DEPARTAMENTO DE ARQUITECTURA Y TECNOLOGÍA DE SISTEMAS INFORMÁTICOS

Facultad de Informática
Universidad Politécnica de Madrid

Ph.D. Thesis

GLOBAL BEHAVIOR MODELING: A NEW APPROACH TO GRID AUTONOMIC MANAGEMENT

Author

Jesús Montes Sánchez
M.S. Computer Science

Ph.D. supervisors

María de los Santos Pérez Hernández
Ph.D. Computer Science

Alberto Sánchez Campos
Ph.D. Computer Science
M.S. Marketing

June 2010

Thesis Committee

Chairman:

Member:

Member:

External Member:

Secretary:

VLADIMIR: Well? Shall we go?
ESTRAGON: Yes, let's go.

They do not move.

Curtain.

Waiting for Godot
Samuel Beckett

Abstract

Over the last decade, grid computing has paved the way for a new level of large scale distributed systems. The grid can be defined as a set of geographically dispersed, interconnected computational resources, aimed at performing challenging computational tasks. This infrastructure makes it possible to securely and reliably take advantage of widely separated computational resources that are part of several different organizations. Resources can be incorporated to the grid, building a theoretical virtual supercomputer. However, this new step in distributed computing comes along with a completely new level of complexity. Grid management mechanisms play a key role, and a correct analysis and understanding of the grid behavior is needed. Grid systems must be able to self-manage, incorporating autonomic features capable of controlling and optimizing all grid resources and services. Traditional distributed computing management mechanisms analyze each resource separately and adjust specific parameters of each one of them. When trying to adapt the same procedures to grid computing, the vast complexity of the system can make this task extremely complicated.

But grid complexity could only be a matter of perspective. It could be possible to understand the grid behavior as a single system, instead of a set of resources. This abstraction could provide a deeper understanding of the system, describing large scale behavior and global events that probably would not be detected analyzing each resource separately. This abstraction could also be a solid, unified basis on top of which advanced grid autonomic management solutions could be developed.

In this Ph.D. thesis a specific methodology is presented and described in order to create a global behavior model of the grid, analyzing it as a single entity. The purpose of this model is to serve as the above mentioned abstraction of the grid system, providing an unique global behavior understanding. This global behavior model becomes also an extremely valuable tool for developing autonomic management mechanisms and contributes to a service-oriented, unified vision of the grid. This methodology is strongly based on system monitoring, performance estimation tools, knowledge discovery techniques and advanced scientific visualization. As a whole, it provides a unique and new point of view of grid computing, contributing to enrich and develop this technology.

To conclude, the proposed methodology has been tested on a series of typical experimental scenarios, both real and simulated, obtaining statistically meaningful results confirming that a global behavior model of a grid system benefits its understanding and serves as a solid basis to improve its autonomic capabilities.

Keywords: Grid computing, autonomic computing, behavior modeling.

Resumen

Durante la última década, la computación grid ha sentado las bases para alcanzar un nuevo orden de magnitud en los sistemas distribuidos. Se puede definir el grid como un conjunto de recursos (máquinas, elementos de comunicación y otros dispositivos) interconectados y geográficamente distribuidos, que se unen para enfrentarse a grandes desafíos computacionales. Esta infraestructura permite aprovechar simultáneamente múltiples recursos independientes pertenecientes a diferentes instituciones. Los recursos se unen al grid para dar lugar a un gigantesco superordenador virtual. No obstante, este avance en los sistemas distribuidos eleva a su vez su complejidad. Los mecanismos de gestión del grid juegan por tanto un papel decisivo. La mayoría de técnicas de gestión actuales analizan de forma separada cada recurso e intentan optimizar su funcionamiento de forma separada. Cuando se trata de adaptar este enfoque al grid, su elevada complejidad dificulta extremadamente el proceso.

No obstante, la problemática causada por la complejidad del grid podría ser simplemente una cuestión de punto de vista. Se podría intentar observar el comportamiento del grid como si de un único elemento se tratase, en lugar de un gigantesco conjunto de recursos. Esta abstracción ofrecería nuevo conocimiento sobre el sistema de forma global, permitiendo analizar la sinergia existente entre sus componentes en lugar de los detalles específicos de cada uno. Así mismo, proporcionaría una sólida base sobre la que desarrollar mecanismos de auto-gestión global del mismo.

Esta tesis doctoral presenta una metodología específica, diseñada para construir un modelo de comportamiento global de un grid, representándolo como una única entidad. Dicho modelo realiza la función de abstracción anteriormente mencionada, proporcionando una visión unificada y global del sistema. Este modelo sirve, a su vez, como una herramienta de gran valor a la hora de desarrollar mecanismos de gestión del grid. Se trata de una metodología fuertemente basada en la monitorización del sistema, mecanismos de estimación del rendimiento, técnicas de extracción de conocimiento y visualización científica avanzada. En conjunto, proporciona un nuevo punto de vista bajo el que analizar y desarrollar la tecnología grid.

Por último, la metodología propuesta ha sido validada mediante diversos estudios experimentales, en escenarios tanto reales como simulados. Los resultados obtenidos proporcionan evidencias estadísticamente significativas de que el modelo de comportamiento global del grid propuesto mejora el conocimiento del sistema y proporciona las bases para optimizar sus mecanismos de auto-gestión.

Palabras clave: computación grid, computación autónoma, modelado de comportamiento.

Declaration

I declare that this Ph.D. Thesis was composed by myself and that the work contained therein is my own, except where explicitly stated otherwise in the text.

(Jesús Montes Sánchez)

Table of Contents

List of Figures

List of Tables

Part I

INTRODUCTION

Chapter 1

Introduction

Since the appearance of the first cluster computers, distributed computing has become the common basis for the majority of new advances in supercomputing. Network interconnection has enabled to combine independent resources, making it possible to create powerful systems, capable of achieving top levels of computational power and new functional capabilities. Clear proof of this is that most of top 500 computers in the world are of distributed (cluster-like) nature (83% according to the 11/2009 list at TOP500 Supercomputing Site [TSS]).

With the emergence of the Internet and global interconnection, new forms of large scale distributed computing have appeared. Resources could not only be combined within local, private networks, but also geographically dispersed ones, enabling to access to a potentially unlimited pool of computational power. Several initiatives have explored this idea (described in more detail in Chapter 2) but it was in the late 1990s when the idea was deeply explored and developed with the appearance of *grid computing* [KF98] trying to address all possible related issues.

1.1 Motivations

The grid could be seen as the utmost expression of a large scale distributed system. It can be defined as a massive pool of heterogeneous and geographically distributed computational resources. These resources are coordinated by the grid, but not subject to a centralized control. They use stan-

dard, open and general-purpose protocols and interfaces to interact and, finally, the resulting system delivers non-trivial qualities of service. The grid allows us to globally share computing resources, storage elements, specific applications, specific-purpose systems, etc. Most of characteristics presented by grid systems were already present in other large scale computing initiatives. Grid computing puts all these ideas together, coordinating them and further developing their principles and implications. The grid provides a successful environment for applications that require a very large amount of computational and storage resources, such as numerical simulations, genetic analysis, complex natural simulations, etc. Many of these advanced applications are related to great scientific efforts, often called *grand challenge* problems, and have been the main objective of many grid development projects. A *grand challenge* is a fundamental problem in science or engineering, with broad applications, whose solution would be enabled by the application of high performance computing resources that could become available in the near future. Examples of grand challenges are computational fluid dynamics, structure calculations for the design of new materials, development of plasma dynamics for fusion energy, etc. Also, from a different, more market-oriented perspective, the grid can be an ideal infrastructure for developing *utility computing* solutions, *renting* computational resources based on client specific needs.

However, over the past decade grid computing has received some criticism, due to the many technological and social problems that the development of this technology creates. Among the most important technological issues are communication and software protocols integration, management, scalability, dependability and security. Among the non-technological ones, the most important are related to confidence and administrative issues between grid partners sharing resources, given the decentralized nature of the system. Most of these grid issues can be roughly summarized in one word: **complexity**. The extremely complex nature, at many different levels, of this kind of systems is the underlying cause of all these mentioned problems. This has an effect on all aspects of grid operation and needs to be handled properly in order to provide high performance, dependability and quality of service among other possible features. In order to build a grid computing infrastructure, its natural complexity has to be correctly identified and understood. Developing techniques capable of manage this complexity enables to provide scalability, dependability, quality of service, etc.

1.1.1 Managing complexity

One of the most common strategies to handle complexity in large scale distributed systems is the incorporation of *autonomic computing* features. Autonomic computing [IBM06] (discussed in detail in Chapter 3) is an attempt to deal with the system's complexity, in order to increase performance and other features. It was inspired by biological systems that can regulate themselves (like the human

nerve system). In autonomic computing systems multiple management tasks are done automatically and transparently, providing (among other features) reliability, dependability and quality of service.

The different aspects related to autonomic computing have proved to be beneficial for grid computing in many ways, making it possible to achieve some of its most ambitious goals. Incorporating autonomic features into grid management mechanisms strongly facilitates the system administrator task, which in a large scale system like a grid would be otherwise overwhelming, or simply impossible. Many of current grid management techniques include autonomic characteristics [BAG00, SF05, Sán08] dealing with each independent resource's separately and automatically optimizing its behavior in order to achieve an improvement in global performance. This approach seems reasonable, given the variety of individual, heterogeneous resources that can be part of a grid system.

