance and shortening completion times for computational tasks.

8.3 Cloud Computing

Despite early enthusiasm, the Grid has yet not become the ubiquitous general purpose cyberinfrastructure used to access all types of resources. Being mostly focused on sharing of scientific instruments, high-performance computers and large-scale storage, existing Grid infrastructures do not easily meet the requirements of business applications and as a consequence, adoption outside academic environments is slow.

The currently popular vision for a general purpose infrastructure for providing IT capabilities as services goes under the name of cloud computing. Ideally, a cloud infrastructure should on demand adapt to changes in client request load by resizing itself. In order to achieve this adaptability, detailed and up to date monitoring information about server load and client performance metrics is required, as well as a (virtualization-powered) mechanism that enables the efficient migration and/or duplication of servers as needed. Notably, clients (service consumers) should ideally be unaware of the internal structure of the cloud and experience no degradation of QoS during server migration. Whereas the initial motivation for Grids was the interconnection of high-performance computers and expensive instruments mainly used in academic environments, the vision of a cloud as an infinitely scalable data center may better suited for industry needs.

Many problems, including security, accounting, and placement (resource selection) are similar in cloud and Grid environments. To become successful, research in cloud technologies should hence leverage the results obtained by the Grid community. There are however some notable differences between brokering of Grid jobs and provisioning of services in a cloud (i.e., service placement decisions). Services in a cloud may be hosted until further notice (potentially for a very long time), whereas Grid jobs typically have a known, finite duration. Performance metrics for service provisioning include minimization of SLA violations and costs (electricity, cooling etc.), and hence maximization of provider profit. Batch system scheduling (and also Grid brokering) on the other hand typically focus on response time, utilization, throughput, fairness, or a combination of these. These differences impose a series of research problems in how lessons learned from batch system oriented Grid brokering can be applied to service provisioning in time-shared cloud environments. The low-cost migration operation provided by virtualization technology justifies research on migration heuristics and cross-domain placement optimization.

8.3.1 RESERVOIR

The Resources and Services Virtualization without Barriers (RESERVOIR) project [173] is the venue for both ongoing work [224] and planned extensions to the topics studied in this thesis. The overall goal of RESERVOIR is to support service-oriented computing, in particular by dynamic provisioning of services as utilities. The prime motivation for the project is to avoid the costly over-provisioning currently used by data centers to ensure SLA compliance during peaks in demand. The RESERVOIR project will provide an open specification and a reference implementation of an infrastructure for federated clouds, where Grid and virtualization technologies along with business process management will enable efficient delivery of services. By resizing services on demand, potentially including migrating (parts of) them to other physical machines or even other, partnering data centers, a RESERVOIR infrastructure provider (a data center) will be able to optimize placement and hence minimize costs.

The design of heuristics for cost-aware SLA-compliant cross-site service placement share many similarities with the resource brokering scenarios investigated in this thesis. These similarities include complicating issues such as lack of control over remote sites, limited information about the load on these, and differences in usage policies. The existence of multiple brokers (placement decision modules) and competition among these for (potentially scarce) resources are other common characteristics of a federated cloud environment and a decentralized resource brokering scenario.

Bibliography

[1] L. Abeti, P. Ciancarini, and R. Moretti. Service oriented software engineering for modeling agents and services in grid systems. *Multiagent and Grid Systems*, 2(2):135–148, 2006.

[2] D. Abramson, R. Buyya, and J. Giddy. A computational economy for grid computing and its implementation in the Nimrod-G resource broker. *Future Generation Computer Systems*, 18(8):1061–1074, 2002.

[3] R.J. Adair, R.U. Bayles, L.W. Comeau, and R.J. Creasy. A virtual machine system for the 360/40 - Cambridge scientific center report 320. Technical report, IBM, May 1966.

[4] L. Adzigogov, J. Soldatos, and L. Polymenakos. EMPEROR: An OGSA Grid meta-scheduler based on dynamic resource predictions. *J Grid Computing*, 3(1–2):19–37, 2005.

[5] A.H. Alhusaini, V.K. Prasanna, and C.S. Raghavendra. A unified resource scheduling framework for heterogeneous computing environments. In V.K. Prasanna, editor, *8th Heterogeneous Computing Workshop, HCW'99*, pages 156–165, 1999.

[6] A. Ali, A. Anjum, J. Bunn, R. Cavanaugh, F. van Lingen, R. McClatchey, M. A. Mehmood, H. Newman, C. Steenberg, M. Thomas, and I. Willers. Predicting resource requirements of a job submission. In *Proceedings of the Conference on Computing in High Energy and Nuclear Physics (CHEP 2004), Interlaken, Switzerland*, September 2004.

[7] S. Ali, H.J. Siegel, M. Maheswaran, and D. Hensgen. Task execution time modeling for heterogeneous computing systems. In C. Raghavendra, editor, *Proceedings 9th Heterogeneous Computing Workshop (HCW 2000)*, pages 185–199, 2000.

[8] W.E. Allcock, I. Foster, and R. Madduri. Reliable data transport: A critical service for the Grid. Building Service Based Grids Workshop, GGF 11. Available at: www.globus.org/alliance/publications/papers/GGF11_RFTV-Final.pdf.

[9] G. Allen, K. Davis, K.N. Dolkas, N.D. Doulamis, T.Goodale, T.Kielmann, A. Merzky, J. Nabrzyski, J. Pukacki, T. Radke, M. Russell, E. Seidel, J. Shalf, and I. Taylor. Enabling applications on the grid - a gridlab overview. *Int. J. High Perf. Comput. Appl.*, 17(4):449–466, 2003.

[10] G. Allen, K. Davis, T. Goodale, A. Hutanu, H. Kaiser, T. Kielmann, A Merzky, R. van Nieuwpoort, A. Reinefeld, F. Schintke, T. Schott, E. Seidel, and B. Ullmer. The Grid Application Toolkit: Toward generic and easy application programming interfaces for the Grid. *Proceedings of the IEEE*, 93(3):534–550, 2205.

[11] G. Allen, T. Dramlitsch, I. Foster, N. Karonis, M. Ripeanu, E. Seidel, and B. Toonen. Supporting efficient execution in heterogeneous distributed computing environments with Cactus and Globus. In *Proceedings of Supercomputing 2001*, 2001.

[12] Amazon Elastic Compute Cloud (Amazon EC2). aws.amazon.com/ec2, June 2008.

[13] S. Andreozzi, S. Burke, F. Ehm, L. Field, G. Galang, B. Konya, M. Litmaath, P. Millar, and J.P. Navarro. GLUE specification v. 2.0. http://www.ogf.org/Public_Comment_Docs/Documents/2008-06/ogfglue2rendering.pdf, August 2008.

[14] A. Andrieux, K. Czajkowski, A. Dan, K. Keahey, H. Ludwig, T. Nakata, J. Pruyne, J. Rofrano, S. Tuecke, and M. Xu. Web services agreement specification (WS-Agreement). https://forge.gridforum.org/sf/docman/do/downloadDocument/projects.graap-wg/docman.root.current_drafts/doc6090, November 2006.

