Service-based Resource Brokering for Grid-based Applications

Valentin Kravtsov
Service-based Resource Brokering for Grid-based Applications

Research Thesis

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy

Valentin Kravtsov

Submitted to the Senate of the Technion — Israel Institute of Technology
Elul 5769 Haifa August 2009
The research thesis was done under the supervision of Prof. Assaf Schuster in the Computer Science Department.

Many people have made this work possible through their teaching, advise, support, and shared effort. First and foremost I would like to thank my dear wife, Ira Shtern, for her support and patience. None of this would have been possible without you. Many thanks to my co-authors Werner Dubitzky, Thomas Niessen, Vlado Stankovski, Benny Yoshpa, Mark Silberstein, Dennis Wegener, Matthias Rohm, Jernej Trnkoczy, Michael May, Juergen Franke, David Carmeli, Krzysztof Kurowski, Muli Ben-Yehuda, David Breitgand, Michael Factor, Hillel Kolodner, Dan Pelleg, Ariel Orda, Uri Dubin, and Pavel Bar as well as our scientific editor Sharon Kessler for their share in the effort, for their enthusiasm and their belief.

Last and not least I would like to thank Prof. Assaf Schuster, a gifted and generous adviser and a dear friend. Assaf has taught me all I know of the art of research, and so much on the nurturing of students and on academy life.

The generous financial support of the Technion and the Gutwirth family is gratefully acknowledged.
# Contents

Abstract  

1 Introduction  

1.1 Opportunistic Grids  

1.2 Quasi-Opportunistic Grids  

1.3 Cloud Computing  

2 Resource Brokering in Purely Opportunistic Grids  

2.1 DataMiningGrid system  

2.1.1 Introduction  

2.1.2 Requirements of grid-based data mining  

2.1.3 DataMiningGrid system architecture  

2.1.4 Application clients  

2.1.5 Grid-enabling data mining applications  

2.1.6 Summary of main features  

2.1.7 Evaluation  

2.1.8 Comparison to existent solutions  

2.1.9 Conclusion  

2.2 DataMiningGrid Resource Broker  

2.2.1 Introduction  

2.2.2 OGSA approach  

2.2.3 Generic use-case for the resource broker  

2.2.4 Requirements  

2.2.5 The job submission process  

2.2.6 The resource broker
List of Figures

2.1 DataMiningGrid system architecture .......................... 18
2.2 A DataMiningGrid application composed with the Triana workflow editor user interface ......................... 27
2.3 Process view of the DataMiningGrid system ................. 39
2.4 Cross-validation: Grid2 GRAM installation .................. 48
2.5 Cross-validation: Kanin GRAM installation .................. 49
2.6 Cross-validation: Matrix GRAM installation ................ 49
2.7 DataMiningGrid technologies ...................................... 62
2.8 Interaction of services during job submission in the DataMiningGrid system .................................................... 65
2.9 Enhanced GridBus resource broker architecture .......... 69

3.1 QosCosGrid system architecture .............................. 77
3.2 A parallel, topology-aware evolutionary algorithm. ....... 84
3.3 A sample request representation of a topology-aware application ................................................................. 86
3.4 A sample representation of offered machines ................ 87
3.5 Left: resource availability on the time axis. Right: appropriate resource availability matrix $AVAIL$ where $R_3$ is not compliant with $PG_1$ and $R_2$ is not compliant with $PG_2$ .................. 92
3.6 Crossover operation $OX3$ on $|Tasks| = anchor\_distance = 6, p = 4, q = 5$ ....................................................... 98
3.7 Clustering resource offers according to requested ranges. . 99
3.8 Comparison of ordering algorithms vs. simple heuristics .... 105
3.9 Comparison of ordering algorithms vs. exhaustive search. . 106
3.10 Performance measurements of the coallocation algorithm vs. optimal solution .............................................. 107
3.11 Runtime of the heuristic coallocation algorithm vs. optimal algorithm. ................................. 107
3.12 Runtime of the evolutionary algorithm application with optimal and suboptimal allocation on the compute resources. . 109

4.1 NAP – high level architecture .................................................. 115
4.2 Application server times CDF – baseline (30 users) ............ 121
4.3 Application server – increased workload. Topmost chart – 40 users, bottom chart – 50 users ...................... 122
4.4 Application server – times vs. queue length ......................... 122
4.5 Application server – CPU overload CDF ......................... 123
4.6 Application server – CDF after remediating CPU over-load . 123
4.7 Trade6 application server – baseline CDF ......................... 124
4.8 Trade6 application server – increased workload CDF ............ 125
4.9 Trade6 application server – CPU overload CDF ............ 126
4.10 Trade6 application server – remediation CDF ...................... 126
4.11 Trade6 – requests count distribution ................................. 127
4.12 Trade6 – response time distribution ................................. 127
List of Tables

2.1 Comparing major grid-enabling technology with DataMiningGrid. ........................................ 52
Abstract

This work describes research on resource brokering in grid and cloud computing environments. In the first part we describe a system for resource brokering in purely opportunistic grid environments, where the user’s applications are “embarrassingly parallel” and require no communication. In the second part we present a complete system for scheduling and coallocation of tightly-coupled jobs in quasi-opportunistic grid environments. In the third part we present a simple, yet powerful, methodology for application-agnostic diagnostic and remediation of performance hot spots in elastic multi-tiered client/server applications, deployed as collections of black box virtual machines.

We begin by describing the DataMiningGrid system and one of its key components – the DataMiningGrid Resource Broker. The DataMiningGrid system has been designed to meet the requirements of modern and distributed data mining scenarios. Based on the Globus Toolkit and other open technology and standards, the DataMiningGrid system provides tools and services to facilitate the grid-enabling of data mining applications without any intervention on the application side. In particular, the DataMiningGrid Resource Broker facilitates the exploitation of different resources from various domains, in order to give data mining researchers the ability to access and utilize resources needed for modern, distributed and computationally intensive data mining algorithms. The DataMiningGrid Resource Broker was designed to work in purely opportunistic grids that do not support an “advanced reservation” feature; thus it handles only “embarrassingly parallel” applications.

The second part of this thesis describes the QosCosGrid (Quasi-Opportunistic Supercomputing for Complex Systems) system and its decision-making module – the QosCosGrid Resource Broker. The QosCosGrid system is the first
grid technology with the capability to harness the available grid resources and provide service that is computationally equivalent to a supercomputer. The QosCosGrid system was built to support applications with large numbers of highly interconnected heterogeneous elements. Such applications are used to simulate real-world complex systems that typically exhibit non-linear behavior and emergence. Scheduling of such large-scale, distributed topology-aware applications requires not only that the properties of the requested machines be considered, but also the properties of the machines’ interconnections. This requirement severely complicates the scheduling process, as even a matching between a single multi-processors task and available machines in a single time slot becomes an NP-complete problem with no polynomial approximation.

The third part of this thesis depicts a simple, yet powerful, methodology for application-agnostic diagnostic and remediation of performance hot spots in elastic multi-tiered client/server applications, deployed as collections of black box virtual machines. The developed Network Analysis for Remediating Performance Bottlenecks (NAP) system identifies performance bottlenecks that might affect application performance and derives remediation decisions that are most likely to alleviate performance degradation.
Chapter 1

Introduction

"Grid" computing has emerged as an important new field, distinguished from conventional distributed computing by its focus on large-scale resource sharing, innovative applications, and, in some cases, high performance orientation [90]. Typical applications designed for execution in grid environments require an immense number of execution nodes, which can be arranged in some predefined topologies and geographically distributed all over the world. Design of grid systems is usually based on the principles of Service Oriented Architecture (SOA). SOA implies that the system implements many independent services, each capable of carrying out a set of predefined tasks. The integration of all these services into one system with a variety of general and designated clients results in a highly modular, reusable, interoperable, scalable and maintainable system.

One of the most important components of a typical grid system’s architecture is the resource broker service. A grid resource broker examines and keeps track of the resources and their capabilities in a grid-computing environment, matches incoming application job requests and their requirements against those resources, assigns the jobs to available and suitable resources in the grid, and initiates and monitors job execution.

Computational grid environments can be roughly divided into two main types: (a) opportunistic and (b) quasi-opportunistic. An opportunistic environment implies that the resource providers are ready to share their resource in a “best effort” fashion: for each resource, the resource provider always has priority over all other users and is willing to share resources usually when they are idle. The resource provider may choose to evict any running job
or allow it to finish. A quasi-opportunistic environment, on the other hand, implies at least some sort of obligation of a resource provider to a resource consumer. To formalize this obligation, the resource provider and resource consumer usually sign an electronic agreement, in which the consumer commits to “rent” the resources, while the provider commits to “lease” them. Any deviation from this contract usually involves economic “punishment.” The difference between the opportunistic and quasi-opportunistic environments is the guarantee provided by the resource provider to the resource consumer. The quasi-opportunistic grid environments provide more reliable infrastructure.

Consequently, resource brokering in quasi-opportunistic grids is inherently more complex task than it is in completely opportunistic grid environments, as the former allows execution of complex jobs having both computational and network requirements, while the latter supports execution of “embarrassingly parallel” tasks only.

Cloud computing is a style of computing in which dynamically scalable and often virtualized resources are provided as a service over the Internet. Users need not have knowledge of, expertise in, or control over the technology infrastructure in the “cloud” that supports them. Cloud computing can be seen as an incarnation of grid computing that supports a completely non-opportunistic resource provision paradigm. The resource management techniques used in a typical cloud are completely different than those applied in a typical grid, mostly because of the resource and network virtualization that take place in cloud computing. In this thesis we will only describe an out-of-band performance management system whose objective is to enable the dynamic re-balancing of physical resources assigned to virtual machines in order to address changing requirements while ensuring a desired level of performance.

### 1.1 Opportunistic Grids

The vast majority of early grid systems were built in a “purely opportunistic” fashion. In purely opportunistic grids resource providers had full control over their shared resources, while resource consumers usually had no guarantee whatsoever whether their jobs would be allowed to complete execution.

The main goal of such grids was to harvest the idle CPU cycles and storage capacity of shared workstations in laboratories, companies, and universities in order to perform useful computation. The purpose of the resource broker service was to mediate access to distributed resources by (a) discovering suitable data sources for a given analysis scenario, (b) discovering suitable computational resources, (c) optimally mapping analysis jobs to resources, (d) deploying and monitoring job execution on selected resources, (e) accessing data from local or remote data sources during job execution, and (f) collating and presenting results.

In this research we present the DataMiningGrid system and its resource broker service. One challenge was to develop an environment suitable for executing data analysis and knowledge discovery tasks in a wide range of different application sectors, including automotive, biological and medical, environmental and ICT. By analyzing the diverse requirements of these applications, the DataMiningGrid has developed generic technology for grid-based data mining. The DataMiningGrid developed enhancements to open source grid middleware in order to provide the specialized data mining functionality required by its use cases. This includes functionality for tasks such as data manipulation, resource brokering, application searching according to different data mining tasks and methodologies, and supporting different types of functionality for parameter sweeps. The result is a grid with all the generic functionality of its component middleware, but with additional features that ease the development and execution of complex data mining tasks. Some key aims in the development of the DataMiningGrid system included the following:
(1) Grid transparency: Domain-oriented end users should be able to carry out data mining tasks without needing to understand detailed aspects of the underlying grid technology.
(2) Application development support: Developers of data mining solutions should be able to grid-enable existing data mining applications, techniques and resources (e.g., database management systems) with little or no intervention in existing application code.
(3) Service-orientated architecture and interoperability: The system should adhere to existing and emerging grid and grid-related standards such as WSRF, and be based on widely used open source technology.
1.2 Quasi-Oppportunistic Grids

In purely opportunistic grids, the computational and storage resources are usually shared among multiple organizations and typically managed in a “best effort” manner. However, many real-world supercomputing applications, such as computational fluid dynamics, weather forecasting, and complex system simulations, rely on co-allocation of large numbers of reliable resources as well as on a static and stable execution environment, making such “best effort” quality of service inadequate. In this chapter, we describe a quasi-opportunistic grid system that enables execution of demanding parallel applications on massive non-dedicated resources in grid environments by adding as many supercomputing characteristics as possible to the grid.

The vision of grids as huge virtual supercomputers is unattainable without bridging the existing gap in their performance and reliability. However, grids differ radically from supercomputers in a number of characteristics, making the realization of this vision even more challenging.

Non-dedicated resources. Real-world grids typically comprise many distributively owned clusters of resources, each of which often serves a community of local users while executing external jobs arriving from the grid. In such a setup, local users are typically prioritized. This policy results in unpredictable performance degradation of the grid-originating jobs. The fluctuations in resource availability could be prevented by applying mechanisms for negotiation and enforcement of global resource sharing policies (e.g. advanced reservation), coupled with an incentive for the resource providers to maintain these policies. Note that such an incentive, if introduced, should be taken into account during the scheduling process [21], [98], [132], [20].

Network heterogeneity. The network topology of a typical grid can be presented as a graph with multiple cliques (clusters) with high capacity links inside cliques and low capacity links among them. High performance cannot be attained without the grid middleware’s ability to expose such a topology to the application, and to apply the topology-aware resource allocation algorithms that satisfy the topology requirements of a given application [80].

Frequent failures. Even in the presence of dedicated resources, the inherently distributed nature of grids implies unpredictable and frequent failures. While easily handled by embarrassingly parallel workloads, such failures are devastating for the performance of complex parallel computations.
Therefore, a grid infrastructure must provide fault tolerance for all its sub-components during run-time [112], [66], [39], [111], [120].

Data pre-staging orchestration. The resource management system must ensure the availability of input data at all remote resources before the execution begins [75], [14], [41], [45], [101], [79], [129].

We believe that a system which handles the above issues transparently and hides them from the application can be considered a reasonable simulation of a real supercomputer. In the following section we outline our current approaches to the implementation of such a system, called QosCosGrid. Later, we describe a decision-making module of the QosCosGrid system that is responsible for scheduling and co-allocation of topology-aware tasks.

1.3 Cloud Computing

The cloud computing paradigm views dynamically scalable and usually virtualized resources as a service to be provided over the Internet. Server virtualization is typically deployed to make more efficient use of server resources, to improve server availability, and to centralize server administration.

Performance management is an important component of any systems management regime, and even more so in a virtual environment. One of the key advantages of server virtualization is rapid and adaptive response to changing performance requirements. To fully realize this advantage, the infrastructure should be able to dynamically resize and re-provision business processes deployed as collections of virtual machines (VMs), according to the current performance needs. Performance management drives this dynamic re-provisioning by using monitoring and analysis tools to facilitate and automate a “monitor—analyze—remediate” cycle.

We, therefore, argue that automated and application-independent means for handling performance problems of black-box VMs is crucial to realizing the full promise of the compute cloud infrastructure paradigm. It should be stressed that this work presents a generic methodology that is relevant to any environment using server virtualization; we use compute clouds as a motivating example only.

In general, monitoring VMs to manage the performance of the applications running inside them is not trivial. A application-level knowledge is typically required to measure application performance. For example, the Ap-
Application Response Measurement (ARM) standard [123] provides application management capabilities, including measurement of application availability, performance, usage, and end-to-end transaction response time. However, to use ARM, a programmer needs to integrate the application with an ARM software development kit (SDK). Integration with the ARM SDK or other application monitoring products requires effort and cannot be done without access to the application source code. The approach exemplified by ARM is often referred to as in-band monitoring. In-band monitoring is not always possible as it requires intrusive instrumentation of the application.

This leaves out-of-band monitoring as the only generic monitoring option available. In the context of virtual machine performance management, out-of-band monitoring implies that the monitoring agents are deployed on a hypervisor and have no knowledge of the applications running within the monitored virtual machines.

While this work is relevant to any environment in which server virtualization is used, it is particularly motivated by infrastructure compute clouds such as Amazon’s EC2\(^2\), RESERVOIR [113], OpenNebula [44], GoGrid\(^3\).

In an infrastructure compute cloud, the customer is usually responsible for sizing the service to be deployed in the cloud. Unfortunately, exact service sizing is often impossible since it requires a thorough understanding of the application’s inner working as well as costly and complex experiments. This may be especially problematic for small and medium enterprises that wish to leverage a cloud computing infrastructure. In addition, due to variability in the service workload, the sizing needs to be dynamically adjusted to reflect changing demands. Thus, an automated and generic means of monitoring and analyzing performance for cloud computing infrastructures is mandatory to ensure that the full promise of these computing clouds is realized. We thus use computing clouds as a motivating example throughout this chapter.

In this work, we advocate out-of-band monitoring to address the core technological challenges impeding performance management of virtualized multi-tier applications. Our solution, NAP\(^4\), is designed to monitor the performance of multi-tiered client/server applications that receive requests.

\(^{2}\)http://aws.amazon.com/ec2/
\(^{3}\)http://www.gogrid.com
\(^{4}\)Black-box Network Analysis for Remediating Performance Bottlenecks.
to perform tasks and return the results over the network. We address the common scenario of a synchronous request/response communication pattern between the clients and the server VMs. NAP’s objective is to address changing requirements while ensuring a desired level of performance. This is done by enabling the dynamic rebalance of physical resources assigned to virtual machines.

NAP constructs distributions for these metrics and observes their temporal behavior, comparing them to the baseline distributions by means of a non-parametric statistical test. When statistically significant deviations from the baseline are detected, the analysis module of our tool issues alerts that may trigger capacity reallocation process to relieve bottlenecks or to eliminate over-subscription. This way, performance is monitored in a way that is transparent to the application. Moreover, the analysis is application-agnostic.

To the best of our knowledge, we are the first to propose a performance management system with such capabilities. Specifically, our contributions are:

1. A novel methodology for out-of-band, non-intrusive, application-independent diagnostics of performance bottlenecks for virtualized multi-tier client/server applications;

2. A prototype of online tool, NAP, that uses this methodology to monitor, analyze and remediate virtualized servers in the Xen hypervisor environment;

3. An experimental evaluation of our solution and demonstration that it is capable of high precision diagnostics while incurring negligible overhead on the managed system. To evaluate NAP, we use Trade6 [29], an established and very widely used benchmark for e-commerce, as our workload generator.
Chapter 2

Resource Brokering in Purely Opportunistic Grids

2.1 DataMiningGrid system

2.1.1 Introduction

Due to the increased computerization of many industrial, scientific, and public sectors, the amount of available digital electronic data is growing at an unprecedented rate. The effective and efficient management and use of these data, and in particular their transformation into information and knowledge, is considered a key requirement for success in such knowledge-driven sectors. Data mining [15, 48, 76] (also known as knowledge discovery in databases) is the de-facto technology addressing this information need. Data mining is an inductive, iterative process that extracts information or knowledge patterns from volumes of data [115].

The rise of distributed computing environments has profound implications in terms of how distributed data are analyzed and interpreted [138, 106]. Future data mining applications will need to operate on massive data sets and the programs for processing, analyzing, evaluating, and visualizing the data will increasingly reside at geographically distributed sites on heterogeneous computing platforms. Distributed data mining and in particular grid-enabled data mining has become an active area of research and development in recent years [9, 118].

Grid computing can be viewed as a virtual computing architecture that
provides the ability to perform higher throughput computing by taking advantage of many computers connected by a network (usually local and wide area networks) [57, 56].

Grid technology evolves rapidly and this often poses challenges, such as interoperability problems, when building applications on open source technology as the available functionality frequently changes. In the past years, grid standardisation efforts have concentrated on achieving the goal to define an open framework for modeling and accessing stateful resources by using Web services. The actual OASIS standard named Web Services Resource Framework (WSRF) v. 1.2 was approved only recently in April 2006. This work describes a large-scale effort aimed at developing a system that brings WSRF-compliant grid computing technology to users and developers of data mining applications. The resulting system is the output of the DataMiningGrid project, which was largely funded by the European Commission [119].

A challenge undertaken by the DataMiningGrid was to develop an environment suitable for executing data analysis and knowledge discovery tasks in a wide range of different application sectors, including the automotive, biological and medical, environmental and ICT sectors. By analysing the diverse requirements of these applications the DataMiningGrid has developed generic technology for grid-based data mining. While existing grid technology already provides a diverse set of generic tools, its emphasis on generality means that the available functionality may lack the sophistication needed to specifically support advanced data mining use-cases. Therefore the DataMiningGrid developed enhancements to open source grid middleware in order to provide the specialised data mining functionality required by our use-cases. This includes functionality for tasks such as data manipulation, resource brokering, application searching according to different data mining tasks and methodologies, and supporting different types of functionality for parameter sweeps. The result is a grid with all the generic functionality of its component middleware, but with additional features that ease the development and execution of complex data mining tasks. Some key aims in the development of the DataMiningGrid system included the following:

1. Grid transparency: Domain-oriented end users should be able to carry out the data mining tasks without needing to understand detailed aspects of the underlying grid technology.
2. Application development support: Developers of data mining solutions should be able to grid-enable existing data mining applications, techniques and resources (e.g., database management systems) with little or no intervention in existing application code.

3. Service-orientated architecture and interoperability: The system should adhere to existing and emerging grid and grid-related standards such as WSRF, and be based on widely used open source technology.

To enable a detailed assessment of the benefits and shortcomings of the DataMiningGrid technology in light of related developments, this presentation is deliberately intended to be comprehensive and as much as possible self-contained.

2.1.2 Requirements of grid-based data mining

The main function of a grid-based data mining system is to facilitate the sharing of data, data mining application programs, processing units and storage devices in order to improve existing, and enable novel, data mining applications. Such a system should take into account the unique constraints and requirements of data mining applications with respect to the data management and data mining software tools, and the users of these tools. These high-level goals lead to a natural breakdown of the requirements for a grid-based data mining system – we distinguish user, application and system requirements. The user requirements are dictated by the need of end users to define and execute data mining tasks, and by developers and administrators who need to evolve and maintain the system. Application program requirements are driven by technical factors such as resource type and location, software and hardware architectures, system interfaces, standards, and so on. The DataMiningGrid project’s main aim is to identify the requirements for a grid-based data mining system and to design, implement and evaluate a system that meets these requirements.

To determine and specify the detailed requirements of the DataMiningGrid system, the project analyzed a representative set of use cases. The use cases are based on real-world data mining applications in the following areas: medicine, biology, bioinformatics, customer relationship management and car repair log diagnostics in the automotive industry, monitoring of
network-based computer systems and civil engineering (ecological modeling).
Some selected use cases are listed below. Section 2.1.7 provides additional
information on the implementation and evaluation of two of these use cases.

Use Case 1: Evolutionary algorithms for reverse-engineering
gene regulatory networks: This is a compute-intensive data mining ap-
application which derives the interaction topology as well as interaction logic
(i.e., the functional description how one gene expression influences another)
from gene expression data [121].

Use Case 2: Analysis of molecular dynamics protein unfolding
simulation data: Computer simulations based on molecular dynamics
generate large volumes of data (> 100 MB per simulation) [19]. To facilitate
analysis of such large data sets, this use case investigates the shipping
of data mining applications to an execution machine near (i.e., with a very
fast data transfer connection) the data source.

Use Case 3: Data mining of distributed medical databases:
This application involves medical databases residing in several geographic
regions in Slovenia. Privacy and security considerations are essential for this
application. In addition, it is important that the application scales well as
more and larger databases become part of this analysis.

Use Cases 4-6: Text mining: These use cases include fast distributed
text classification (Section 2.1.7) and ontology learning for quality and cus-
tomer relationship management in the automotive industry. Another text
mining use case is concerned with information retrieval in digital libraries.
A critical aspect of this is the distribution of the source documents.

Use Case 7: Ecological modeling: This application is concerned
with the task of building mathematical models of ecosystems, in particular
population dynamics in aquatic ecosystems [125]. Data mining is used to
discover equations that reflect the dynamics of the ecosystem (Section 2.1.7).

Use Case 8: Grid monitoring scenarios: Traditional monitoring
services in distributed applications are based on gathering local logs. Experts
browse these logs manually or by using software. The aims of this data
mining application are to provide more powerful monitoring functionality,
to allow ongoing analysis of the system’s condition, and to provide critical
diagnostic information [96].

The main requirements arising from the analysis of the use cases are
described in the following subsections.
User requirements

The typical end user is mainly driven by concepts, issues, tasks and requirements arising from his or her application domain. It is not uncommon that such end users have only limited knowledge of the technical details of the underlying data mining and grid technology. A biologist, for example, may want to classify scientific texts in terms of a given set of cancer classes in order to identify relationships between cancer mechanisms and genes. Of key importance to this type of user are the data reflecting domain concepts and particular type of data mining analysis that needs to be performed. Besides efficiency and effectiveness, the main concerns of such a user include ease-of-use, responsiveness, interactivity, flexibility and conceptual simplicity of a grid-enabled data mining application.

At the other end of the spectrum are end users who possess considerable knowledge of data mining and grid technology or both. Such users may want to make specific choices in terms of defining and configuring a data mining process (selecting algorithms, setting parameters, defining workflows, and specifying preferences for grid resources used to execute a particular data mining application). An analyst, for example, may want to define a detailed workflow designed to pre-process and subsequently cluster data in a specific way with particular algorithms. Such a user is concerned with openness and flexibility in configuring and executing data mining applications. Clearly, efficiency and effectiveness is of great importance to this type of user.

Other categories of users are application and system developers, and system administrators. While rarely performing actual analyses, such users are concerned with requirements such as extensibility, maintainability, user management, system integration, and so on.

From the above considerations, and the use case scenarios analyzed in the DataMiningGrid project, the following key user requirements of the DataMiningGrid system have been identified:

1. Effectiveness, efficiency, novel use. (1) A grid-enabled version of an existing data mining application should offer one or more of the following benefits to the end user: (a) be more effective, (b) be more efficient (higher throughput, which relates to speed-up), or (c) provide a novel feature or a novel way of using the application in comparison to the non-grid version. (2) The DataMiningGrid system should be scalable,
i.e., it should allow seamless adding of grid resources to accommodate increasing numbers of users and user demands without performance loss.

2. **Scope of application domain and task.** (1) The system should be capable of accommodating a widely differing set of existing end user application domains, and data mining tasks and scenarios (e.g., text mining, relational mining, pre-processing, clustering, network induction, and so on). (2) Furthermore, it should be flexible enough to permit entirely novel data mining applications that are impractical or impossible outside a grid environment.

3. **Ease and flexibility of use.** (1) Application-oriented end users should be able to use the system without needing to know technological details, in particular those relating to the underlying grid computing technology. (2) Technology-aware end users should be able to define, configure and parameterize details of the data mining application, the data mining process, and various grid aspects. (3) Users should be able to search for available data mining applications based on various criteria.

4. **Monitoring.** Users should be able to monitor the progress of their applications and be able to respond with the appropriate action (e.g., abort a running application based on exceptions or intermediate results).

5. **Extensibility.** (1) Application developers should be able to grid-enable existing data mining applications with little or no modification to the original data mining application program. (2) System developers should be able to extend the features of the core system without major modifications to the main system components.

6. **Maintenance and integration.** (1) Application developers, system developers, and administrators should be able to easily integrate new applications and core system components with other technology (networks, Web services, grid components, user interfaces, etc). (2) Maintenance (e.g., upgrades, user management) of the core system and the already grid-enabled applications should be simple, seamless and non-interruptive.
Application and system requirements

To meet the user requirements presented in Section 2.1.2, the DataMining-Grid system needs to fulfill a number of additional technical requirements relating to data mining application software (data, programs) and the underlying grid components. We call these requirements data mining application and (data mining grid) systems requirements. The following is a list of the most important requirements in this category:

1. *Resource sharing and interoperation.* The system is required to facilitate the seamless interoperation and sharing of critical data mining resources, in particular, data mining application programs, data resources, storage devices and processing units.