However, this management approach focused on individual resources seems to be the opposite of what it is traditionally done when managing computing systems. Typical, less complex systems such as individual machines or clusters are usually regarded and analyzed as single entities, and management techniques address global issues that affect not only its individual components, but also the sum of its parts. This creates the idea of *the system as a separate concept from the sum of its parts*. This idea is made possible by an abstraction process that isolates the top level user from the machine's specifics.

When noticing this apparent difference between the grid and other systems, several questions arise, regarding its autonomic management: Why is grid management different? If the concept of **grid** exists from theoretical point of view, why there is no translation of it into practical terms? Is a grid abstraction as a single entity possible?

1.1.2 Antecedents

This Ph.D. thesis originated from the research lines initiated by the *Operating Systems Group* of the *Facultad de Informática* (Computer Science School) and the *Supercomputing and Visualization Center of Madrid* (CeSViMa), both institutions belonging to the *Universidad Politécnica de Madrid* (UPM). These research lines focused on high performance computing (HPC), parallel I/O, grid computing, advanced information analysis and scientific visualization.

In past years, the continuous research of the Operating Systems Group on HPC, and more specifically, on high performance I/O, led to the development of the MAPFS file system [Pér03] and its grid extension, MAPFS-Grid [Sán08]. The latter project led the group's research into the grid computing field, addressing new challenges and facing with the biggest issue of grid complexity.

Handling this complexity requires its understanding. Systems can be observed and behavior information can be collected, but in most situations extracting useful knowledge from that data is not a trivial task. The experience in this area of both Operating Systems Group and CesViMa provides the necessary background to make successful contributions in grid complexity analysis and behavior modeling. In particular, advanced knowledge discovery techniques and scientific visualization have proved to be of high value for this Ph.D. thesis.

1.2 Hypothesis and objectives

It may be argued that the grid is a large and complex infrastructure and that a global, unified vision of it is just not possible. However, this apparently unsolvable problem could be simply caused by a matter of perspective. Maybe current grid management techniques are excessively focused on low level technological aspects and, based on them, it is difficult to conceive how a global model can contain and explain all these little details. Given the service-oriented nature of the original grid idea, a hypothetical global model of the whole grid could be strongly beneficial for both user interaction and system management. Following this flow of reasoning, **this Ph.D. thesis aims at demonstrating that a high-level, global, unified and service-oriented model of the grid behavior can be constructed, and then it can be used to improve the grid's management and autonomic capabilities**. This also serves, therefore, as the basic hypothesis this work is focused on.

Based on this hypothesis, the main objective of this thesis is to create the necessary mechanisms to construct such a global behavior model and to develop the adequate scientific experimentations in order to validate this approach and demonstrate its benefits. In more detail, the following separated objectives can be distinguished:

- Investigation and analysis of the current behavior modeling and abstraction mechanisms in order to establish the global model basic characteristics.
- Formal development and validation of the necessary theoretical formalisms on which the grid global behavior model is based on.
- Study and identification of the theoretical benefits and implications of a global behavior approach regarding grid management.
- Formalization, design and implementation of a behavior analysis methodology capable of creating a grid service-oriented global model.

- Validation of the proposed methodology, focusing on its capabilities to provide and benefit from autonomic characteristics.

To fulfill these objectives, the following milestones were proposed, as the three capital achievements that should have been accomplished during the development of this Ph.D. thesis:

1. Formalization of the theoretical concepts required for the definition of a grid global model. These must include a basic and general behavior approach but, at the same time, they should be designed to define a model that could benefit from autonomic management.

2. Definition of a practical methodology capable of creating such a model, describing and validating its principal characteristics.

3. Demonstration of the benefits of this global behavior approach on different scenarios, focusing on autonomic management aspects of grid computing.

As a summary, this work attempts to address the fundamental issues regarding the definition and formalization of a global, abstract vision of the grid, highlighting its features and benefits.

1.3 Document organization

The rest of this document is divided into three thematic blocks. Each one is divided in chapters to facilitate its understanding:

- **State-of-the-art:** the first part gathers the current state of the researching areas related to this Ph.D. thesis work.
  - Chapter 2 describes several distributed computing technologies designed to solve challenge problems that demand great computing and storage capacities. Most of its content is dedicated to grid computing, as it is the main scope of this work. Other distributed technologies such as cluster and cloud computing are also discussed.
  - Chapter 3 describes the advances made to deal with the growing complexity of computer systems that led to the software complexity crisis. It focuses on autonomic computing, aiming at finding automatisms to obtain the maximum performance of applications without having expert knowledge or complex control systems.
  - Chapter 4 studies several statistical and knowledge discovery methods that can help to model large scale distributed systems.

- **Problem statement and proposal**: the second part constitutes the core of this work. It states the problem to be solved and the proposed approach. The proposal addresses the combination of the researching lines of autonomic computing, behavior modeling and data mining, in order to analyze the grid and create a global behavior model. The model construction methodology proposed is thoroughly analyzed and validated in different experimental scenarios. Finally some extensions to this methodology are proposed, to incorporate behavior prediction capabilities.

  - Chapter 5 describes the initial problem statement, developing the idea of a single-entity, service-oriented, global model. It addresses these issues from a basic, theoretical perspective, analyzing its options, alternatives and implications. It focuses on an autonomic management perspective.
  - Chapter 6 further extends the autonomic computing discussion and formalizes the basic behavior modeling concepts, from a global grid service-oriented perspective.
  - Chapter 7 focuses on formally defining a practical methodology to create a global model of the grid behavior. This chapter provides an elaborated study of the proposed methodology and presents experimental results to validate the approach.
  - Chapter 8 empirically illustrates how grid autonomic management can be benefited from global behavior modeling, presenting detailed experimental results.
  - Chapter 9 explores an advanced extension to the global grid model, incorporating behavior prediction capabilities.

- **Conclusions and future work**: the last part shows the conclusions of this Ph.D. thesis and the open lines of this proposal.

  - Chapter 10 extracts the most important conclusions obtained from the achievements of this work.
  - Chapter 11 describes the possible researching lines that arise at the end of this thesis development.

Part II

STATE OF THE ART

Chapter 2

Large-scale distributed computing

Moore's law, stated by Gordon Moore in 1965, predicted that "the number of transistors per chip that yields the minimum cost per transistor is roughly doubled every two years" [Moo65]. This was an ambitious attempt to foresee the technological evolution of the computing field. A few years later the law was slightly revised, reducing the time period to 18 months. Surprisingly, Moore's law predictions continue being fulfilled today, almost 45 years after it was stated. However, as the available computing resources increase, the demands the society imposes on this technology grow accordingly. The possibilities that the new computing infrastructures leave open stimulate the industrial and scientific communities, continuously producing new challenges of increasing complexity.

Along with these never-ending increasing demands, one of the basic principles of modern society economics is to maximize the benefit/cost ratio. Computer Science is not an exception in this case, and computing technological advances have followed this premise from the beginning. The effect of this basic principle has had a very clear impact on the field in the last two decades, drastically changing the way the most basic computational resources are exploited. Years ago many computing challenges could only be faced by means of highly expensive, hardly scalable supercomputers. Nowadays the focus is on relatively cheap, regular resources, not very powerful independently, connected by means of communication networks and programmed to work together. The synergy that occurs during this distributed interaction is the key to new levels of computing evolution.

A very clear example of this evolution is the CERN's[1] particle accelerator Large Electron-Positron (LEP). In the early years of its operation, Cray supercomputers were used in order to analyze the data it produced. These were replaced by clusters in the late nineteens and finally, the newer Large Hadron Collider (LHC) uses grid computing technology.

As correctly stated in [Mar02], the main problem of this modern approach is that, in order for it to be successful, carrying out distributed computing among several resources must have lesser cost than in a single machine, and achieving this is not always an easy task. On the one hand, High Throughput Computing (HTC) applications are easily adaptable because they can be divided in parts or jobs, which can be assigned to different machines. High Performance Computing (HPC), on the other hand, usually presents a high level of interaction between tasks, requiring reliable and high-speed communication, so the application can efficiently progress. In this case computing distribution in independent blocks is much more complicated.

2.1 Cluster computing

The first alternative to multiprocessor systems was cluster computing. The main objective of this technology was to improve performance vs. cost ratio. A *cluster* is a set of dedicated, independent machines (each one with its own CPU or CPUs, memory, etc.) interconnected by a private, fast, commonly dependable and dedicated network. Nowadays clusters provide high performance, availability, dependability and scalability. The independent machines that make up a cluster (commonly referred to as *nodes*) are normally homogeneous, presenting identical hardware and software characteristics. However, there are many clusters that present certain degree of heterogeneity, incorporating various types of nodes nodes, usually dedicated to different specific tasks.