[15] A. Andrieux, K. Czajkowski, A. Dan, K. Keahey, H. Ludwig, J. Pruyne, J. Rofrano, S. Tuecke, and M. Xu. Web Services Agreement Negotiation Specification (WS-AgreementNegotiation). https://forge.gridforum.org/sf/go/doc6092?nav=1, November 2006.

[16] A. Anjomshoaa, F. Brisard, M. Drescher, D. Fellows, A. Ly, A. S. McGough, D. Pulsipher, and A. Savva. Job Submission Description Language (JSDL) specification, version 1.0. http://www.ogf.org/documents/GFD.136.pdf, August 2008.

[17] The Apache software foundation. The Apache ant project. http://ant.apache.org/, September 2008.

[18] C.G. Atkeson, A.W. Moore, and S. Schaal. Locally weighted learning. *Artificial Intelligence Review*, 11(1–5):11–73, 1997.

[19] F. Azzedin, M. Maheswaran, and N. Arnason. A synchronous co-allocation mechanism for Grid computing systems. *Cluster Computing*, 7(1):39–49, 2004.

[20] Emir M. Bahsi, Emrah Ceyhan, and Tevfik Kosar. Conditional workflow management: A survey and analysis. *Scientific Programming*, 15(4):283–297, 2007.

[21] X. Bai, L. Bölöni, D.C. Marinescu, H.J. Siegel, R.A. Daley, and I-J. Wang. A brokering framework for large-scale heterogeneous systems. In *Parallel and Distributed Processing Symposium, IPDPS 2006*, 2006.

[22] C. Baru, R. Moore, A. Rajasekar, and M. Wan. The SDSC Storage Resource Broker. In S.A. MacKay and J.H. Johnson, editors, *Proceedings of CASCON'98*, 1998.

[23] J. Basney, M. Humphrey, and V. Welch. The MyProxy online credential repository. *Softw. Pract. Exper.*, 35(9):801–816, 2005.

[24] D. Bella, T. Kojo, P. Goldsack, S. Loughran, D. Milojicic, S. Schaefer, J. Tatemura, and P. Toft. Configuration description, deployment, and lifecycle management (CDDLM) foundation document. www.ogf.org/documents/GFD.50.pdf, August 2008.

[25] A. Bellouma, E. Deelman, and Z. Zhao (Eds.). Scientific workflows. *Scientific Programming*, 14(3–4), 2006.

[26] E. Benson, G. Wasson, and M. Humphrey. Evaluation of UDDI as a provider of resource discovery services for OGSA-based grids. In *Parallel and Distributed Processing Symposium, IPDPS 2006*, 2006.

[27] F. Berman, G.C. Fox, and A.J.G Hey (editors). *Grid computing: making the global infrastructure a reality.* John Wiley and Sons Ltd, 2003.

[28] B. Bierbaum, C. Clauss, M. Pöppel, S. Lankes, and Thomas Bemmerl. The new multidevice architecture of metampich in the context of other approaches to grid-enabled mpi. In B. Mohr, J. Larsson Träff, J. Worringen, and J. Dongarra, editors, *Recent Advances in Parallel Virtual Machine and Message Passing Interface, LNCS 4192*, pages 184–193, 2006.

[29] S. Boag, D. Chamberlin, M.F. Fernández, D. Florescu, J. Robie, and J. Siméon (editors). XQuery 1.0: An XML Query Language. www.w3.org/TR/xquery, August 2008.

[30] N. Bobroff, L. Fong, S. Kalayci, Y. Liu, J.C. Martinez, I. Rodero S.M. Sadjadi, and D. Villegas. Enabling interoperability among meta-schedulers. In T. Priol et al., editors, *CCGRID 2008 Eighth IEEE International Symposium on Cluster Computing and the Grid*, pages 306–315, 2008.

[31] R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin. Resource ReSerVation Protocol (RSVP). http://tools.ietf.org/html/rfc2205, August 2008.

[32] E.-K. Byun and J.-S. Kim. Dynagrid: An adaptive, scalable, and reliable resource provisioning framework for WSRF-compliant applications. *J Grid Computing*, 2008. To appear, DOI:10.1007/s10723-008-9107-y.

[33] L.-C. Canon, E. Jeannot, R Sakellariou, and W. Zheng. Comparative evaluation of the robustness of dag scheduling heuristics. In S. Gorlatch, P. Fragopoulou, and T. Priol, editors, *Grid computing - achiements and prospects*, pages 63–74, 2008.

[34] H. Casanova and J. Dongarra. Netsolve: A network server for solving computational science problems. *Int. J. Supercomput. Appl.*, 11(3):212–223, 1997.

[35] H. Casanova, G. Obertelli, F. Berman, and R. Wolski. The AppLeS Parameter Sweep Template: User-level middleware for the Grid m{1}. *Scientific Programming*, 8(3), 2000.

[36] C Castillo. *On the Design of Efficient Resource Allocation Mechanisms for Grids*. PhD thesis, North Carolina State University, 2008.

[37] C. Castillo, G.N. Rouskas, and K. Harfoush. On the design of online scheduling algorithms for advance reservations and qos in grids. In *2007 IEEE International Parallel and Distributed Processing Symposium*, 2007.

[38] CERN. CASTOR CERN Advanced STORage manager. http://www.cern.ch/castor, August 2008.

[39] A Chakrabarti, R. Dheepak, and S. Sengupta. Integration of scheduling and replication in data grids. In L. Bougé et al., editors, *High Performance Computing - HiPC 2004, LNCS 3296*, pages 375–385, 2004.

[40] C. Chapman, M. Musolesi, W. Emmerich, and C. Mascolo. Predictive resource scheduling in computational Grids. In *Parallel and Distributed Processing Symposium, IPDPS 2007*, 2007.

[41] K. Chard and K. Bubendorfer. A distributed economic meta-scheduler for the Grid. In T. Priol et al., editors, *CCGRID 2008 Eighth IEEE International Symposium on Cluster Computing and the Grid*, pages 542–547, 2008.

[42] A.L. Chervenak, N. Palavalli, S. Bharathi, C. Kesselman, and R. Schwartzkopf. Performance and scalability of a replica location service. In *Proceedings of the International IEEE Symposium on High Performance Distributed Computing (HPDC-13)*, 2004.

[43] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web Services Description Language (WSDL) 1.1. http://www.w3.org/TR/wsdl, June 2008.

[44] D. Churches, G. Gombas, A. Harrison, J. Maassen, C. Robinson, M. Shields, I. Taylor, and I. Wang. Programming scientific and distributed workflow with Triana services. *Concurrency Computat.: Pract. Exper.*, 18(10):1021–1037, 2006.