2. *Applications.* (1) The DataMiningGrid system should be able to run a wide range of data mining application programs. (2) To facilitate ease of use and flexibility in defining data mining tasks and processes, the system needs to provide a flexible workflow component with a graphical user interface. (3) In order to execute data mining applications within the DataMiningGrid system, the system needs to understand the requirements, constraints, and user-defined settings associated with an application. For this purpose, each application needs to provide an application description (see Section 2.1.5).

3. *Resource brokering and job scheduling.* Like any grid system, the DataMiningGrid system needs to (1) match available resources to job requests (resource broker), (2) schedule the execution of the jobs on matched resources (scheduler), and (3) manage and monitor the execution of jobs (job execution and monitoring) [77]. Unique requirements for resource brokering and job scheduling of a data mining grid system include data-oriented scheduling, parameter sweep support, and consideration of: (a) the type of data mining task (e.g., classification, clustering), (b) the data mining technique (artificial neural network, decision tree), and (c) the data mining method (e.g., C4.5, C5.0).

4. *Standardization.* To facilitate interoperation and evolution of the DataMiningGrid system, the design and implementation should be based on existing and emerging grid and grid-related standards and open technology.
2.1.3 DataMiningGrid system architecture

The DataMiningGrid architecture is depicted in the diagram of Figure 1. Generally, components in higher layers make use of components organized in lower layers. The bottom layer labeled Resources shows grid software (data, data mining applications) and hardware resources (storage devices, processing elements and computer networks). The main grid middleware layer (large box labeled Globus Toolkit 4) provides the core grid middleware functionality to the DataMiningGrid system. High-level DataMiningGrid Services are organized in the layer above the middleware layer (box labeled DMG High-Level Services). Finally, the top layer depicts the client side components of DataMiningGrid applications (large box labeled DMG Client Applications). Below we will describe the different layers and their relationships in detail, in particular those components developed by the DataMiningGrid project (highlighted in red in Fig. 2.1).

Resources

The box labeled Hardware and Software Resources at the bottom of the diagram in Fig. 2.1 illustrates the grid resources layer in the DataMiningGrid system architecture. The main purpose of the DataMiningGrid system is to facilitate sharing and interoperation of such resources in the context of data mining.

Typical basic hardware resources include processing units (CPUs) and primary and secondary storage devices (e.g., RAM, magnetic disks). These are crucial for processing and storing large quantities of data.

Clusters (or cluster computers), nowadays common in many organizations, are a special kind of resource consisting of collections of loosely coupled computers. In the diagram, the label Local Scheduler represents a typical distributed batch or scheduler system (a grid middleware) used to facilitate convenient access to computers within a cluster while preserving the owner rights of the machines in the cluster. Common batch systems or local schedulers of this kind include Condor [93] Platform’s Load Sharing Facility (LSF) [137] Sun Grid Engine [61], etc. In the current DataMiningGrid test bed we make extensive use of Condor.

Ultimately data is the main ‘substrate’ of all data mining applications. Typically, electronic data is provided in files, spreadsheet applications, database
Figure 2.1: DataMiningGrid system architecture
management systems and other IT systems. Grid mechanisms facilitating
the management of data resources are of critical importance to the Data-
MiningGrid system.

Finally, data mining applications are a fundamental type of software
resource in the DataMiningGrid system. We define a data mining application
as an executable software program, or a collection of such programs, that
performs one or more data mining tasks. Data mining applications include
executable data mining algorithms, data mining tools, components, libraries,
and so on. Enabling this type of resource for operation in grid computing
environments facilitates the development of flexible, scalable and distributed
data mining applications.

Globus Toolkit 4

The middleware layer of the DataMiningGrid architecture is concerned with
core grid functionality, such as virtual organization management, resource
management (discovery, registration, allocation, scheduling), job and execu-
tion management, data management, monitoring, security, and so on. In
the DataMiningGrid system, these core capabilities, functions and services
are provided mainly by the Globus Toolkit 4 software [117]. The Globus
Toolkit 4 meets the requirements of OGSA and it implements middleware
services adequate for grid applications and the WSRF. Some of the ele-
ments of Globus Toolkit 4 that are relevant to the DataMiningGrid system
are described below.

Monitoring and Discovery System 4

Globus Toolkit 4’s Information Services: Monitoring and Discovery System
4 (MDS4) provides information about the available grid resources and their
status. It is used to monitor (e.g., to track usage) and discover (e.g., to
assign computing jobs and other tasks) the services and resources in a grid
system. To facilitate monitoring and discovery MDS4 has the ability to
collect and store information from multiple, distributed information sources.
The DataMiningGrid high-level services (in particular the Resource Broker
and Information Services) are using MDS4 to implement their functionality.
Data management

The data management components from Globus Toolkit 4, used by the Data-MiningGrid architecture, are GridFTP, the Reliable File Transfer (RFT), and the data services provided by OGSA-DAI [52, 6]. In addition, we use the Java CoG Kit [131] for file manipulation on application client machines that do not have a Globus Toolkit 4 installation. GridFTP is a basic platform on which a variety of higher-level functions can be built. The RFT facilitates reliable management of multiple GridFTP transfers. The Globus Toolkit 4 data access and integration tools (OGSA-DAI component) provide grid-enabled access to files, relational databases, and XML databases. None of Globus Toolkit 4’s data replication services are used in the data DataMiningGrid, as the data sets to be data mined are either exposed to the grid as OGSA-DAI data resources, or they are uploaded to the grid via the Triana workflow editor.

In the DataMiningGrid system, GridFTP servers are used to receive any data that is introduced to the grid via Triana. Data transfers between grid nodes are orchestrated by the DataMiningGrid Resource Broker service (Section 2.1.3) and are performed using the GridFTP and RFT components (see also Fig. 2.3). OGSA-DAI is used to provide access to various distributed data resources, in particular to relational databases. The data services it provides can be used to query, update, transform and manipulate data from these resources in various ways. OGSA-DAI provides these functions as activities. Activities are the operations that data service resources can perform on behalf of a client. They expose capabilities of the underlying data resource and are generally used to carry out data resource manipulations or data delivery operations, such as executing SQL or XPath statements, performing XSL-T transformations, and data delivery using GridFTP. The DataMiningGrid activities are extensions to OGSA-DAI. They can be used by DataMiningGrid services to provide basic data preparation and transformation operations relevant to data mining, such as data aggregation, data discretization, computation of derived data, and cross-validation.

Using the APIs provided by OGSA-DAI, with the DataMiningGrid extensions, specialized clients can be constructed to perform data access and transformation, and these clients can be integrated into the Triana workflow editor. Such clients are able to access data from distributed databases
and files, integrate these data into a single data set, filter and transform them (e.g., into training and test data sets), and then convert them into the format required by a particular data mining algorithm, such as the attributed-relationship file format (ARFF) of Weka [134].

**Addressing and security**

The information contained in a service’s endpoint reference reveals how it can be contacted by messages. The endpoint reference contains an address property (URL) and reference properties and parameters. A binding directive (binding in short) constitutes how this information is copied to the message and protocol fields that are to be sent to this service. The WS-Addressing specification defines the SOAP binding as the default binding mechanism for WSRF-compliant Web services.

The Grid Security Infrastructure (GSI) was developed by the Globus Alliance for the Globus Toolkit. It enforces security by using a Public-Key-Infrastructure (PKI) implemented in X.509 compliant certificates for authorization [62]. For communication with a grid system built on the Globus Toolkit, so-called ‘proxy certificates’ are used. These are only valid for fixed periods of time, and are created for a user using the Globus client-side security API. When the client contacts the service, it has to authenticate the user by first passing the proxy. This is implemented inside the Globus API, which is used by the DataMiningGrid clients. GSI supports two levels of security, message-level and transport-level security [133], which differ in concept, performance, and standards conformity. By default, the DataMiningGrid uses transport-level security, the Globus Toolkit 4 default security setting, as this gives the best performance.

**Execution management**

Globus Toolkit 4’s execution management tools handle the initiation, monitoring, management, scheduling, and coordination of remote computations. Globus Toolkit 4 provides the Grid Resource Allocation and Management (GRAM) interface as a basic mechanism for these purposes. A Web services version of the GRAM (the WS-GRAM) supports a suite of Web services with which clients can interact to submit, monitor, and cancel jobs on local or remote computing resources. The DataMiningGrid Services are interact-
ing with services provided by the WS-GRAM to achieve their functionality (Section 2.1.3).

**Enhanced Condor Adapter.** The original Globus Toolkit 4 Condor Adapter is designed to submit relatively simple batch programs to Condor pools. It lacks two critical functions that are needed in the DataMiningGrid system for submitting a wide range of programs to Condor pools. First, the current Condor implementation and the Globus Toolkit 4 Condor Adapter restrict data movement to simple copying of files. Second, despite the sophisticated Java submission environment provided by Condor, it is not possible with the Globus Toolkit 4 Condor Adapter to submit arbitrary Java jobs to Condor pools. In the DataMiningGrid project we developed two mechanisms to address these shortcomings in the Globus Toolkit 4 Condor Adapter: to cope with the data movement restrictions, we enabled it to compress recursive directory structures into a single archive file, move the file to the execution machine and extract the archive’s content before the actual job execution; to facilitate flexible Java job submission, we made it capable of handling JVM parameters, class path information, and so on.

**Enhanced Fork Adapter.** Similar to the Globus Toolkit 4 Condor Adapter, the Globus Toolkit 4 Fork Adaptor does not support flexible execution of Java jobs. The Globus Toolkit 4 Fork Adaptor was modified to handle JVM parameters, class path information, and so on.

**High-level services**

The three high-level services of the DataMiningGrid system are represented by the DataMiningGrid Resource Broker, Information Services and Data Services.

**Resource Broker**

A (grid) job could be anything that needs a (grid) resource, e.g., a request for bandwidth or disk space, an application or a set of application programs [105]. In the DataMiningGrid system, jobs consist mainly of one or more DataMiningGrid-enabled data mining applications which need data and processor resources in order to be executed. Based on a grid scheduler component, a grid resource broker must make resource selection decisions involving resources spanning over multiple administrative domains (hosts,
routers and networks managed by a single administrative authority). In such a distributed environment, a resource broker has no control over the local resources and the information about the resources is often limited or obsolete.

Specifically, a grid resource broker addresses the following tasks [105]:

1. **Resource discovery.** The resource broker determines the list of available, grid-enabled resources.

2. **Resource selection.** Based on the description of one or more jobs and their requirements, the resource broker selects those resources that best match the requirements in the job description. To do this, the resource broker matches the job requirements against the resource descriptions provided by an information service (Section 2.1.3 describes the DataMiningGrid Information Services).

3. **Job preparation and submission.** Before the resource broker submits a job or a collection of jobs to the selected resources, it has to make sure that the resources have all they need to run. In particular, this may involve certain setup activities and data staging or provision.

4. **Job monitoring and clean-up tasks.** The resource broker needs to inform the user about the progress of the job(s), make sure that the results are available to the user, and initiate possible clean-up operations after the jobs have been completed.

In particular with applications involving multiple resources, users, jobs and sites, resource brokering becomes very complex. Grid resource broker technology is an area of active research [105].

Based on the requirement of the DataMiningGrid system, the specific requirements of the DataMiningGrid Resource Broker [84], are as follows: The Broker needs to (a) facilitate the discovery of data mining applications, (b) accommodate complex workflows (pre-defined or user-defined), (c) identify and select suitable grid resources to execute such workflows, (d) execute such workflows, (e) provide execution monitoring and result retrieval, (f), operate in such a way that the underlying grid details are transparent to the user, (g) should adhere to grid standards (mainly WSRF), and (h) should be able to evolve without requiring users having to install each new version.
Critical challenges in this scenario include intelligently matching user and application requests and requirements with available computation and data resources, and performing the automated staging of data.

After careful investigation of several resource brokers (Community Scheduling Framework, GridWay, GridLab Resource Management System, Nimrod/G and GridBus Resource Broker) we found that none satisfied the requirements. However, we found that with some additional development, the GridBus Resource Broker could be made to meet the requirements.

GridBus has the advantage of being able to submit jobs to the Globus Toolkit 4 execution subsystem, it is clearly structured and well designed, and, because it does not require any specific information or security subsystem, it can be easily integrated with other systems.

Currently the GridBus is not service-based, and so it cannot be used for a global grid application as it needs to be installed on every client machine. To address this, a wrapper was developed which exposes its main functions through a pre-defined WSRF-compliant service interface. GridBus also lacked automated resource compilation and MDS4 querying mechanisms. These mechanisms are implemented via the DataMiningGrid Information Services (Section 2.1.3). An illustration of how the different services interact when a DataMiningGrid application is launched, including the job submission steps, is provided in Section 2.1.5 below.

**Information Services**

Any comprehensive grid system needs information services to discover, characterize and monitor resources, services and computation [38]. Typically, a grid information service feeds into other grid components and services, including services for discovery, replication, scheduling, troubleshooting, application adaptation, and so on. The DataMiningGrid Information Services include the InformationIntegrator Service, which is responsible for creating the application registry from DataMiningGrid Application Description Schema, and grid information services provided by Globus Toolkit 4. In particular, they provide information on (a) all resources available in a DataMiningGrid grid system, (b) the properties of these resources in terms of storage and CPU capacity, operating system, and so on, and (c) the current load of these resources. The DataMiningGrid Resource Broker requires this information to plan, allocate, carry out, and monitor job execution. The
Information Services component automatically queries the MDS4 and compiles a list of the resource entries maintained there. This approach has the following advantages.

First, no user interaction is required for discovering available resources as this is done automatically via MDS4 queries. As a result, the overall system becomes more user-friendly.

Second, the original design of the resource broker facilitates the inclusion of different kinds of schedulers (e.g., based on a cost, round-robin or any other policy). However, in some scenarios requirements may arise with constraints for which the actual scheduler cannot cater, for example, certain confidential data may only be processed on machines belonging to certain organizations. For this purpose it is reasonable to exclude all resources not belonging to the required organizations from the applied scheduler. As the scheduler must select resources from a list automatically compiled by the Information Services, the resource broker can perform a preliminary step of filtering this list before the actual scheduler is called.

Data Services

The DataMiningGrid’s data services are OGSA-DAI data services that utilize DataMiningGrid activities for performing data operations on data sets exposed to the grid as data resources. For the use-cases investigated so far, these data sets usually reside in distributed relational databases (Section 2.1.3). Typically the result of these operations will be one or more URIs to pre-processed data sets residing in a file, or a set of files, which the DataMiningGrid Resource Broker then schedules for processing with a data mining algorithm.

2.1.4 Application clients

In its current version, the DataMiningGrid architecture comprises two different DataMiningGrid application clients: A workflow editor and manager, and a Web client. Both clients are able to handle multi-component data mining applications. The workflow editor and manager client are designed to facilitate detailed user control of the individual data mining application components, including workflow composition, parameter settings, input and output setting, etc. The DataMiningGrid workflow editor and manager are
based on Triana [122]. The Web client is intended for end users (e.g., biologists, medics, customer relationship manager, etc.) with a domain-oriented view of the data mining application. This client is useful for end users with limited knowledge of the underlying data mining and grid technologies.

Both types of clients have an Application Enabler component which is normally operated by a data mining application developer and not the user. The Application Enabler client components are discussed in Section 2.1.5, Grid-enabling data mining applications.

**Triana workflow editor and manager client**

The workflow editor and manager used to design complex DataMiningGrid application workflows is based on Triana [122]. Triana is an open source problem-solving environment developed at Cardiff University, UK, as part of the GridLab and GridOneD projects. The Triana tool is structured into two major parts: (1) a graphical workflow editor to help users compose workflows from a set of building-blocks that are dragged into a work-space window and connected using a mouse, and (2) a workflow manager or engine to execute workflows.

The diagram in Fig. 2.2 illustrates a DataMiningGrid application and its presentation via the Triana workflow user interface. Here, the output of the Load Description unit is used to instantiate the Application Control unit with the application description for a data mining application. In this case the application is an evolutionary algorithm and for simplicity, the input data is already located at a URI on the grid. The Application Control unit is used to define the options or parameters for the evolutionary algorithm, to map data inputs from the GridURI units to the algorithm’s inputs, and finally to define certain execution requirements, such as the operating system and minimum amount of memory required. In this case, the evolutionary algorithm defines 100 jobs, each job representing an independent evolutionary population. The 100 jobs are distributed for execution on different grid nodes, based in Slovenia, Germany and Northern Ireland. The Execution unit shows the execution status of these jobs. The final two units in the workflow, the Provenance Manager and Grid Explorer units, are used (a) to display a record of all information concerning the execution of this evolutionary algorithm application, and (b) to view and download the resulting output data files.
Figure 2.2: A DataMiningGrid application composed with the Triana workflow editor user interface
Triana is capable of discovering and binding to Web services and WSRF-compatible services. To develop a user-friendly DataMiningGrid interface, it was necessary to implement the WSRF binding for most DataMiningGrid units. Additionally, all DataMiningGrid units contain bindings for WS-Security and WS-SecureConversation [133]. This permits the passing of the end user proxy credentials to the contacted WSRF service and facilitates secure conversations between the units and the corresponding WSRF services.

**Credentials Generator.** Security plays an important role in grid computing. As each user requires a proxy certificate, which is a copy of the user’s original X.509 [62] certificate with limited lifetime, the workflow editor has to provide an easy-to-use means to create and manage such proxies. The Credentials Generator unit will generate a new proxy certificate. If users possess several certificates (e.g., from different authorities) they can choose which one to use. Furthermore, they can select a suitable private key and specify the lifetime of the proxy (12h, 24h, 1 week, 1 month). Interaction with this unit is only required once per workflow.

**Application Explorer.** The Application Explorer is a client to the MDS4 service (Section 2.1.3). It is used to search the grid-wide application registry for data mining applications. Search criteria include application ID, application name, group, data mining technique or method, functional area (data mining task such as classification, clustering, etc.), CRISP-DM phase [115], and so on. The parameters are translated into an XPath query which is passed to MDS4. The query results are displayed in a table from which the application developer may select a suitable data mining application. Once the data mining application is selected, MDS4 is contacted again to obtain the full description of this grid-enabled data mining application (Steps 3-8 in Fig. 2.3). The obtained application description is passed to the next unit in the workflow, which is usually the Application Control unit.

**Data Manipulator.** The DataMiningGrid Data Manipulator component consists of a collection of units that is divided into three groups. These groups are units that manipulate file systems, relational databases, and a fairly large number of units for expert users that provide direct access to the functionality provided by OGSA-DAI.

The file manipulation units have two main functions. First, they transfer data between the client applications and the grid infrastructure, most
importantly this includes uploading data sets from the user’s own machine to the grid for data mining, and downloading the data mining results to the user’s machine. Second, the units are used to browse file systems and view files on remote machines hosting a GridFTP server. These units are implemented using the Java CoG Kit.

The relational database units are specialized OGSA-DAI clients, providing many functions that are hidden from the user. For example, a user simply enters an SQL query - on execution of the workflow the data is accessed from distributed databases, filtered and transformed, and then delivered to a set of files which are formatted for cross-validation by a particular algorithm.

The output from a Data Manipulator unit will always be a data set (files or directories of files), and one or more URIs pointing to the data. The URIs are subsequently mapped onto application parameters using the Application Control unit.

**Application Control.** The Application Control unit is a central point of control when developing a workflow for a DataMiningGrid application. In this unit, the end user specifies all application parameters and assigns input data as well as application parameters. The unit takes as input (1) a description of a previously selected grid-enabled data mining application, and (2) any number of data URIs, which may be used as input to the data mining application. The graphical user interface is created dynamically, based on the description of the selected data mining application.

The unit displays general information about the data mining application, a description of the necessary input data (e.g., a file in coded ARFF), and the default settings of the application parameters. Application parameters (also referred to as options) can be of different types, such as strings and integers, as defined by the DataMiningGrid Application Description Schema (Section 2.1.5). For each application parameter, an input field is presented to the end user. The input field depends on the data type of the application parameter. For example, the value of a decision tree learning parameter $A$ is a real number, with the default value of $A_{def} = 0.43$. In order to execute the application in a distributed way, the end user can also carry out a sweep of values of parameter $A$ for a specified interval, $i_A$, and step-size, $s_A$. For example, the interval $i_A = [0.40, 0.45]$ and step-size, $s_A = 0.01$ will result in exactly six job execution requests. Similarly, if the data mining application uses a parameter $B$, which specifies the class in a decision tree training data
file (e.g., iodine) and it belongs to a string data type, the end user can also specify a whole list of strings prompting the system to generate and distribute and execute one job per class in the list.

The input data is denoted by a URI, which may point to either a remote directory containing sub-directories and files (e.g., text documents for ontology-learning) or a single file, which may be a data file (e.g., a data file coded in ARFF) or a file containing some additional parameter settings (e.g., differential equations for an equation discovery learning algorithm). In case of a directory URI, the Application Control unit always allows the end user to specify that a sweep is to be performed over all files in the directory, meaning that there will be as many jobs produced as there are files contained in the directory. For example, the directory may contain a set of task files making it possible that the different tasks can be executed in a distributed fashion, one job per task. The file sweep can only be performed if the application input is of type file. If it is of type directory, a subdirectory sweep is performed.

The end user can also customize the execution requirements for a particular application. For example, the user may decide to specify the minimum memory (or any other resource capacity) needed for execution. Furthermore, the end user is allowed to specify an exact WS-GRAM service on which all jobs are to be executed. This is especially important when the application has to be moved to a location where the data resides.

The output of the Application Control unit is a fully specified instance of the Application Description Schema (Section 2.1.5), which represents a multi-job description. This description is passed on to the Execution Manager unit for further processing (starting from Step 9 in Fig. 2.3; see also Section 2.1.4).

**Execution Manager.** The Execution Manager provides a Triana user interface client to the WSRF-compliant DataMiningGrid Resource Broker service. This client was built so as to hide the grid complexity from the user. The main Execution Manager provides the user with mechanisms facilitating the (a) submission of jobs, (b) monitoring of jobs, and (c) propagation of the result URIs. The execution of the unit commences when a fully specified data mining application description is received from the Application Control unit. This description is then passed as input to the Resource Broker service for execution of the jobs. After the job execution has been initiated,
end users can monitor the execution status of the jobs until all jobs are completed (either successfully or not). Upon completion of all jobs, the unit propagates to the next units in the workflow a URI of the results location and provenance information.

**Provenance Manager.** The Provenance Manager unit displays all the relevant provenance and metadata and permits these data to be saved for later use on a (local) computer specified by the user. It represents the final operation in a simple DataMiningGrid Triana workflow. The information presented by the Provenance Manager is part of the DataMiningGrid Application Description Schema. This schema includes the following information:

1. Complete description of the data mining application.
2. Timestamp recording when the application was submitted to the DataMiningGrid Resource Broker and when it was completed.
3. Status of the application, in particular whether it completed successfully or failed.
4. URI specifying the location of the results obtained from a successful execution of the application.
5. Information about the execution of each individual job of the data mining application. For each job, the following information is available: (a) job ID; (b) job status (success/failure); (c) submission to WS-GRAM time and completion time; (d) failure description (exception code indicating the nature of the failure); and (e) application parameters with which a job started.

Saving the provenance information enables users to search and discover algorithms, training data, and prediction models including scores indicating how these fared. It also permits users to reconstruct the process that has led to a particular result.

**Monitor.** The Monitor unit provides general grid monitoring capabilities. It displays, in intuitive graphical form, the current status of the grid and its components. For each WS-GRAM in the system the unit presents the WS-GRAM’s status: the number of busy CPUs, number of available CPUs and the total number of CPUs. This unit is intended as stand-alone unit and
may not be connected with other units in the workflow. The Monitor unit’s information is periodically updated to reflect changes in the WS-GRAMS available in the grid environment.

**Custom units.** Custom units can be used to develop specific data operations, result visualization and other custom functions. These functions can then be added to the data mining process in the usual manner.

**Triana Application Enabler.** Before a data mining application can be used within the DataMiningGrid, a description of the application needs to be provided. Since this task is concerned with grid-enabling data mining applications, it is described in more detail in Section 2.1.5.

**Web-based client**

Similar to the Triana client, the Web client provides a developer and an end user component for grid-enabling and running data mining applications on the DataMiningGrid.

**Web Application user client.** The Web Application client of the DataMiningGrid system is intended for users who do not want to concern themselves with developing detailed data mining workflows. Instead, the Web Application client permits the running of complex DataMiningGrid applications while hiding many of the underlying complexities of the both the data mining application and the grid from the user. Thus, the user can concentrate on controlling important parameters of the application without needing to know the low-level ‘wiring’ of the application and the grid.

We have investigated several ways of building Web-based clients for executing DataMiningGrid applications in a grid environment. Generally speaking, Web-based clients are less flexible when compared to a workflow editor like Triana, but can offer a greater degree of user-friendliness for those users who do not require detailed technological control. A user has to specify application information via the Web Application client, and this includes information on the following (see Section 2.1.5): (1) general information about the application; (2) general execution details; (3) input data requirements; (4) where to store output data; and (5) specific execution requirements.

While it is relatively straightforward to realize a very simple Web-based DataMiningGrid application, the developer effort needed to provide a Web-based user interface for complex, multi-step data mining applications can be
considerable. In this respect the Triana solution provides more modularity and greater flexibility.

**Web Application enabler client.** Before a Web client can be used within a grid environment, a description of the associated data mining application needs to be provided. The Web Application Enabler is a DataMiningGrid tool that allows developers to carry out this task through a Web browser. Since this task is concerned with grid-enabling of data mining applications, it is described in more detail in Section 2.1.5.

### 2.1.5 Grid-enabling data mining applications

In the DataMiningGrid, grid-enabling existing data mining programs is achieved through the use of metadata and the associated Information and Resource Broker Services. This is a generic approach and may be extended to make use of Semantic Grid technologies.

Grid-enabling data mining applications for the DataMiningGrid requires two elements. First, the data mining application and its properties and requirements need to be described in a uniform way and be made known to the system. This description of data mining applications could be viewed as a data mining application resource abstraction. Second, the data mining application’s software resources need to be registered and stored in the grid. This section and its subsections describe the basic tasks, components, processes and tools involved.

**Application Description Schema**

The DataMiningGrid Application Description Schema (ADS) is a central feature of the DataMiningGrid system. It is the basis for registering an application in the grid and providing flexible application search mechanisms, dynamic configuration of GUIs, resource brokering, and so on. The ADS, whose instances are realized as XML documents, describes various aspects about data mining applications that are enabled to run on the DataMiningGrid system. The ADS is divided into two parts: the first part describes aspects that are common to all applications enabled to run on the system; the second part describes information that specifically relates to data mining applications. Below we summarize the structure and content of the ADS.
The common part of the ADS captures application-relevant information that falls into three major categories: general, execution and application.

The general part specifies different ‘general’ aspects of the application which will be useful for different purposes (searching for applications, provenance, administration, and so on). The information contains elements like a global unique identifier and a time stamp (both created by the system), a textual description of application, vendor and version information, etc.

The execution part contains information relevant to the execution of the application program, such as the underlying programming environment (e.g., Bash Shell or Windows operating system), programming language (e.g., C, Python, Java), executable file(s) and required libraries, commands and arguments used at start-up, directory information, and so on.

The application part is by far the most comprehensive one. It provides configuration information of the application such as

- **Options** or parameters used when the data mining algorithm implemented by the executable is executed. All options or parameters are typed and can be optional (a default may or may not exist and may be overwritten by the user) and hidden (an option is transparent to the user).