The cluster management system is on top of all the interconnected hardware and software from independent machines that together form a cluster. Its basic function is to provide the necessary mechanisms to make all these cluster nodes work together and provide service. The cluster management system handles user interaction and internal coordination, controlling aspects such as load balancing, fault tolerance and single system image. To carry out this tasks it usually incorporates tools such as monitoring mechanisms and predefined management policies aimed at control and improve performance. The cluster management software usually works very close to the node's operating system, performing numerous operations at system level (otherwise tasks such as job migration could not be possible). In short, the cluster management system works as a middleware layer between the user and

[1] *Conseil Européen pour la Recherche Nucléaire* (English: European Organization for Nuclear Research)

the cluster nodes, providing:

- Maintenance and optimization tools. Modern cluster management software performs tasks such as job checkpointing, load balancing, fault-tolerance, advance scheduling, etc.
- A single system access interface, commonly named *single system image*. This presents the user with an unified point of access and interaction, providing the same response that when operating a single computer. This isolates the user from the inherit complexity of the cluster distributed nature. This is important, considering that modern clusters sometimes incorporate hundreds or even thousands of nodes.
- Scalability. The cluster management system should be prepared to automatically detect new nodes or other resources incorporated to the cluster and take advantage of them.

Nowadays there are several cluster management systems in use. Among those most frequent, MOSIX, OpenMosix, LinuxSSI and Beowulf can be distinguished. MOSIX [mos], and its extended open source release OpenMosix [OMo], provide load balancing in a transparent way, making possible for a cluster to behave as a *sort-of* single machine. LinuxSSI [LSS] is an operating system for clusters developed over the existing Linux-based Kerrighed technology [Ker], which provides single system image. Beowulf [Beo] is a cluster technology based on Linux computers designed to construct a parallel virtual supercomputer. Besides these typical cases, there are more simplistic queue managers, designed to send jobs to each node depending of its workload, like OpenPBS [PBS] and Condor [Con]. These can not be consider cluster middlewares *per se*, as the are not aimed at providing single system image. However they strongly simplify the use of a set of independent, interconnected nodes.

Although cluster computing provides a relatively cheap alternative to more traditional supercomputers, there is sometimes the misconception that any piece of software would execute faster on a cluster, as it is supposed to provide higher performance. However, it is important to remember that, to achieve this, the software must undergo an adaptation to this different, distributed environment. This means, in most situations, re-devoloping most part of it, trying very specifically to avoid non-dependent parallel operations that would hurt performance.

Finally, there are other issues and limitations regarding cluster computing. This technology enables the efficient use of distributed resources, but under a very controlled, secure conditions. This makes impossible a real combination of resources from different administrative domains (i.e. putting together computational resources from different organizations) as different security policies will always interfere with the natural cluster operation. Furthermore, most cluster technology is not based on open standards, which complicated the growth and communication among applications.

2.2 Grid computing

In 1995, during the 8th ACM/IEEE conference on Supercomputing in San Diego, California (SC'95) an innovative distributed computing experiment was presented. This experiment demonstrated how it was possible to execute distributed applications in 17 separated research facilities owned by different centers, connected by a high speed network. This *proof of concept* experiment inspired a new research line, aimed at making possible to share large-scale computational resources in a distributed way.

Taking advantage of these different available computational resources, physically separated but all connected by means of the Internet network is what gave birth to *grid computing*. The term *grid* was used for the fist time in this context by Foster and Kesselman in [KF98]. It came inspired by the idea of making computer power as easy to access as an *electric power grid*, offering an abstract, uniform and cheap access point to electricity anywhere. Foster and Kesselman idea was to make grid computing capable of providing computing power and data storage under analogous terms.

Therefore, a grid can be intuitively defined as an enormous set of machines connected by any communications network (the Internet, WAN, etc.) joining efforts to carry out costly computing tasks and providing specific services. In short, a grid is a pool of geographically dispersed distributed computing resources that work together.

Unlike most traditional distributed systems, grid computing is based on different organizations sharing resources in a non-dedicated way. This means that grid partners do not need to reserve and dedicate specific resources to the grid computational pool. Grid resources can be simultaneously and independently used by their respective owners. This means that some grid nodes can be, for example, part of a grid and available to grid users under certain conditions and, at the same time, part of a private corporative network and be used by the institution employees under completely different terms. Grid computing technology provides the necessary mechanisms to guarantee that this duality takes place without security concerns of any kind. The conditions under which each grid partner shares its resources can be set independently, and the resources owner always keeps full administrative control of its property.

This non-dedicated way of sharing resources makes much more easy for grid partners to contribute to the common resource pool, as they do not need to dedicate specific hardware and are able to fully control how their resources are being used. Grid computing can take advantage of low workload periods of all connected non-dedicated resources, enabling to sum up enormous quantities of comput-

ing power. This makes possible to solve extremely complex computational problems, very difficult to process using a single machine. Grid computing makes the most of these spare computational power from non-used resources, creating a *de facto* virtual supercomputer.

From a user-level perspective, the grid is capable of simultaneously accepting an enormous amount of service requests from users belonging to all grid partners. Correct autentification and other security related issues are normally handled in by means of electronic certificates and other advanced trustworthy computational solutions.

It is important to stress that grid computing is not intended to replace other previous distributed computing technologies. It is aimed at addressing more global, large-scale problems, making it possible for different authorities or virtual organizations to share their computational resources, without compromising the private security policies and management tools [Fos01]. In consequence grid computing must respect certain rules:

- It must not interfere with site or organization management.
- It must not put user and organization security at risk.
- It must not force to install or modify previously existing nodes basic tools such as operating systems, networks protocols or services.
- It must enable organizations, sites or resources to join or leave the grid at any moment.
- It must provide a reliable infrastructure.
- It must provide support for heterogeneous components.
- It must use existing standards and technologies.

In [Fos02], I. Foster proposed a three-point checklist of requirements any grid should meet:

1. "Coordinates resources that are not subject to centralized control...". Following the idea above introduced, each grid partner keeps full control of its own resources, allowing a decentralized structure with possible different security and management policies for each site or virtual organization.
2. "...using standard, open, general-purpose protocols and interfaces...". This is necessary in order to make possible interaction among resources in the heterogeneous, decentralized structure of the grid.

3. "..to deliver nontrivial qualities of service.". This stresses the idea that grids are built to share resources and achieve new height in computing power capabilities, providing quality of service in aspects such as response time, throughput, availability and security.

In short, among all the benefits of grid computing, the following ones can be stood out:

- Potentially unlimited computational power provided by thousands or even millions of globally interconnected computers.
- Possible bottleneck elimination in some computing processes due to a very high resource availability, offering the chance to select the most adequate computing solution in each case.
- Possibility of managing and taking advantage of multiple sites and domains corresponding to different virtual organizations, with all the computational resources in them.
- Integration of systems and heterogeneous devices. A grid can be built from heterogeneous resources and include a large range of different technologies.
- Scalability. Grids can grow from being formed of a few resources to millions of them. In traditional distributed systems, this very often leads to operation degradation as the system size increases. Grid computing is designed taking the heterogeneity, size and resource global dispersion into account, developing services adapted to these conditions. This guarantees scalability and quality of service at the same time.
- Adaptability. The frequent variability of resources is a basic aspect of grid technology. Grid services and applications are designed with this in mind, handling this increased complexity and adapting its behavior dynamically to use the available resources at any given time.

2.2.1 Grid middleware

The software infrastructure responsible of creating and managing the grid is called the *grid middleware*. Its functions make possible to provide the above mentioned grid features. It coordinates the grid resources, manages the grid services and nodes, controls all security aspects involved and provides a unified user interface. The grid middleware is composed of the following several service layers:

1. *Grid fabric*. This represents all the grid distributed resources that are accessible from any location of Internet. These resources can be any kind of computing nodes (single PCs, multiprocessors, clusters...) with possibly different operating systems (Windows, UNIX, Linux...), storage devices, databases, and scientific instruments such as telescopes, climate sensors or other devices.

2. *Core grid middleware*. This represents basic services such as remote process management, resource assignment, storage access, information discovery, security, and other different quality of service (QoS) related aspects, such as resource reservation and brokering.

3. *User level grid middleware*. This includes mechanisms and tools for developing user level applications intended to access and manage global resources.

4. *Grid portals*. This represents a web service infrastructure where final users can run applications and obtain results in a transparent way.

In the following subsections each of these service layers will be described in detail.

2.2.1.1 Grid fabric

This layer contains a set of services designed to make possible the efficient use of local resources. There can be many different types of services within the grid fabric, but most grid computing solutions contain at least the following three:

- The *queue manager* is the service usually in charge of managing and distributing the jobs that are executed in each local resource. Sun Grid Engine [SGE] and Condor [Con] stand out among typical queue managers, due to their widespread use.

- *Parallel computing libraries* are almost always present, such as MPI [MPI] and PVM [PVM]. They are necessary to take advantage of compound local resources, such as clusters or shared-memory supercomputers.

- *Resource monitoring* services, such as Hawkeye on Condor [Haw] and Ganglia [gan] are a necessary part of the grid fabric. They are used to observe each single local resource performance at any given moment.

2.2.1.2 Core grid middleware

On top of the grid fabric rests the core grid middleware. This is a critical layer of the grid system as it drastically conditions the whole system's functions and structure. Nowadays there are several core grid middleware technologies. Among those, Unicore, Globus and gLite should be noted for their widespread use and scientific relevance.