[45] J. Clark and S. DeRose (editors). XML Path Language (XPath) version 1.0. http://www.w3.org/TR/xpath, May 2008.

[46] The NextGRID consortium. NextGRID: Architecture for next generation grids. www.nextgrid.org, August 2008.

[47] CoreGRID. CoreGRID annual report 2007. http://www.coregrid.net/mambo/content/view/310/301/, May 2008.

[48] C. Coti, T. Herault, and S. Peyronnet. Grid services for MPI. In T. Priol et al., editors, *CCGRID 2008 Eighth IEEE International Symposium on Cluster Computing and the Grid*, pages 417–424, 2008.

[49] S. Crosby and D. Brown. The virtualization reality. *ACM Queue*, pages 34–41, December/January 2006-2007.

[50] K. Czajkowski, I. Foster, and C. Kesselman. Resource co-allocation in computational Grids. In *Proceedings of the Eighth IEEE International Symposium on High Performance Distributed Computing (HPDC-8)*, pages 219–228, 1999.

[51] J. Decker and J. Schneider. Heuristic scheduling of grid workflows supporting co-allocation and advance reservation. In B. Schulze et al., editors, *Seventh IEEE International Symposium on Cluster Computing and the Grid (CCGrid '07)*, pages 335–342. IEEE Computer Society, 2007.

[52] E. Deelman. Looking into the future of workflows: the challenges ahead. In I. Taylor et al., editors, *Workflows for e-Science*, pages 475–481. Springer-Verlag, 2007.

[53] E. Deelman, D. Gannon, M. Shields, and I. Taylor. Workflows and e-science: An overview of workflow system features and capabilities. *Future Generation Computer Systems*, 25(5):528–540, 2009.

[54] E. Deelman, G. Singh, M. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G.B. Berriman, J. Good, A. Laity, J.C. Jacob, and D.S. Katz. Pegasus: a framework for mapping complex scientific workflows onto distributed systems. *Scientific Programming*, 13(3):219–237, 2005.

[55] E. Deelman and I. Taylor. Special issue on scientific workflows. *Journal of Grid computing*, 3(3–4):151–304, 2005.

[56] T. Delaitre, T. Kiss, A. Goyeneche, G. Terstyanszky, S. Winter, and P. Kacsuk. GEMLCA: Running legacy code applications as grid services. *Journal of Grid Computing*, 3(1–2):75–90, 2005.

[57] T. Dierks and C. Allen. The TLS protocol version 1.0. http://www.ietf.org/rfc/rfc2246.txt, May 2008.

[58] J. Clark (editor). W3C XSL Transformations (XSLT) Version 1.0. www.w3.org/TR/xslt.html, August 2008.

[59] J. Treadwell (Editor). Open grid services architecture glossary of terms version 1.5 (gfd81). http://www.ogf.org/documents/GFD.81.pdf, May 2008.

[60] K. Zeilenga (editor). Lightweight Directory Access Protocol (LDAP): Technical specification road map. http://www.ietf.org/rfc/rfc4510.txt, November 2006.

[61] W. Allcock (editor). GridFTP: Protocol extensions to FTP for the Grid. http://www.ogf.org/documents/GFD.20.pdf, May 2008.

[62] EGEE. gLite - lightweight middleware for grid computing. www.cern.ch/glite, August 2008.

[63] EGEE. JRA1: workload management. http://egee-jra1-wm.mi.infn.it/egee-jra1-wm/, September 2008.

[64] A. Elghirani, R. Subrata, and A.Y. Zomaya. Intelligent scheduling and replication in datagrids: a synergistic approach. In B. Schulze et al., editors, *Seventh IEEE International Symposium on Cluster Computing and the Grid (CCGrid '07)*, pages 179–182, 2007.

[65] M. Ellert, M. Grønager, A. Konstantinov, B. Kónya, J. Lindemann, I. Livenson, J. L. Nielsen, M. Niinimäki, O. Smirnova, and A. Wäänänen. Advanced resource connector middleware for lightweight computational Grids. *Future Generation Computer Systems*, 27(2):219–240, 2007.

[66] E. Elmroth and P. Gardfjäll. Design and evaluation of a decentralized system for Grid-wide fairshare scheduling. In H. Stockinger et al., editors, *First International Conference on e-Science and Grid Computing*, pages 221–229. IEEE CS Press, 2005.

[67] E. Elmroth and J. Tordsson. A Grid resource broker supporting advance reservations and benchmark-based resource selection. In J. Dongarra, K. Madsen, and J. Waśniewski, editors, *Applied Parallel Computing - State of the Art in Scientific Computing, LNCS 3732*, pages 1061–1070, 2006.

[68] European Telecommunications Standards Institute. ETSI GRID. http://portal.etsi.org/grid, August 2008.

[69] T. Fahringer, R. Prodan, R. Duan, F. Nerieri, S. Podlipnig, J. Qin, M. Siddiqui, H.-L. Truong, A. Villazon, and M. Wieczorek. ASKALON: A Grid Application Development and Computing Environment. In *6th International Workshop on Grid Computing*, pages 122–131. IEEE, 2005.

[70] T. Fahringer, R. Prodan, R.Duan, J. Hofer, F. Nadeem, F. Nerieri, S. Podlipnig, J. Qin, M. Siddiqui, H.-L. Truong, A. Villazon, and M. Wieczorek. ASKALON: A development and Grid computing environment for scientific workflows. In I. Taylor et al., editors, *Workflows for e-Science*, pages 450–471. Springer-Verlag, 2007.

[71] U. Farooq, S. Majumdar, and E. W. Parsons. Impact of laxity on scheduling with advance reservations in Grids. In *MASCOTS '05: Proceedings of the 13th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, pages 319–324, Washington, DC, USA, 2005. IEEE Computer Society.

[72] International Organization for Standardization. ISO/IEC 2382-1 information technology - vocabulary - part 1: Fundamental terms, 1993.

[73] The Common Component Architecture Forum. The Common Component Architecture forum. http://www.cca-forum.org/, September 2008.

[74] I. Foster. What is the Grid? a three point checklist. www-fp.mcs.anl.gov/~foster/Articles/WhatIsTheGrid.pdf, May 2008.

[75] I. Foster. Globus toolkit version 4: Software for service-oriented systems. In H. Jin et al., editors, *IFIP International Conference on Network and Parallel Computing,* LNCS 3779, pages 2–13. Springer-Verlag, 2005.

[76] I. Foster. Globus toolkit version 4: Software for service-oriented systems. In H. Jin, D. Reed, and W. Jiang, editors, *IFIP International Conference on Network and Parallel Computing, LNCS 3779*, pages 2–13, 2006.

[77] I. Foster and C. Kesselman (editors). *The GRID: Blueprint for a New Computing Infrastructure.* Morgan Kaufmann Publishers, Inc., 1999.

[78] I. Foster, J. Geisler, W. Nickless, W. Smith, and S. Tuecke. Software infrastructure for the I-WAY high performance distributed computing experiment. In *Proc. 5th IEEE Symposium on High Performance Distributed Computing*, pages 562–571, 1997.