- **Parameter lists and loops** define parameter lists and information for executing iterations over loop counters, files or directories. The Application Control unit uses this part of the ADS (Section 2.1.4). The list element facilitates a ‘sweep’ over a list of numeric or symbolic values. Such a list may either be provided explicitly by the user, or generated automatically by the system if a repeated execution with a list of different values is required. The loop element is used for ‘sweeps’ or iterations over a user-defined interval with a fixed step size.

- **Data input and output** slots are used to describe the application’s input and output data, i.e., data types (file or directory), transfer protocols permissible for particular data, physical location of data, other descriptors (e.g., whether data input is optional or mandatory), etc.

- The requirements slot of the ADS captures the application’s system requirements. This information is processed by the DataMiningGrid
Resource Broker and used to match applications to resources (Section 2.1.3). Typical entries include requirements for memory, disk space, the type of WS-GRAM job manager (Fork or Condor), optional user-defined IP addresses of execution machines, operating systems and processor architectures (e.g., Motorola PowerPC, Intel i386 and higher).

- **Environment variables** that need to be set at the execution machine before execution of the application.

The data mining part of the ADS describes application information that is specific to the data mining aspect of the application. The content structure of the ADS is mainly based on CRISP-DM [115] and to a lesser extent on the Data Mining Ontology for Grid Programming [22]. This information is used to facilitate fast discovery of data mining applications and data-mining-aware resource brokering. Among other things, this information includes the:

1. Description (free text) of the data mining application and the domain-specific problem it is used to address.
2. CRISP-DM phase the application is used for.
3. Data mining task to which the application applies.
4. Data mining method or methodology the application employs.
5. Data mining algorithm used by the application.
6. Particular software implementation of the algorithm.
7. Software suite the application is part of (e.g., Weka, Clementine).

**Application development with the Triana user client**

Triana-based application development with the DataMiningGrid system has two aspects: application *deployment* and *use*. Deployment is concerned with the grid-enabling of data mining applications for the DataMiningGrid system, and use refers to employing applications that have already been grid-enabled for the DataMiningGrid system to solve an actual problem.
This approach of deploying and using data mining applications in a grid offers the following advantages. Firstly, all applications are handled by the system in a uniform manner. This ensures that the system can be easily extended and accommodate new applications. Secondly, end users need concern themselves only with information required to find, parameterize and launch applications. Domain-oriented users who do not want to delve too deep into the grid technicalities will find this feature especially appealing.

Deploying a data mining application requires that an initial instance of the ADS is created for this particular application. This ADS instance defines all aspects of the application including default options for certain parameters that the user may override later. A DataMiningGrid developer usually performs this task. The developer specifies all required information needed to instantiate the ADS for the particular application. The Application Enabler generates the corresponding initial ADS instance, registers the application details in the MDS4 registry, and uploads and stores all application files (e.g., executables, libraries) in the MDS4 database. A developer (as opposed to the end user) is normally concerned with operating the Application Enabler, and this is reflected in Fig. 2.1 by the dashed outline of the boxes representing the Application Enablers in both the Triana and Web client case.

To use an already grid-enabled DataMiningGrid application, the user typically identifies the application he or she needs and then specifies the input data, output data location, application options and parameters, and other information required to launch and execute the application in the DataMiningGrid environment. They usually perform these tasks by using the Application Explorer, Application Control, Execution Manager, and Data Manipulator (Section 2.1.4) Triana units.

DataMiningGrid Generic Job Template (GJT) has been developed to support the use of any data mining application enabled for the DataMiningGrid system. This is similar to the work flow shown at the top of Fig. 2.2. The GJT consists of different Triana units for (a) selecting a data mining application, (b) selecting input data, (c) specifying application options, parameters, and mapping data inputs to the application, and (d) remote execution of the job on the grid.

To locate and select a data mining application registered in a grid, the user employs either the Load Description or the Application Explorer units.
These units load the application’s ADS description either from the MDS4 or from a local file.

The user has many ways to specify what data the application is to mine, e.g., by uploading a local data file, by executing an application-specific OGSA-DAI client, by selecting a file on the grid, etc. The Grid Explorer and Grid URI units and the Data Manipulator units provide a means to carry out this task, and all ultimately communicate at least one URI to the Application Control unit.

The Application Control unit is used to specify parameters of the application. The output of this unit is a fully specified ADS instance containing all information needed to execute the application. If a parameter sweep or a file sweep is set, then the ADS instance describes the execution of multiple jobs rather than a single execution.

The Execution Manager unit receives the ADS instance and initiates the execution of the application by communicating the ADS instance to the Resource Broker. Finally, the execution results are passed on to a storage server and details about the execution process are passed on to the Provenance Manager.

The Provenance Manager and Grid Explorer units are used to view the results of the application execution and details about the execution process. The user may also chose to store the metadata contained in the Provenance Manager for further use.

**Application development with the Web user client**

The basic purpose of the Web-based Application Enabler is the same as the Application Enabler Triana unit. The Web Application Enabler is a Web-based tool used by the developer to define an initial DataMiningGrid ADS instance, register the application in the grid-wide application registry (MDS4), and store the application resources in the system. This facilitates easy discovery and use by both users and developers. The Web version of the Application Enabler consists of several form-based Web pages. These guide the developer through the entire process of creating a DataMiningGrid ADS instance, and uploading and registering the application in the grid. The steps include filling in forms for General Information, Execution Information, Input Data, Output Data, Requirements and Upload, and once these are completed the user may initiate the application’s execution with...
the Start page.

**Application run-time**

Once an application is grid-enabled to operate in the DataMiningGrid environment, the system architecture (Fig. 2.1) comes to ‘life’ by:

1. Collecting the necessary information of the data to be processed, the data application to be used, and the available resources.
2. Matching the application requirements to suitable resources.
3. Executing the distributed data mining application.

A more detailed account of this process is provided in the diagram of Fig. 2.3 and the accompanying description.

Fig. 2.3 depicts a scenario involving four different organizations (labeled Org A to Org D) from a virtual organization which are part of a grid. In the scenario, data from Org A and an application from Org D are staged by the system on resources at Org B and then executed there. This somewhat simplistic scenario is chosen for reasons of diagrammatic clarity. Clearly, the system supports more complex scenarios involving multiple processes running simultaneously at many different sites involving multiple data and application program resources. The numbers in the diagram denote the different steps involved in this process:

**Steps 1-2:** Selection, query, extraction and pre-processing of data from Org A. This involves the workflow component Data Manipulator (Section 2.1.4).

**Steps 3-8:** Querying information on available data mining applications. This process extracts the information about the data mining applications as described in the corresponding DataMiningGrid Application Description Schema instance (Section 2.1.5) and produces the corresponding job descriptions.

**Steps 9-10:** The Resource Broker queries MDS4’s Information Services directly to retrieve up-to-date information on the grid, including available machines, workload statistics, and so on.

**Step 11:** Passing job descriptions to Globus Toolkit 4’s Job Execution Services running on the selected execution machine(s) at Org B. In
Figure 2.3: Process view of the DataMiningGrid system
the depicted scenario this is performed only at a single organization. In a different scenario, Globus Toolkit 4 Job Execution Services in several organizations may be called simultaneously. After Step 11, processing splits into two parallel streams denoted by a and b respectively.

**Steps 12-15:** The Job Execution machine calls RFT/GridFTP components to initiate the transfer of data (prepared by Step 2) from Org A, and the files related to the application from Org D, to the storage component in Org B. The steps labeled 13a, 14a and 15a relate to the transfer of data and the steps 13b, 14b and 15b describe the transfer of application files such as executables, libraries and other files. The optimization of the stage-in process is based on the assumption that data movement is an expensive operation. To minimize data movement, the stage-in process is performed once per GRAM (as opposed once per job) and each job is provided with the complete URL of the local data storage location. Usually, the GRAM is responsible for at least one cluster of machines. The per-GRAM approach has the advantage that the stage-in operation is performed far less than would be necessary in a per-job approach. After successful stage-in of all the executables, data and libraries, the executable is launched, and the execution is monitored until completion.

**Step 16:** Finally, this step describes the execution request, in which the job description is passed from Globus Toolkit 4’s Job Execution Services to the Local Scheduler in organization Org B.

### 2.1.6 Summary of main features

This section summarized the main features of the DataMiningGrid system. Some of these features partially overlap or are interdependent to some degree.

**Scope of application.** The DataMiningGrid system is able to accommodate data mining applications from a wide range of platforms, technologies, application domains and sectors.

**Service-oriented architecture.** The DataMiningGrid system is designed around SOA principles. The system implements many independent services, each capable of carrying out a set of predefined tasks. The integration of all these services into one system with a variety of general and designated clients, results in a highly modular, reusable, interoperable, scal-
able and maintainable system.

**User interfaces.** The DataMiningGrid system provides two types of flexible user interfaces based on a sophisticated workflow technology (Triana [122]) and a Web client. The workflow client facilitates the composition and execution of complex data mining processes by users with comprehensive data mining expertise. The Web client is designed for domain-oriented end users who do not require detailed knowledge of the underlying data mining and grid technology.

**Dynamic configuration of graphical user interfaces.** The metadata included DataMiningGrid Data Mining Application Description Schema instances allows data miners to operate data mining applications through a dynamically configured graphical user interface, providing slots for specifying public options, algorithm parameters, data inputs, and so on. This frees the developer from the need to develop such interfaces for each and every application.

**Searchable applications.** The DataMiningGrid’s repository of searchable applications, which contains meta-data about each application available at that particular time, provides data miners with a mechanism to search for data mining applications available on the grid. The search can be performed based on different criteria such as general attributes (application name, vendor, version, etc.) or data mining specific attributes (CRISP-DM phase, functional area or data mining task, data mining technique or method). Additionally, users may also specify custom queries using XQuery.

**Shipping of applications.** The DataMiningGrid system facilitates the ‘shipping’ of data mining applications to targeted machines on the grid (as determined by either the user manually, or by the DataMiningGrid Resource Broker automatically). This feature has two main advantages: (a) there is no need for pre-installation of applications (this supports a dynamic grid infrastructure), and (b) shipping of algorithms to data (this is useful if data cannot be transferred because of large data volumes, privacy issues, etc.). See also *ease of deployment* below.

**Data Mining Application Description Schema.** The DataMiningGrid ADS is a central feature of the DataMiningGrid system. Each data mining application is characterized by an instance of the ADS. Particular ADS instances are used for registering applications in a grid, searching applications, creating Triana-based user interfaces dynamically, resource bro-
kering, etc.

Data Services. The DataMiningGrid Data Services are represented by a set of clients based on the OGSA-DAI software [6]. They facilitate integrating distributed databases, ‘on-the-fly’ data formatting to support data mining operations, performing data assays or summaries of accessed data sets, and the extension of OGSA-DAI services to provide additional data preparation and transformation operations. In addition, remote file browsing and transfer utilities provide easy user access to file-based applications and are implemented using the Java CoG Kit [131].

Ease of deployment. An important feature of the DataMiningGrid system is that it allows users to mine their data without the need to install grid middleware or any data mining software at the location where their data resides. Many users will have data sets that they wish to mine which are not already exposed as data resources on the DataMiningGrid. To use the DataMiningGrid system, a user must have the Triana workflow editor installed and a valid DataMiningGrid certificate. The DataMiningGrid software consists of Java libraries and these, along with important Java libraries from grid middleware (CoG Kit, GridBus, OGSA-DAI and Globus Toolkit, are simply bundled together and copied to a library folder of the Triana workflow editor. These libraries contain the client software that is used to manipulate data resources, select DataMiningGrid applications and orchestrate all the DataMiningGrid services needed to execute these applications.

Resource Broker. The DataMiningGrid Resource Broker is essential to the DataMiningGrid system. It combines the following features: (a) fully automated resource aggregation, (b) data-oriented scheduling, (c) zero-footprint on execution machines, (d) extensive monitoring of resources, (e) brokering based on application requirements typical to data mining (e.g., parameter sweep, data mining task, etc.), (f) support for user-friendly applications, (g) interoperability, and (h) adherence to relevant existing and emerging grid and grid-related standards (e.g., interoperability, security). Extensive performance experiments indicate that the Resource Broker exhibits good speed-up\(^1\) and scale-up\(^2\) behavior [84, 68, 86].

\(^1\)Speed-up is typically defined as the ratio of serial execution time of the fastest known serial algorithm (requiring a certain number of basic operations to solve the problem) to the parallel execution time of the chosen algorithm.

\(^2\)Scale-up may be defined as the ability of a greater number of processors to accommodate a proportionally greater workload in a more-or-less constant amount of time.
**Standardization.** Even in the fast-changing field of grid computing adherence to standards (and de-facto standards) is crucial for building flexible, interoperable and future-proof systems. A critical standard the system supports is WSRF, which has been widely adopted by the grid community. Globus Toolkit 4 is the key middleware upon which the DataMiningGrid system is built. Globus Toolkit 4 implements the WSRF and follows the OGSA standard.

**Re-use of (open-source) technology.** A critical consideration in developing the DataMiningGrid technology was to ensure that it will be taken up by the industrial and R&D communities and can evolve into the future. To support this goal the DataMiningGrid has been developed based on existing open technology such as Globus Toolkit 4, OGSA-DAI, Triana and GridBus.

### 2.1.7 Evaluation

The section presents experiments that were carried out to evaluate the performance of the integrated system components.

**Test bed**

To evaluate the DataMiningGrid system, a comprehensive test bed spanning the three European countries UK, Germany and Slovenia was set up. The test bed includes several servers with Globus Toolkit 4 installations and three sites with Condor compute clusters of different composition. The basic test bed hardware configuration was as follows:

- The Grid1 GRAM in Germany: Pentium 1.4 GHz, 1 GB RAM (running middleware).
- The Matrix GRAM installation in the UK, with 64 Itanium II 900 MHz, 2 GB RAM; and 10-25 Pentium, 2 to 3 GHz, 1 GB RAM machines in the condor pool.
- The Grid2 GRAM installation in Germany, with 5 Pentium 1.4 GHz, 1 GB RAM machines in the condor pool.
- The Kanin GRAM installation in Slovenia, with 40 Pentium 1.4 GHz, 512 MB RAM machines in the condor pool.
In addition to performing comprehensive data mining studies based on the main use case scenarios, numerous tests were run using a wide range of application programs. Experiments relating to two main use case scenarios are presented below [126].

**Ecosystem modeling**

The increasing pollution of aquatic systems has triggered intensive efforts in the monitoring and management of such systems. These efforts include ecological modeling approaches supported by information technology. Computational ecosystem models allow decision makers to run simulations under different conditions corresponding to alternative environmental policies. In this area decision-support tools from machine learning, data mining and other fields are increasingly employed to model and simulate ecosystems. Because of the underlying complexity (non-linear dynamics phenomena, multiple experiments under different conditions) the necessary calculations are conceptually complex and compute-intensive. In this use case scenario, the DataMiningGrid system was employed to (1) induce models of aquatic ecosystems from modeling knowledge and measurement data, and (2) simulate these models under changing (different) environmental conditions. The main purpose of the developed models was to predict phytoplankton dynamics of two European lakes: Lake Glumse (Denmark) and Lake of Bled (Slovenia) [125].

This study is based on Lamgme [124], which is a system for equation discovery. Equation discovery is the area of machine learning concerned with the automated discovery of quantitative laws, expressed in the form of equations, from collections of measured data. Equation discovery methods could be viewed as an extension to system or model identification methods, as they aim to identify both an adequate structure of the equations and appropriate values of the constant parameters. In order to perform complex scenarios with Lagmame, parallelization of the discovery process is very attractive as it can reduce the process time significantly. We implemented the entire procedure of model building using the DataMiningGrid system.

To execute the experiments on the DataMiningGrid system, ecological modeling experts developed two workflows, which were then combined into a (two-step) complex workflow. The first step corresponds to the model induction task. It is concerned with discovering an equation structure and model
parameters that fit the training data well. The second step corresponds to the simulation task. This step calculates the variables representing the ecological system over time under different conditions with the model(s) derived in step one.

The first step (model induction) of the workflow is the one that is computationally demanding. The induced model is expressed in the form of an ordinary differential equation. This part of the workflow interacts with the measurement data, which are files stored on the grid as data resources, and with the model storage directory, in order to store the output results, i.e., the induced models.

The second (model simulation) part of the workflow interacts with (a) the model storage directory (a data resource in the DataMiningGrid system) in order to obtain the previously induced model(s), and (b) with the measurement data (data resource) to obtain the data set defining the initial conditions for simulation as well as with (c) another storage directory in order to store the simulation results, which are then analyzed and interpreted by the domain experts.

In order to test the system, seven test cases were prepared and executed in the test bed. Each test case contains different ecological modeling scenarios, resulting in different number of jobs. For these experiments only 18 out of ca. 120 machines in the test bed were available due to operating system requirements of Lagrange 2.0, which runs only under Linux. The speed-up was evaluated for a workload of 60 jobs, by varying the number of machines to process the jobs. Changing the number of machines from 1 to 10 corresponded to speed-up of 6.70. Furthermore, increasing the number of machines by a factor of 2.50 and the workload by a factor of 2.00 produced a scale-up of 1.80. In addition to speed-up in the model construction process, the ecological modeling experts found the user-friendly access to the data mining tools to be a great benefit.

Text classification

In customer relationship management and quality management, Daimler-Chrysler maintains distributed repositories with about 15 million text documents, and in the order of 104 new documents are added daily. In this application (a) these documents need to be classified according to subject – this facilitates their redirection to suitable subject matter experts, (b) new
emerging subjects need to be discovered, and (c) keywords need to be extracted from documents or groups of documents. It is infeasible to handle and aggregate this information manually or on a centralized server. This application requires the automation of these tasks, handling of their distribution, and features methods for distributed text classification. Because these data are confidential they cannot be described here. Instead, we have applied the same techniques on newswire data available from the German news agency dpa (Deutsche Presse-Agentur).

We performed measurements of runtimes for cross-validation with standard SVM kernels (called SVM-CV) and cross-validation with structured graph kernels (called Hypergraph-CV) [25]. The measurements were run identically on three different GRAM installations with 5 (Grid2 Condor pool), 24 (Kanin Condor pool) and 64 (Matrix condor pool) machines respectively. There was no run which included two or more of the geographically separated GRAMs at the same time. The measurements contain 1, 2, 4, 8, 16, 32, 64 and 128 jobs on each Condor pool with the same input data for SVM-CV and 1, 2, 4, 8, 16 jobs for Hypergraph-CV (this uses unpublished algorithms from Fraunhofer). The Hypergraph-CV measurements were also executed on an input data volume that is 50% of the size of the original data. Each set of jobs was run in two modes:

- **Fork mode** denotes the sequential execution of each job on the GRAM installation machine in question to obtain reference values.
- **Condor mode** denotes the distributed execution of the jobs on the computing pool, which is controlled by the Condor batch job system.

Cross-validation was performed on 1000 documents of the dpa collection of October 2004. The pre-processing yields approximately 10000 variables (distinct content words), the exact number depends on the pre-processing parameters. The source XML file of October 2004 is 39 MB, the pre-processed data file is 1.70 MB. The XML data resides at Fraunhofer IAIS, Germany, and was transferred to the remote GRAM machines. Pre-processing was performed locally at the Condor pools.

To determine the performance of a classifier on the given data set, cross-validation consisted of pre-processing documents, training the classifier, and classifying in \( N \) identical runs. We executed the runtime measurements on cross-validation for both classifier variants: SVM-CV and Hypergraph-CV.
Speed-up depends on the location of input data, in particular because input files can become rather large. For a central file server with Gigabit Ethernet, we measured a linear speed-up only for up to 5 machines. Unlimited linear speed-up is possible and was measured only if the input files are mirrored on locally installed disks. Fig. 2.4 to Fig. 2.6 show the test results of the SVM-CV application.

- **Comparison of Fork / Condor execution**: Both Fork and Condor execution have a linear increase in runtime. Execution on a Condor pool is faster than Fork execution on a single machine so the increase is more moderate.

- **Comparison of Fork execution**: Fork execution is nearly the same at each GRAM installation machine, the Matrix machine is marginally faster, because it has faster hardware.

- **Comparison of Condor execution**: The more machines the pool has the faster is the Condor execution.

### 2.1.8 Comparison to existent solutions

The purpose of this section is to briefly review some of the main efforts in the area of distributed, and in particular grid-enabled, data mining technologies and compare these, as much as this is possible, to the DataMiningGrid system. The criteria used to compare and discuss these systems are listed below, and a summary of the comparison is provided in Table 2.1. Although relevant and interesting, due to space limitations, this review does not consider work on distributed data mining in peer-to-peer networks, privacy-preserving data mining, distributed data stream mining, data mining in mobile and embedded devices, distributed data mining in sensor networks, or parallel data mining (unless it is related to distributed or grid computing environments).

The following list describes a number of criteria we deemed relevant in discussing and comparing the DataMiningGrid system to related systems:

- **Domain restrictions.** The system is restricted to applications in certain domains.
Figure 2.4: Cross-validation: Grid2 GRAM installation
Figure 2.5: Cross-validation: Kanin GRAM installation

Figure 2.6: Cross-validation: Matrix GRAM installation
• **Data mining task/technique restrictions.** The system is restricted to certain data mining tasks (e.g., classification, clustering) or methods or algorithms implementing the tasks (e.g., C5.0, c-means).

• **Data mining technology restrictions.** The system is restricted to certain data mining platforms, tools or implementations (e.g., Weka).

• **Modification of data mining application source.** The system requires modification at source-code level of the data mining application program to be grid-enabled.

• **Data mining standards supported.** Major data mining standards that are directly supported by the system (e.g., CRISP-DM, PMML).

• **Data mining-aware resource brokering.** The system provides a grid resource broker that takes into account requirements and constraints relevant to data mining.

• **Data mining application metadata** The system allows the characterization of data mining application programs or components with computable metadata. In particular, this metadata should describe the constraints, properties and requirements of the application in terms of data mining dimensions (task, method, implementation) and computing requirements (memory, hardware architecture, software and operating system requirements). The grid resource broker should be able to access and use this information for effective resource brokering.

• **Shipping of data mining application.** The system facilitates the automated shipping of data mining application programs to different locations within a grid.

• **User interface.** The user interface the system provides for different types of users (end user, developer, administrator).

• **Workflow support.** The system provides a means to define and execute complex data mining workflows.

• **Application search.** The system provides a way for the user to identify suitable data mining applications within a grid (based on metadata used to characterize applications).
Service-oriented architecture. The system is based on principles of service-oriented architecture.

Distributed data management. The system facilitates management of distributed data sources, i.e., access to distributed data sources and shipping/staging of data across grid nodes.

Grid standards supported. The grid standards supported by the system.

Middleware. Critical grid middleware the system is based on (e.g., Globus Toolkit, Unicore, etc.).

Inherent data distribution. The system supports direct mining of data which are inherently distributed, i.e., data which cannot be merged or staged on a single grid site or node. Typically, such scenarios exist where data are partitioned horizontally (a shared set of variables across distributed sets of observations) or vertically (shared set of observations over distributed sets of variables).

Fine-grained parallelism. The system allows portions of a single data mining algorithm instance to be run in parallel on distributed grid nodes.

Open source. Whether the system software is available as open source license (and if yes, what kind of license).

Seamless application handling. Data mining applications can be flexibly added, modified, removed without any changes to user components.

GridMiner [18] has been designed to support data mining and online-analytical processing (OLAP) in distributed computing environments. GridMiner implements a number of common data mining algorithms, some as parallel versions, it also supports some text mining tasks. Each data mining service is implemented as standalone grid service specified by OGSA. The system architecture is service-oriented and a Dynamic Service Composition Engine facilitates the integration and execution of services as a workflow. GridMiner provides the integration of distributed databases based on
Table 2.1: Comparing major grid-enabling technology with DataMiningGrid.

<table>
<thead>
<tr>
<th>Feature / System</th>
<th>DataMiningGrid</th>
<th>GridMiner</th>
<th>K-Grid</th>
<th>DMGA/WekaG</th>
<th>SODDM</th>
<th>FAEHIM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Domain restrictions</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>DM task/technique restriction</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Part</td>
<td>HDM</td>
<td>No</td>
</tr>
<tr>
<td>DM technology restriction</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Part</td>
<td>–</td>
<td>No</td>
</tr>
<tr>
<td>Modification of DM application</td>
<td>No</td>
<td>–</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>DM application metadata</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes*</td>
<td>Yes</td>
<td>–</td>
</tr>
<tr>
<td>DM standards supported</td>
<td>CRISP</td>
<td>PMML</td>
<td>–</td>
<td>Weka</td>
<td>No</td>
<td>Weka</td>
</tr>
<tr>
<td>DM-aware resource brokering</td>
<td>Yes</td>
<td>–</td>
<td>Yes*</td>
<td>No</td>
<td>Yes*</td>
<td>–</td>
</tr>
<tr>
<td>Shipping of DM application</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>User interface</td>
<td>G/W</td>
<td>G</td>
<td>G</td>
<td>OBPD</td>
<td>G</td>
<td>–</td>
</tr>
<tr>
<td>Workflow support</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes*</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Application search</td>
<td>Yes</td>
<td>–</td>
<td>Yes</td>
<td>Yes*</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Service-oriented architecture</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Distributed data management</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Grid standards supported</td>
<td>WSRF</td>
<td>OGSA</td>
<td>WSRF</td>
<td>WSRF</td>
<td>WSBP</td>
<td>WS</td>
</tr>
<tr>
<td>Middleware</td>
<td>GT4</td>
<td>GT3</td>
<td>GT4</td>
<td>GT4</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Inherent data distribution</td>
<td>Part</td>
<td>–</td>
<td>Yes</td>
<td>No</td>
<td>HO</td>
<td>No</td>
</tr>
<tr>
<td>Fine-grained parallelism</td>
<td>Part</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Part</td>
</tr>
<tr>
<td>Open source</td>
<td>AOSL2</td>
<td>–</td>
<td>No</td>
<td>Weka</td>
<td>No</td>
<td>GPL</td>
</tr>
<tr>
<td>Seamless application handling</td>
<td>Yes</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>Yes</td>
</tr>
</tbody>
</table>


OGSA-DAI. A Knowledge Base component of GridMiner handles all elements of the data mining process, and serves as a service and user registry. The task of bridging the Web and grid environment gap is performed by a Dynamic Service Control Engine component.
From a high-level architecture perspective GridMiner seems to have a lot
in common with the DataMiningGrid. Perhaps one of the main advantages
of the DataMiningGrid over GridMiner is that DataMiningGrid complies
with the more recent trend towards WSRF.

Knowledge Grid (K-Grid) [23], [30] is a high-level, service-oriented
framework providing grid-based data mining tools and services. The K-Grid
system facilitates a wide range of data mining tasks and related tasks such as
data management and knowledge representation. The system architecture
is organized into a High-level K-Grid Services and a Core-level K-Grid Ser-
vices layer, which are built on top of a Basic Grid Services layer. The Core
K-Grid Services handle (a) publication and search of data sources and search
for mining results, (b) publication and search for data extraction, mining
and visualization tools, (c) definition and management of abstract execution
plans used to describe complex data mining processes, and (d) presentation
of data mining results. The High-level K-Grid Services are responsible for
(a) managing metadata describing K-Grid resources, and (b) mapping of the
requests described in the abstraction execution plan to available resources,
and managing and executing the abstract execution plan. The main com-
ponents of K-Grid are implemented using the VEGA (Visual Environment
for Grid Applications), which is itself based on Globus Toolkit. Recent de-
velopments of K-Grid [30] seek to re-implement an earlier pre-WSRF [23]
as a WSRF-compliant version.