UNICORE (Uniform Interface to COmputer REsources) [ES01] is a grid middleware initiative developed in two projects funded by the German ministry for education and research (BMBF). UNICORE provides seamless, secure, and intuitive access to distributed grid resources such as supercomputers or clusters and information stored in databases. Applications distributed by means of

UNICORE are divided in parts, which run in different computers in an asynchronous or sequentially synchronized way. An UNICORE job contains a multi-part application indicating its resources requirements and dependences among different parts. The UNICORE technology is developed as open source under BSD license.

The aims of the UNICORE's design are based on:

- Having a simple graphical user interface.
- Architecture based on the concept of abstract jobs including security.
- Minimal interference with the local procedures.
- Use of existing technologies taking into account the new growing technologies.

UNICORE is designed to support batch jobs using a distributed system in which the different parts of the application can be run. In order to standardize the accesses to the computers, the user id is unique.

The Globus toolkit [FK97, Fos05] is the *de facto* standard grid computing middleware. Nowadays it is being used by leading computing companies, such as HP, Cray, Sun Microsystems, IBM, Microsoft, Veridian, Fujitsu, Hitachi and NEC. The *Globus Alliance* [GlA] was born from the joint efforts of research and development groups of the Argonne National Laboratory (Illinois), the University of Southern California Information Sciences Institute, the University of Chicago, the University of Edinburgh and the Swedish Royal Institute of Technology. It is headed by Ian Foster and Carl Kesselman, co-authors of the grid seminal paper [KF98]. Globus is an open-source and open-architecture project that incorporates protocols and basic services for constructing grids. Furthermore, it provides support to run applications taking advantage of any available grid computational power.

The Globus structure and basic elements are shown in Figure 2.1. As it can be seen, it builds upon the grid fabric in order to provide the grid core middleware services.

Globus includes software for security, information infrastructure, resource management, data management, communication, fault detection, and portability. It is packaged as a set of components that can be used either independently or together to develop applications. Every organization has its own modes of operation, and collaboration between multiple organizations is hindered by incompatibility of resources such as data archives, computers, and networks. The Globus Toolkit was conceived to remove obstacles that prevent seamless collaboration. Its core services, interfaces and protocols allow users to access remote resources as if they were located within their own machine room while

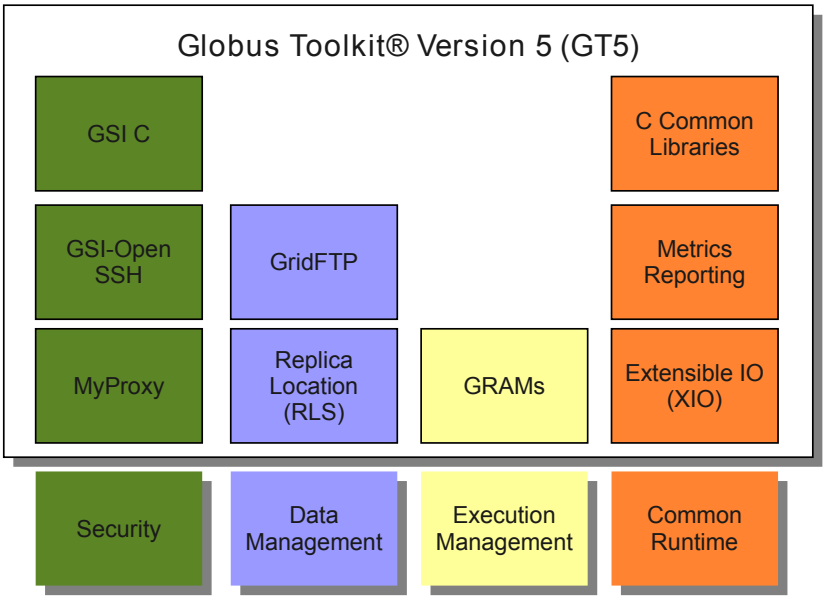


Figure 2.1: Globus structure and services [AGl]

simultaneously preserving local control over who can use resources and when. The Globus toolkit has grown through an open-source strategy and collaborative effort, similar in many ways to the Linux operating system.

gLite [Gli] is the core grid middleware used by the CERN LHC experiments (Worldwide LHC Computing Grid, WLCG) [wlc] and a very large variety of scientific domains. It was born from the collaborative efforts of more than 80 people in 12 different academic and industrial research centers as part of the EGEE Project [ege]. It was originally based on an early version of the Globus toolkit, although since its creation gLite has evolve in a completely independent way. The gLite middleware includes a complete set of services for building a production grid infrastructure. It provides a framework for building grid applications tapping into the power of distributed computing and storage resources across the Internet. gLite middleware is currently deployed on hundreds of sites as part of the EGEE project and enables global science in a number of disciplines, notably serving, as above mentioned, the WLCG project [Gli, wlc].

Nevertheless, operating systems underneath have no notion about the core grid middleware running on top (whether it is UNICORE, Globus or any other). Integration between grid middleware and operating systems could make possible further different improvements, such as better individual resource management and accurate job monitoring.

Thus, the XtreemOS project [Mor07] has taken the first steps in this direction, introducing some grid concepts into the Linux operating system. XtreemOS determines which new services should be added to current operating systems to build grid infrastructures in a simpler way. In essence, the idea behind XtreemOS is to integrate the core grid middleware into the operating system.

XtreemOS addresses many aspects regarding to grid services. Among those the following should be highlighted:

- Support for virtual organizations.
- Provide the users with interface an tools similar to those that can be found o a traditional computer.
- Abstraction from the hardware and secure resource sharing between different users.
- Robust, secure and easy-to-manage infrastructure for system administrators.

From a different point of view, a complete grid operating system could be built to improve the deployment of grid middleware. In this sense, GridOS [PW03] is an operating system that makes a regular computer more suitable for being incorporated into grid systems.

2.2.1.3 User level grid middleware

In 2002 the grid computing community adopted a unified orientation, based on a services model [FKNT02]. This was subscribed with the definition of a new basic grid architecture, called *Open Grid Services Architecture* (OGSA), by the *Global Grid Forum* (GGF). OGSA defines the necessary mechanisms to provide both creation and maintenance of different services, offered within a multiple *Virtual Organizations* (VO) framework.

The Global Grid Forum was an international organization in charge of defining standards for grid computing protocols and services. It was created in Amsterdam in 2001 with the aim of becoming the international forum about grid technology. Among its most important partners were universities and research centers worldwide, along with companies such as IBM, Microsoft, Sun and HP. In 2006 the GGF and the *Enterprise Grid Alliance* (EGA) joined efforts to create the *Open Grid Forum* [OGF]. The EGA was formed in 2004 to focus on the grid adoption in enterprise data centers. Nowadays the OGF has two principal functions, being the first one development of grid standards, and second helping to build communities and partnerships within the overall grid world, including extending it to encompass wider participation from both academia and industry.

In February 2003, the Global Grid Forum, the Globus Alliance and IBM proposed together a convergence of grid computing technology towards Web services. Thus, the GGF defined OGSA [FKNT02, FSB+06, ogs], the necessary architecture to provide grid services. OGSA defines the services that must be offered by grid computing systems. More specifically it defines the following service capabilities:

- Infrastructure services.
- Execution Management services.
- Data services.
- Resource Management services.
- Security services.
- Self-management services.
- Information services.

Specifically, grid services are a considerable extension of Web services. Web services can not give support to grid applications because of the following limitations:

- They are stateless, that is, they are not persistent services.
- They do not give support to services, like notifications and life cycle management.

On the other hand, grid services have the following features:

- They are stateful. The state of the grid service is kept during invocations.
- They can be persistent. Some instances of the same services can be created at a given moment destroying them when they are not necessary.
- They have support services.

Thanks to the relation between the OGF and Consortium for World Wide Web (W3C)[2], grid services have been fused with Web services (WS) in an single research line named WS-Resource Framework (WSRF) [GKM+06]. WSRF provides access to stateful Web services.

[2]W3C was founded in October 1994 to promote the World Wide Web, developing standard protocols to ensure its interoperability.

Although there are many known grid projects related to computational power, like Worldwide LHC Computing Grid (WLCG) and Enabling Grids for E-SciencE (EGEE), computing is not the only important aspect of a grid. Other applications such as data access and utility computing are for many applications more important than the access to high performance computing resources.

In short, grids can provide several types of services [BBL02]:

- **Computing services**. Grids can provide tools to run complex jobs on distributed computing resources. Some examples of computational grids are NASA Information Power Grid (IPG) [CFF+01], WLCG [wlc] and NSF TeraGrid [TeG].
- **Data services**. Grids can also provide secure data access and data management. Data can be catalogued and replicated in order to manage it. A data grid offers an environment in which is possible to manage huge data volumes in a distributed way. The projects European Data Grid (EDG) [CFF+01] and CrossGrid [CrG] can be stood out among all the projects belonging to this type.
- **Application services**. Another typical grid use is to provide application management and a transparent access to remote software and libraries. They are built to use computing and data services provided by the grid. A system used to develop such services is NetSolve [SYAD05].
- **Information services**. Most grids also include services designed to obtain and show information observed and collected about computing performance, data access and application services.
- **Knowledge services**. These are referring to the way the system knowledge is acquired, used, recovered and published to help to the users to fulfill its aims. The knowledge is understood as information applied to reach an aim, solve a problem or make a decision.