[79] I. Foster, A. Grimshaw, P. Lane, W. Lee, M. Morgan, S. Newhouse, S. Pickles, D. Pulsipher, C. Smith, and M. Theimer. OGSA© basic execution service version 1.0. http://www.ogf.org/documents/GFD.108.pdf, August 2008.

[80] I. Foster, C. Kesselman, C. Lee, B. Lindell, K. Nahrstedt, and A. Roy. A distributed resource management architecture that supports advance reservations and co-allocation. In M. Zitterbart and G. Carle, editors, *7th International Workshop on Quality of Service*, pages 27–36. IEEE, 1999.

[81] I. Foster, C. Kesselman, G. Tsudik, and S. Tuecke. A security architecture for computational Grids. In *Proc. 5th ACM Conference on Computer and Communications Security Conference*, pages 83–92, 1998.

[82] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the Grid: Enabling scalable virtual organizations. *Int. J. Supercomput. Appl.*, 15(3), 2001.

[83] I. Foster, H. Kishimoto, A. Savva, D. Berry, A. Grimshaw, B. Horn, F. Maciel, F. Siebenlist, R. Subramaniam, J. Treadwell, and J. Von Reich. The Open Grid Services Architecture, version 1.5, 2006. http://www.ogf.org/documents/GFD.80.pdf, October 2007.

[84] I. Foster and S. Tuecke. Describing the elephant: The different faces of IT as service. *ACM Queue*, 3(6):26–34, 2005.

[85] Geoffrey C. Fox and Dennis Gannon. Special issue: Workflow in grid systems. *Concurrency Computat.: Pract. Exper.*, 18(10):1009–1331, 2006.

[86] D. Gannon, R. Bramley, G. Fox, S. Smallen, A. Rossi, R. Ananthakrishnan, F. Bertrand, K. Chiu, M. Farrellee, M. Govindaraju, S. Krishnan, L. Ramakrishnan, Y. Simmhan, A. Slominski, Y. Ma, C. Olariu, and N. Rey-Cenvaz. Programming the Grid: Distributed software components, P2P and Grid Web Services for scientific applications. *Cluster Computing*, 5(3):325–336, 2002.

[87] P. Gardfjäll, E. Elmroth, L. Johnsson, O. Mulmo, and T. Sandholm. Scalable Grid-wide capacity allocation with the SweGrid Accounting System (SGAS). *Concurrency Computat.: Pract. Exper.*, 20(18):2089–2122, 2008.

[88] T. Glatard, J. Montagnat, D. Lingrand, and X. Pennec. Flexible and efficient workflow deployement of data-intensive applications on grids with MOTEUR. *Int. J. High Perf. Comput. Appl.*, 22(3):347–360, 2008.

[89] T. Glatard, G. Sipos, J. Montagnat, Z. Farkas, and P. Kacsuk. Workflow-level parametric study support by MOTEUR and the P-GRADE portal. In I. Taylor et al., editors, *Workflows for e-Science*, pages 279–299. Springer-Verlag, 2007.

[90] Globus. http://www.globus.org. February 2009.

[91] A. Goyeneche, G. Terstyanszky, T. Delaitre, and S. Winter. Improving Grid computing performance prediction using weighted templates. In S. Cox, editor, *Proceedings of the UK e-Science All Hands Meeting 2007*, pages 361–368, 2007.

[92] S. Graham and B. Murray (editors). Web Services Base Notification 1.2 (WS-BaseNotification). http://docs.oasis-open.org/wsn/2004/06/wsn-WS-BaseNotification-1.2-draft-03.pdf, June 2008.

[93] S. Graham and J. Treadwell (editors). Web Services Resource Properties 1.2 (WS-ResourceProperties). http://docs.oasis-open.org/wsrf/wsrf-ws_resource_properties-1.2-spec-os.pdf, June 2008.

[94] S. Graham, A. Karmarkar, J. Mischkinsky, I. Robinson, and I. Sedukhin (editors). Web Services Resource 1.2 (WS-Resource). http://docs.oasis-open.org/wsrf/wsrf-ws_resource-1.2-spec-os.pdf, June 2008.

[95] F. Gregoretti, G. Laccetti, A. Murli, G. Oliva, and U. Scafuri. MGF: A grid-enabled MPI library. *Future Generation Computer Systems*, 24(2):158–165, 2008.

[96] Grid Interoperability Now. http://wiki.nesc.ac.uk/read/gin-jobs. September 2006.

[97] C. Grimme, J. Lepping, A. Papaspyrou, P. Wieder, R. Yahyapour, A. Oleksiak, O. Wäldrich, and W. Ziegler. Towards a standards-based grid scheduling architecture. In S. Gorlatch, P. Fragopoulou, and T. Priol, editors, *Grid computing - achiements and prospects*, pages 147–158, 2008.

[98] A. S. Grimshaw and W. A. Wulf. The Legion vision of a worldwide virtual computer. *Communications of the ACM*, 40(1):39–45, 1997.

[99] R. Gruber, V. Keller, P. Kuonen, M-C. Sawley, B. Schaeli, A. Tolou, M. Torruella, and T-M. Tran. Towards an intelligent Grid scheduling system. In R. Wyrzykowski, J. Dongarra, N. Meyer, and J. Wasniewski, editors, *Parallel Processing and Applied Mathematics, LNCS 3911*, pages 751–757. Springer Verlag, 2005.

[100] M. Gudgin, M. Hadley, N. Mendelsohn, J-J. Moreau, H. Frystyk Nielsen, A. Karmarkar, and Y. Lafon. SOAP version 1.2 part 1: Messaging framework. http://www.w3.org/TR/soap12-part1/, June 2008.

[101] P. Hasselmeyer, H. Mersch, B. Koller, H.-N. Quyen, L. Schubert, and Ph. Wieder. Implementing an SLA negotiation framework. In *Exploiting the Knowledge Economy: Issues, Applications, Case Studies (eChallenges 2007)*, 2007.

[102] F. Hernández, P. Bangalore, J. Gray, Z. Guan, and K. Reilly. GAUGE: Grid Automation and Generative Environment. *Concurrency Computat.: Pract. Exper.*, 18(10):1293–1316, 2006.

[103] T. Hey and A.E. Trefethen. The UK e-Science Core Programme and the Grid. *Future Generation Computer Systems*, 18(8):1017–1031, 2002.

[104] A. Hoheisel. User tools and languages for graph-based Grid workflows. *Concurrency Computat.: Pract. Exper.*, 18(10):1101–1113, 2006.

[105] R. Housley, W. Polk, W. Ford, and D. Solo. Internet X.509 public key infrastructure certificate and certificate revocation list (CRL) profile. http://www.ietf.org/rfc/rfc3280.txt, May 2008.

[106] http://uddi.xml.org/. UDDI. June 2008.