K-Grid is not too dissimilar to DataMiningGrid, except that DataMin-
ingGrid was conceived from the outset with OGSA and WSRF in mind.
Recent K-Grid developments are geared towards WSRF compliance [30].
Unlike DataMiningGrid, K-Grid is not available as open source.

Data Mining Grid Architecture (DMGA) [110] is a flexible data
mining grid architecture, based on the main phases of a data mining process:
pre-processing, data mining itself and post-processing. This architecture
is composed of generic, data grid and specific data mining grid services.
WekaG [109] is an implementation of this architecture based on Weka [134],
one of the most well known and used data mining tools and Globus Toolkit
4. The main advantages of the DMGA/WekaG combination include: (1)
DMGA/WekaG can be adapted to the requirements of complex data mining
processes. (2) WekaG is very easy to use for data mining users; the user
interfaces of Weka and WekaG are the same. (3) New data mining services
can be added in a flexible way. (4) It is possible to use a combination of different data mining services that make use of trading and negotiation protocols for selecting the most suitable service. (5) DMGA provides the composition of services, that is, the ability to create workflows. DMGA offers both horizontal composition, in which different functional services are composed, and vertical composition, in which several instances of the same service access different data sets. (6) Finally, WekaG is able to support parallel implementations of data mining algorithms. Weka is available as open source and WekaG is likely to be open source in the future.

While DMGA is flexible with regard to underlying data mining applications, its WekaG implementation (which is based on Weka) is restricted to applications implemented in WekaG. DataMiningGrid is not restricted to specific data mining applications.

**Anteater** [73] is a service-oriented architecture for data mining that relies on Web services to achieve extensibility and interoperability. Anteater is designed to handle large volumes of data and high computational demands, and to handle a diverse user population. A particular feature of the Anteater system is its capability to distribute fine-grained parallel data mining applications and to provide high degrees of scalability. Anteater’s architecture revolves around a set of servers, including the: data server, mining server, application server and visualization server. Anteater uses visual metaphors to present the system’s functionality to domain-oriented end users by keeping technical details transparent to them. To exploit parallelism while maintaining performance, Anteater provides a runtime system called Anthill. Anthill requires data mining applications to be decomposed into *filter-stream* subcomponents, which abstract a data mining application into a series of *filters* (computation) and *streams* (communication). Anteater has been evaluated on a number of algorithms and real-world data mining applications.

Anteater requires data mining applications to be converted into a filter-stream structure. While this provides scalability, this overhead cuts down on the number of applications that will actually be ported to this platform. It is not clear to what extent Anteater provides resource brokering. This highlights one of the key features of the DataMiningGrid system – it is very easy to grid-enable existing data mining applications and to seamlessly distribute its execution within a grid computing environments based on the
resource allocation and management provided by the DataMiningGrid Resource Broker.

**Service-oriented distributed data mining (SODDM):** Work by Cheung and colleagues [27] is based on a platform which focuses on service-oriented distributed data mining (we refer to this platform as SODDM). This approach concentrates on real-time, on-demand, self-adaptive, distributed and privacy-preserving mining of inherently distributed data. Privacy preservation is achieved via a learning-from-abstractions approach, which is based on statistical properties of private data rather than individual data items. The overall mining process is performed via a global analysis of local data abstractions. SODDM is based on Web services and the underlying data mining processes are specified in the Business Process Execution Language for Web Services (BPEL4WS also known as WSBPEL. In addition to supporting long-running (stateful) interactions, BPEL4WS provides a model and a grammar for specifying interactions between a business process and its partners through Web services interfaces. The learning-from-abstraction and self-adaptive approach to distributed data mining was demonstrated using different data mining applications, including clustering and manifold unfolding for visualization. To achieve self-adaptation (trade-off between overall data mining quality and source data granularity) SODDM implements a negotiation procedure between the SODDM global service broker and the local data sources (which are endowed with autonomous negotiation properties). Cheung et al. consider service-orientation a critical step towards future distributed data mining since it will help to find better ways to control and manage aspects such as accuracy, privacy, efficiency and resource utilization.

In contrast to DataMiningGrid, which has no restrictions with regard to data mining domains, applications, techniques or technology, SODDM is geared towards data mining applications focusing on high-dimensional data. Also, SODDM seems to have some limitations in terms of resource brokering and other grid-related aspect (see Table 1).

**Federated Analysis Environment for Heterogeneous Intelligent Mining (FAEHiM)** [2] implements a toolkit for supporting data mining based on a grid services approach. The toolkit consists of data mining grid services and a workflow engine to enable users to compose those services into a solution for a particular problem. Three types of grid services are provided
to implement data mining functions: (1) classification, (2) clustering and (3) association rules. The algorithms used to implement the data mining functions are taken from Weka [134]. In addition, visualization grid services based on GnuPlot and Mathematica are also provided to visualize the output of the data mining grid services. Data sets may be read by the grid services from the local file space or streamed from a remote location.

A limitation of FAEHIM seems to be its reliance on Weka. It is not clear if FAEHIM supports any form of intelligent resource brokering. Like Data-MiningGrid, FAEHIM uses Triana [122] to define data mining processes.

Platform Independent Text Mining Engine Tool (Pimiento) [1] is an object-oriented application framework (OOAF), i.e., an application ‘template’, for text mining. Implemented in Java Standard Edition, the Pimiento OOAF gives application developers the primary benefits of an OOAF, such as modularity, reusability, extensibility, and inversion of control. It implements several text mining functions, including text categorization, language identification, clustering and similarity analysis. The overall aims of the Pimiento OOAF platform are to let application developers incorporate text mining functionalities without having to worry about the complexity implicit in a text mining engine, the scalable management of text documents, or access control. Addressing these aims, the Pimiento development was driven by the following key requirements: (1) Interoperability: Based on an open architecture and open application interfaces, the Pimiento system should enable seamless interaction and integration between the application and the text mining platform. (2) Modularity: Applications should continue to function (using the text mining platform via services or components) after changes to the functionality of the platform. (3) Simplicity: Software developers should be able to add text mining functionalities to their applications, without requiring in-depth knowledge on text mining. (4) Sustainability: Existing applications should not be negatively affected when new algorithms, languages and features are added to the platform. (5) Scalability and performance: In particular, the platform should be able to perform and scale well as the amount of analyzed documents increases. (6) Security.

Pimiento is an interesting mining tool with focus on text mining. While this tool supports distributed text mining tasks, it does not seem to facilitate sophisticated resource brokering.
Discovery Net [114] focuses on scientific discovery from high-throughput data generated in life science, geo-hazard and environmental domains. Discovery Net provides a service-oriented computing model for knowledge discovery, allowing users to access and use data analysis software and data sources made available by third parties. The system is based on the Globus Toolkit and provides components (a) to declare the properties of analytical software components and scientific data stores, (b) to retrieve and compose Knowledge Discovery Services, (c) to integrate structured and semi-structured data from different data sources using XML schemas, and (d) to deploy and publish existing knowledge discovery procedures as new services. The system also provides a Discovery Process Markup Language (DPML) which is an XML-based representation of the Discovery Net workflows. Processes created with DPML workflows can be deployed and shared as new services on the grid. Another feature of Discovery Net allows dynamic access and integration of various data sets in the workflows. Interfaces to SQL databases, OGSA-DAI sources, Oracle databases and custom designed wrappers are built to enable data integration.

Discovery Net does not seem to support WSRF. It is not clear to what extent the system supports resource brokering, specifically with sensitivity to data mining applications.

Algorithm Development and Mining (ADaM)\(^3\) consists a wide variety of data mining and image processing components designed for analyzing scientific and remote sensing data. The components can be configured to create customized data mining processes. In its latest version, individual ADaM data mining operations (called toolkits) are available as executables, libraries (e.g., C/C++), modules (Python), and can be accessed via multiple external interfaces facilitating the implementation of data mining and image processing components as Web and grid services.

ADaM focuses on image processing tasks. While the ADaM system supports a wide range of interfaces and a composition mechanism to realize customized data mining processes, its seem to lack a proper workflow editing and management facility and grid resource brokering.

\(^m\)y\(g\)rid\(^4\) aims to exploit the growing interest in grid technology, with an emphasis on the information grid and bioinformatics. In addition to basic is-

---

\(^3\)http://datamining.itsc.uah.edu/adam/

\(^4\)www.mygrid.org.uk
sues of storage, computation and resource management, $myGrid$ is designed to provide higher level functionalities over an existing grid infrastructure that supports scientists in making use of complex distributed resources. In particular, this includes supporting the (a) scientific process of experimental investigation, evidence accumulation, and result assimilation, (b) scientist’s use of the community’s information, and (c) facilitation of scientific collaboration, allowing dynamic groupings to tackle emergent research problems. $myGrid$ consists of a collection of loosely-coupled middleware components and services designed to support data intensive in silico experiments in biology. Workflows and query specifications link together third party and local resources using Web services technology. Experiment Knowledge Discovery is the part of the in silico life cycle that relates to data mining. The $myGrid$ system includes a sophisticated workflow tool called Taverna Workbench, which allows users to construct complex analysis workflows from components located on both remote and local machines, run these workflows on their own data and visualize the results. The $myGrid$ software can be freely downloaded and has been used for building discovery workflows for investigations into various life science studies. The $myGrid$ project is part of the Open Middleware Infrastructure Institute UK Consortium (OMII-UK). The integration of the $myGrid$ software stack with the production level OMII-UK software promises to increase the commitment of the existing user community, and to encourage a significantly wider deployment of the workflow and semantic grid capabilities that $myGrid$ offers.

In contrast to DataMiningGrid, $myGrid$ focuses on distributed information and knowledge management in the context of the life sciences. Part of the system is devoted to data mining activities. DataMiningGrid is much leaner and concentrates on distributing data mining tasks to enhance performance; to facilitate the sharing of data, data mining applications, and hardware; and to promote resource exploitation. For users and developers interested solely in grid-enabled data mining, DataMiningGrid seems to be more suited than $myGrid$.

Overall we believe the collection of features of the DataMiningGrid technology make it a unique and competitive contender for grid-enabling existing and developing new data mining applications for grid computing environments.
2.1.9 Conclusion

As the demand for automated analysis of large and distributed data grows, new data mining challenges in distributed computing environments emerge. The aim of DataMiningGrid project is to address some important requirements arising from modern data mining scenarios. The technology developed by the project facilitates grid-enabling of existing, and the development of novel, data mining applications. Major features of the DataMiningGrid technology include high performance, scalability, flexibility, ease of use, conceptual simplicity, compliance with emerging grid (e.g., WSRF) and data mining standards (e.g., CRISP-DM), and use of mainstream grid and open technology. The DataMiningGrid technology has been evaluated on the basis of a wide range of representative modern data mining applications. In comparison with related technologies, the DataMiningGrid system offers an attractive alternative. The software is freely available under the Apache Open Source License version 2 via the project Web site [119] or SourceForge. Future developments of the DataMiningGrid technology include to explore the technology in the context of new applications and to further develop the system accordingly. This may be approached by the DataMiningGrid partners individually or in concert or indeed by anyone who wishes to do so.

2.2 DataMiningGrid Resource Broker

2.2.1 Introduction

As Grid [57] and pervasive computing environments are becoming commonplace, the availability of data mining [115] operations, algorithms and services available within proprietary Grid environments and the visible Web is constantly increasing. Such scenarios give rise to novel ways of building data mining applications that dynamically integrate distributed data mining operations or services into a (logical) single system or tool [118]. Currently, there is a considerable variety of distributed, computational and data intensive data-mining applications and algorithms that simply cannot be fully exploited without Grid infrastructure. Nevertheless, we could not identify any software that provides a complete, user-friendly solution for data mining researchers willing to perform long-running data mining tasks on massive data on the Grid. Generally, data mining researchers are not Grid experts.
Therefore, any solution that will be eventually adopted by them has to focus on user-friendly interfaces, which have to hide all the Grid complexity from the user. The success of such Grid-based data mining frameworks is dependent on the implementation of an essential component that will make "intelligent" decisions regarding all the Grid aspects, such as location of job execution, destination of data movement, error handling, error propagation etc. In a distributed system, this component is the resource broker.

Resource brokering is not a new concept in Grid realm. Resource Brokers were in the past and continue to be developed by different commercial companies and academic research groups. Software packages like CSF\(^5\) and GridWay\(^6\) are available for download from Globus Alliance [54] Web page. The list of resource broker implementations includes also GridLab Resource Management System (GRMS) [4], Nimrod/G [20], GridBus Resource Broker [130] and others that are already in use in diverse scientific areas and projects. Each one of the above tools has different advantages and integration constraints. However, at the time of the design release none of the above brokers had a WSRF-compliant implementation with the tested ability of submitting jobs to Grid Resource Allocation Manager 4 (WS-GRAM). The meaning and the importance of WSRF will be discussed in the next paragraph.

2.2.2 OGSA approach

The vision shared by many ongoing research projects is that future Grid systems will be based on the Open Grid Service Architecture (OGSA) [56]. OGSA is a distributed interaction and computing architecture based around the Grid Computing service, assuring interoperability on heterogeneous systems so that different types of resources can communicate and share information. OGSA addressed a number of requirements, which led to the definition of stateful Web services. Namely, as Web services are static entities, they do not give enough flexibility for creating on-demand systems. Such systems require dynamic creation and destruction of entities that represent virtual resources.

In terms of Grid middleware, both Globus Toolkit 3, which implements

\(^5\)www.globus.org/Grid\_software/computation/csf.php
\(^6\)www.globus.org/Grid\_software/computation/Gridway.php
the Open Grid Standards Infrastructure (OGSI), and further, the Globus Toolkit 4\(^7\), which implements the Web Services Resource Framework (WSRF), follow the OGSA. Web Services Resource Framework (WSRF) is aimed at defining a generic framework for modeling and accessing persistent resources using Web services so that the definition and implementation of a service and the integration and management of multiple services is made easier. WSRF narrowed the meaning of Grid services to those services that conform to the WSRF specification\(^8\), although, a broader, more natural meaning of a Grid service is still in use. As defined in Foster’s "Physiology of the Grid" [56], a Grid service is any service used in the Grid environment that conforms to specific interface conventions accepted as standard throughout this Grid. However, the low-level fabric provided by WSRF is insufficient for developing complex-problem solving environments. Complex higher-level functionality is necessary to provide a seamless, transparent, manageable, reliable, efficient and secure common computing environment, in which a resource broker is an integral part.

Figure 2.7 illustrates the software components to be used and services to be developed in the DataMiningGrid project:

The DataMiningGrid project focuses mainly on the development of the components and services belonging to the high-level DataMiningGrid Technology Layer shown in Figure 2.7. Analysis services and data services are also being developed, which make use of different data mining algorithms (located in the layer above them) in a distributed environment possible. The graphical workflow editor hides the complexity of the underlying Grid technology layer from the users, who are expected to be experts on data mining, but not on Grid computing. In doing so data miners can concentrate on defining data mining workflows appropriate to their problem, while all Grid related tasks (e.g. resource allocation, stage-in and stage-out of the required data and results) are carried out by the resource broker and underlying Grid system.

\(^7\)www.globus.org/toolkit/

\(^8\)www.oasis-open.org/committees/
2.2.3 Generic use-case for the resource broker

The common usage scenario of the system is a combination of three main steps:

- Step 1: Workflow composition by the user
- Step 2: Execution of the workflow in the Grid
- Step 3: Retrieval and annotation of the results

Step 1: Workflow generation is done by the user via a graphical user interface, namely the workflow editor. A data mining workflow may be large and complex, include conditional loops and branching as well as many other advanced features.

Step 2: After the workflow has been created, it is submitted to the workflow manager for execution. The workflow manager interacts with the resource broker service, which is responsible for intelligent matching between the user’s requests (jobs) and available computation and data resources, as
well as for later execution of jobs on the Grid. The resource broker service not only has to balance the load on the Grid nodes, but also has to make decisions sophisticated enough to satisfy a broad range of the requests. Many data mining algorithms pose very strict requirements regarding execution environments, data transfer policies, parallelisation constraints, free temporary and permanent storage necessities, etc. After the resource broker has successfully matched the jobs with the proper resources, the jobs are submitted to the selected machine or cluster.

Step3: After the execution has completed, the final results of the data mining operation are stored either in the Grid itself or on the user’s client machine. These results may also be annotated with meta-data, e.g. for providing provenance information about the workflow by which these results were obtained. However, this step is independent of the implementation of the resource broker.

2.2.4 Requirements

Over the past years, data mining has been adopted by many diverse application areas, including biology, medicine, ecological modeling and customer relationship management. There are requirements that any system for performing data mining in the Grid has to satisfy in order to provide surplus value for data miners when compared to traditional data mining applications. The following list explains the key requirements:

1) Defining and performing data mining operations in a Grid must be user-friendly.

Professional data miners, while being experts in their own field, cannot be assumed to be experts on Grid technology. Due to the fact that Grid technology is just a new method for enabling data mining to them, as much Grid-related details as possible have to be hidden from these users.

2) Data mining operations must be executed on suitable machines in the Grid without direct user interaction.

While usually only a few computational resources in a Grid serve a specific purpose (e.g. data base servers, central registries), many machines simply share their computing power. However, the execution of any data mining operation on such a machine contains three subtasks:

a) A list of resources needs to be compiled.
b) The most suitable machine for executing the operation can be selected.
c) The data and the data mining application have to be transferred to the
selected machine for execution.

However, Grids are dynamic and possibly fast-changing distributed sys-
tems, based on virtualization of a large number of heterogeneous resources.
Therefore, the user has no easy means of compiling a reasonably complete
list of these resources. Yet, even if the user could provide such a resource
list, it would probably contain only a small fraction of all available resources.
This again may lead to reduced performance and an increase of idle time
while waiting for the results of the operation. Furthermore, even if such a
complete list were available, matching the individual machines’ capabilities
with the requirements of the data mining operation would be a difficult and
error-prone task. Finally, transferring the required data and application to
the selected machine is a tedious operation and depends on the knowledge
of the transport mechanisms provided by the Grid middleware.

For the above reasons, a resource broker must address the following re-
quirements:
a) Automated compilation of a reasonably complete list of resources cur-
rently available
b) Automated selection of the most suitable execution machine from this
list
c) Interaction with the Grid middleware in order to transfer the data and
the application to the selected machine.

The broker has to carry out these operations without direct user inter-
action.

3) The Grid middleware must provide mechanisms for monitoring jobs.
One of the basic features of job execution in general, and on the Grid in
particular - is the monitoring job state capability. Users must be able to
retrieve information about the current job’s execution status from the time
it is submitted until the time when the user receives the computational re-
sults. This also includes as much additional information as possible (e.g.
error logs, percentage of completion, etc). As a resource broker should per-
form all matching and submission activities in the system, it is clear the
broker is the only component, which has all the information needed to pro-
vide sophisticated monitoring functionality.

4) Any resource broker must be extensible without forcing all users to
install subsequent updates. Resource brokering in a heterogeneous, fast-changing, distributed environment is a non-trivial task. Therefore, it is reasonable to assume that any implementation of a resource broker will pass through many development cycles. In the process, the broker’s functionality and performance may be enhanced. Additionally, new emerging Grid technology and middleware may require revising the broker. However, acquiring and installing subsequent updates is a tedious task for the user to do and usually results in bad customer satisfaction.

5) All services must be compatible with existing Grid standards. Even in the fast changing area of Grid technology adherence to standards and de-facto standards used by large parts of the community is crucial for building future-proof and accepted systems.

2.2.5 The job submission process

The system developed in the DataMiningGrid project is based on the most recent version of the Globus Toolkit. Figure 2.8 illustrates the interaction of services and components during composition of the workflow and job submission, which correspond to steps one and two mentioned in section 2.2.2. The figure does not describe step three, retrieval and annotation of results, as this step does not depend on the resource broker itself.

Figure 2.8: Interaction of services during job submission in the DataMiningGrid system
The process of job submission starts at the workflow editor, the only component to be installed on the client machine. This step of composing the workflow involves querying the Information Integrator service framework for available Data Services and Analysis Services (1-4). While the user composes the workflow, the system automatically creates a job description in the background according to the user’s choices regarding the data and applications he wishes to use. Thereby, the physical location of the data and applications incorporated in the workflow is entered automatically into the job description without direct interaction from the user. During this process the user may specify hundreds of simultaneous executions (e.g. parameter sweep). However, this job description does not specify the machines to execute the jobs on. After the workflow has been compiled it is submitted to the workflow manager for execution (5). The workflow manager resides on a dedicated machine in the Grid and is responsible for executing the workflow in the correct order. During execution of the workflow, the workflow manager automatically passes all job descriptions to the resource broker (6). The broker chooses one or multiple optimal execution machines based on the information about application requirements, data sizes, data permissions and workload on individual execution machines in the Grid. It retrieves this information partially from the job descriptions themselves and partially from the Information Integrator framework (7-10). According to the chosen execution machine the broker completes the job description regarding the stage-in and stage-out of input and output files. It finally passes the descriptions on to the job execution service from Globus running on the selected execution machine (11). The job execution service contacts the data services on the respective machine, which use GridFTP [3] to transfer the files before and after execution and executes the actual application (12-17). This also includes the respective application executables themselves. If a selected machine is residing in a cluster, the scheduler from that cluster is contacted, which eventually schedules the execution. During the whole process of composing the workflow and executing it in the Grid, both the physical location of the data and applications incorporated in the workflow, as well as the operations performed for resource discovery and scheduling are transparent to the user.
2.2.6 The resource broker

Unfortunately, the Globus Toolkit does not provide a resource broker that is capable of scheduling jobs over sites from different organizations (A and B), as displayed in Figure 2.8. We selected the GridBus Grid Service Broker and Scheduler v.2.4 [130] to fulfill this task for the following reasons:

- The GridBus resource broker is capable of submitting jobs to Globus’ execution subsystem as well as to many other computational resources (e.g. Alchemi, Unicore, XGrid, and others). To achieve this purpose the broker translates the job description it receives into the format of the specific middleware controlling the selected resource. The translated job description also contains information for the Grid middleware regarding transfer of the required data and application. The transfer operation itself is carried out without any direct user interaction (requirement 2).

- The GridBus broker’s architecture is clearly structured and well designed from a software engineering point of view.

- Unlike many other resource brokers, this one does not require any particular information or security system. It is designed as a stand-alone piece of software, which can be integrated with various existent components.

However, despite being the most reasonable choice, a careful investigation of the resource broker also revealed several drawbacks in its design, which prevented us from using the broker in its original state.

1. The broker was not service based, but needed to be installed on every client machine.

2. The broker did not provide mechanisms for automated compilation of a list of the computational resources currently present in the Grid, but required the user to do so. This list should be compiled from the Monitoring and Discovery System 4 (MDS4), which is the implementation of an information system in the Globus Toolkit.

3. The broker was unable to query MDS4 for obtaining the status of the resources contained in the list.
Internal design of the resource broker

Figure 2.9 illustrates the internal design of the resource broker including the modifications (in red) described in this work. The design of the broker is composed of three main sub-systems:

- The application interface sub-system
- The core-sub-system
- The execution sub-system

The interface layer presents the inputs of the broker. There are two main inputs:

A) An application description list, which corresponds with the job description created during composition of the workflow, and represents the user request (job) that needs to be executed.

B) A resource description list, which represents the available Grid resources for executing the job. The application description list (A) together with the delegated user’s credentials is received from the client through the Web service interface of the broker. The list containing resource descriptions (B) is compiled for each execution, based on the data from the Information System. The second list represents the current state of the available computational resources in the system.

In the broker core layer the above inputs are converted into "jobs" and "servers" - where "job" is the abstraction for a unit of work that will be assigned to some computational node, abstracted by "server". Once the jobs are prepared and the servers are discovered, the scheduler is started. The scheduler maps jobs (i.e. submits jobs using the actuator component in the execution sub-system) to suitable servers based on its algorithm. The actuator is a middleware specific component, which dispatches the job to the remote Grid node. On receiving a job submission, each server uses its associated server-manager to actuate the middle-ware specific job-submitters (also known as Agents). The job-monitor updates the book-keeper by periodically monitoring the jobs using the services of the execution sub-system. As the job gets completed, the agent takes care of cleaning up and gathering the output of the jobs. The scheduler stops after all the jobs have been scheduled. The scheduling policy determines whether failed jobs are restarted or ignored [130].
Figure 2.9: Enhanced GridBus resource broker architecture
WSRF wrapping

The original version of the GridBus resource broker (2.4) comes as a stand-alone application that has to be installed on every client that wishes to use its functionality. While this original design may be feasible if interfacing with Grid middleware not based on a service-based architecture, it is highly undesirable in service-based systems. In order to fit it into a service-oriented architecture we wrapped it up as a WSRF-compliant service, exposing its main features through a predefined service interface. This modification provides three key advantages:

1. The broker’s service interface adheres to the WSRF specifications. WSRF is currently the de-facto standard in the field of Grid services. Although its specifications have been submitted to the standard bodies of OASIS, it is not yet an adopted standard. Nevertheless, WSRF has been widely adopted by the Grid community already. For these reasons, the WSRF wrapper raises the acceptance of the resource broker and makes it more future-proof (requirement 5).

2. It is now possible to change the broker’s internals and to introduce new functionality such as more sophisticated schedulers without forcing each user to download and install a new version of the broker as long as the service interface does not change. This facilitates short development cycles with many possible updates, without resulting in discomfort for the users (requirement 4).

3. As the broker is implemented in Java, it already provides a certain degree of platform independence. However, wrapping it as a service also provides language independency, thus enabling any application to use the broker, regardless of the application’s programming language, as long as it is capable of interfacing with Web services. This is a crucial advantage for later exploitation of the individual packages resulting from the DataMiningGrid project.

4. According to the development team from GridBus a WSRF-compliant version of its broker, which will provide the full set of features through a Web interface is scheduled. Thus, exposing the broker’s main functions such as job scheduling and monitoring through our own service
at this time reduces the risk of incompatibilities of future versions of the broker with the rest of the Grid system that performs the job submission process explained in section 2.2.5.

Introduction of the Information Integrator Service Framework

In the context of the resource broker the role of the Information Integrator service framework developed in the DataMiningGrid project, is to provide the following information about computational resources:

- All computational resources, currently available in the Grid, from which the scheduler can choose.
- The capabilities of these resources regarding number of CPUs, main memory, etc.
- The current load of these resources.

The broker requires these pieces of information for choosing a suitable resource for executing the job described in the job description without any interaction by the user.

The framework automatically queries the information services provided by the Grid middleware (MDS4) and compiles a list of the resource entries maintained there. This approach results in two key advantages:

1. No user interaction is required for discovering available resources. By querying MDS4, which contains entries for each computational resource, we automate this process (requirement 2). As a result, the overall system becomes more user-friendly (requirement 1).

2. The original design of the broker allows to include different kinds of schedulers (e.g. cost-based, round robin, own developments, etc.). However, in some scenarios requirements may arise that are independent of the actual scheduler, for example certain data may only be processed on machines belonging to certain organizations. For this purpose, it is reasonable to exclude all other resources independent of the applied scheduler. As the Information Integrator framework automatically compiles a list of available resources, which the scheduler may choose, it can perform a preliminary step of filtering the list of resources before the actual scheduler is called (requirement 2). Thus, any subsequent scheduler applied may remain as is and can be exchanged with a different one without any modifications.
Chapter 3

Resource Brokering in Quasi-Opportunistic Grids

3.1 QosCosGrid System

3.1.1 Introduction

Supercomputer are dedicated, special-purpose multiprocessor computing systems that provide close-to-best achievable performance for demanding parallel workloads [67]. Supercomputers possess a set of characteristics that enable them to process such workloads efficiently. First, all the high-end hardware components, such as CPUs, memory, interconnects and storage devices are characterized not only by considerable capacity levels but also by a high degree of reliability and performance predictability. Second, supercomputer middleware provides a convenient abstraction of a homogeneous computational and networking environment, automatically allocating resources according to the underlying networking topology [8]. Third, the resources of a conventional supercomputer are managed exclusively by a single centralized system. This enforces global resource utilization policies, thus maximizing hardware utilization while minimizing the turnaround time of individual applications. Together, these features give supercomputers their unprecedented performance, stability and dependability characteristics.