In order to search, discover and select the most suitable services or resources an element (or set of elements) containing information about the numerous grid resources is required. This element is usually called *broker* [SFT00, Tor04]. The grid services capabilities depend on the way the broker handles clients and resources. Brokering is a key aspect of modern service-oriented grids.

The way a grid is used and the user experience required are directly determined by the broker's operation and performance. Modern grid brokers identify, characterize, evaluate, select and reserve the most suitable resources or services for a certain job, according to the user needs. However, brokers can simply be traditional job allocators, simplifying its configurations and management, but losing some interesting high-level features. In this sense, many queue managers used in other distributed systems can be adapted to be applied in grid systems. Some examples of this adaptation are PBS and

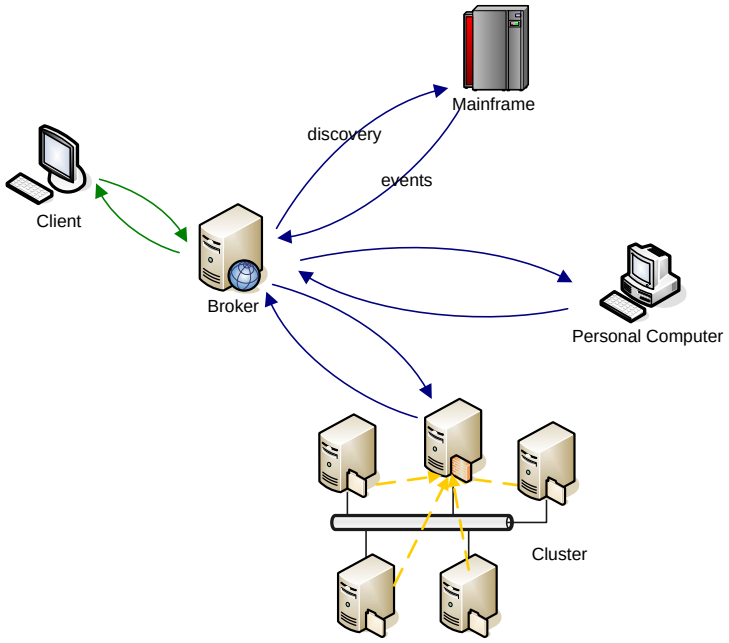


Figure 2.2: Brokering in a grid system

Condor-G [CoG], a grid specific version of Condor.

Different grid middleware projects include brokering mechanisms with different characteristics, but most approaches can be classified in two possible groups:

- **Client broker**. This kind of broker is designed to handle client needs selecting the services according to client requirements without having a global vision of other clients or service requests. In client brokers most of brokering tasks are incorporated into the client software, in the form of access and decision making libraries on top of which the application is build. A very popular example of this approach is GridWay [HML05, HHML05, GWM]. This widely used client broker can carry out the most usual brokering tasks required, such as resource assignment, task migration and fault-tolerance. For GridWay the client-broker integration becomes an advantage because it is more scalable and resistant to dynamic changes, since it focused only on the needs of a single client. GridWay can be used as a part of Globus and/or gLite.

- **System broker**. In this case, the broker has a complete and global vision of all resources and clients of the grid. Thus, it can select the most suitable resources for the client jobs with the aim of improving the performance of the whole grid. System brokers usually operate as independent entities, designed to communicate and interact both with clients and grid resources. Nevertheless, it concentrates the decision making and consequent load on a single point, raising dependability issues and creating a possible service's bottleneck. To avoid these problems,

broker hierarchies and other redundancy mechanisms have been proposed.

Nowadays the scientific community is working on improving different aspects of grid services. Service scalability, introduction of autonomic computing techniques, global management improvement and dependability related issues (fault tolerance, quality of service...) are important research areas. This Ph.D. thesis is mainly focused on the second and third of these research lines. Although several improvements can be performed, there are a large number of working projects, such as WLCG [wlc], EGEE [ege], TeraGrid [TeG] and CrossGrid [CrG], that solve compute and data-intensive problems. In addition, there are many finished projects that demonstrate that the grid technology is possible, like GridSim [BM02a], IPG [JGN99], EDG [CFF+01], Gridbus (GRID computing and BUSsines) [Gbu]. Among them EGEE and WLCG can be emphasized.

The Large Hadron Collider (LHC) is the CERN's particle accelerator. The discovery of new fundamental particles, like specifically the Higgs boson or God particle[3], is one of the most important aims of the LHC. It is possible to know the existence of this kind of particles analyzing the properties by means of a statistical analysis of massive amounts of data collected by the LHC's detectors ATLAS, CMS, ALICE and LHCb. 15 petabytes per year is the data size expected to collect by the LHC from 2007. Then, a comparison by means of compute-intensive theoretical simulations is required.

The analysis of this great amount of data means one of the most important scientific challenges in the world. The aim of the Worldwide LHC Computing Project (WLCG) [wlc] is to build an analysis infrastructure in order to understand this data. Furthermore, data has to be stored in a distributed way since its size is too big to be stored in a single storage system. The WLCG project inter-operate with other known grid projects, such as EGEE, Grid3 [GR3] and The Globus Alliance.

The Enabling Grid for E-sciencE (EGEE) [ege] project aims to develop a grid infrastructure that is available to the geographically scattered scientist community to solve different kinds of complex problems in different domains. EGEE tries to cover a wide-range of both scientific and industrial applications, including Earth Sciences, High Energy Physics, Bioinformatics, Astrophysics and so on.

Two pilot applications were selected to test the performance and functionality of the EGEE's infrastructure. Both the LHC Computing Grid and Biomedical Grids, where scientists facing with bioinformatics and health-care problems, certify its use.

[3]The Higgs boson is the key particle to understand why matter has mass.

The problem of this infrastructure is constituted by the fact that not all the grid requirements stated by Foster [Fos02] are fulfilled. EGEE has stated several requirements that a researching center must fulfill to join it to the initiative. The aim was to create quickly a simple grid infrastructure to solve problems that do not have solution with other technologies. Thus, the most of computers used in EGEE have the same architecture and operating system. Besides each VO or researching center has lost some of control on its own resources, since there is a central organization that manages the EGEE's resources.

2.2.1.4 Grid Portals

To facilitate interaction between users and the system, grids can benefit from unified, user-friendly access interfaces such as web portals. These tools make possible to access the grid from any location, providing a easy to use, graphical interface to its services (job execution, data access, etc). Furthermore, the specific costs of grid software installation and setup in every client machine is eliminated, as operations are carried out through a standard web interaction.

Most big grid projects include web portals that make different grid services available via web, such as information services requests, file transference and job management. However, these interfaces are usually specific of each project. In addition, there are some initiatives trying to facilitate grid web interfaces development, providing generic tools for grid portal construction that can be used in different systems. Among these initiatives GridSphere and the OGCE portal and gateway toolkit could be highlighted.

GridSphere [GSp] is a project to make easy the creation of Grid Portals based on *portlets*. A *portlet* is a component that can be integrated in a portal web and works as a web application managed by a central manager.

The OGCE (Open Grid Computing Environments) portal and gateway toolkit [ogc] is a set of technologies designed to contribute in the development of scientific portals for computational grids and cloud computing resources. By means of OGCE, developers can extend the functionality of a web portal to run complex applications in a grid. OGCE has been used to develop scientific portals and it is compatible with commonly used grid middleware software such as Globus.

2.2.2 Grid performance

Performance parameters such as throughput, network bandwidth or response time are typically used in distributed systems. Grid operation can be observed using these metrics, but its special char-

acteristics require for additional parameters to be considered, such as:

- Resource quality of service.
- Service availability.
- suitability of the process assignment to resources.

In short, it is very important to take into account the adaptation of applications to the infrastructure knowing the principal features of a grid:

1. Diversity of resources.
2. Dynamism of resources.

Therefore, it is necessary to analyze the way the used resources have been selected, above all its suitableness and efficiency.

[GWB+04] shows the required conditions that an analysis performance tool for grids must fulfill. Grid performance analysis has special requirements, emphasizing:

1. Data acquisition should include both application and infrastructure monitoring, since grid performance is determined by the configuration of the used resources and the running applications.
2. Monitors should have an intelligent part in order to filter and preprocess monitored data because monitoring a grid can involve a great amount of data.
3. Monitoring techniques should be able to detect performance problems automatically from their symptoms.
4. Monitoring tools should be able to be customized since a grid is heterogeneous and dynamic.

Grid benchmarking and grid monitoring are usual methods as a fisrt step to identify and understand grid behavior.

2.2.2.1 Grid benchmarking

Benchmarks can be used to analyze grid performance, define the expected quality of service, evaluate resources assignment policies and compare service implementations or even complete systems. The main effort in building standard grid benchmarks was mainly directed by the Grid Benchmarking Research Group at the now extinct Global Grid Forum.

Most grid benchmarks are computation intensive and based on NAS Parallel Benchmark[4]. Among them, NAS Grid Benchmark [FdW02] and GridBench [TD03] can be emphasized. Among other types, the Arithmetic Data Cube (ADC) [FS03] can be distinguished as the single known possibility for data intensive grid benchmarking.