[107] http://www.sun.com/service/sungrid/SunGridUG.pdf. Sun^{TM} Grid compute utility - reference guide. August 2008.

[108] R. Huang, H. Casanova, and A.A. Chien. Using Virtual grids to simplify application scheduling. In *Parallel and Distributed Processing Symposium, IPDPS 2006*, 2006.

[109] E. Huedo, R.S. Montero, and I.M. Llorente. A framework for adaptive execution on Grids. *Softw. Pract. Exper.*, 34(7):631–651, 2004.

[110] Cluster Resources inc. Torque resource manager. http://www.clusterresources.com/pages/products/torque-resource-manager.php, August 2008.

[111] Institute of Electrical and Electronics Engineers. IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries., 1990.

[112] International Organization for Standardization. Information technology – database languages – SQL – part 1: Framework (SQL/framework). ISO/IEC 9075-1:2003.

[113] J. Joseph, M. Ernest, and C. Fellenstein. Evolution of grid computing architecture and grid adoption models. *IBM Systems journal*, 43(4), 2004.

[114] P. Kacsuk, G. Dozsa, J. Kovcs, R. Lovas, N. Podhorszki, Z. Balaton, and G. Gombas. P-GRADE: a grid programming environment. *Journal of Grid Computing*, 1(2):171–197, 2003.

[115] S. Kannan, P. Mayes, M. Roberts, D. Brelsford, and J. Skovira. *Workload Management with LoadLeveler.* IBM Corp., 2001.

[116] N. H. Kapadia, J. A. B. Fortes, and C. E. Brodley. Predictive application-performance modeling in a computational grid environment. In *Proceedings of the Eighth IEEE International Symposium on High Performance Distributed Computing (HPDC-8)*, pages 71–80, 1999.

[117] K. Keahey, I. Foster, T. Freeman, and X. Zhang. Virtual workspaces: Achieving quality of service and quality of life in the Grid. *Scientific Programming*, 13(4):265–276, 2005.

[118] A. Kertesz and P. Kacsuk. Meta-broker for future generation grids: A new approach for a high-level interoperable resource management. In D. Talia et al., editors, *Proceedings of the CoreGRID Workshop on Grid Middleware*, 2007.

[119] A. Kertész, I. Rodero, and F. Guim. BPDL: A data model for grid resource broker capabilities. Technical report, CoreGRID, 2007. http://www.coregrid.net/mambo/images/stories/TechnicalReports/tr-0074.pdf, August 2008.

[120] G. Khanna, U. Catalyurek, T. Kurc, R. Kettimuthu, P. Sadayappan, and J. Saltz. A dynamic scheduling approach for coordinated wide-area data transfers using GridFTP. In *Parallel and Distributed Processing Symposium, IPDPS 2008*, 2008.

[121] T. Kosar and M. Livny. A framework for reliable and efficient data placement in distributed computing systems. *Journal of Parallel and Distributed Computing*, 65(10):1146–1157, 2005.

[122] K. Krauter, R. Buyya, and M. Maheswaran. A taxonomy and survey of Grid resource management systems for distributed computing. *Softw. Pract. Exper.*, 32(2):135–164, 2002.

[123] K. Kurowski, B. Ludwiczak, J. Nabrzyski, A. Oleksiak, and J. Pukacki. Dynamic grid scheduling with job migration and rescheduling in the Grid-Lab resource management system. *Scientific Programming*, 12(4):263–273, 2004.

[124] H. Lamehamedi, B. Szymanski, Z. Shentu, and E. Deelman. Data replication strategies in grid environments. In *ICA3PP'02: Proceedings of the Fifth International Conference on Algorithms and Architectures for Parallel Processing*, pages 378–383, 2002.

[125] M. Lei, S.V. Vrbsky, and Z. Qi. Online Grid replication optimizers to improve system reliability. In *Parallel and Distributed Processing Symposium, IPDPS 2007*, 2007.

[126] P. Leong, C. Miao, and B-S. Lee. A survey of agent-oriented software engineering for service-oriented computing. *International Journal of Web Engineering and Technology*, 4(3):367–385, 2008.

[127] H. Li, J. Chen, Y. Tao, D. Groep, and L. Wolters. Improving a local learning technique for queue wait time predictions. In S.J. Turner et al., editors, *Sixth IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2006)*, pages 335–342, 2006.

[128] H. Li, D. Groep, and L. Wolters. Efficient response time predictions by exploiting application and resource state similarities. In *6th International Workshop on Grid Computing (GRID 2005)*, pages 234–241, 2005.

[129] H. Li, D. Groep, L. Wolters, and J. Templon. Job failure analysis and its implications in a large-scale production grid. In *Second IEEE International Conference on e-Science and Grid Computing*. IEEE CS Press, 2006.

[130] A. Litke, D. Skoutas, K. Tserpes, and T. Varvarigou. Efficient task replication and management for adaptive fault tolerance in mobile Grid environments. *Future Generation Computer Systems*, 23(2):163–178, 2007.

[131] L. Liu and S. Meder (editors). Web Services Base Faults 1.2 (WS-BaseFaults). http://docs.oasis-open.org/wsrf/wsrf-ws_base_faults-1.2-spec-os.pdf, June 2008.

[132] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E.A. Leen, J. Tao, and Y. Zhao. Scientific workflow management and the Kepler system. *Concurrency Computat.: Pract. Exper.*, 18(10):1039–1065, 2006.

[133] B. Ludäscher and C. Goble. Special issue on scientific workflows. *SIGMOD Record*, 34(3), 2005.

[134] J. MacLaren. Advance reservations state of the art. http://www.fz-juelich.de/zam/RD/coop/ggf/graap/sched-graap-2.0.html, September 2006.

[135] J. MacLaren. HARC: the highly-available resource co-allocator". In *Proceedings of GADA'07, LNCS 4804*, pages 1385–1402, 2007.

[136] R.K. Madduri, C.S. Hood, and W.E. Allcock. Reliable file transfer in grid environments. In A. Jacobs, editor, *27th Annual IEEE Conference on Local Computer Networks (LCN)*, pages 737–738, 2002.

[137] T. Maguire, D. Snelling, and T. Banks (editors). Web Services Service Group 1.2 (WS-ServiceGroup). http://docs.oasis-open.org/wsrf/wsrf-ws_service_group-1.2-spec-os.pdf, June 2008.

[138] M.W. Margo, K. Yoshimoto, P. Kovatch, and P. Andrews. Impact of reservations on production job scheduling. In E. Frachtenberg and U. Schwiegelshohn, editors, *Job Scheduling Strategies for Parallel Processing*, pages 116–131. Springer, 2008. LNCS 4942.

[139] G. Mateescu. Quality of Service on the Grid via metascheduling with resource co-scheduling and co-reservation. *Int. J. High Perf. Comput. Appl.*, 17(3):209–218, Fall 2003.