The vision of grids becoming powerful virtual supercomputers can be attained only if their performance and reliability limitations can be overcome. Due to the considerable differences between grids and supercomputers, the
realization of this vision poses considerable challenges. Some of the main challenges are briefly discussed below.

**The co-allocation of a large number of participating CPUs.** In conventional supercomputers, where all CPUs are exclusively controlled by a centralized resource management system, the simultaneous allocation (co-allocation) and invocation (co-invocation) of processing units is handled by suitable co-allocation and co-invocation components [58]. In grid systems, however, inherently distributed management coupled with the non-dedicated (opportunistic) nature of the underlying resources makes co-allocation very hard to accomplish. Previous research has focused on co-allocation in grids of supercomputers and dedicated clusters [87], [82], [10]. The co-allocation problem has received little attention in the high-performance grid computing community. While co-allocation issues arise in other situations (e.g., co-allocation of processors and memory, co-allocation of CPU and networks, setup of reservations along network paths), the dynamic, non-dedicated nature of grids presents special challenges [37]. The potential for failure and the heterogenous nature of the underlying resource pool are examples of such special challenges.

**Synchronous communications.** Typically, synchronous communications form a specific communication topology pattern (e.g., *stencil exchange* in MM5 [72] and *local structures* in complex systems [46]). This is satisfied by supercomputers via special-purpose, low-latency, high-throughput hardware as well as optimized allocation by the resource management system to ensure that the underlying networking topology matches the application’s communication pattern [8]. In grids, however, synchronous communication over a wide area network (WAN) is slow, and topology-aware allocation is typically not available despite the existing support of communication libraries [53].

**Allocation of resources does not change during runtime.** While always true in supercomputers, this requirement is difficult to satisfy in grids, where low reliability of resources and WANs, as well as uncoordinated management of different parts of the grid, contribute to extreme fluctuations in the number of available resources.

**Fault tolerance.** In large-scale synchronous computation, the high sensitivity of individual processes to failures usually leads to termination of the entire parallel run in case of such failures. While rare in supercomputers
because the high specifications of their hardware, system and component failures are very common in grid systems.

We define a quasi-opportunistic supercomputer as a grid system which could address the challenges mentioned above but still hide many of the grid-related complexities from applications and users.

In this work we present some of the early results coming out of the QosCosGrid project\(^1\) which is aimed at developing a quasi-opportunistic supercomputer. The main contributions of this work are that we

- Introduce and motivate the concept of a quasi-opportunistic supercomputer.
- Summarize the main requirements of a quasi-opportunistic supercomputer.
- Present a detailed system architecture designed for the QosCosGrid quasi-opportunistic supercomputer.

### 3.1.2 Requirements of a quasi-opportunistic supercomputer

Many real-world systems involve large numbers of highly interconnected heterogeneous elements. Such structures, known as complex systems, typically exhibit non-linear behavior and emergence [59]. The methodologies used to understand the properties of complex systems involve modeling, simulation and often require considerable computational resources that only supercomputers can deliver. However, many organizations wanting to model and simulate complex systems lack the resources to deploy or maintain such a computing capability. This was the motivation that prompted the initiation of the QosCosGrid project, whose aim it is to develop core grid technology capable of providing quasi-opportunistic supercomputing grid services and technology. Modeling and simulation of complex systems provide a huge range of applications requiring supercomputer or supercomputer-like capabilities. The requirements derived from the analysis of nine diverse complex systems applications are summarized below.

1. **Co-allocation.** Complex systems simulations require simultaneous execution of code on very high numbers of CPUs. In this context,
co-allocation also means that resources for a certain task are allocated in advance. Those resources must be negotiated in advance and guaranteed to be available when the task’s time slot arrives. This implies the need for a sophisticated distributed negotiation protocol which is supported by advance reservation mechanisms.

2. **Topology aware resource management.** A complex system simulation is usually composed of multiple agents performing tasks of different complexity. The agents are arranged in a dynamic topology with different patterns of communication. To execute such a simulation, appropriate computational and network resources must be allocated. To perform this task, resource co-allocation algorithms must consider the topological structure of resource requests and offers and match these appropriately.

3. **Economics-based grid.** In “best-effort” grids, local cluster administrators are likely to increase the priorities of local users, possibly disallowing remote jobs completely and thus disassembling the grid back into individual clusters. This is because administrators lack suitable incentives to share the resources. We believe that resource co-allocation must be backed up by an *economic model* that motivates resource providers to honor their guarantees to the grid user and force the user to carefully weigh the cost of resource utilization. This model is also intended to address the “free-rider” problem [5].

4. **Service-level agreements.** The economic model should be supported by formal agreements, whose fulfilment can later be confirmed. Thus, an expressive language is required to describe such agreements, along with monitoring, accounting and auditing systems that can understand such a language.

5. **Cross-domain fault tolerant MPI and Java RMI communication.** The majority of current fault-tolerant MPI and Java RMI implementations provide transparent fault tolerance mechanisms for clusters. However, to provide a reliable connection within a grid, a cross-domain, fault-tolerant and grid-middleware-aware communication library is needed.
6. **Distributed checkpoints.** In grids, nodes and network failures will inevitably occur. However, to assure that an entire application will not be aborted after a single failure, distributed checkpoints and restart protocols must be used to stop and migrate the whole application or part of it.

7. **Scalability, extensibility, ease of use.** In order to be widely accepted by the complex systems community, the QosCosGrid system must offer easy interfaces yet still allow extensions and further development. Additionally, for real-world grids, the system must be scalable in terms of computing resources and supported users.

8. **Interoperability.** The system must facilitate seamless interoperation and sharing of computing and data resources.

9. **Standardization.** To facilitate interoperation and evolution of the QosCosGrid system, the design and implementation should be based on existing and emerging grid and grid-related standards and open technology.

### 3.1.3 QosCosGrid System Architecture

The working hypothesis in QosCosGrid project is that a quasi-opportunistic supercomputer (as characterized in Section 3.1.1) can be built by means of a **collaborative grid** which facilitates sophisticated resource sharing across multiple **administrative domains (ADs)**. Loosely based on Krauter [83], a collaborative grid consists of several organizations participating in a **virtual organization (VO)** and sharing resources. Each organization contributes its resources for the benefit of the entire VO, while controlling its own administrative domain and own resource allocation/sharing policies. The organizations agree to connect their resource pools to a trusted “grid level” middleware which tries to achieve optimal resource utilization. In exchange for this agreement, partners gain access to very large amounts of computational resources.

The QosCosGrid architecture is depicted in Figure 3.1. The diagram depicts a simplified scenario involving three administrative domains (labeled Administrative Domain 1, 2 and 3). Administrative Domain 3 consists of two resource pools, each of which is connected to an AD-level service, located
in the center of the diagram. AD-level services of all the administrative domains are connected to a trusted, distributed grid-level service. The grid-level services are designed to maximize the global welfare of the users in the entire grid.

Grid fabric

The basic physical layer consists of computing and storage nodes connected in the form of computing cluster. The computing cluster is managed by a local resource management system (LRMS) – in our case a Platform Load Sharing Facility (LSF Cluster) – but may be replaced by other advanced job scheduling systems such as SGE or PBSPro. The LSF cluster runs batch or interactive jobs, selecting execution nodes based on current load conditions and the resource requirements of the application. The current load of CPUs, network connections, and other monitoring statistics are collected.
by the cluster and networking monitoring system, which is tightly integrated with LRMS and external middleware monitoring services. In order to execute cluster-to-cluster parallel applications, the QosCosGrid system supports the advance reservation of computing resources, in addition to basic job execution and monitoring features. Advance reservation is critical to the QosCosGrid system as it enables the QosCosGrid middleware to deliver resources on-demand with significantly improved quality of service.

**FedStage Open DRMAA Service Provider and advance reservation APIs**

A key component of the QosCosGrid architecture is a LRMS offering job submission, monitoring, and advance reservation features. However, for years LRMS provided either only proprietary script-based interfaces for application integration or nothing at all, in which case the command-line interface was used. Consequently, no standard mechanisms existed for programmers to integrate both grid middleware services and applications with local resource management systems. Thanks to Open Grid Forum and its Distributed Resource Management Application API (DRMAA) working group [127], the DRMAA 1.0 specification has recently been released. It offers a standardized API for application integration with C, Java, and Perl bindings. Today, DRMAA implementations that adopt the latest specification version are available for many local resource management systems, including SGE, Condor, PBSPro, and Platform LSF, as well as for other systems such as GridWay or XGrid. In the QosCosGrid we have successfully used FedStage² DRMAA for LSF and integrated those APIs with the Open DRMAA Service Provider (OpenDSP). OpenDSP is an open implementation of SOAP Web Service multi-user access and policy-based job control using DRMAA routines implemented by LRMS. As a lightweight and highly efficient software component, OpenDSP allows easy and fast remote access to computing resources. Moreover, as it is based on standard Web Services technology, it integrates well with higher level grid middleware services. It uses a request-response-based communication protocol with standard JSDL XML and SOAP schemas protected by transport level security mechanisms such as SSL/TLS or GSI. However, neither DRMAA nor OpenDSP provide standard advance reservation and resource synchronization APIs required by cross-domain parallel applications. Therefore, in the QosCosGrid project,

²http://www.fedstage.com/wiki/
we have extended DRMAA and proposed standard advance reservation APIs that are suited to the various APIs of underlying local resource management systems.

**QosCosGrid parallel cross-domain execution environments**

The QosCosGrid Open MPI (QCG-OMPI) is an implementation of the message passing interface that enables users to deploy and use transparently MPI applications in the QosCosGrid testbed, and to take advantage of local interconnects technology\(^3\). QCG-OMPI supports all the standard high-speed network technologies that Open MPI supports, including TCP/Ethernet, shared memory, Myrinet/GM, Myrinet/MX, Infiniband/OpenIB, or Infiniband/mVAPI. In addition, it supports inter-cluster communications using relay techniques in a manner transparent to users and can be integrated with the LSF Cluster. QCG-OMPI relates to a check-pointing interface that provides a coordinated check-pointing mechanism on demand. To the best of our knowledge, no other MPI solution provides a fault-tolerant mechanism on a transparent grid deployment environment. Our intention is that the QCG-OMPI implementation will be fully compliant with the MPI 1.2 specification from the MPI Forum\(^4\).

QosCosGrid ProActive (QCG-ProActive) is a Java-based grid middleware for parallel, distributed and multi-threaded computing integrated with OpenDSP. It is based on ProActive, which provides a comprehensive framework and programming model to simplify the programming and execution of parallel applications. ProActive uses by default the RMI Java standard library as a portable communication layer, supporting the following communication protocols: RMI, HTTP, Jini, RMI/SSH, and Ibis \([36]\).

**Administrative domain- and grid-level services**

Grid fabric software components, in particular OpenDSP, QCG-OMPI and QCG-ProActive, must be deployed on physical computing resources at each administrative domain and be integrated with the AD-level services. AD-level services, in turn, are connected to the Grid-level services in order to share and receive information about the entire grid as well as for tasks that cannot be performed within a single administrative domain. We distinguish five main high-level types of services:

\(^3\)http://www.open-mpi.org/
\(^4\)http://www.mpi-forum.org/
Grid Resource Management System

The Grid Resource Management System (GRMS) is a grid meta-scheduling framework which allows developers to build and deploy resource management systems for large-scale distributed computing infrastructures at both administrative domain and grid levels. The core GRMS broker module has been improved in QosCosGrid to provide both dynamic resource selection and mapping, along with advance resource reservation mechanisms. As a core service for all resource management processes in the QosCosGrid system, the GRMS supports load-balancing among LRMS, workflow execution, remote job control, file staging and advanced resource reservation. At the administrative level, the GRMS communicates with OpenDSP services to expose the remote access to underlying computing resources controlled by LRMS. Administrative domain-level GRMS is synchronized with the Grid-level GRMS during the job submission, job scheduling and execution processes.

At the grid level, the GRMS offers much more advanced co-allocation and topology-aware scheduling mechanisms. From the user’s perspective, all parallel applications and their requirements (including complex resource topologies) are formally expressed by an XML-based job specification language called QCG Job Profile. Those job requirements, together with resource offers, are provided to the GRMS during scheduling and decision making processes.

Accounting and economic services

Accounting services support the needs of users and organizations with regard to allocated budgets, credit transactions, auditing, etc. These services are responsible for (a) Monitoring: Capturing resource usage records across the administrative domains, according to predefined metrics; (b) Usage record storage: Aggregation and storage of the usage records gathered by the monitoring system; (c) Billing: Assigning a cost to operations and charging the user, taking into account the quality of service actually received by the user; (d) Credit transaction: The ability to transfer credits from one administrative domain to another as means of payment for received services and resources; (e) VO management: Definition of user groups, authorized users, policies and priorities; (f) Accounting: Management of user groups’ credit accounts, tracking budget, economical obligations, etc.; (g) Budget planning: Cost estimations for aggregation of resources according to the
pricing model; and (h) Usage analysis: Analysis of the provided quality of service using information from usage records, and comparison of this information to the guaranteed quality of service.

**Resource Topology Information Service (RTIS)**

The RTIS provides information on the resource topology and availability. Information is provided by means of the Resource Topology Graph (RTG) schema, instances of which depict the properties of the resources and their interconnections. For a simpler description process, the RTG does not contain a “point-to-point” representation of the desired connections but is based instead on the communication groups concept, which is quite similar to the MPI communicator definition. The main goals of the RTIS are to facilitate topology-aware services to discover the grid resources picture as well as to disclose information about those resources, on a “need-to-know” basis.

**Grid Authorization System**

Currently, the most common solution for mutual authentication and authorization of grid users and services is the Grid Security Infrastructure (GSI). The GSI is a part of the Globus Toolkit and provides fundamental security services needed to support grids [55]. In many GSI-based grid environments, the user’s identity is mapped to a corresponding local user identity, and further authorization depends on the internal LRMS mechanisms. The authorization process is relatively simple and static. Moreover, it requires that the administrator manually modify appropriate user mappings in the gridmap file every time a new user appears or has to be removed. If there are many users in many administrative domains whose access must be controlled dynamically, the maintenance and synchronization of various gridmap files becomes an important administrative issue. We believe that more advanced mechanisms for authorization control are required to support dynamic changes in security policy definition and enforcement over a large number of middleware services. Therefore, in QosCosGrid we have adopted the Grid Authorization Service (GAS), an authorization system integrated with various grid middlewares such as the Globus Toolkit, GRMS or OpenDSP. GAS offers dynamic fine-grained access control and enforcement for shared computing services and resources. Taking advantage of the strong authentication mechanisms implemented in PKI and GSI, it provides crucial security mechanisms in the QosCosGrid system. From the QosCosGrid architecture perspective, the GAS can also be treated as a trusted
single logical point for defining security policies.

**Service-Level Agreement Management System**

In order to enforce the rules of the economic system, we employ a *service-level agreement* (SLA) protocol [94]. A SLA defines a dynamically established and managed relationship between the resource providers and resource consumers. Both parties are committed to the negotiated terms. These commitments are backed up by organization-wide policies, incentives, and penalties to encourage each party to fulfill its obligations. For each scheduled task, a set of SLAs is signed by the administrative domain of the task owner, and by each of the provider administrative domains. The SLA describes the service time interval, and the provided QoS – resources, topology, communication, and mapping of user processes to provider’s resources. SLAs are represented using the RTG model, and are stored in RTIS. The SLA Compliance Monitor analyzes the provided quality of service for each time slot, and calculates a weighted compliance factor for the whole execution. The compliance factor is used by the pricing service (which is a part of accounting services) to calculate the costs associated with the service if it is provided successfully, or the penalties that arise when a guarantee is violated.

### 3.1.4 Conclusions

The main objective of the QosCosGrid project is to address some of the key technical challenges to enable development and execution of large scale parallel experiments across multiple administrative, resource and security domains. In this chapter we presented the main requirements and important software components that make up a consistent software architecture. This high-level architecture perspective is intended to give readers the opportunity to understand a design concept without the need to know the many intricate technical details related in particular to Web service, WSRF technologies, remote protocol design, and security in distributed systems. The QosCosGrid architecture is designed to address key quality-of-service, negotiation, synchronization, advance reservation and access control issues by providing well-integrated grid fabric, administrative domain- and grid-level services. In contrast to existing grid middleware architectures, all QosCosGrid system APIs have been created on top of carefully selected third-party services that needed to meet the following requirements: open
standards, open source, high performance, and security and trust. Moreover, the QosCosGrid pluggable architecture has been designed to enable the easy integration of parallel development tools and supports fault-tolerant cluster-to-cluster message passing communicators and libraries that are well known in high-performance computing domains, such as Open MPI, ProActive and Java RMI. Finally, we believe that without extra support for administrative tools, it would be difficult to deploy, control, and maintain such a big system. Therefore, as a proof of concept, we have developed various client tools to help administrators connect sites from Europe, Australia and the USA. After the initial design and deployment stage, we have begun carrying out many performance tests of the QosCosGrid system and cross-domain application experiments. Collected results and analysis will be taken into account during the next phase of system re-design and deployment.

3.2 QosCosGrid Resource Broker

3.2.1 Introduction

Many distributed, tightly-coupled applications such as complex systems simulations, weather forecasting, and fluid dynamics, rely on co-allocation of large numbers of reliable resources. While this requirement has traditionally been met by supercomputing facilities, it is appealing to use the production multi-cluster grids (e.g., Egee\textsuperscript{5}, OSG\textsuperscript{6}) as a more economic computing resource. However, executing such applications on large-scale cross-site grids poses the following challenges: a) as opposed to supercomputers, the grids have heterogeneous network topology of loosely coupled clusters, complicating the co-allocation of grid resources in accordance with the application communication requirements, and b) it is not trivial to achieve high utilization of the resources while scheduling multiple complex applications on the time axis.

The QosCosGrid’s novel topology-aware development paradigm allows application modelers to design adaptive, topology-aware applications by aggregating different processes into communication groups with certain topology requirements. This approach provides more flexibility to the meta-

\textsuperscript{5}http://public.eu-egee.org/
\textsuperscript{6}http://www.opensciencegrid.org/
scheduling framework, allowing it to optimize the co-allocation process. The meta-scheduler may, for example, choose to split a group of scarcely interacting processes between two remote clusters and place frequently interacting processes on some high performance network.

Figure 3.2: A parallel, topology-aware evolutionary algorithm.

An example of a topology-aware application from the QosCosGrid [26, 35] is depicted in Figure 3.2. This shows an evolutionary algorithm that is used to reverse-engineer gene regulatory networks. In this evolutionary algorithm the population is divided into subpopulations that can be distributed over different compute clusters at different sites or physical locations. This is an example of the distributed island model of an evolutionary algorithm, where multiple subpopulations are optimized in order to overcome problems associated with local minima, and individuals are exchanged or “migrated” between subpopulations at periodic intervals. The application has a two-layer topology: the lowest layer corresponds to a subpopulation distributed upon a lattice topology according to the cellular model of evolutionary algorithms, where each individual exchanges genetic material only with its immediate neighbours; and the highest layer corresponds to a ring topology for communication between subpopulations. The example in Figure 3.2 may be extended to include more sites, with one or more subpopulations at each site.

In this work we propose an entire grid-level framework for scheduling topology-aware applications. This framework consists of a schema for efficient description of the requests and offers, and pluggable components that are responsible for all the steps of the scheduling process: assignment of
tasks on the time axis, clustering of the resources, and matching the graphs of resource requests and offers (we call this stage topology-aware co-allocation).

The proposed scheduling framework relies on the advance reservation mechanism of the local job schedulers to enforce its decisions, and in practice the system is tested with the LSF scheduler\(^7\). The final tests will include real-life applications from the domains of stellar dynamics, protein folding, social influence and discrete choice, ecological evolution, and business supply chain evolutionary dynamics, under the umbrella of the QosCosGrid project.

### 3.2.2 Scheduling Input Format

Large-scale, topology-aware parallel applications are usually composed of hundreds to thousands of processes. However, in practice, these applications can be efficiently described in terms of “process groups” rather than in terms of single processes. This description can significantly minimize the size of a given problem with no loss of description accuracy. For instance, the MPI communicators abstraction usually implies that all the machines in a certain communicator should have similar properties and should be interconnected by similar all-to-all links.

Figure 3.3 depicts a typical request of a topology-aware application as described in the proposed framework. In this request, a user asks for four process groups called PG1 – PG4. Each process group is mapped to a computing resource template that describes the properties of the requested identical machines. For example, PG4 requires between 4 and 6 computing resources, each of which conforms to a certain computing resource template. The properties of a computing resource template are described in terms of ranges. For example, the computing resource template requested by both PG4 and PG1 might be specified as follows: [clock rate in range of 2...4MHz], [memory in range of 1...2GB], and [free disk space in range of 2...∞ GB].

All the ranges mentioned in the text have the following origins: the minimal value is dictated by application requirements, as the application cannot run with less than min resources, while the maximum is usually dictated by the maximal amount that the user can/wishes to pay for the task’s execution. This description allows flexibility for the resource description.

\(^7\)http://www.platform.com/Products/Platform.LSF.Family/
Additionally, it helps in dealing with the fluctuating nature of the grid, as resources might fail during different stages of a job life cycle.

Process groups (PGs) are arranged in process communication groups (PCGs), where it is assumed that all processes within a PCG would like to have all-to-all interconnections with certain properties. The quantitative properties of these interconnections are specified by means of the network resource templates. For instance, PCG2 (green oval) contains two process groups – PG2 and PG3, which means that all the processes of these two PGs must have all-to-all interconnections as described in the PCG2’s network resource template. Like the computing resource templates, the network resource templates are described in terms of ranges. A sample template might be specified as follows: \([\text{bandwidth in range of } 10...54\text{Mbit/sec}], \text{ and } [\text{latency in range of } 0.001...0.01\text{sec}]\).

The aforementioned description format is called a resource topology graph (RTG). In addition to the computing and network resources description, the request RTG also contains the estimated runtime of the requested task and optional time bounds for the task’s execution.

Like the description of the requests, the offered machines are also described in terms of an RTG. Figure 3.4 depicts a sample offer, as published by three administrative domains (ADs). Each AD publishes the offered resources along with their properties, such as memory and disk space, and the properties of its network resources. The network resources are described.
in terms of resource communication groups (RCGs), which, by definition, means that all the resources within an RCG have identical all-to-all inter-connections. Additionally, the inter-cluster connections connect two RCGs in different ADs. An offer RTG also contains the description of resource availability on the time axis.

![Resource Communication Group (RCG)](image)

Figure 3.4: A sample representation of offered machines.

It is important to stress that the purpose of the request and offer RTGs is to simplify and make more efficient the description of the requested and available resource topologies by users and system administrators. Additionally, to simplify the description and avoid errors, the real values can be replaced by several predefined parameter groups. For instance, the clock-rate parameter can be described as “fast/medium/slow”. In this way, users can describe their requirements more accurately while requesting resources according to their current budgets.

### 3.2.3 Model Formalization

The scheduling system has three inputs: resource topology graph (RTG) of requests, RTG of offers, and the scheduling time slot. The scheduling time slot specifies the time bounds of the scheduling round. It is dictated by the scheduler invoker and may vary according to whether the scheduling scenario is, for instance, recurrent or ad hoc.

#### Formalizing the resource request

The RTG of a requested task contains the description of required resources, the task’s estimated runtime, and optional runtime bounds: \( \text{requested}_{\text{start}} \) and \( \text{requested}_{\text{end}} \) between which the task must be executed (\( \text{requested}_{\text{end}} - \text{requested}_{\text{start}} \)).
requested_{start \geq runtime}). This RTG is translated into a graph \( G = (V, E) \), where \(|V| = n \) vertexes denote the task’s process groups (PGs). Each PG \((v_i)\) requires a number of identical resources (a cluster of identical machines) according to its size. The sizes of PGs are given as ranges denoted by the vectors \( \text{CAP}_{\text{min}} = [\text{cap}_{\text{min}1}, \ldots, \text{cap}_{\text{min}n}] \) and \( \text{CAP}_{\text{max}} = [\text{cap}_{\text{max}1}, \ldots, \text{cap}_{\text{max}n}] \). The properties of each machine in the requested clusters are presented as property vectors \( C_{\text{min}} = [c_{\text{min}1}, \ldots, c_{\text{min}n}] \) and \( C_{\text{max}} = [c_{\text{max}1}, \ldots, c_{\text{max}n}] \), where \( c_{\text{min}i}, c_{\text{max}i} \) denote the minimal and the maximal quantitative properties of each required computing resource in cluster \( v_i \) (e.g., FLOPS). Different quantitative properties might be described by multiple property vectors.

The process communication groups are translated into graph edges \( E \), denoted by \( n \)-by-\( n \) adjacency matrices \( B_{\text{min}} \) and \( B_{\text{max}} \), where \( b_{\text{min}ij}, b_{\text{max}ij} \) refer to the minimal and maximal connectivity level of the machines in cluster \( v_i \) and \( v_j \). Matrices \( B_{\text{min}} \) and \( B_{\text{max}} \) represent the communication latency between and within the requested clusters as requested by the user.

**Formalizing the resource offer**

Analogously, an offer RTG is translated to a graph \( \hat{G} = (\hat{V}, \hat{E}) \), where \(|\hat{V}| = m \). Each vertex \( \hat{v}_j \) represents a cluster of identical offered machines. A capacity vector \( \hat{\text{CAP}} = [\hat{\text{cap}}_1, \ldots, \hat{\text{cap}}_m] \) denotes the number of available machines in each cluster. To simplify the formalization, we will assume that each machine has a single CPU, although extending the model to handle an unrestricted number of CPUs per machine is straightforward, moreover, in the following sections, this assumption will be discarded. The quantitative properties (e.g., FLOPS) of machines in each cluster are denoted by the vector \( \hat{C} = [c_1, \ldots, c_n] \). In contrast to the request RTG, the offer RTG has no ranges, as it represents the values of real machines in the grid.