The NAS Grid Benchmark (NGB) [FdW02] is an intensive computing benchmark. The NGB tasks are defined in terms of flow charts, where each node represents the computing elements and the computing times within them, and each arch represents the communication between the computing elements. The grid performance is calculated from operation logs, which include running times for each node and transmission times in each arch. This benchmark is very useful to analyze the critical path for a certain application.

GridBench is a benchmark definition tool for specific grid configurations, and requires these configurations to be defined. In order to do it, GridBench has its own *Grid Benchmark Description Language* (GBDL) [TD05] that can be automatically translated to other job description languages like JDL (Condor) or RSL (Globus) commonly used in grid computing systems.

In order to be user-friendly, GridBench provides a graphical interface that allows to define and execute benchmarks and analyze the obtained results. These results are based on the architecture R-GMA defined for the project European Data Grid (see section 2.2.2.2).

GridBench benchmark results are stored in a XML database, along with the benchmark GBDL definition. Besides, additional monitored data observed during the benchmark's execution can be included in order to enrich the results and improve the analysis. Benchmark results from desired computing elements can be published in a monitoring and discovery service such as MDS (see section 2.2.2.2), making its access easy to users and grid brokering systems.

Arithmetic Data Cube [FS03] is an application of Data Cube Operator in an arithmetic data set. Data Cube Operator [GCB+97] is a tool of On-Line Analytical Processing[5] that processes views of a data set. ADC can be used as grid benchmark since it manages large distributed data sets. Furthermore, it is possible to control the intensity of the benchmark controlling the size of the views of the arithmetic data set.

The main disadvantage of grid benchmarks is that they run using grid resources but the resources

[4]NAS Parallel Benchmark is composed of a set of 8 modules that evaluate the performance of parallel supercomputers.

[5]On-Line Analytical Processing (OLAP) provides quick answers to analytical queries in a database. It allows users to discover patterns in a data set.

assignment can constantly vary, due to the natural dynamism of the system. Resources are dynamically selected according to an assignment policy. Thus, the benchmark results are only representative for the specific selected resources in its execution. The resources selected do not have to be the same in future benchmark executions and, even though the same resources were selected, its characteristics can change from one execution to the next (as they are non-dedicated). This makes difficult to predict grid performance taking benchmark results as a basis. The system variability and natural evolution can render the benchmark results obsolete very quickly.

2.2.2.2 Grid monitoring

The other basic approach to observe and measure grid performance is grid monitoring. This allows to obtain real-time behavior data, while the system is under production, with real resources, services and users. Grid monitoring can be performed at different levels, such as:

- Specific of the application.
- Node or server level.
- Cluster or site level.
- Grid level.

The basic idea behind grid monitoring is just to observe and gather information. These data can subsequently be studied using performance analysis and behavior modeling tools.

Several working groups within the Open Grid Forum, such as the Grid Monitoring Architecture Working Group [GPW], have worked to build a monitoring architecture designed for the specific components that characterize every grid platform. This architecture has been named GMA (Grid Monitoring Architecture), and its basic structure can be seen in Figure 2.3. GMA defines *producer*, *consumer* and *registry* elements. The producer registers its monitoring skill regarding a part of the grid (set of resources, specific application, etc.) in the directory service (registry). The consumer uses this directory service contained in the registry to locate those producers that provide the specific monitoring information it requires. The producer is selected according to brokering policies. Finally, the consumer communicates with the selected producer to obtain the necessary information.

Different architectures, like R-GMA [BCC+02], MDS [mds], Hawkeye [Haw] and NWS [nws], based on the GMA generic architecture have been implemented.

Relational Grid Monitoring Architecture (R-GMA) [BCC+02, GMA] implements the GMA architecture model defined by the Global Grid Forum by means of Java servlets and relational databases.

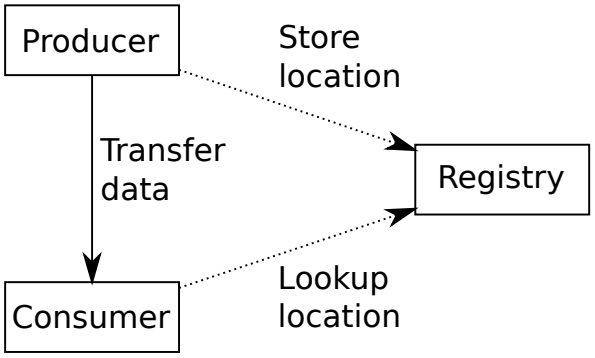


Figure 2.3: Grid Monitoring Architecture (GMA) [GMA]

This makes transparent the details of the producer/consumer model to the user. It is used for searching grid services and applications monitoring.

Monitoring and Discovery System (MDS) [mds] is the Globus Toolkit's information services component. It uses an extensible framework with a structure arranged in a hierarchical order to manage static and dynamical information. It provides an information services architecture offering mechanisms of resource discovery and monitoring.

MDS provides information about available resources in the grid and its state. It is often used in order to publish some results obtained by means of benchmarks in a certain grid. In spite of being a monitoring service, it is more used as discovery service, because it does not provide an historical archive showing how system has evolved. Furthermore, it can be combined with other grid protocols in order to build high-level services, like brokering and fault tolerance.

Hawkeye [Haw] is the Condor-G scheduler monitoring system. It combines the Globus's resource management with local management Condor methods. Hawkeye provides mechanisms for job monitoring, notification, etc. The main advantage of Hawkeye is that it is designed to automatically detect security and dependability issues and other possible problems. It is based on Condor's ClassAd technology, which identifies and characterizes the resources available in a set of machines. ClassAds are based on <attribute,value> pairs that are used to automatically determine if a resource fulfill the requirements of a certain application.

The Network Weather Service (NWS) [nws] was created by the University of California. It is a distributed system that monitors periodically the system and forecasts the performance of the computing resources and network load at any given time.

MonALISA (Monitoring Agents using a Large Integrated Services Architecture) [NLB01, NLG+03, mal] provides a distributed service to monitor, control and optimize complex systems. The combination of a service based architecture with the utilization of mobile agents facilitates the creation of a services hierarchy that can be easily expanded to manage very complex systems, like grids. The scalability of the system is achieved thanks to an engine that runs several dynamic services. These services can be discovered and used by other users or services that need the information they provide. Thus, there are two kind of services or agents: *data collector agents* (monitor service) and *decision making agents* (aggregation service).

The monitoring system performs real-time supervision of grid elements, network and running processes. Collected information is essential when high-level services are developed, especially when it is needed to make decisions regarding grid performance optimization. Different monitoring tools can integrated in MonALISA as low-level monitoring modules to collect information about grid resources.

GMonE (Grid Monitoring Environment) [Sán08] is a monitoring framework for large-scale distributed systems based on the *publish-subscribe* paradigm. GMonE runs a process called *resource monitor* on every grid node to be monitored. Each such node publishes monitoring information to one or more *monitoring archives* at regular time intervals. These monitoring archives act as the subscribers and gather the monitoring information in a database, constructing a historical record of the system's evolution. The *resource monitors* can be customized with *monitoring plugins*, which can be used to adapt the monitoring process to a specific scenario by selecting relevant monitoring information.

2.3 Cloud computing

After the development of the grid in the last 15 years, Cloud computing emerged in 2006 as a new distributed computing paradigm. Cloud computing [Wei07] promises to provide reliable services, delivered through next-generation data centers, and built over virtualized compute and storage technologies. The idea is to make possible for users (also called *consumers*, as Cloud computing is strongly market-oriented) to access applications and data from a *cloud*, anywhere in the world and on demand. The consumers are assured that the Cloud infrastructure is sufficiently robust, guaranteeing availability at any time.

As can be seen, this generic concept of Cloud computing, if analyzed carefully, incorporates ele-

ments such as highly reliable, scalable and autonomic services, ubiquitous access, dynamic discovery, etc. In particular, Cloud computing presents itself as a very market-oriented infrastructure, where services are exploited from an economic perspective and, therefore, quality of service becomes a matter of capital importance.

From a scientific point of view, there have been several attempts to define Cloud computing [Clo], and the subject is still a matter of discussion at some levels. Nevertheless, most part of the scientific community seems to agree in an intuitive idea of what Cloud computing is, and what could be expected of it. A possible definition of this new distributed paradigm could be (from [FZRL09]):

Cloud computing is a large-scale distributed paradigm that is driven by economies of scale, in which a pool of abstracted, virtualized, dynamically-scalable, managed computing power, storage, platforms and services are delivered on demand to external customers over the Internet .

There are several key points in this definitions. First, Cloud computing is a specialized distributed computing paradigm. It differs from other distributed systems in that *i*) it is massively scalable, *ii*) can be encapsulated as an abstract entity that delivers different levels of services to customers outside the Cloud, *iii*) it is driven by economies of scale [Sil87], and *iv*) the services can be dynamically configured (via virtualization or other approaches) and delivered on demand [FZRL09].

Based on current Cloud computing industrial and research projects, its four main characteristics can be distinguished:

- **Elasticity and scalability**. The cloud is elastic, meaning that resource allocation can get higher or lower on demand. Elasticity enables scalability, enabling the cloud to gracefully accept increasing computational demands.