[140] C. Mateos, A. Zunino, and M. Campo. A survey on approaches to gridification. *Softw. Pract. Exper.*, 38(5):523–556, 2008.

[141] A. S. McGough, A. Afzal, J. Darlington, N. Furmento, A. Mayer, and L. Young. Making the Grid predictable through reservations and performance modelling. *The Computer Journal*, 48(3):358–368, 2005.

[142] A.S. McGough, W. Lee, J. Cohen, E. Katsiri, and J. Darlington. ICENI. In I. Taylor et al., editors, *Workflows for e-Science*, pages 395–415. Springer-Verlag, 2007.

[143] A.S. McGough and A. Savva. Implementation and interoperability experiences with the Job Submission Description Language(JSDL). http://www.ogf.org/Public_Comment_Docs/Documents/2008-07/draft-gwdejsdlexperience1.0007.pdf, August 2008.

[144] W. Lee A.S. McGough and J. Darlington. Performance evaluation of the GridSAM job submission and monitoring system. In S. Cox and D.W. Walker, editors, *Proceedings of the UK e-Science All Hands Meeting 2005*, pages 915–922, 2005.

[145] Message Passing Interface Forum. http://www.mpi-forum.org, June 2008.

[146] R.A. Meyer and L.H. Seawright. A virtual machine time-sharing system. *IBM Systems Journal*, 9(3):199–218, 1970.

[147] H. H. Mohamed and D. H. J. Epema. Experiences with the KOALA co-allocating scheduler in multiclusters. In *Proceedings of the International Symposium on Cluster Computing and the Grid (CCGRID2005)*, pages 784–791. IEEE Computer Society, 2005.

[148] G. Moltó, V. Hernández, and J.M. Alonso. A service-oriented WSRF-based architecture for metascheduling on computational grids. *Future Generation Computer Systems*, 24(4):317–328, 2008.

[149] M. Morgan. ByteIO specification 1.0. http://www.ogf.org/documents/GFD.87.pdf, August 2008.

[150] M.M. Morgan and A.S. Grimshaw. Genesis II - standards based grid computing. In B. Schulze et al., editors, *Seventh IEEE International Symposium on Cluster Computing and the Grid (CCGrid '07)*, pages 611–618. IEEE Computer Society, 2007.

[151] M.A.S. Netto and R. Buyya. Rescheduling co-allocation requests based on flexible advance reservations and processor remapping. In *Proceedings of the 9th IEEE International Conference on Grid Computing, (Grid 2008)*, 2008.

[152] C. Neuman, T. Yu, S. Hartman, and K. Raeburn. The Kerberos network authentication service (v5). http://www.ietf.org/rfc/rfc4120.txt, May 2008.

[153] S. Newhouse and A. Grimshaw. Independent software vendors (ISV) remote computing usage primer. http://www.ogf.org/Public_Comment_Docs/Documents/200807/ISV-V93.pdf, August 2008.

[154] E. Di Nitto, R.J. Hall, J. Han, Y. Han, A. Polini, K. Sandkuhl, and A. Zisman (editors). *The 2006 International Workshop on Service Oriented Software Engineering (IW-SOSE'06).* ACM, 2006.

[155] OASIS. OASIS Web Services Resource Framework (WSRF) TC. http://www.oasis-open.org/committees/wsrf/, October 2007.

[156] OASIS Open. Reference Model for Service Oriented Architecture 1.0. http://www.oasis-open.org/committees/download.php/19679/soa-rm-cs.pdf, June 2008.

[157] T. Oinn, M. Addis, J. Ferris, D. Marvin, M. Senger, M. Greenwood, T. Carver, K. Glover, M.R. Pocock, A. Wipat, and P. Li. Taverna: A tool for the composition and enactment of bioinformatics workflows. *Bioinformatics*, 20(17):3045–3054, 2004.

[158] T. Oinn, M. Greenwood, M. Addis, M. Nedim Alpdemir, J. Ferris, K. Glover, C. Goble, A. Goderis, D. Hull, D. Marvin, P. Li, P. Lord, M.R. Pocock, M. Senger, R. Stevens, A. Wipat, and C. Wroe. Taverna: Lessons in creating a workflow environment for the life sciences. *Concurrency Computat.: Pract. Exper.*, 18(10):1067–1100, 2006.

[159] OMII Europe. OMII Europe - open middleware infrastructure institute. http://omii-europe.org, August 2008.

[160] M.P. Papazoglou. *Web services: principles and technology.* Pearson Education Limited, 2008.

[161] M.P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann. Service-Oriented Computing: State of the art and research challenges. *Computer*, 40(11):38–45, 2007.

[162] S.G. Parker, D.M. Weinstein, and C.R. Johnson. The SCIRun computational steering software system. In E. Arge et al., editors, *Modern Software Tools in Scientific Computing*, pages 1–40. Birkhauser press, 1997.

[163] C. Pautasso and G. Alonso. Parallel computing patterns for grid workflows. In *Proc. of the HPDC2006 Workshop on Workflows in Support of Large-Scale Science (WORKS06) Paris France*, June 2006.

[164] M. Pereira, O. Tatebe, L. Luan, and T. Anderson. Resource namespace service specification. http://www.ogf.org/documents/GFD.101.pdf, August 2008.

[165] G. Pierantoni, B. Coghlan, E. Kenny, O. Lyttleton, D. O'Callaghan, and G. Quigley. Interoperability using a Metagrid Architecture. In *ExpGrid workshop at HPDC2006 The 15th IEEE International Symposium on High Performance Distributed Computing*, Paris, France, February 2006.

[166] M.L. Pinedo. *Scheduling — Theory, Algorithms, and Systems.* Springer Verlag, 2008.

[167] K. Plankensteiner, R. Prodan, T. Fahringer Attila Kertész, and P. Kacsuk. Fault-tolerant behavior in state-of-the-art Grid workflow management systems. Technical report, CoreGRID, 2007. http://www.coregrid.net/mambo/images/stories/TechnicalReports/tr-0091.pdf, August 2008.

[168] A. Pugliese, D. Talia, and R. Yahyapour. Modeling and supporting grid scheduling. *J Grid Computing*, 6(2):195–213, 2008.

[169] C. Qu. A Grid advance reservation framework for co-allocation and co-reservation across heterogeneous local resource management systems. In R. Wyrzykowski et al., editors, *PPAM 2007*, 2007.

[170] I. Raicu, Y. Zhao, C. Dumitrescu, I. Foster, and M. Wilde. Falkon: a Fast and Light-weight tasK executiON framework. In *Proceedings of IEEE/ACM Supercomputing 07*, 2007.

[171] H. Rajic, R. Borbst, W. Chan, F. Ferstl, J. Gardiner, A. Haas, B. Nitzberg, D. Templeton, J. Tollefsrud, and P. Tröger. Distributed resource management application API specification 1.0. http://www.ogf.org/documents/GFD.133.pdf, August 2008.