An \( m \)-by-\( m \) adjacency matrix \( \hat{B} \) represents the edges’ properties (e.g., the latency within and between the \( m \) clusters in the grid), assuming identical connectivity properties between all the machines in each cluster. Additionally, each machine \( r \) in cluster \( j \) has a list of time slots \( T = \{(t_{\text{start}rj}^1, t_{\text{end}rj}^1), (t_{\text{start}rj}^2, t_{\text{end}rj}^2), \ldots\} \) during which the resource is marked as “available.”
Formalizing the output

The goal of the grid-level scheduler is to provide a scheduling plan for a large number of requests and offers. For each task, we define an $n$-by-$m$ allocation matrix $X$ in which the term $X_{ij} = k$ represents an allocation of $k$ processes of a requested process group $v_i$ to an offered cluster $\hat{v}_j$. The allocation matrix $X$ must satisfy the following constraints:

$$\forall i : 1 \leq i \leq n, \quad cap_{\text{min}}_i \leq \sum_{j=1}^{m} X_{ij} \leq cap_{\text{max}}_i,$$  \quad (3.1)

denoting that each process group must be served by a range of $cap_{\text{min}}_i \ldots cap_{\text{max}}_i$ offered resources;

$$\forall j : 1 \leq j \leq m, \quad \sum_{i=1}^{n} X_{ij} \leq \hat{cap}_j,$$  \quad (3.2)

denoting that an offered cluster $j$ can serve at most $\hat{cap}_j$ processes;

$$\forall i, j : 1 \leq i \leq n, 1 \leq j \leq m, \quad \text{if (NOT } c_{\text{min}}_i \leq \hat{c}_j \leq c_{\text{max}}_i \text{) then } (X_{ij} = 0),$$  \quad (3.3)

denoting that if offered resources in cluster $j$ do not match the requested process group $i$, then no allocations are allowed;

$$\forall i, j, k, l : 1 \leq j, l \leq m, 1 \leq i, k \leq n, \quad sgn(X_{ij}X_{kl})b_{\text{min}_{ik}} \leq sgn(X_{ij}X_{kl})b_{\hat{j}l} \leq b_{\text{max}_{ik}},$$  \quad (3.4)

denoting that the connectivity (edge) properties of the requests must match the connectivity properties of offers\,\footnote{$\text{sgn}(X_{ij}) = \begin{cases} 1 & \text{if } X_{ij} > 0 \\ 0 & \text{if } X_{ij} = 0 \end{cases}$, as $X_{ij} \geq 0$} (formalized only for the latency metric due to space limitations); and

$$\forall i, j : 1 \leq i \leq n, 1 \leq j \leq m, X_{ij} \in \{0, \ldots, \text{min}(cap_{\text{max}}_i, \hat{cap}_j)\},$$  \quad (3.5)

denoting that the number of allocations is an integer.

In addition to the constraints on matrix $X$, there are the following time constraints:
\begin{align}
task_{\text{start}} & \geq \text{requested}_{\text{start}} \land \task_{\text{end}} \leq \text{requested}_{\text{end}} \land \\
\task_{\text{end}} - \task_{\text{start}} & = \text{runtime}, \tag{3.6}
\end{align}

denoting that the actual task’s start and end times must fit the given bounds; and

\begin{align}
\forall i, j : 1 \leq i \leq n, 1 \leq j \leq m, \quad X_{ij} > 0 \Rightarrow \exists R_j = \{ r_{1j}, r_{2j}, \ldots \}, \\
\forall r \in R_j, \exists k : \task_{\text{start}} \geq t_{\text{start}, r, j} \land \task_{\text{end}} \leq t_{\text{end}, r, j} \land |R| = X_{ij} \tag{3.7}
\end{align}

denoting that only “available” resources can be matched.

### 3.2.4 Scheduling Framework

Scheduling general topology-aware applications is very difficult. In [85] it was shown that even the coallocation of a task consisting of single processes to homogeneous clusters in a single time slot is NP-complete and cannot be approximated by polynomial time algorithms. Therefore, our strategy in the entire framework is to simplify the problem as far as possible by dividing it into small sub-problems, minimize the problems’ size by means of clustering, and eventually solve hard problems by repetitively solving easy ones. The main task-by-task scheduling algorithm of the proposed framework is depicted in algorithm 3.2.4.

**Algorithm 1** Scheduling Algorithm

**Require:** readyTasks : set of tasks to schedule

\begin{algorithmic}
\While {readyTasks \neq \emptyset}
\State \text{task} \leftarrow \text{get next task from the ordering algorithm}
\State run clustering algorithm on \text{task}
\State run coallocation algorithm on \text{task}
\If {coallocation succeeded}
\State update available resources
\EndIf
\State remove \text{task} from readyTasks
\EndWhile
\end{algorithmic}

The following subsections describe in detail the implementation of the ordering, clustering, and coallocation algorithms.
**Ordering algorithm**

It is well known that the classic parallel machine scheduling problem for minimizing the makespan ($P_m/|C_{max}$) is NP-hard [91]. Moreover, given that the allocation of a single task cannot be approximated by the polynomial time algorithms, it is clear that scheduling several such tasks can only be handled by means of heuristics. In this subsection we propose two novel generic scheduling algorithms that, in practice, achieve very reasonable results and can work with any given utility function.

The goal of the ordering phase in a scheduling algorithm is to find the order $O$ of tasks so that scheduling according to this order will maximize a given utility function. The proposed scheduling framework is built to support any utility function, as long as it allows the comparison of $utility(O_1)$ and $utility(O_2)$. We experimented with utility functions such as maximization of resource utilization, fairness maximization, task satisfaction maximization, makespan minimization, and a combination of the above.

Before running any ordering algorithm, we have to know the potential start times for a given task. To this end, an initial resource availability matrix, $AVAIL$, is calculated. The rows of this matrix contain the Cartesian product of process groups (PGs) and offered computing resource clusters ($R_x$). The rows where an offered cluster does not comply with a certain PG are discarded. Figure 3.5 illustrates a sample transformation of resource availability data (on the left) to the resource availability matrix $AVAIL$ (on the right). Each column represents a start or end time of resources availability. The gray column headers denote tasks’ potential start times, as at these times one or more resources were added. The numbers in the table represent CPUs that are available starting from $t_i$ (inclusive) until $t_{i+1}$ (exclusive). Resource availability periods that are smaller than the task’s runtime do not appear in the table.

Finding the start time of a task is not trivial. In order to be sure that a task can run at a certain time $t$, the coallocation algorithm (Section 3.2.4) must be executed. The coallocation algorithm takes into account all the computing (CPU, memory, etc.) and connectivity (latency) constraints, i.e., equations 3.1–3.5, and returns an accurate answer: coallocation matrix $X$ if a task can run at time $t$ and “failure” otherwise. However, the coallocation algorithm is fairly heavy and we would like to reduce the number of its
invocations, namely, to check as few potential start times $t$ as possible. In order to speed up the ordering algorithm, we used two additional verification procedures, which are less accurate than the coallocation algorithm but much faster. The goal of these verification procedures is to rule out the start times $t$ at which not enough resources are available.

The verification procedure 1 (VP1), depicted in algorithm 2, returns the first potential start time $t$ for a task, when seemingly enough resources are available. This is the fastest but most inaccurate verification test. If VP1 returned a potential start time $t$ for a task, this still does not guarantee that the task will be able to run at $t$ (possible false positives). Verification procedure 1 does a very rough counting of resources suitable for each process group (PG), and if at least one PG does not have enough resources, time $t$ is marked as unsuitable for the task.

**Algorithm 2 VP1: Finding a potential start time**

**Require:** AVAIL: availability matrix

```
for all $t \in AVAIL_{start\_times}$ do {gray columns}
    if $\forall i \in 1..n : \sum_{j=1..m} AVAIL[PG_i, R_j][t] \geq cap_{min}$, then
        return $t$
    end if
end for
return task cannot be executed – insufficient resources
```

Verification procedure 2, depicted in algorithm 3, is more computationally intensive than the VP1, and therefore it returns a more accurate answer. In this procedure, we check separately whether mapping of vertexes (requested and offered machines) and mapping of edges (connectivity) can
be done at time $t$. This algorithm is supposed to cover the drawback of the first verification procedure, which could count the same offered resource more than once for several process groups. The second test involves solving a linear programming problem that contains equations 3.1–3.3. This test is relatively quick, as these equations form a capacitated transportation problem [40], which is solvable by pure linear programming techniques (e.g., the interior-point method) very efficiently. A capacitated transportation problem has a totally unimodular constraint matrix [47], and its solution is thus guaranteed to also satisfy equation 3.5 (i.e., it is an integer solution).

Algorithm 3 VP2: Testing whether a task can run at $t$

Require: task: task to verify
  $t$: task’s potential start time
cluster $R$ by algorithm 6
solve the following linear problem:
maximize $\sum_{i=1}^{n} \sum_{j=1}^{m} X_{ij}$ subject to:
$\forall i: 1 \leq i \leq n, \quad \text{cap}_{\min} \leq \sum_{j=1}^{m} X_{ij} \leq \text{cap}_{\max}$;
$\forall j: 1 \leq j \leq m, \quad \sum_{i=1}^{n} X_{ij} \leq \hat{c}_j$;
$\forall i,j: 1 \leq i \leq n, 1 \leq j \leq m,$
  if (NOT $c_{\min} \leq \hat{c}_j \leq c_{\max}$) then ($X_{ij} = 0$);
if no feasible solution was found then
  return task cannot be executed at $t$
end if
for all $e \in E$
  if $\nexists \hat{e} \in \hat{E}$ so that $\hat{e}$ satisfies $e$ then
    return task cannot be executed at $t$
  end if
end for
return passed 2nd verification procedure for task at $t$

Simulated Annealing Ordering Algorithm

The first ordering algorithm that we present is based on the simulated annealing (SA) technique applied to the cutting stock problem [88]. Algorithm 4 starts from an initial order $O$ and starting temperature. Each step of the SA algorithm replaces the current solution with a random “nearby” solution: the probability of the replacement depends on the current temperature and the utility of the new solution. The main idea of this algorithm is
to evaluate different orders of task submission by means of an “optimistic” approach, namely VP1 and VP2 only. Repeatedly, as the temperature decreases, the algorithm fixes the start times of more and more tasks in the currently best order. The times are fixed by means of the coallocation algorithm, which returns an exact answer to the question of whether “a task can run at \( t \).” If the coallocation is successful, the task is marked as scheduled; otherwise the algorithm tries to find another start time for a task. This procedure is repeated until maximal runtime is exceeded or a target utility is reached.

**Algorithm 4** Simulated annealing ordering algorithm

```
Require: Tasks: list of tasks to schedule

anchor_distance ← |Tasks|

repeat
    repeat
        \( O \leftarrow \text{generate an order of Tasks} \)
        utility(\( O \)) ← \text{calculate utility of } O
        if accept(temperature, change-in-utility) then
            current_order ← \( O \)
            \( O^* \leftarrow \text{best}(O, O^*) \)
        end if
    until equilibrium not reached

decrease temperature
update anchor_distance
if anchor_distance decreased then
    fix first_task
end if
Tasks ← Tasks\( \setminus \text{first_task} \)
until runtime ≥ MAX or utility(\( O \)) ≥ TARGET
try coallocating unallocated tasks

return \( O^* \)
```

Lines 3–10 depict the search for a “better” task ordering with constant temperature \( T \). In line 4, a new order \( O \) of Tasks is generated. In the first iteration of the repeat loop, this order is generated either randomly or by means of a simple heuristic (e.g., longest tasks first). In all subsequent iterations, the order \( O \) is generated by swapping the places of two random tasks in the current order. The created order \( O \) is evaluated by means of the
“calculate utility” function (line 5). The calculate utility function creates a tentative scheduling of tasks on the time axis applying VP1 and VP2 (no coallocation algorithm is involved yet), and then calculating the utility of the scheduled tasks. The utility is defined by the given utility function, for instance, maximize offered resource utilization. The pseudo-code of the “calculate utility” function is as follows:

\[
\text{for all } task \in \text{Tasks do}
\begin{align*}
&\text{repeat} \\
&t \leftarrow \text{get potential start time for } task \text{ from VP1} \\
&\text{run } \text{VP2}(t, task) \\
&\text{until } \text{VP1} \& \text{VP2 passed or } t \text{ not found} \\
&\text{if } \text{VP2}(t, task) \text{ passed then} \\
&&\text{mark } task \text{ as scheduled} \\
&&\text{update available resources} \\
&\text{end if} \\
\end{align*}
\text{end for}
\]
\[
\text{return calculate utility of scheduled tasks}
\]

In lines 6–9 the decision is made whether to replace the current order with the newly created order \( O \) or to retain the current one. This decision is made using the following accept function:

\[
\text{if } \text{change-in-utility} > 0 \text{ then } \{ \text{better solution found}\}
\]
\[
\text{return true}
\]
\[
\text{return } \exp(-\alpha/\text{temperature}^2) > \text{random}(0, 1)
\]

The goal of the accept function is to allow non-improving steps in the beginning of the algorithm, when the temperature is high, and to allow only improving steps in the final stages of the algorithm, when the temperature is low. During the search, the best order \( O^* \) is recorded and used to check whether “equilibrium” (lines 3–10) was reached. Equilibrium means that the system cannot find an improving order \( O' \) after a certain number of iterations [88].

Line 11 is responsible for updating the temperature parameter. The temperature is decreased according to the function \( T_{i+1} = \frac{T_i}{(1+\beta T_i)} \), where

\[
\beta = \frac{T_{\text{initial}}-T_{\text{final}}}{\text{MAX_STEPS} \cdot T_{\text{initial}} \cdot T_{\text{final}}} \]

The anchor distance parameter (line 12) is used to represent the number
of tasks that can be swapped (each iteration of the inner repeat loop of the SA algorithm involves the swapping of 2 tasks). In the beginning, the anchor distance equals the number of tasks, meaning that swapping is allowed between all the tasks, i.e., any task can move. As the algorithm proceeds, the anchor distance decreases to \(i\), meaning that swapping is allowed only between the last \(i\) tasks. The rationale behind this extension of the classical simulated annealing technique is that the tasks are ordered using inexact information – verification procedures 1 and 2. At some point we would like to fix the first task in the currently best order and check that this task can be coallocated as expected; this is done in lines 13–15.

Line 14 denotes the execution of the coallocation algorithm on the first_task of the currently best order. The task’s start time \(t\) was previously found by VP1. If the coallocation succeeds, the first_task is marked as scheduled and the availability of appropriate resources is updated accordingly. The failure to coallocate the first_task means that the calculations of the last steps (since fixing the previous task) were based on the incorrect optimistic assumption that the first_task could run at \(t\). In order to avoid recalculations of the entire step, a recovery procedure is performed. The recovery procedure temporarily fixes all tasks except the first_task in the Tasks list as if they were scheduled in the current order \(O\) on resources found by VP2. Then it tries coallocating the first_task at some other time \(t'\). If the coallocation at \(t'\) succeeded, the first_task is marked as scheduled and the availability of the appropriate resources is updated. Otherwise, the task is marked as “not allocated.” In any case, the first_task is removed from the Tasks list (line 16). At the end of the algorithm, all the tasks that were not allocated are rechecked for possible allocation (line 18). Eventually, the best order \(O^*\) is returned.

The ordering algorithm is executed until one of the two conditions is met (lines 2–17): either the runtime of the algorithm exceeds a given maximal runtime or some target utility is reached (for instance, utilization of 95%).

**Genetic Ordering Algorithm**

The second ordering algorithm we present is based on the genetic algorithm applied to the cutting stock problem [74]. In this algorithm, an initial population \(P\) of different orders is generated. Each individual (an order) in \(P\) is evaluated by calculating its utility (by the aforementioned “calculate utility” function). The best \(L\) orders are marked as “elite” and are passed to the
next generation. A fixed number of worst individuals (orders) is discarded and replaced in the next generation by new random orders – immigrants \( I \). The \( C = |P| - |L| - |I| \) orders participate in reproducing new \( C \) orders by means of crossover operation OX3\(^3\) [42]. This evolutionary process is repeated until maximal runtime is exceeded or a target utility is reached.

**Algorithm 5 Genetic ordering algorithm**

| generate population \( P \) of random orders |
| \( \forall O \in P \) calculate utility of \( O \) |
| \( utility \leftarrow \) utility of the best \( O \) |
| anchor\_distance \( \leftarrow |Tasks| \) |
| **while** runtime < \( MAX \) **and** utility < \( TARGET \) **do** |
| generate new population \( P \) as: |
| \( L \) best orders from last round |
| \( I \) new random orders |
| \( C \) orders generated by OX3 |
| \( \forall O \in P \) calculate utility of \( O \) |
| \( utility \leftarrow \) utility of the best \( O \) |
| update anchor\_distance |
| **end while** |
| **return** best \( O \) in \( P \) |

In algorithm 5, lines 1–3 describe the creation of the initial population \( P \) of random orders and its evaluation. The genetic ordering algorithm will terminate when either maximal runtime is exceeded or a target utility is reached (lines 5–10). The new generation \( P \) in line 6 is created according to the aforementioned rules, where the crossover operation OX3 is defined as follows: Let \( P^a \) and \( P^b \) be the selected parents. We generate two random numbers \( p, q \) (\( p < q \)) in the interval \( (|Tasks| - anchor\_distance + 1, |Tasks|) \). The offspring are two individuals, \( O^a \) and \( O^b \), generated as follows. The crossover copies elements \( P^a_p, \ldots, P^a_q \) (respectively, \( P^b_p, \ldots, P^b_q \)) to the same positions in \( O^a \) (respectively, \( O^b \)). \( O^a \) and \( O^b \) are then filled up by selecting the missing elements from \( P^b \) and \( P^a \), respectively, and placing them in the same order in which these elements appear in \( P^b \) and \( P^a \), respectively [74]. A sample result of the OX3 crossover operation is depicted in Figure 3.6. The two offspring are evaluated and the best one passes to the next round. This “reproduction” process is repeated \( C \) times per round. Each individual’s probability of being selected is directly proportional to its utility.
Each order in the created population $P$ is evaluated according to its utility (lines 7–8) in the same way as in the SA algorithm. Afterwards, the anchor distance is recalculated (line 9). The anchor distance decreases according to the following calculation: The creation and evaluation time of the initial population $P$ is measured and treated as a benchmark. Given this benchmark and the maximal runtime, the anchor distance is linearly decreased so that, in the last allowed round, $\text{anchor\_distance}! \approx |P|$.

**Clustering algorithm**

Modern computational grids comprise thousands of computing resources. Therefore, the allocation matrix $X$ defined in Section 3.2.3 can be very large, resulting in poor performance of any coallocation algorithm. However, in practice, the offered resources can be efficiently clustered without effecting the accuracy of the resource description. This results in a considerable reduction in the size of the coallocation matrix. The proposed clustering algorithm is based on the fact that the users requests are described in terms of ranges. Within this range, the offered machines are “compatible” with the request, while outside the range they are not. At this stage, we will assume that the user is indifferent to the resources’ exact properties as long as they are compatible. If required, user’s preferences can be added to the utility function described in Section 3.2.4.

Figure 3.7 depicts the resource requests and offers plotted on the memory/CPU coordinate system (in practice, there will be as many dimensions as there are resource properties). The resource requests are presented as transparent rectangles, while the offered resources are marked as yellow dots. The offered resources have no ranges, as they describe the real hardware.
properties of the offered machines.

Algorithm 6 gathers together all the offered resources that can serve the same requests. In the example presented in Figure 3.7, 10 suitable computing resources out of 13 were clustered into 4 clusters (marked in red in the figure). All the machines in each cluster can serve exactly the same resource requests. Additionally, the clustering algorithm takes into account the constraints on resource interconnections. In our model, we allow clustering of resources only if they belong to the same resource communication groups (RCGs).

**Algorithm 6 Clustering Algorithm**

**Require:** $PGs$ : requested process groups of a task
                            $Offers$ : a set of offered machines

**for all** $offer \in Offers$ **do**
   $offer.hash \leftarrow$ empty string
   **for all** $pg \in PGs$ **do**
       if $offer$ is compatible with $pg$ then
           $offer.hash += pg.id$
       end if
   end for
end for

group $Offers$ according to their hash and RCGs

Figure 3.7: Clustering resource offers according to requested ranges.
Coallocation algorithm

A coallocation algorithm provides an allocation plan for a single task composed of multiple process groups satisfying equations 3.1–3.5. Here we propose a heuristic algorithm that is based on the observation that equations 3.1–3.3 form a capacitated transportation problem, which can be efficiently solved by linear programming techniques. The solution of this problem is guaranteed to be an integer that also satisfies equation 3.5. In order to find the allocation matrix that will also satisfy the equation 3.4, we apply a hill climbing technique, while ensuring in advance that a start point is chosen on a “promising” hill.

The proposed algorithm contains three steps. In the first step, we are trying to find the “most promising” hill to climb. During the first step, a weight matrix $W$ is calculated by means of a modified version of the graduated assignment graph matching algorithm [65]. The matrix $W$ contains weights between 0 and 1, which denote the “profitability” of each allocation $X_{ij}$. The calculation of $W$ takes into account equations 3.1–3.5. The first step is performed only once and the calculated weights will not change during the rest of the algorithm (steps 2 and 3).

Algorithm 7 Step 1: calculating the weights matrix $W$

\[
\beta \leftarrow \beta_0
\]

while $\beta \leq \beta_f$ do

while \#iterations $\leq I_0$ do

$Q_{ij} \leftarrow$ calculate the profitability of each cell

$W_{ij} \leftarrow \exp(\beta \cdot Q_{ij})$

while \#iterations $\leq I_1$ do

normalize $W$ across the columns

normalize $W$ across the rows

end while

end while

$\beta \leftarrow \beta \cdot \beta_r$

end while

return $W$

During algorithm 7, $W_{ij}$ expresses how well each cell (an allocation of request $i$ to offer $j$) fits the entire current allocation proposal. The “calculate profitability” step calculates for each term $W_{ij}$ the correlation of an alloc-
tion \([i, j]\) to all the other allocations \([k, l]\). Namely, the following calculation is performed:

```plaintext
for \(i = 1\) to \(n\), \(j = 1\) to \(m\) do
  if \(c_{\min_i} \leq \hat{c}_j \leq c_{\max_i}\) then \(\{v_i\text{ and }\hat{v}_j\text{ are compatible}\}\)
    \(Q_{ij} \leftarrow \sum_{k=1}^{n} \sum_{l=1}^{m} \alpha \cdot W_{kl} \cdot \min(cap_{max_k}, c\hat{p}_j)\)
  else \(\{v_i\text{ and }\hat{v}_j\text{ are incompatible}\}\)
    \(Q_{ij} \leftarrow 0\)
  end if
end for
```

\(W\) is normalized across rows and columns in order to keep the values of \(W_{ij}\) between 0 and 1, while making sure that the “better” terms \([i, j]\) will get values closer to 1. This is done by the following calculation:

```plaintext
for \(j = 1\) to \(m\) do \(\{\text{normalizing across columns (offers)}\}\)
  \(\forall i, 1 \leq i \leq n : W_{ij} \leftarrow \min(1, \frac{W_{ij}}{\sum_{k=1}^{\text{MAX}_j} \text{smallest}_k})\)
end for
for \(i = 1\) to \(n\) do \(\{\text{normalizing across rows (requests)}\}\)
  \(\forall j, 1 \leq j \leq m : W_{ij} \leftarrow \min(1, \frac{W_{ij}}{\sum_{k=1}^{\text{MAX}_i} \text{smallest}_k})\)
end for
```

where \(\text{smallest}_k\) stands for the sum of the smallest terms (weights) in column \(j\) and row \(i\). The number of such terms, \(\text{MAX}_j\) and \(\text{MAX}_i\), is dictated by the capacity of offer \(j\) and the maximal capacity of request \(i\). If \(W\) is normalized on column \(j\), \(\sum_{k=1}^{\text{MAX}_j} \text{cap}_{\text{smallest}_k} \leq c\hat{p}_j\), and if it is normalized on row \(i\), \(\sum_{k=1}^{\text{MAX}_i} \text{cap}_{\text{smallest}_k} \leq cap_{max_i}\).

In the second step (algorithm 8) of the coallocation algorithm, we find a solution that satisfies equations 3.1–3.3 while taking into the account the weight matrix \(W\). To this end, a new matrix \(M\) is added and initialized as follows: \(M_{ij} \leftarrow \min(cap_{max_i}, c\hat{p}_j)\). The purpose of matrix \(M\) is to bound the values of \(X_{ij}\). In the first iteration of algorithm 8, adding matrix \(M\) has no effect; however, the values of matrix \(M\) will be later updated by algorithm 9. In algorithm 8, the following capacitated transportation problem is created and solved (as mentioned above, the solution of this problem satisfies equation 3.1–3.3 and 3.5):

In the third step, we have to make sure that no connectivity constraints were violated by step 2. To do so, we analyze all the allocation pairs \((X_{ij}, \)
Algorithm 8  Step 2: addressing equations 3.1 – 3.3 and 3.5.

solve the following linear problem:

maximize \( \sum_{i=1}^{n} \sum_{j=1}^{m} W_{ij} X_{ij} \) subject to:

\( \forall i : 1 \leq i \leq n, \quad \text{cap}_{\text{min}} \leq \sum_{j=1}^{m} X_{ij} \leq \text{cap}_{\text{max}}; \)

\( \forall j : 1 \leq j \leq m, \quad \sum_{i=1}^{n} X_{ij} \leq \text{cap}_{j}; \)

\( \forall i, j : 1 \leq i \leq n, 1 \leq j \leq m, \quad \text{if} \ (\text{NOT} \ c_{\text{min}} \leq \hat{c}_{j} \leq c_{\text{max}}) \ then \ (X_{ij} = 0); \)

\( \forall i : 1 \leq i \leq n, \quad X_{ij} \leq M_{ij}. \)

\( X_{kl} \) and count the number of connectivity-violating allocation pairs in which each allocation \( X_{ij} \) appears. If no connectivity violations were detected, the algorithm terminates, returning the matrix \( X \). Otherwise, the “worst” allocation \( X_{ij} \) (the one that appears in the most violating pairs) is removed from the allocation matrix, from then on forcing \( X_{ij} = 0 \) (which is done by setting \( M_{ij} \) to 0), and step 2 is repeated.

The first line of algorithm 9 initializes a list of coordinates \([i,j]\) of non-zero terms of the allocation matrix \( X \). In line 4, the algorithm goes through all the pairs in this list, and checks whether the requested connectivity between two process groups was violated by the allocated offers. If the connectivity was violated (lines 5–12), the matrix \( X \) is marked as “illegal,” and the number of violations is calculated per term \( X_{ij} \). Line 7 (and 10) checks whether an allocation \( X_{ij} \) (and \( X_{kl} \)) can be zeroed. This is done by checking all the other “potential” allocations \( M_{ij} \) (and \( M_{kl} \)) of the process group \( i \) (and \( k \)), and if enough potential allocations are found, it is assumed that the allocation \( X_{ij} \) (and \( X_{kl} \)) is dispensable. In that case, the appropriate problem_cost is increased (lines 8 and 11). In line 15 the algorithm checks whether any violation was found; if not, the legal matrix \( X \) is returned. Otherwise, in line 18, the algorithm checks whether any term \( X_{ij} \) was found to be dispensable. If not, matrix \( X \) is still illegal, but no more improving steps are possible. Therefore, the execution of the entire algorithm is stopped and an “infeasible solution” exception is thrown. If matrix \( X \) is illegal but an additional improvement step is possible (lines 21 – 23), the “worst” term \( X_{ij} \) is zeroed, and step 2 is executed again.
Algorithm 9 Step 3: validation of equation 3.4.

Require: $X$: allocations matrix returned by Algorithm 8

$curr_{alloc} \leftarrow \{(i,j)| i \in \{1..n\} \land j \in \{1..m\} \land X_{ij} > 0\}$

is\_legal\_X \leftarrow true

problem\_cost \leftarrow ZERO\_MATRIX

for all $([i,j], [k,l]) \mid [i,j], [k,l] \in curr\_alloc$ do

if NOT $b_{min_{ik}} \leq b_{jl} \leq b_{max_{ik}}$ then

is\_legal\_X \leftarrow false

if $\sum_{a=1}^{m} (M_{ia}) - X_{ij} > cap_{min_{i}}$ then

problem\_cost_{ij}++;

end if

if $\sum_{a=1}^{m} M_{ka} - X_{kl} > cap_{min_{k}}$ then

problem\_cost_{kl}++;

end if

end if

end for

if is\_legal\_X = true then

return $X$

end if

if problem\_cost is ZERO\_MATRIX then

return “No feasible solution was found”

end if

$[i,j] \leftarrow$ index of the biggest number in problem\_cost

$M_{ij} \leftarrow 0 \{ \text{dropping the “worst” } X_{ij}\}$

goto step 2

3.2.5 Evaluation

In this section we present the evaluation of the proposed ordering and allocation algorithms. All the inputs described in this section were generated synthetically, as we are not aware of any supercomputer logs that contain information about topology-aware applications and their requirements. The performance of the ordering algorithms was compared with four different well-known heuristics:

- LTF – Longest Time First, where the tasks are sorted according to their runtime in descending order;
- BGF – Biggest Gang First, where tasks are sorted according to the number of required CPUs in descending order;
• Txs – Time x Size, where tasks are sorted in descending order according to the number of requested CPUs multiplied by the task’s runtime;

• FCFS – First-Come, First-Served, where tasks are sorted according to the time of their submission in ascending order.