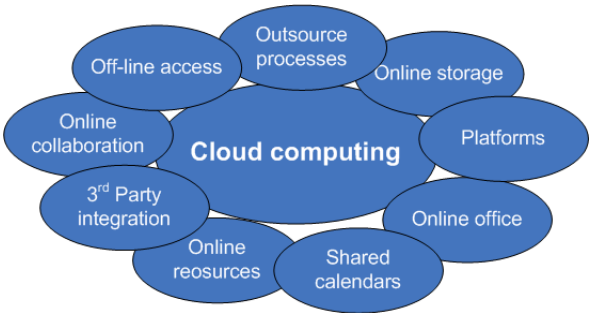


Figure 2.4: Cloud computing and some related ideas included in it.

- **Self-service provisioning**. Cloud customers can provision cloud services without going through a lengthy process. Users request an amount of computing, storage, software, process, or more from the service provider. After these resources are used, they can be automatically deprovisioned.

- **Standardized interfaces**. Cloud services should have standardized APIs, which provide instructions on how two application or data sources can communicate with each other. A standardized interface lets the customer more easily link cloud services together.

- **Billing and service usage metering**. Users can be billed for resources as they use them. This *pay-as-you-go* model means usage is monitored and users pay only for what they consume.

2.3.1 Cloud layers

The Cloud computing paradigm is organized in the following five distinguished layers:

1. **Client**: A cloud client is a computer hardware and/or computer software that relies on cloud computing for application delivery. Examples include some computers, phones and other devices, operating systems and browsers.

2. **Application**: Cloud application services or *Software as a Service* (**SaaS**) deliver software as a service over the Internet, eliminating the need to install and run the application on the customer's own computers and simplifying maintenance and support.

3. **Platform**: Cloud platform services or *Platform as a Service* (**PaaS**) deliver a computing platform and/or solution stack as a service, often consuming cloud infrastructure and sustaining cloud applications. It facilitates deployment of applications without the cost and complexity of buying and managing the underlying hardware and software layers.

4. **Infrastructure**: Cloud infrastructure services or *Infrastructure as a Service* (**IaaS**) delivers computer infrastructure, typically a platform virtualization environment, as a service. Rather than purchasing servers, software, data center space or network equipment, clients instead buy those resources as a fully outsourced service.

5. **Server**: The servers layer consists of computer hardware and/or computer software products that are specifically designed for the delivery of cloud services, including multi-core processors, cloud-specific operating systems and combined offerings.

2.3.2 Cloud computing projects

Since its conception, cloud computing has been a strongly market-oriented initiative, with wide acceptation in the industry and private sector. This has motivated many leading private companies (Amazon, Google, IBM, Microsoft, etc) to develop and offer cloud-based computing solutions. Simultaneously, cloud computing is being also developed within scientific research projects and open source initiatives.

The Amazon Elastic Compute Cloud (commonly known simply as Amazon EC2) [ec2] is probably the first well known Cloud computing initiative. As stated in Amazon's own web site, "EC2 is a web service that provides resizable compute capacity in the cloud. It is designed to make web-scale computing easier for developers". A user can create, launch, and terminate server instances as needed, paying by the hour for active servers, hence the term *elastic*. EC2 provides users with control over the geographical location of instances which allows for latency optimization and high levels of redundancy.

Google App Engine [app] is a platform for developing and hosting web applications in Google-managed data centers. It was first released as a beta version in April 2008. Google App Engine is a cloud computing technology. It virtualizes applications across multiple servers and data centers. Google App Engine is a free service, up to a certain level of used resources. Fees are charged for additional storage, bandwidth, or CPU cycles required by the application.

Microsoft's Azure Services Platform [azu] is an application platform in the cloud that allows applications to be hosted and run at Microsoft datacenters. It provides a *cloud operating system* called Windows Azure that serves as a runtime for the applications and provides a set of services that allows development, management and hosting of applications off-premises. All Azure Services and applications built using them run on top of Windows Azure.

abiCloud [abi] is an open source Cloud computing platform manager that allows to easily deploy a private cloud infrastructure. One of its key features is a rich Web interface design to make easy the system management tasks. In abiCloud users can deploy a new service just dragging and dropping a virtual machine. It allows to deploy instances of general purpose virtual machines such as VirtualBox, VMware, KVM, and Xen. It features user management through ACL, infrastructure and network management, an appliance repository, and the ability to easily design virtual data centers.

OpenNebula [ope, SMLF08] is an open-source toolkit to easily build cloud infrastructures of three types: private, public and/or hybrid. OpenNebula has been designed to be integrated with any

networking and storage solution and so to fit into any existing data center. OpenNebula orchestrates storage, network and virtualization technologies to enable the dynamic placement of multi-tier services (groups of interconnected virtual machines) on distributed infrastructures, combining both data center resources and remote cloud resources, according to allocation policies.

Nimbus [KF08] is a set of open source tools that together provide an "Infrastructure-as-a-Service" (IaaS) cloud computing solution. Its developer's objective is to evolve the infrastructure with emphasis on the needs of science, but many non-scientific use cases can be supported as well. Nimbus allows clients to lease remote resources by deploying virtual machines (VMs) on those resources and configuring them to represent an environment desired by the user.

RESERVOIR (REsources and SERvices Virtualisation withOut barrIeRs) [RBL$^+$09] is a Cloud computing research project partly funded by the European Commission as an Integrated Project under the Seventh Framework Programme (FP7) sponsorship program. According to the project description, their goal is "to support the emergence of Service-Oriented Computing as a new computing paradigm. In this paradigm, services are software components exposed through network-accessible, platform and language independent interfaces, which enable the composition of complex distributed applications out of loosely coupled components."

2.3.3 Cloud computing and the grid

A crucial question that was raised by many voices shortly after the cloud became an established paradigm was if there was really something new in Cloud computing and, more specifically, which were the differences between it and the previously existing concept of grid computing. On the one hand, a quick comparison shows many similarities between both initiatives, something that from the beginning led to some people to suggest that the cloud was nothing but the grid, simply presented from a new, market-oriented, perspective. Other voices, on the other hand, claimed that, although grids and clouds are both large scale distributed initiatives and therefore share many basic characteristics, cloud computing introduces several key aspects, creating a whole new paradigm. It is important to indicate that not only grid and cloud, but other similar large scale distributed computing concepts such as utility computing and Internet computing are also involved in this debate.

So, is Cloud computing just another name for grid? Evidently that question does not have a straightforward answer, as many aspects have to be considered. Both paradigms share a common vision: "to reduce the cost of computing, increase reliability and increase flexibility by transforming computers from something that we buy and operate ourselves to something that it is operated by a third party" [FZRL09]. However, there are important differences, especially when considering archi-

tectural aspects. Grid computing, on the one hand, is about independent institutions all around the world sharing resources in an harmonious way, and therefore it deals not only with large scale distribution and scalability issues but incorporates elements regarding security, trust between partners, decentralized control and so on. Cloud computing on the other hand, bases its infrastructure on large scale distributed data centers, in most cases centrally managed by an unique company (like Amazon in the case of EC2 or Google in the case of the Google App Engine). These centralized computing resources are nevertheless of a very large scale, and therefore its managers are faced with scalability, distribution and other related issues, as in the case of the grid. However, this more controlled (to some extent) environment allows cloud service providers to develop more reliable low-level infrastructures and therefore focus on high-level service related issues and its market-oriented model.

Nevertheless, most grid and cloud problems are still the same. Both need to be able to manage large scale (yet somehow different) facilities. They both need to define methods by which users/consumers discover, request and use resources provided by the system. Additionally, they both need to provide the users/consumers with the necessary mechanisms to develop the often highly parallel computations that execute on those resources.

The high expectations created by the grid community at its conception led over the years to disappointment and criticism, when many realized that the required efforts to create the grid were too great. In this sense, Cloud computing could be regarded as a compromise, where some of the technologically more challenging problems that grid computing created are eliminated by basing the cloud on a simpler infrastructure. This apparent simplification has led to the creation of new interesting platforms that provide grid-like services (specially if we remember the *electric power grid* analogy [KF98]), with very promising scientific but most of all commercial results. But in a world where everything in computer science is moving towards a totally distributed, shared model (the popularity of, for instance, the Wikipedia project and the peer-to-peer systems are *living proofs* of this trend) it does not seem logical to expect that data center based, centralized structures such as most current commercial clouds (Amazon EC2, Google App Engine, OpenNebula, etc.) should not in time evolve into a more widely distributed, resource-sharing approach.

Chapter 3

Autonomic computing

With the development of the grid and other large scale distributed initiatives, computing systems have made considerable advancements in the last two decades. Along with this advance, software tools and services become more and more sophisticated, causing a continuous increase of complexity in information technologies.

This incessant complexity growth, the introduction of different system standards, and the existence of new distributed and heterogeneous infrastructures make system management an extremely complicated task. In the end, it could lead to a evident loose of performance due to the difficulty to know and handle all the features of the infrastructure.

In addition, it is necessary to take into account the physical and logical continuous growth of the system. For instance, the number of users that access a server can change in an unpredictable way. The human participation and/or control of the system's managing becomes a very costly task. Additionally, sometimes it is not possible (or extremely complicated) to monitor the whole system in order to manually control the whole system and adapt its services and applications to its variable characteristics.