[172] K. Ranganathan and I. Foster. Simulation studies of computation and data scheduling algorithms for data grids. *Journal of Grid computing*, 1(1):53–62, 2003.

[173] Reservoir. Reservoir. http://www.reservoir-fp7.eu/, September 2008.

[174] M. Riedel and D. Mallmann. Standardization processes of the UNICORE Grid system. In J. Volkert et al., editors, *Proceedings of 1st Austrian Grid Symposium*, pages 191–203, 2005.

[175] T. Röblitz and A. Reinefeld. Co-reservation with the concept of virtual resources. In *Proceedings of the Fifth IEEE International Symposium on Cluster Computing and the Grid (CCGrid'05)*, pages 398–406, 2005.

[176] I. Rodero, J. Corbalán, R. M. Badia, and J. Labarta. eNANOS Grid Resource Broker. In P. M. A. Sloot, A. G. Hoekstra, T. Priol, A. Reinefeld, and M. Bubak, editors, *Advances in Grid Computing - EGC 2005, LNCS 3470*, pages 111–121, 2005.

[177] I. Rodero, F. Guim, J. Corbalan, L.L. Fong, Y.G. Liu, and S.M. Sadjadi. Looking for an evolution of Grid scheduling: Meta-brokering. In *Proceedings of the Second CoreGRID Workshop on Middleware at ISC2007 (CoreGRID-2007)*, 2007.

[178] H.G. Rotithor. Taxonomy of dynamic task scheduling schemes in distributed computing systems. *IEE Proceedings of Computers and Digital Techniques*, 141(1):1–10, 1994.

[179] D. De Roure, C. Goble, and R. Stevens. The design and realisation of the $^{\mathrm{my}}$Experiment virtual research environment for social sharing of workflows. *Future Generation Computer Systems*, 2008. To appear, DOI:10.1016/j.future.2006.06.010.

[180] A.J. Rubio-Montero, E. Huedo, R.S. Montero, and I.M. Llorente. Management of virtual machines on globus Grids using GridWay. In *Parallel and Distributed Processing Symposium, IPDPS 2007*, 2007.

[181] J.M. Schopf. Ten actions when Grid scheduling. In J. Nabrzyski, J.M. Schopf, and J. Węglarz, editors, *Grid Resource Management State of the art and future trends*, chapter 2. Kluwer Academic Publishers, 2004.

[182] U. Schwiegelshohn, A. Tchernykh, and R. Yahyapour. Online scheduling in grids. In *Parallel and Distributed Processing Symposium, IPDPS 2008*, 2008.

[183] J. Seidel, O. Wäldrich, P. Wieder, Philipp, R. Yahyapour, and W. Ziegler. Using SLA for resource management and scheduling - a survey. In Domenico Talia, Ramin Yahyapour, and Wolfgang Ziegler, editors, *Grid Middleware and Services - Challenges and Solutions*, CoreGRID Series. Springer, 2008. Also published as CoreGRID Technical Report TR-0096.

[184] M. Siddiqui, A. Villazón, and T. Fahringer. Grid capacity planning with negotiation-based advance reservation for optimized QoS. In *the 2006 ACM/IEEE Conference on Supercomputing SC—06*, 2006.

[185] W. Smith. Improving resource selection and scheduling using predictions. In J. Nabrzyski, J.M. Schopf, and J. Węglarz, editors, *Grid Resource Management State of the art and future trends*, chapter 16. Kluwer Academic Publishers, 2004.

[186] W. Smith. Prediction services for distributed computing. In *Parallel and Distributed Processing Symposium, IPDPS 2007*, 2007.

[187] W. Smith, I. Foster, and V. Taylor. Using run-time predictions to estimate queue wait times and improve scheduler performance. In D. G. Feitelson and L. Rudolph, editors, *Job Scheduling Strategies for Parallel Processing, LNCS 1459*, pages 202–219, 1999.

[188] W. Smith, I. Foster, and V. Taylor. Scheduling with advance reservations. In *14th International Parallel and Distributed Processing Symposium*, pages 127–132, 2000.

[189] Q. Snell, M. Clement, D. Jackson, and C. Gregory. The performance impact of advance reservation meta-scheduling. In D. G. Feitelson and L. Rudolph, editors, *Job Scheduling Strategies for Parallel Processing: IPDPS 2000 Workshop, JSSPP 2000, LNCS 1911*, pages 137–153. Springer Berlin / Heidelberg, 2000.

[190] B. Sotomayor and L. Childers. *Globus Toolkit 4 - Programming Java Services.* Elsevier, 2006.

[191] B. Sotomayor, K. Keahey, and I. Foster. Combining batch execution and leasing using virtual machines. In *HPDC - The ACM/IEEE International Symposium on High Performance Distributed Computing*, 2008.

[192] L. Srinivasan and T. Banks (editors). Web Services Resource Lifetime 1.2 (WS-ResourceLifetime). http://docs.oasis-open.org/wsrf/wsrf-ws_resource_lifetime-1.2-spec-os.pdf, June 2008.

[193] H. Stockinger. Defining the grid: a snapshot on the current view. *J Supercomput*, 42(1):3–17, 2007.

[194] Z. Stojanovic and A. Dahanayake. *Service-oriented Software System Engineering: Challenges and Practices.* Idea Group Inc, 2005.

[195] A. Streit, D. Erwin, Th. Lippert, D. Mallmann, R. Menday, M. Rambadt, M. Riedel, M. Romberg, B. Schuller, and Ph. Wieder. UNICORE - from project results to production grids. In L. Grandinetti, editor, *Grid Computing: The New Frontiers of High Performance Processing, Advances in Parallel Computing 14*, pages 357–376. Elsevier, 2005.

[196] I. Taylor, E. Deelman, D. Gannon, and M. Shields. *Workflows for e-Science.* Springer-Verlag, 2007.

[197] I. Taylor, M. Shields, I. Wang, and A. Harrison. The Triana workflow environment: architecture and applications. In I. Taylor et al., editors, *Workflows for e-Science*, pages 320–339. Springer-Verlag, 2007.

[198] I. Taylor, M. Shields, I. Wang, and O. Rana. Triana applications within grid computing and peer to peer environments. *J Grid Computing*, 1(2):199–217, 2003.

[199] J. Taylor. News from the e-Science programme. http://www.rcuk.ac.uk/escience/news/firstphase.htm, July 2008.

[200] D. Thain, T. Tannenbaum, and M. Livny. Distributed computing in practice: The Condor experience. *Concurrency Computat. Pract. Exper.*, 17(2–4):323–356, 2005.

[201] The Grid Infrastructure Research & Development (GIRD) project. Umeå University, Sweden. http://www.gird.se, June 2008.

[202] N. Tonellotto, R. Yahyapour, and Ph. Wieder. A proposal for a generic Grid scheduling architecture. Technical report, CoreGRID, 2006. http://www.coregrid.net/mambo/images/stories/TechnicalReports/tr-0015.pdf, August 2008.