Figure 3.8 depicts the comparison of the genetic ordering algorithm (GA), the simulated annealing ordering algorithm (SA), and the aforementioned heuristics\(^9\). In this experiment, the utility function was dependent on several parameters. The first of these was utilization, the percentage of utilized CPU seconds out of all those available. The second parameter was satisfaction, the average task satisfaction in terms of received CPUs. Task satisfaction is defined as:

\[
task\_satisfaction = \begin{cases} 
0 & \text{if task is not scheduled} \\
\frac{1}{2} + \frac{\text{received} - \text{min} + 1}{2(\text{max} - \text{min} + 1)} & \text{otherwise}
\end{cases}
\]

where received, min and max denote, respectively, the number of received CPUs, the minimum requested CPUs, and the maximum requested CPUs.

The third parameter was fairness, the average fairness of task scheduling compared to its submission time. Task fairness is defined as:

\[
task\_fairness = \begin{cases} 
0 & \text{if task is not scheduled} \\
1 & \text{if only task is scheduled} \\
1 - \frac{\text{bypassed}}{\text{total} - 1} & \text{otherwise}
\end{cases}
\]

where bypassed denotes the number of tasks scheduled to run before the given task although they were submitted after it, and total denotes the total number of scheduled tasks. The final parameter is makespan. In general, makespan defines the endtime of the last scheduled task and is subject to minimization. However, in order to allow utility maximization, the makespan was defined as: \(\text{makespan} = 1 - \frac{\text{last\_task\_endtime} - \text{start}}{\text{end} - \text{start}}\), where last_task_endtime denotes the end time of the last scheduled task, while start and end denote the start and end time of the scheduling round.

The utility function was defined as maximize\((0.2 \times \text{utility} + 0.2 \times \text{satisfaction} + 0.2 \times \text{fairness} + 0.4 \times \text{makespan})\). The first three components of the utility function encourage high utilization and are supposed to balance the makespan, which might give precedence to partial scheduling solutions (scheduling a small number of tasks). However, minimizing the makespan is important as we would like to get a schedule which is as dense as possible.

\(^9\)Complementary heuristics such as Shortest Time First and Smallest Gang First showed very poor performance.
The experiment depicted in Figure 3.8 was performed with the following settings: 10 tasks were generated, each composed of a random number of process groups (PGs) chosen in range of 1...20. Each PG required a random number of CPUs in range of 80...120. The tasks’ runtime was randomly generated from the range of 20 minutes to 5 hours. The offered resource topology graph comprised 100 clustered offers, where each cluster contained a random number of CPUs in the range of 80...120. On average, the number of requested and offered CPUs in this experiment was about 10,000 in each RTG. The offered resources were available for 10 hours with a single unavailability slot of 10 minutes, randomly distributed on the time axis. The probability of a single requested machine to be compliant with a single offered machine was 40%. The experiment depicts average values measured in 120 non-trivial experiments (i.e., experiments where the inputs were randomized in a way that at least two tested heuristics returned different solutions).

In the upper chart in Figure 3.8, the results of the FCFS heuristic were used as the baseline, so that its components sum up to exactly 100%. The lower chart of Figure 3.8 denotes the average number of tasks scheduled by each algorithm. As shown in the graph, the two proposed ordering algorithms outperform the classic heuristics by about 30%. It is worth mentioning that the best order was found by the genetic algorithm within 40 moves on average (tentative scheduling of 40 different orders), while the simulated annealing algorithm found the best order within 114 moves on average.

Figure 3.8: Comparison of ordering algorithms vs. simple heuristics.

Figure 3.9 depicts the performance of the GA and SA algorithms com-
pared to the results achieved by an exhaustive search algorithm. Given the two aforementioned heuristics – running the task at the first possible time slot, and maximizing the number of resources allocated per task – the exhaustive search provides the optimal result that can be achieved by an ordering algorithm. As shown in figure 3.9, both ordering algorithms achieved results that are very close to the optimum.

![Figure 3.9: Comparison of ordering algorithms vs. exhaustive search.](image)

Figure 3.9: Comparison of ordering algorithms vs. exhaustive search.

Figure 3.10 depicts an experiment with the coallocation algorithm presented in Section 3.2.4. In this experiment, the inputs of the coallocation algorithm were generated as follows: the requested task contained 10 process groups (PGs), each with a number of CPUs in a range of 50...150, (in total between 500 and 1500 CPUs). The offered resource topology graph contained 15 clusters, each with a number of CPUs randomized in range of 80...120 (about 1500 CPUs). The compatibility between a single offered and requested resource was 70%. The compatibility between a single link in the requested and offered graphs is presented on the x-axis.

The optimal coallocation algorithm is defined by equations 3.1–3.5, where quadratic equation 3.4 was replaced by an equivalent linear equation. To this end, \( n \cdot m \) binary variables \( Y_{ij} \) were added, where \( Y_{ij} = 0/1 \) indicates whether \( X_{ij} \) should be zeroed or not. Equation 3.4 was later transformed to:

\[
(Y_{ij} + Y_{kl} - 1)b_{\min ik} \leq b_{jl} \land (Y_{ij} + Y_{kl} - 1)b_{jl} \leq b_{\max ik} \\
\land X_{ij} \leq Y_{ij} \min (cap_{\max i}, cap_j) \\
\downarrow \\
Y_{ij} + Y_{kl} \leq 1 + \min \left( \frac{b_{jl}}{b_{\min ik}}, \frac{b_{\max ik}}{b_{jl}} \right) \\
\land X_{ij} \leq Y_{ij} \min (cap_{\max i}, cap_j).
\]

106
This transformation was possible because all the values in the proposed model are positive. The created integer programming model was fed into the GLPK [97] solver, which returned an optimal solution for the co-allocation problem.

As the compatibility between the links increases, the performance of the proposed co-allocation algorithm increases. This behavior can be explained by the number of possible solutions, which rises as link compatibility increases. When number of possible solutions is large enough, the hill-climbing algorithm (algorithm 7) chooses a “good” hill with high probability and a fair solution is found.

Figure 3.10: Performance measurements of the co-allocation algorithm vs. optimal solution.

Figure 3.11: Runtime of the heuristic co-allocation algorithm vs. optimal algorithm.

Figure 3.11 depicts the runtime of the proposed co-allocation algorithm compared with the runtime of the GLPK solver in four different experiments. The experiments are denoted on the x-axis as \(|\text{requestedPGs}| \times |\text{offered_clusters}|\). Each experiment depicts an average runtime of 100 independent executions. The number of machines in each PG was random-
ized in range of 50...150. The number of machines in each offered cluster was randomized in the range of 80...120 (thus, the 6x9 experiment denotes matching of about 600 requested processes to 900 available machines). The probability of a single requested resource to be compliant with a single offered resource was 70%, and the probability of network compatibility was 36%. Note that bigger graphs could not be evaluated in this experiment due to the limitations of the optimal algorithm.

**Evolutionary algorithm evaluation**

An experiment with the real-life topology-aware application was performed on the French Grid5000 [24] infrastructure, using clusters located in Orsay and Bordeaux. The machines in Orsay cluster have the CPU clock rate of 2.0 GHz, while the machines in the Bordeaux cluster have 2.6 GHz clock rate. The evolutionary algorithm shown in Figure 3.2 was executed twice with the subpopulation divided between the Orsay and Bordeaux clusters. These results are shown in Figure 3.12. The red or upper line gives the average execution time when the two subpopulations are both distributed evenly over the two clusters: this is a suboptimal allocation because it requires frequent communication to be performed between the clusters (which are physically sited 100 kms away from each other). The green or lower line gives the average execution time when each subpopulation is allocated to just one cluster: this is a more optimal allocation because frequent communications between the individual processes in the subpopulation are localized to just a single physical site.

From this simple experiment, we can deduce two important conclusions. First, the application runtime is affected by the proper matching. When the same application was improperly placed, the execution time was increased. Second, we see that when the application is placed correctly, it scales gracefully. Namely, its execution time is dictated only by the slowest machines selected for the execution and some additional constant overhead derived from the inter-cluster communication.

### 3.2.6 Summary

We have presented a novel framework for executing tightly-coupled topology-aware applications in large scale environments. The proposed framework is
Figure 3.12: Runtime of the evolutionary algorithm application with optimal and suboptimal allocation on the compute resources.

We showed that the scheduling module is very generic and capable of handling a large variety of utility functions. The presented clustering module is designed to minimize the size of a co-allocation problem, which is very significant in large-scale environments. One of the key features of the clustering module is that it makes no approximations and therefore can be used in systems that provide a certain guarantee about the provided Quality-of-Service. Additionally, we showed a heuristic co-allocation algorithm that handles an NP-complete problem of matching resource requests and offers with very reasonable results.

We believe that the combination of all the aforementioned components into a single scheduling system provides a full solution for large-scale resource management systems that must handle topology-aware applications. The proposed scheduling framework was integrated into the QosCosGrid system, where it is used as the main decision-making module.
Chapter 4

Resource Brokering in Cloud Computing

4.1 Black-box Network Analysis for Remediating Performance Bottlenecks

4.1.1 Model

We first describe our assumptions about the monitored applications in Section 4.1.1. In Section 4.1.1 we outline the NAP monitoring approach. Section 4.1.1 depicts performance remediation policies based on our model.

Application Assumptions

In this work we report results for synchronous multi-tier client/server applications, where an application is provisioned as a collection of VMs. As a typical example, consider a three-tier commercial application consisting of a Web front-end tier, an application server tier, and a database server tier. A minimal configuration would include three VMs running a front-end, an application server and a database server respectively. With elastic computing, the application can be resized on demand to meet required performance levels cost-effectively. Specifically, VMs can be resized by increasing or decreasing the CPU power and/or the number of CPUs and other resources, such as e.g., memory and bandwidth, and tiers can be re-dimensioned by adding or removing VM instances. NAP aims at facilitating these perfor-
mance management decisions without deploying in-band application-level monitoring solutions.

Making application-level performance decisions without application knowledge is hard. Fortunately, we can exploit a typical virtualization setting for our needs. While different applications may utilize the same VMs, e.g., a VM running a DNS server, to make a more efficient use of the infrastructure, in typical virtualization scenario, there is one-to-one mapping of the application components to VMs. The TCP/IP traffic is directly observable for VMs by monitoring their virtual network interfaces at the hypervisor level. Thanks to the one-to-one mapping of the application components to VMs, the TCP/IP traffic observed for a VM can be reliably attributed to the activities of the application component hosted by the VM\(^1\).

NAP handles both multi-threading and multi-tasking models of request processing at the server side. In what follows, we assume the multi-threaded model, which, arguably, is more a more complex one. In this model a thread from the working pool is allocated to an incoming request and each request is served until completion by the thread initially assigned to it.

Monitoring Approach

NAP aims at being application-agnostic. It performs traffic monitoring at the TCP/IP level. Since the monitored application is a multi-tier synchronous client/server service, the TCP/IP traffic coming from the upper tiers to the back-end tiers, which we call inbound traffic, can be attributed to application requests. The TCP/IP traffic returning in the opposite direction is termed outbound traffic and is associated with the application responses.

Thanks to synchronous communication, we can correctly demarcate the TCP/IP segments that correspond to requests and responses on the TCP/IP connections. As we show, this information is sufficient for identifying performance hot spots.

NAP uses a simple technique matching the last TCP/IP segment of a request sent on the direct session of the TCP/IP connection with the first

\(^1\)If more than one application is deployed on a single VM, which is not a typical virtualization scenario, NAP still will be able to identify performance hot spots and perform remediation. However, in this scenario, NAP is likely to over-provision capacity to some degree, as it is not clear how to scale up an application component independently from other application components deployed on this VM and, possibly, having different capacity demands and performance targets.
TCP/IP segment of the response sent on the return session of the same connection by using TCP segment sequence and acknowledgement numbers. This request/response pair matching is explained in more detail in Section 4.1.2. At this point it is important to stress that we want to capture the total processing time of requests at the server components, excluding network delays. To achieve this, monitoring of requests and responses is always performed at the server side. Furthermore, in case of a multi-segment request we capture the time at which we observe the last segment of the request as request arrival time for this request. For responses, on the contrary, we record the time at which we observe the first TCP segment of the response as the response arrival time\(^2\).

It is important to notice that pipelining of requests and responses as used, e.g., in HTTP/1.1 does not break the NAP model. HTTP/1.1 allows multiple HTTP requests to be written out to a socket together without waiting for the corresponding responses. The requestor then waits for the responses to arrive in the order in which they were requested.

The requests observed at the network level using the above technique, are not equivalent to the user-level transactions, for which performance service level objectives (SLO) might be defined, as these transactions may comprise multiple requests. Machine learning techniques similar to [78] can be used to recognize user-level transactions from the requests. We do not take this approach in NAP, however, observing that detecting statistically significant changes in the measures of central tendency of the requests themselves, might be sufficient to trigger successful resource reallocation decisions.

Let \(\Delta\) denote a single sampling interval duration. Let \(N_i\) denote the number of pending requests. At any time instance during \(\Delta\) when a new request or response is detected, \(N_i\) is updated, where \(i\) stands for “i-th request or response detection”. Thus, \(N_i\) is a random variable assuming integer values between 0 and \(\infty\).

Let \(\{N_i\}_{\Delta}\) be a sample of \(N_i\) of size \(n\) during the monitoring interval \(\Delta\). Then, using configuration information readily available in a typical deployment scenario, as explained in Section 4.1.2, we define the mean queue length \(L\) of an application component running in the monitored VM as:

\[^2\text{Note that NAP knows which TCP segment of the request is “last” only a posteriori, i.e., when it matches response TCP segments on the reverse TCP session.}\]
If $\Delta$ is chosen to be sufficiently long, most of requests are matched to the responses, i.e., the system is balanced. The unmatched requests can be discarded after some additional timeout as explained in Section 4.1.2 in more detail.

From Little’s law [92] applied to the queuing part of the system we derive an approximation of the mean queue time $W$ as follows:

$$W = \frac{L}{\lambda}$$

where $L$ is approximated by Equation 4.1 and $\lambda$ is computed directly from observing the mean inter-arrival time of requests during $\Delta$.

The mean total response time, $T$, is calculated using the following equation:

$$T = \frac{\sum_{t_{req} \in \Delta} t_{resp} - t_{req}}{\text{reqs}(\Delta)}$$

where $t_{req}$ denotes a request’s arrival time and $t_{resp}$ denotes the corresponding response’s dispatching time. $T$ is calculated for each $\Delta$.

The mean service time $S$ is approximated using Equation 4.2 and Equation 4.3 as follows:

$$S = T - W$$

NAP collects a number of data points for the mean queue time $W$, mean total response time $T$, and mean service time $S$, and constructs statistical distributions of these variables. The analysis module of NAP compares these distributions to the baselines for the application. If considerable deviation is detected, an alert is sent to the remediation module that takes performance management decisions such as capacity reapportionment.

The monitoring process described above is performed at every tier. This allows isolation of the problematic tier, i.e., the one that is either over-utilized, causing performance hot spots, or under-utilized, causing cost-ineffectiveness.
Remediation Policies

A few deviations from the baseline distributions of the mean $S$ and $W$ are particularly important. We use them to guide NAP’s remediation policy selection:

- **The distribution of both $W$ and $S$ are stochastically greater than their respective baselines.** This is the case of a typical performance hot spot resulting from insufficient capacity. In general, this case is indicative either of an increase in the average complexity of the requests in the current workload or of a hardware insufficiency that may slow down processing. In any case, the most efficient remediation action is the resizing of the corresponding VM by allocating more resources to it.

- **The distribution of both $W$ and $S$ are stochastically smaller than their respective baselines.** This case is indicative of underutilization. The corresponding VMs may be resized by taking away excess capacity.

- **The distribution of $W$ is stochastically greater than its baseline, while the distribution of $S$ is stochastically equal to its baseline.** This may seem counter-intuitive. Yet, as we show, it arises in practical scenarios. This phenomenon is observed when the workload volume, e.g., the number of clients issuing requests increases drastically, while the capacity demand of the requests does not increase on the average. While allocating more resources per machine, e.g., increasing CPU speed, may be helpful in some cases, the CPU might not be the primary bottleneck. An appropriate remediation in this case would be adding more VM instances (or, in some cases increasing a number of CPUs per VM) to shorten the queuing time.

- **The distribution of $W$ is stochastically smaller than its baseline, while the distribution of $S$ is stochastically equal to its baseline.** This is indicative of under-utilization. To improve cost-effectiveness the corresponding tier may be scaled-down by removing excess VM instances.
Given the model outlined above, the key questions for experimental analysis are: (a) the choice of statistical test and (b) the threshold for the disparity that triggers the alerts. The NAP analysis module can be implemented using any statistical test. In fact, the NAP’s architecture is modular and allows plugging different tests. In Section 4.1.4 we show an analysis based on a simple percentile test that can be easily understood and adopted by system managers that may wish to follow the decision logic of the autonomic NAP policies.

### 4.1.2 Architecture and Implementation

The NAP system can be roughly divided into three main components: monitoring, analysis, and remediation. Figure 4.1 depicts the high-level architecture of NAP in a hypervisor environment. In this setting, the host runs two virtual machines, one of which is a service virtual machine running the NAP system and the other one is the virtual machine running an application. In order to service client requests, the application VM communicates with the outside world over a virtual network interface. All the network traffic of the server passes through the service virtual machine, where the NAP monitoring module collects the needed information.

![Figure 4.1: NAP – high level architecture](image)

Figure 4.1: NAP – high level architecture
We note that having a special virtual machine handle all network traffic for other virtual machines is very common and is the default mode of operation for the Xen hypervisor [13], regardless of NAP. The overhead added by NAP is confined to the monitoring module’s overhead.

Monitor

The application-agnostic feature of the monitoring system stems from the fact that the system observes the TCP/IP headers only and makes no assumptions about higher-layer protocols beyond the very general assumptions of Section 4.1.1. The monitoring module treats the packet incoming to the monitored server and containing some data (as opposed to headers only) as requests, and packets outgoing from the server and containing some data as responses. Packets belonging to the same request/response pair have the same sequence and acknowledgment numbers respectively. Therefore, no request or response is counted more than once. Additionally, the TCP packets of a request contain the acknowledgement number of the corresponding response, thus allowing the exact calculation of a request’s response time.

The monitoring module is a light-weight packet sniffer, which is responsible for collecting traffic information such as the headers of data packets sent and received by clients during a monitoring interval $\Delta$. The actual packets capturing is performed via the JPCAP library\(^3\). The packet headers are processed online and the main statistics such as $L$, $\lambda$, $T$, $W$, and $S$ are calculated for each $\Delta$ as described in Section 4.1.1.

The monitoring component is configurable, allowing the administrator to set a timeout after which an unanswered request would be either discarded or counted as request with a maximal response time $T_{max}$.

Other helpful (optional) parameters include:

**Filter String** : specifies which ports should be monitored by NAP for this VM, e.g., ”port 80”;

**Worker Thread Pool Size** : number of worker threads configured at the server side. Currently, we obtain this information from the configuration files, e.g., from inspecting Apache MPM common directives.

---

\(^3\)http://netresearch.ics.uci.edu/kfujii/jpcap/doc/index.html
If these parameters are not set, as may be a common case in the black box model, NAP infers them from the out-of-band traffic monitoring. In particular, if the port that should be monitored at the virtual interface of a server VM is not specified, NAP monitors this interface in a promiscuous mode capturing all IP datagrams, extracting the port numbers from the encapsulated TCP segments and listening to all ports with activity.

If the maximal number of working threads is unknown, NAP performs a special test to estimate it. Namely, NAP chooses a request from the observed traffic, and sends it to the VM under the test at a constant rate, increasing this rate gradually. For these requests, NAP measures total average response time. When the number of worker threads becomes smaller than the number of requests that arrive simultaneously, some requests get queued. This is manifested by an increase in the average wait time and, consequently total average response time. NAP captures this change point to estimate the number of workers in the pool.

The monitoring module of NAP works as follows. The monitoring module stores the key attributes of request and response TCP/IP segments headers such as arrival timestamp, sequence and acknowledgment numbers using internal data structure. Each response is matched with its request, or discarded after the maximal response time timeout, which is set to be a number of Δ-s. Consequently, Δ-s with the end time earlier then the current time – maximal response time are marked as “aged”. For the aged Δ, the monitoring module computes the values of $S$ and $W$ using Equations 4.1–4.4 and passes them to the analysis module. Eventually, the aged data is discarded.

In the current version of NAP Δ is manually set and it is not adaptive. However, a simple adaptive heuristic is possible to automate this process. Namely, let $\epsilon$ be a maximal error in the estimation of averages that we want to admit at the $\alpha$ confidence level. Then, window should is adaptively set in such a way that the number of requests (a) approximately equals the number of requests and (b) the number of request-response pairs in the window $n > (s \cdot \frac{Z_{\alpha/2}}{\epsilon})^2$, where $Z_{\alpha/2}$ is the critical value of unit Normal variate at confidence level $\alpha$ and $s$ is the sample standard deviation computed over the total response time measured for each request/response pair. Inability to determine $\Delta$ that satisfies these conditions may in itself be an indicator of performance instability as the job flow in the system is highly unbalanced.
on every time scale. Another reason for inability to set good $\Delta$ may be large sample standard deviation, which is indicative of significant differences between requests in the system. If this is the case, univariate datapoints clustering will be performed and then $W$ and $S$ will be computed per each cluster, as explained in Section 4.1.4.

**Analysis and Remediation**

The analysis module stores the values of $S$, and $W$ for a sliding time window comprising a number of $\Delta$-s. Using the data in the sliding window, the distributions of $S$ and $W$ are calculated and compared to the corresponding baseline distributions. The comparison of the distributions is implemented as a pluggable component, which is easy to change and reconfigure. In the current version of NAP, the comparison algorithm is a simple percentile analysis, which is depicted in Algorithm 10. Based on the comparison results, the decision is made whether to raise a performance hot spot (or under-utilization) alarm. If an alarm is raised, the remediation module is engaged.

The third, remediation component, decides which (if any) action to take in order to remediate the problem detected by the analysis component. Remediation is usually done by either scaling-up the tier (i.e., adding more VM instances at that tier) or scaling-up (i.e., adding more resources to each instance) the service layer where the problem is detected. One of the advantages of the NAP system is that it can clearly indicate in which tier of the $n$-tier application the problem occurred and suggest the proper remediation action.

**4.1.3 Implementation Details**

The monitoring module is responsible for capturing and processing of the TCP/IP packets heading to and from the monitored server. Assuming that a server and a client synchronously exchange messages by means of a conversation pattern, it is possible to understand when a new request or reply is started. Additionally, based on the TCP/IP packets’ sequence and acknowledgment numbers, a reply can be matched to a proper request allowing the exact calculation of a request’s total response time. In order to discard the TCP/IP auxiliary packets (such as empty acknowledgments for received
Algorithm 10 Decision making algorithm

\[
\text{percBaseS} \leftarrow Xth \text{ percentile of baseS} \\
\text{percBaseW} \leftarrow Xth \text{ percentile of baseW} \\
\text{percWinS} \leftarrow Xth \text{ percentile of winS} \\
\text{percWinW} \leftarrow Xth \text{ percentile of winW} \\
\text{if} \quad \text{percBaseW} \cdot (1+\text{threshold}) < \text{percWinW} \quad \text{and} \quad \text{percBaseS} \cdot (1+\text{threshold}) < \text{percWinS} \quad \text{then} \\
\quad \text{alert(“over-utilization – more resources per instance are needed”)} \\
\quad \text{return} \\
\text{end if} \\
\text{if} \quad \text{percBaseW} \cdot (1+\text{threshold}) < \text{percWinW} \quad \text{then} \\
\quad \text{alert(“over-utilization – more instances needed”)} \\
\quad \text{return} \\
\text{end if} \\
\text{if} \quad \text{percBaseW} \cdot (1-\text{threshold}) > \text{percWinW} \quad \text{and} \quad \text{percBaseS} \cdot (1-\text{threshold}) > \text{percWinS} \quad \text{then} \\
\quad \text{alert(“under-utilization – less resources per instance are needed”)} \\
\quad \text{return} \\
\text{end if} \\
\text{if} \quad \text{percBaseW} \cdot (1-\text{threshold}) > \text{percWinW} \quad \text{then} \\
\quad \text{alert(“under-utilization – less instances needed”)} \\
\quad \text{return} \\
\text{end if}
\]

packets) we discard the packets which contain no data. Algorithm 11 depicts the processing of a TCP/IP packet that satisfies the supplied “filter string”:

4.1.4 Experimental Evaluation

The primary purpose of our evaluation study is to usefulness of simple out-of-band black box traffic monitoring for detecting performance hot-spots reliably. In this initial study we leave some important questions out of the scope. First, our testbed is static and we do not evaluate NAP in a virtualized infrastructure environment that uses migrations. Second, we exclude network perturbations from our evaluation experiments. Third, we did not include hardware failures or software bugs into our evaluation scenarios. These issues are important and they are deferred to the future work.
NAP was evaluated using two setups: (1) a synthetic client-server application tailored specifically to sanity-check the model and gain intuition and (2) a real world application simulation using Trade6. The synthetic application provided full control over the client workloads and the server-side application behavior. This simple application allows reproducing problematic server-side states and inspecting the monitoring system behavior before and after the problem remediation. After gaining intuition and acquiring insight with this simple system, we demonstrate that our approach works well for Trade6, which is considerably more complex.

Our toy synthetic application consists of three VMs deployed on three different physical hosts: (1) client workload generator VM, (2) application server VM, and (3) database VM. There were two types of requests in our application: SELECT and UPDATE. With probability 50% clients invoke either SELECT or UPDATE. When received by the application server, both SELECT and UPDATE trigger a series of intensive I/O mixed with intensive calculations, which can heavily tax CPU. Processing SELECT requests takes approximately twice as long as processing the UPDATE requests. After finishing the calculations, the application server submitted an appropriate SELECT or UPDATE query to the second VM, which ran the PostgreSQL database server. On the DB server, the SELECT query again took approximately twice as long as the UPDATE command.

The baseline workload was set for 30 users. Several executions (totalling 120,000 client queries) were performed with this workload. We measured the baseline mean service time \( S \), the mean queue time \( W \), and the mean total response time \( T \) on the application server VM and on the DB VM. Fig. 4.2 depicts the cumulative distribution function (CDF) for \( S \), \( W \), and \( T \) as measured on the application server during the baseline executions.

In order to simulate a problematic state, we increased the number of users. Two additional experiments were conducted, in which the number of users was raised first to 40 and then to 50. Fig. 4.3 depicts the CDF of the times as measured on the application server VM during these two experiments.