[203] H. Topcuoglu, S. Hariri, and W. Min-You. Task scheduling algorithms for heterogeneous processors. In V.K. Prasanna, editor, *Heterogeneous Computing Workshop, 1999. (HCW '99) Proceedings*, pages 3–14, 1999.

[204] D. Tsafrir, Y. Etsion, and D. G. Feitelson. Modeling user runtime estimates. In *The 11th Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP), LNCS 3834*, pages 1–35, 2005.

[205] S. Tuecke, K. Czajkowski, I. Foster, J. Frey, S. Graham, C. Kesselman, T. Maguire, T. Sandholm, P. Vanderbilt, and D. Snelling. Open Grid Services Infrastructure (OGSI) version 1.0, Global Grid Forum Draft Recommendation. http://www.globus.org/alliance/publications/papers/Final_OGSI_Specification_V1.0.pdf, June 2008.

[206] S. Tuecke, V. Welch, D. Engert, L. Pearlman, and M. Thompson. Internet X.509 Public Key Infrastructure (PKI) Proxy Certificate Profile. http://www.ietf.org/rfc/rfc3820.txt, May 2008.

[207] S. Ullah and I. Ahmad. Non-cooperative, semi-cooperative, and cooperative games-based grid resource allocation. In *Parallel and Distributed Processing Symposium, IPDPS 2006*, 2006.

[208] W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, and A.P. Barros. Workflow patterns. *Distributed and Parallel Databases*, 14(1):5–51, 2003.

[209] J.L. Vázquez-Poletti, E. Huedo, R.S. Montero, and I.M. Llorente. A comparison between two grid scheduling philosophies: EGEE WMS and GridWay. *Multiagent and Grid Systems*, 3(4):429–439, 2007.

[210] S. Venugopal, X. Chu, and R. Buyya. A negotiation mechanism for advance resource reservations using the alternate offers protocol. In *Quality of Service, 2008. IWQoS 2008. 16th International Workshop on*, pages 40–49, 2008.

[211] S. Venugopal, K. Nadiminti, H. Gibbins, and R. Buyya. Designing a resource broker for heterogeneous grids. *Softw. Pract. Exper.*, 38(8):793–825, 2008.

[212] J. Vollbrecht, P. Calhoun, S. Farrell, L. Gommans, G. Gross, B. de Bruijn, C. de Laat, M. Holdrege, and D. Spence. AAA authorization framework. http://www.ietf.org/rfc/rfc2904.txt, August 2008.

[213] G. von Laszewski and M. Hategan. Workflow concepts of the Java CoG Kit. *J. Grid Computing*, 3(3–4):239–258, 2005.

[214] W3C. Web Services Activity. http://www.w3.org/2002/ws/, June 2008.

[215] O. Wäldrich, P. Wieder, and W. Ziegler. A meta-scheduling service for co-allocating arbitrary types of resources. In R. Wyrzykowski, J. Dongarra, N. Meyer, and J. Wasniewski, editors, *Parallel Processing and Applied Mathematics, LNCS 3911*, pages 782–791. Springer Verlag, 2005.

[216] J. Wang, L-Y. Zhang, and Y-B. Han. Client-centric adaptive scheduling for service-oriented applications. *J. Comput. Sci. and Technol.*, 21(4):537–546, 2006.

[217] V. Welch. Grid Security Infrastructure message specification. http://www.ogf.org/documents/GFD.78.pdf, May 2008.

[218] M. Welsh, D. Culler, and E. Brewer. SEDA: An architecture for well-connected scalable internet services. *Operating System Review*, 35(5):230–243, 2001.

[219] M. Wieczorek, R. Prodan, and T. Fahringer. Scheduling of scientific workflows in the ASKALON grid environment. *SIGMOD Record*, 34(3):56–62, 2005.

[220] R. Wolski, J. Brevik, G. Obertelli, N. Spring, and A. Su. Writing programs that run EveryWare on the computational Grid. *IEEE transactions on parallel and distributed systems*, 12(10), 2001.

[221] R. Wolski, J. Brevik, J. S. Plank, and T. Bryan. Grid resource allocation and control using computational economies. In F. Berman, G. Fox, and A. Hey, editors, *Grid Computing: Making The Global Infrastructure a Reality*, chapter 32. John Wiley & Sons, 2003.

[222] M.Q. Xu. Effective metacomputing using LSF multicluster. In *Cluster Computing and the Grid, 2001*, pages 100–105, 2001.

[223] R. Yahyapour. Considerations for resource brokerage and scheduling in Grids. In G. R. Joubert, W. E. Nagel, F. J. Peters, and W. V. Walter, editors, *Parallel Computing: Software Technology, Algorithms, Architectures and Applications, PARCO 2003, Dresden, Germany*, pages 627–634, 2004.

[224] M. Ben Yehuda, O. Biran, D. Breitgand, K. Meth, B. Rochwerger, E. Salant, E. Silvera, S. Tal, Y. Wolfsthal, J. C'aceres, J. Hierro, W. Emmerich, A. Galis, L. Edblom, E. Elmroth, D. Henriksson, F. Hernández, J. Tordsson, A. Hohl, E. Levy, A. Sampaio, B. Scheuermann, M. Wusthoff, J. Latanicki, G. Lopez, J. Marin-Frisonroche, A. Dörr, F. Ferstl, S. Beco, F. Pacini, I. Llorente, R. Montero, E. Huedo, P. Massonet, S. Naqvi, G. Dallons, M. Pezzé, A. Puliato, C. Ragusa, M. Scarpa, and S. Muscella. RESERVOIR - an ICT infrastructure for reliable and effective delivery of services as utilities. Technical report, IBM Haifa Research Laboratory, 2008.

[225] C.S. Yeo and R. Buyya. A taxonomy of market-based resource management systems for utility-driven cluster computing. *Softw. Pract. Exper.*, 36(13):1381–1419, 2006.

[226] B. Plale Y.L. Simmhan and D. Gannon. A survey of data provenance in e-Science. *SIGMOD Record*, 34(3):31–36, 2005.

[227] K. Yoshimoto, P. Kovatch, and P. Andrews. Co-scheduling with user-settable reservations. In *IEEE Mass Storage Conference*, 2005.

[228] L. Young, S. McGough, S. Newhouse, and J. Darlington. Scheduling architecture and algorithms within the ICENI Grid middleware. In Simon Cox, editor, *Proceedings of the UK e-Science All Hands Meeting*, pages 5 – 12, 2003.

[229] L. Youseff, R. Wolski, B. Gorda, and C. Krintz. Paravirtualization for HPC systems. In G. Min et al., editors, *Frontiers of High Performance Computing and Networking ISPA 2006 Workshops*, pages 474–486, 2006.

[230] J. Yu and R. Buyya. A taxonomy of workflow management systems for Grid computing. *J. Grid Computing*, 3(3–4):171–200, 2006.