Increasing the workload to 40 and 50 users resulted in growth of the 95th percentile of the mean total queue time \( W \) by 55% and 111% respectively, compared to the baseline. It is important to notice that the service time \( S \) during the 3 aforementioned executions stayed approximately constant.
(this counter-intuitive situation was discussed in Section 4.1.1). Since the server’s mean service time did not change (it took the same time to complete a query as it took in the baseline), but the total queue time exceeded some predefined threshold, the problem should be remediated by scaling-up the application server tier, i.e., by adding more application servers. Doubling the number of application servers and splitting the users equally between them, brings the measured $T$, $S$, and $W$ very close to the baseline shown in Fig. 4.2. Scaling-up VMs resulted in less successful remediation since CPU was not the primary bottleneck in this case.

Another way to see what happens in the system during this experiment that varies the number of users, while keeps the computational complexity of requests the workload constant, is to plot the $S$, $T$, and $W$ times vs. the mean queue length $L$ (see Eq. 4.1), as shown in Fig. 4.4. As expected, $T$ and $W$ in $N$ exhibit a linear dependency on $L$, while the mean service time, $S$, stays constant.

Another set of experiments was conducted to show NAP’s ability to detect the problematic states in a system where the client workload remains constant but some internal server problem occurs. For example, such a problematic state can be caused by a database which has grown too large, unexpected CPU load outside the system’s control, or insufficient disk bandwidth. In order to simulate such a problematic state, a new process running an endless loop of calculations was launched on the application server VM. This resulted in heavy CPU load in the virtual machine bringing it up to 100% usage and, thus, severely degrading performance in the application server. The workload used in this run was the same baseline workload pro-

Figure 4.2: Application server times CDF – baseline (30 users)
duced by 30 client threads as shown in Fig. 4.2. Fig. 4.5 depicts the measured times during the period of CPU overload. It is notable that the 95th percentile of all three measurements ($S$, $T$, and $W$) increased by approximately 500%. This, naturally, was more than enough to trigger an alert.

In this case, both the mean queue time $W$ and the mean service time $S$ were much higher than the baseline. This can indicate either the increased difficulty of the workload (each query takes more time), or some internal misbehavior of the server (which, as we know, was the case due to con-

---

**Figure 4.3:** Application server – increased workload. Topmost chart – 40 users, bottom chart – 50 users

**Figure 4.4:** Application server – times vs. queue length
trolled problem injection that we performed). In any case, the VMs in the problematic tier should be scaled up (each instance given more resources) in order to remediate the problem. In our setup, this was done by adding an additional virtual CPU. Fig. 4.6 depicts the measurements of the monitoring system after the problem remediation.

The results of this run suggest that the system’s responsiveness has almost returned to normal. The measured times still show a degradation of 10-20% when compared to the baseline, but contrasted with the previous 500% degradation, this is a successful remediation.

We note that performance degradation tolerance level is a configurable system parameter. In the current version of NAP, this parameter is controlled manually. If Service Level Objective (SLO) compliance for the application is being monitored and the compliance information is available to NAP, dynamic and adaptive threshold setting techniques can be used to au-
tomate threshold setting with controllable levels of false positive and false negative alerts [17, 16].

Contrary to the application server, the DB server showed no serious fluctuations in the system responsiveness, as this VM was not affected during the experiment. On the DB VM, the queue time was zero during all runs, therefore $S$ was always equal to $T$.  

To show the applicability of the NAP monitoring system to real-life complex applications, we tested NAP with the 3-tier IBM Trade6 application that simulates stock exchange trade environment. Trade6 consists of WebSphere Application Server, the DB2 database, and the Rational Performance Tester 6.1 application, which generates the client load. We used the default server and client configurations.

This experiment was set up as follows: one VM running the WebSphere application server, a second VM running the DB2 database server, and two physical machines acting as clients. The application server (WebSphere) VM was initially configured to run with a single virtual CPU. Fig. 4.7 shows the CDF for the baseline time measurements, made during 15 minutes with a workload generated by 350 users.

![Figure 4.7: Trade6 application server – baseline CDF](image)

In the next experiment, the number of users was increased to 600.

---

4It is reasonable to expect that under extreme load conditions the statistics NAP calculates for the DB tier may also be affected. For example, if the application server slows down to the point that transmission window would become zero often, NAP will assume that service time has increased. We were not able to create such adverse conditions, however.

Fig. 4.8 depicts the values of $T$, $W$, and $S$ during the intensified workload as measured by NAP on the application server VM. It is notable that the $95th$ percentile of $W$ increased by 154% while the $95th$ percentile of $S$ stayed unchanged, suggesting that more application server instances are needed in the relevant tier. Expanding the application service tier by means of a new application server and redirecting half of the users to a new server resolves the problem and brings the system to a state that is nearly identical to the baseline.

Another problematic state was simulated by means of an additional process, running an infinite loop on the Trade6 application server VM, which resulted in heavy CPU load. The CPU overload process is a multi-threaded process where by controlling the number of computational threads, we controlled the overload level. Gradually increasing the number of threads resulted in proportional CPU load increase. Fig. 4.9 depicts the measurements for the maximal over-load level. In this state the mean queue time and mean service time are higher by approximately 121% and 48% compared to the baseline. This suggests that the remediation policy that allocates more resources to the application server is most likely to succeed.

The remediation policy allocated virtual CPUs one by one, resulting in 4 total virtual CPUs allocated. Fig. 4.10 depicts the measured times after the problem remediation. As the graphs show, the mean service time and the mean queue time $95th$ percentiles are now nearly 16% and 29% lower then the respective baseline times.

It is notable that during all of the experiments mentioned above, the average CPU utilization by the monitoring system was 0%. The memory con-
sumption of the monitoring system depended on several parameters, such as number of deltas considered during the distribution calculations, the number of baseline measurements, and the maximal request time. These parameters could be tuned to achieve the right trade-off of memory vs. accuracy. During our tests, the memory usage of the monitoring system implemented in Java was about 50MB. This memory usage also included packet capture library and an additional statistics library. This suggests good scalability even though an explicit scalability study was not performed at this stage.

The main appeal of the percentile-based statistic analysis is its intuitiveness and simplicity. However, this approach should be exercised with caution in cases where requests differ dramatically making analysis of the CDF of mean values of $S$ and $W$ insufficiently sensitive. For instance, the aforementioned Trade6 application contains 15 different request types. The distributions of the number of these requests and their response time are presented in Fig. 4.11 and Fig. 4.12 respectively. Gradual CPU overload-
ing of the Trade6 server showed that the performance of all the requests degraded almost equally as the load increased. However, for the sake of a thought experiment, let us assume that the “Trade Home” request would be the only CPU intensive request out of all the 15 request types, while the rest of the queries would be I/O intensive. In such case, doubling the CPU load on the Trade6 server will case 100% degradation for the “Trade Home” requests. Nevertheless, the 95th percentile of the mean total response time increases only by approximately 9% which may be not enough to trigger an alert.

In such cases, the request-response data can be clustered, using algorithms such as X-means or PG-means [49, 70, 108, 116]. This is even an easier task than usual, since the data is univariate (using directly measured total response time). After clustering, data in each sub-cluster can be treated as a separate source of information exactly as presented above.

![Trade6 – requests count distribution](image1)

**Figure 4.11: Trade6 – requests count distribution**

![Trade6 – response time distribution](image2)

**Figure 4.12: Trade6 – response time distribution**
4.1.5 Conclusions

We have presented a novel out-of-band performance management system NAP. Major features of NAP include application and protocol agnostic monitoring and analysis, negligible performance overhead, accurate detection of hot spot tiers and of problematic VMs within these tiers, and an ability to trigger efficient remediation action in an autonomous manner. We evaluated NAP using the well accepted Trade6 benchmark with several types of workload. Our future research directions include:

- Evaluating NAP with real production systems;
- Evaluating and integrating NAP with a large scale, highly dynamic infrastructure cloud environment, where VM migrations are possible, such as RESERVOIR [113];
- Automatically learning – at the infrastructure level – of the elasticity rules that govern services capacity allocation decisions to satisfy fluctuations in demand in a cost-effective manner;
- Evaluating our solution in presence of network problems and software bugs.

We believe that wherever gray-box or white-box performance management is feasible, it is generally preferable over the black box one. However, there are many important settings, where gray-box or white-box approach is not applicable. As we show in this work, in such cases NAP offers an attractive and powerful alternative.
Algorithm 11 TCP/IP packets processing

Require: packet – TCP/IP packet
    req[] – requests’ hashmaps per connection
    res[] – responses’ hashmaps per connection
    stat[] – statistics per ∆

if packet.data.len == 0 then
    return
end if

∆ ← delta of the packet receive time
if packet.dst==server and (req[packet.conn].contains(packet.seq) or
    req[packet.conn].get(packet.seq).ack != packet.ack) then
    {a first packet of a request}
    req[packet.conn].put(packet.seq+packet.data.len,packet)
    stat[∆].requests.put(packet.seq+packet.data.len,packet)
    return
end if
if packet.dst==server then
    {not a first packet of a multi-packet request}
    req[packet.conn].remove[packet.seq]
    req[packet.conn].put(packet.seq+packet.data.len,packet)
    stat[∆].requests.remove[packet.seq]
    stat[∆].requests.put(packet.seq+packet.data.len,packet)
    return
end if
if packet.src==server and !res[packet.conn].contains(packet.ack) then
    {first packet of a reply}
    res[packet.conn].put(packet.ack,packet)
    stat[∆].responses.put(packet.ack,packet)
    return
end if
if packet.src==server then
    {not a first packet of a multi-packet response}
    do nothing
end if
Chapter 5

Related Work

5.1 Grid Resource Brokers

The purpose of this section is to briefly review some of the main resource brokering systems over distributed computing environments.

The Community Scheduler Framework (CSF)\(^1\) is an open source framework for implementing a grid resource broker, with the use of the Globus Toolkit Services, which provides an environment that can dispatch jobs to various resource managers. CSF was developed by Platform Computing in cooperation with Jilin University, China. CSF is an Open Source project.

CSF coordinates communication among multiple heterogeneous schedulers that operate at the local or cluster level. It supports advance reservation booking, query and control. The Reservation Service enables users to reserve resources under the control of a Resource Manager, in order to guarantee the resources will be available to run the job at a specified time. Reservations can be made according to the number of CPUs, start and end time, user name, host name or host type. The service supports reservations for any type of resource, including hosts, software licenses or network bandwidth, and requires a resource manager that supports advance reservations. CSF by default offers only simple scheduling capabilities. Round-robin and reservation based algorithms are available. Nevertheless, the main objective of CSF is to provide a simple API for writing user-defined policies.

\(^1\)http://sf.net/projects/gcsf
The Grid Service Broker\(^2\) is a resource broker developed in the Grid Computing and Distributed Systems Laboratory at the University of Melbourne, Australia. It was created as a part of the Gridbus project and supports access to both computational and data grids. The Gridbus Grid Service Broker is distributed under LGPL (GNU Lesser General Public License).

Gridbus can transparently access computational resources that are exposed by various low-level grid middleware solutions, such as Globus Toolkit (2.4.x - 4.0.x), Alchemi or XGrid. Access to resources available through Unicore (4.1) middleware will be possible after the next release of Gridbus. It supports scheduling systems such as Condor, PBS and Sun Grid Engine (SGE). Gridbus can interact with the scheduling systems, either with the help of Globus Toolkit or by using the remote SSH execution directly. The former approach to interaction relies on the services provided by the Globus Toolkit Pre-WebServices GRAM (Grid Resource and Allocation Management) interface.

There are two general strategies for the scheduling policy on computational and data grids. Each takes into account a specified deadline and budget, and then optimizes resource use either in time or space (number of nodes). Additionally, the design of the broker allows for writing a custom scheduler that implements a custom scheduling algorithm. Job-monitoring and status-reporting features are provided. Gridbus can gather the output of completed jobs and transfer it to user-defined locations.

GridWay\(^3\) is a lightweight resource broker developed by a team working for the Distributed Architecture Group from Universidad Complutense in Madrid, Spain. GridWay is an Open Source project and started releasing subsequent versions of the software on a GPL v.2 license.

GridWay is a metascheduling solution that fits into the Globus Toolkit ecosystem. It sits atop a grid stack composed of various scheduling systems – such as (Sun Grid Engine / N1 Grid Engine, Platform Computing LSF, PBS) and Globus Toolkit (2.4.x – 4.0.x) – and serves as a low-level grid middleware. Consequently, it is a slim solution that relies on Globus Toolkit components wherever possible. All communication between GridWay and the aforementioned scheduling systems is mediated by Globus Toolkit.

\(^2\)http://www.Gridbus.org/broker/
\(^3\)http://www.gridway.org
The GridWay project supports open standards, and therefore DRMAA (Distributed Resource Management Application API), a GGF standard, is used for communication with scheduling systems that provide appropriate DRMAA bindings (e.g., N1 Grid Engine). In order to facilitate the cooperation between GridWay and Grid Engine (Sun Grid Engine or N1 Grid Engine), the 'GE-GT adapter', developed by GridwiseTech, can be used. The emerging "Grid Scheduling Architecture," a GGF standard, is followed by the GridWay project to a significant extent.

The Moab Grid Scheduler\(^4\) is a feature-abundant resource broker developed by Cluster Resources Inc. It manages the resources on any system where Moab Workload Manager (a part of Moab Cluster Suite) is installed. Moab Workload Manager is a policy engine that allows sites to control the allocation of available resources to jobs. The metascheduler is available on a proprietary license. A limited variant, called Maui Grid Cluster Scheduler, is available on a specific Open Source, non OSI-compliant, license.

The metascheduler can also rely on the Globus Toolkit grid middleware for security and user account management. If the Globus Toolkit has not been deployed for user account management, third-party allocation management systems are used. The metascheduler supports fine-grained grid level fairness policies. Using these policies, the system manager may configure complex throttling rules, fair share, a hierarchical prioritization, and cooperation with allocation managers [60].

The GUR\(^5\) system was developed within the context of the TeraGrid\(^5\), an NSF-funded project to create and link several supercomputer class systems across one of the world’s fastest networks. GUR provides automatic negotiation of coordinated cross-site (pipeline and co-scheduled) and first possible run-time. GUR does not require centralized control over all the grid resources. It automatically reserves resources from a scientist’s laptop, not only making it easier for the scientist, but also reducing load on the remote “login” nodes. This decentralized local scheduler works like a “universal remote,” where it commands and coordinates access to multiple, heterogeneous distributed resources according to where you “point it,” much like a universal electronics remote.

In the TeraGrid project, the GUR tool is implemented as a python script

\(^4\)http://www.clusterresources.com/products/mgs/docs/
\(^5\)http://www.teragrid.org/
that uses the ssh and scp commands to help users make reservations, compile programs, and co-schedule jobs.

HARC, the Highly-Available Resource Co-allocator, is a system for creating and managing resource reservations. The implementation of HARC was supported in part by the National Science Foundation “EnLIGHTened Computing” project 6.

The goal of the HARC Architecture was to provide a co-allocation service without creating a single point of failure. HARC is based upon Lamport and Gray’s Paxos Commit protocol [69], which is where the term Acceptor originates. The Acceptors are a distributed group of services that act in a coordinated fashion. The overall system works normally if a majority of the Acceptors stay in a working state. For a deployment of 7 Acceptors, this would allow a Mean-Time-To-Failure of a couple of years.

5.2 Resource Matching Algorithms

One of the main problems we will discuss in chapter 3 is the graph matching problem. Namely, a graph of the requested processes must be properly matched to a graph of available grid machines. The purpose of this section is to briefly survey the existent graph matching algorithms, which are roughly divided into two types: exact and inexact.

Exact matching

Exact graph matching is characterized by edge-preserving mapping between the nodes of the two graphs, in the sense that if two nodes in the first graph are linked by an edge, they are mapped to two nodes in the second graph that are linked by an edge as well. In the graph monomorphism version of the problem, each node of the first graph is mapped to a distinct node of the second one, and each edge of the first graph has a corresponding edge in the second one; the second graph, however, may have both extra nodes and extra edges. Such a matching problem was proved to be NP-complete; therefore, all the exact graph matching algorithms are of exponential time complexity in the worst case.

6http://enlightenedcomputing.org/
Most of the algorithms for exact graph matching are based on some form of tree search with backtracking. The basic idea is that a partial match (initially empty) is iteratively expanded by adding to it new pairs of matched nodes; the pair is chosen using some necessary conditions that ensure its compatibility with the constraints imposed by the matching type with respect to the nodes mapped so far, and some heuristic condition to prune unfruitful search paths as early as possible is usually used as well. The first important algorithm of this family was by Ullmann [128] in 1976. Ullmann’s algorithm is widely known and, despite its age, is still prevalent: it is probably the most popular graph matching algorithm. To prune unfruitful matches, Ullmann proposes a so-called refinement procedure, which works on a matrix of possible future matched node pairs to remove, on the basis of a suitably defined necessary condition, the ones that are not consistent with the current partial matching.

Another interesting monomorphism algorithm based on backtracking was proposed by Ghahraman et al. [63] in 1980. In this paper the authors, in order to prune the search space, use a technique that is somewhat resembles the association graph.

A more recent algorithm for slightly different versions of the graph matching problems, isomorphism and subgraph isomorphism, is the VF algorithm by Cordella et al. [34]. The authors define a heuristic that is based on the analysis of the sets of nodes adjacent to the ones already considered in the partial mapping. This heuristic computes quickly, leading to a significant improvement over Ullmann’s and other algorithms in many cases.

Probably the most interesting matching algorithm that is not based on tree search is Nauty, developed by McKay [99] in 1981. The algorithm deals only with the isomorphism problem, and is regarded by many authors as the fastest isomorphism algorithm available today. It is based on group theory. In particular, it uses some results coming from this theoretical framework to efficiently construct the automorphism group of each of the input graphs. From the automorphism group, a canonical labeling is derived; it introduces a node ordering that is uniquely defined for each equivalence class of isomorphic graphs. Thus, two graphs can be checked for isomorphism by simply verifying the equality of the adjacency matrices of their canonical forms. The equality verification can be done in $O(N^2)$ time, but the construction of the canonical labeling can require exponential time in the worst
case (Miyazaki [100] in 1997 showed some classes of graphs that exhibit this exponential behavior). Nonetheless, in the average case, this algorithm has quite impressive performance, although in [43] and [51] it has been verified that under some conditions it can be outperformed by other algorithms such as aforementioned VF2. [31]

**Inexact matching**

In a 1980 paper, Gharaman et al., [64] proposed an optimal inexact graph monomorphism algorithm that is based on the use of branch and bound together with a heuristic derived from the netgraph. Tree search with backtracking can also be used for inexact matching. In [71], the $A^*$ algorithm is used with a fast and simple heuristic that takes into account only the future cost of unmatched nodes. A radically different approach is to cast graph matching, which is inherently a discrete optimization problem, into a continuous, nonlinear optimization problem. One of the pioneering works for this approach is that of Fischler and Elschlager [50].

Another inexact algorithm has been proposed by Cordella et al. in two papers [32], [33] in 1996 and 1997. This algorithm deals with deformations by defining a transformation model in which, under appropriate conditions, a subgraph can be collapsed into a single node. The transformation model is contextual, in the sense that a given transformation may be selectively allowed depending on the attributes of neighboring nodes and edges.

In [104], a new matching algorithm based on a probabilistic relaxation framework is proposed, which introduces the definition of a Bayesian graph edit distance. Gold and Rangarajan [65] presented the graduated assignment graph matching (GAGM) algorithm. In this algorithm a technique known as graduated nonconvexity is employed to avoid poor local optima [31].

### 5.3 Cloud Computing Monitoring Systems

Elastic computing has become increasingly popular since the end of 1990s with the spread of electronic commerce [7]. With recent advances in virtualization, elastic computing becomes more cost-effective and potentially easier to manage [135, 113]. Traditionally, performance diagnostics requires correlating the high level application state to the low level performance metrics of
the managed application [7, 28, 16, 17, 102]. This is achieved using in-band, application-aware performance monitoring and analysis. This approach is not always possible in the virtualized server setting.

Hypervisor-level out-of-band monitoring of virtual machines and statistical analysis of their data streams have been explored in Vigilant [107]. In one incarnation, Vigilant monitors the resource requests made by a virtual machine and applies machine learning methods to the data stream in order to detect problems [107]. While Vigilant shares common roots with this work, it has never attempted monitoring of service quality at this level of abstraction.

In [135], black-box and gray-box monitoring strategies for identifying performance hot spots in virtualized data centers are compared. The remediation mechanism studied by the authors uses migration of VMs from the physical hosts that are constrained on resources to the hosts that have excess resources. With respect to the network traffic, the out-of-band black-box monitoring mechanism used in Sandpiper [135] is analogous to NAP. However, the analysis of the traffic observed on the virtual interfaces of VMs is fundamentally different.

In particular, Sandpiper does not capture high-level application performance behavior by analyzing the TCP/IP level traffic as done by NAP. To assess demand (i.e., request arrival rate), Sandpiper relies on gray-box in-band monitoring that uses OS and application instrumentation. Proper appreciation of the request rate is critical for efficient remediation. In this work we show that it is possible to accurately deduce application level behavior, e.g., request arrival rate using solely black-box out-of-band monitoring of the TCP/IP traffic.

B-hive Networks’s 7 technology shares NAP’s goals: analyzing virtual machine network traffic in order to understand application performance, and sizing an organization’s virtual infrastructure in order to achieve specific service levels. Unfortunately, no information has been published about B-hive’s algorithms or system design.

In [81] the authors took a similar to NAP, albeit application dependent approach to identifying performance problems. Their approach is based on detecting sharp changes in the average service time. While very useful, this approach may be insufficient in two respects. First, it requires knowledge

---

of the application level communication protocol. Second, while an increase in average service time necessarily causes an increase in average total response time (ceteris paribus) – indicating, perhaps, that the CPU is the a bottleneck – the opposite is not always true. In our work we show a practical scenario when service time remains constant yet the average wait time grows dramatically, contributing to an increase in the average total response time, i.e., performance degradation. This case is indicative of the CPU not being a bottleneck, and the remediative action is to alleviate queuing by adding more instances of the bottleneck VM (scale-out). Thus, it is beneficial to consider both service and wait time behavior to improve diagnostics – the approach we take in our work.

The status of applications running on a distributed network is typically monitored passively or through a combination of application-level and passive network monitoring, for which popular standards such as NetFlow [11] and IPFIX [12] are used. Data flows are identified and data is collected on a per-flow or per-application basis, enabling calculation of network related performance metrics and detection of a wide spectrum of aberrant behaviors, some of which may be responsible for application performance degradation [103, 89]. Usually, these solutions are deployed on the routing elements inside the network rather than on the virtual network interfaces of the virtual machines. This makes it difficult to accurately differentiate between the network delay and service and wait time at the server. NAP, on the contrary, focuses on the server side and aims at the accurate evaluation of the server side contribution to performance degradation.
Bibliography


ניהול והקצאת משאבי בסביבת סרג
חישובי המבוסס על שירוטי ווב

וולטינ קרבצוב
ניהול והקצאת משאבים בסביבת סריג
הרשימה המבוסס על שיר 오히려 ווב

חיבור על מחקר

לשם מילוי החלק של הדירישוט לקבלת התואר
ודוקטור לפילוסופיה

וולטרה קרפקוב

הוגש לסנטה הטכניון – מכון טכנולוגי לישראל
הפקה ופרסום: הטכניון
יום אוגוסט 2009
המחקרenguשהבערכהפורופ'אסףשוסטרבפקולתהלמדעיהלמהשב.

אני מדע להכינוולמשלחתגאווהיהעלותחפכתהכפייתהolvencyבעקרופה.
תקציר

בעבודה זו מתואר חישובי סריג בסביבת משאבים וניהול הקצאה של סימולציות, התפתחות האלגוריתמים של סימולציות, בגלקסיו, וכתוף תנגשות ת'הנה אפליקציות מחשבים על לרוץ במקור נועדו ל-על, על הרצתן ובנוסף המשקפים כריית מחשבים שונים שכתוצאה מכך יכולים להתמקם במערכת DataMiningGrid Resource Broker ומאוד יש Caucus משאבי משימה של סימולציות על מ✄ surtoutקא מחשבים על ביצוע يقوم במערכת DataMiningGrid Resource Broker ומאוד יש Caucus משאבי משימה של סימולציות על מ✄ surtoutקא מחשבים על ביצועċיים באפליקציות שרת/洛克,_EC, במפעלה שמתאימו לשהית במערכת וורטואלים. 

אフォורטניסטית מתאימה (quasi-opportunistic) קיימם חסרי השכבה, ובחלק השולח על העתיד של מערכות פיתוח לעבודה. בצורת-אפליקציות מתאימה (quasi-opportunistic) קיימם חסרי השכבה, ובחלק השולח על העתיד של מערכות פיתוח לעבודה. בצורת-אפליקציות מתאימה (quasi-opportunistic) קיימם חסרי השכבה, ובחלק השולח על העתיד של מערכות פיתוח לעבודה. בצורת-אפליקציות מתאימה (quasi-opportunistic) קיימם חסרי השכבה, ובחלק השולח על העתיד של מערכות פיתוח לעבודה. בצורת-אפליקציות מתאימה (quasi-opportunistic) קיימם חסרי השכבה, ובחלק השולח על העתיד של מערכות פיתוח教え במערכת וורטואלים.

אנו מתמקמים מתוך มกราคม חישובי סריג בעבר, DataMiningGrid Resource broker. מחויבים החשוביםCOLUMN), AGRID (DataMiningGrid Resource broker) ומאוחדים ל האי-ל الكريمי, וב,</DOCUMENT>
Technion - Computer Science Department - Ph.D. Thesis PHD-2009-12 - 2009

The authors of the thesis detail the challenges faced by grid platforms and their platforms being built, which are both physical and virtual. The thesis addresses the need for accurate resource estimation, which is very difficult in many cases. It is argued that the success of the platform depends on the virtual resources of the platform, which are often difficult to estimate accurately.

The thesis also examines the processes of deploying the platform and its components. It is argued that the platform is not just a single client, but rather a set of servers working together to provide the required resources. The thesis discusses the need for accurate resource estimation, which is very difficult in many cases. It is argued that the success of the platform depends on the virtual resources of the platform, which are often difficult to estimate accurately.

The thesis also examines the processes of deploying the platform and its components. It is argued that the platform is not just a single client, but rather a set of servers working together to provide the required resources. The thesis discusses the need for accurate resource estimation, which is very difficult in many cases. It is argued that the success of the platform depends on the virtual resources of the platform, which are often difficult to estimate accurately.
The application is a collection of modules that is hosted by a server that is dedicated to the application of the internal structure of the organization. Understanding this requires that the matter and its applications, the client's needs, and its applications be altered in accordance with the virtual resources in the amount of change that is required in the application. Therefore, we can do that automatically in a system that is dependent on the potential is full of that is able to use it in a necessary manner. That allows the application of all the locations of the application to be executed in the cloud of thought of the server.

This is a problem of work very efficient, but very simple and technology of performance of problems of isolation for finding the application of virtual servers and the client's computer, among other things, that is large and flexible is a client's application. Therefore, the system that is dependent on the performance of the problem is able to tune the moment of the action that is used in the application.

Little NAP that is involved in the communication of information that is generated in the virtual computer and comes out of the input of the source, it is legitimate that is involved in the activity is determined by the fact that it is able to use the data that is processed in the application or to the type that refers to the data. Therefore, there is a moment, and the responses to the requests are between the difference that is given to the request in an individual. In addition, NAP that is involved in the time is determined by the entries that are given for the request. The use of a small number of additional parameters, that is, the average time of operation and the evaluation of the system are able to find the system's state in comparison with the situation.


developed at the Computer Science Department - Ph.D. Thesis PHD-2009-12 - 2009

.Properties of the system, measured in a process and then evaluated. Little NAP. Therefore, the time of the events that are involved in the application and the treatment of the moment after and keeping time is involved in the time of the system's state in comparison with the situation.