At the beginning of the year 2002, aiming at finding ways to handle this growing complexity, the idea of *autonomic computing* was conceived [IBM06, RAC]. The concept was created by IBM as a way to enable infrastructures to automatically adapt themselves to the demands of the applications running on them, and therefore increasing the system's performance. Other related previous

proposals, based on similar ideas, were self-healing technology, adaptive architecture, introspective computing and holistic computing.

The idea of self-healing technology [HL03] was created in 2001 by Intel, Shindy and F5 to designate the intelligent software responsible of warning in case of applications running near to the maximum capacity. There is a similar software offered by Concord Communications that automatically fits the network connections according to the current characteristics of the network.

Adaptive infrastructure [Hoo05] is the HP's contribution to the construction of self-managing infrastructures and autonomic systems. This approach has been implemented in the HP Utility Data Center managing the whole pool of resources of a data center in a dynamic way.

The idea of introspective computing [Isa02] was conceived as a dynamical adjustment of algorithms that run in a system from a previous analysis of it. Different enhancements can help to this adaptation, such as power saving or fault tolerance.

Holistic computing was the IBM's previous idea, just before autonomic computing. The aim of this sort of computation was to check and control database systems, trying to get a better efficiency.

Autonomic computing unites all these previous concepts and approaches in a single one: system self-adaptation in changing, heterogeneous environments, incorporating learning skills. Autonomic computing was inspired by biological systems, more specifically on the human central nervous system. This system performs multiple tasks, but human beings are not conscious of all of them and, more important, do not take part in its decisions. Some of the most important tasks performed are blood pressure reviews, control of the cardiac frequency, adjustment of the body's temperature, management of the food digestion and so on. All these tasks are carried out in an unconscious, autonomous fashion. In fact, the central nervous system is able to attend to all these tasks and other more important ones concurrently and with a high level of dependability. Moreover, it is also capable of adapting itself to the changing corporal needs. Following these inspirations, autonomic computing aims at the construction of a computing system that works in a similar way to the central nervous system: transparent, reliable and adaptable self-management.

Autonomic computing refers to systems that are capable of self-managing according to changes occurred in the environment, and they can heal and protect themselves in case of failure or attack. This kind of computation requires much less human intervention regarding the system's management and configuration. Nowadays, the biggest problem of most large scale systems is to define its ideal

configuration parameters in order to maximize aspects such as performance, dependability and/or quality of service. System administrators and/or application programmers are usually in charge of deciding the value of these parameters. In many cases, these values are not easy to define, since they depend on the knowledge that the administrator has about how the system works. Besides, in order to obtain maximum performance any given application, if running on different infrastructures, may need a drastically different configuration. Even more, the system characteristics can be dynamic, requiring for the configuration to be modified depending on system conditions.

As a summary, autonomic computing attempts to provide a system with the necessary capabilities to self-manage, automatically adapting to changing conditions in the environment, continuously optimizing its configuration parameters and repairing itself in case of failure. All this is made possible by these newly acquired autonomic skills, and without any human control or intervention. Therefore, autonomic computing requires the design of systems that are capable of adjusting themselves according to the changing conditions of the environment and manage its resources in an efficient way, facing different, unpredictable workloads.

3.1 Autonomic elements

In [Hor01] it is emphasized the urgency of "...design and build computing systems capable of running themselves, adjusting to varying circumstances, and preparing their resources to handle most efficiently the workloads we put upon them. These autonomic system must anticipate needs and allow users to concentrate on what they want to accomplish rather than figuring how to rig the computing systems to get them there ..."

In order to focus on this idea, it is important to understand the nature of autonomic computing. In [Hor01], IBM, one of the most active supporters of autonomic computing, defines the following eight key elements of this discipline:

1. "To be autonomic, a computing system needs to 'know itself' - and comprise components that also possess a system identity". An autonomic computing system is aware of all its components and their status and it must able to act on them.

2. "An autonomic computing system must configure and reconfigure itself under varying and unpredictable conditions". The environment in which an autonomic computing system works is dynamic, and according to these dynamic conditions, the autonomic computing system must be able to reconfigure itself. Although the conditions are unpredictable, it is possible and desirable

to use a system that can predict, in some sense, the future behavior. In this way, the configuration makes feasible the performance enhancement. Besides the system adjustment must be carried out in a constant way. Like this the system must adapt itself to the environment in the best possible way at any time.

3. "An autonomic computing system never settles for the status quo - it always looks for ways to optimize its workings". An autonomic computing system monitors the overall status of the system and decides, according to an optimization plan, the parameters to be changed.

4. "An autonomic computing system must perform something skin to healing - it must be able to recover from routine and extraordinary events that might cause some of its parts to malfunction". For doing it, it can make use of the available resources without interrupting the activities being performed by them. In case of failures that can not be handled by the system, like a hardware failure, it must present the administrator with the proper warning. In short, an important feature of an autonomic computing is its ability for self-healing: a system must be able to identify the problems causes and, if possible, solve them.

5. "A virtual world is no less dangerous that the physical one, so an autonomic computing system must be an expert in self-protection". An autonomic computing system must protect itself from attacks, detecting them and alerting system administrator in case of danger. In the same way the human immune system works, an autonomic system must be able to identify suspicious code, analyze it and distribute a *cure* for the whole system.

6. "An autonomic computing system knows its environment and the context surroundings its activity, and acts accordingly". An autonomic computing system must be able to discover resources and obtain information about them. Furthermore, according to the information of its neighbors, the system makes decisions.

7. "An autonomic computing system cannot exist in a hermetic environment". An autonomic computing system interacts in an open and heterogeneous environment with other elements by means of open standards. This feature is especially compatible with the grid philosophy (see Section 2.2).

8. "Perhaps most critical for the user, an autonomic computing system will anticipate the optimized resources needed while keeping its complexity hidden". An autonomic computing system must be able to act in advance and in a optimized fashion in order to maintain and increase the system performance. This ability must be transparent to the user. The final aim is clear: users have to achieve their aims without worrying about the system operation and its implementation.

Figure 3.1: Autonomic computing principles [RAC]

In order to achieve all these features, four generic principles (see Figure 3.1) are embedded into any autonomic computing strategy [IBM06, KC03], namely:

1. **Self-configuration**, that is, the ability for configuring itself according to high-level policies. The components are adapted dynamically to the system changes. The aim is the flexible adjustment to the environment, being able to face diverse workload and variable resources.

2. **Self-optimization**, that is, the capability to seek ways to control and enhance performance. The system monitors all available resources and it makes decisions to optimize the system operation according to the information obtained.

3. **Self-healing**, that is, the feature that allows the system to detect, diagnose and repair hardware and software problems. This skill means discovering, diagnosing and reacting when faced with system failures, according to the policies indicated by the administrator.

4. **Self-protection**, that is, the ability for preventing the system against possible attacks. In fact, it must detect them and act as necessary.

3.2 Autonomic levels

To measure the autonomic level of a given computing system, a scale from manual to autonomic it is defined in [RAC] (see Figure 3.2). Each level incorporates new features that replace certain areas of human intervention and decision. It is important to emphasize that complexity is present at all system levels, hardware, software and management [GC03].

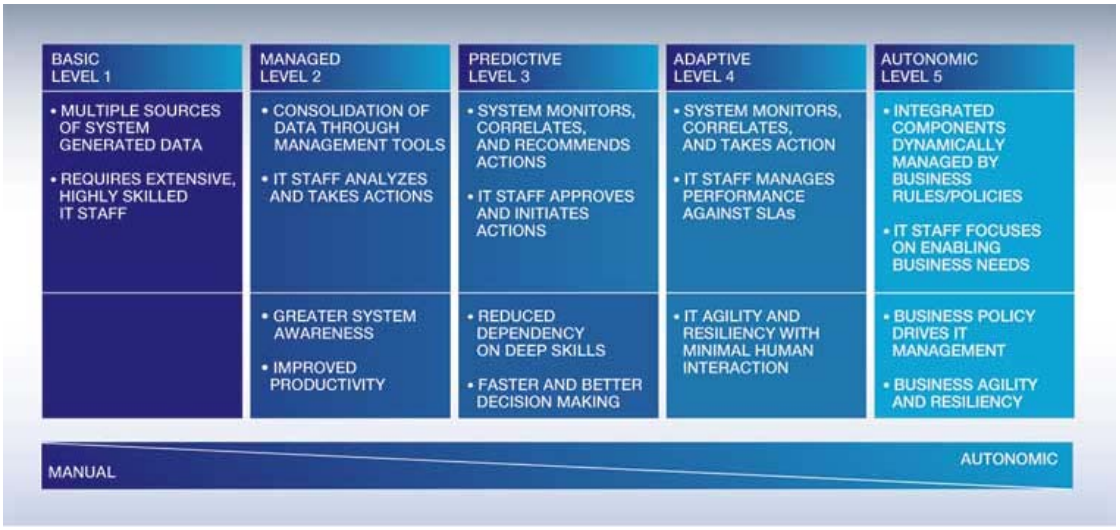


Figure 3.2: Maturity levels of autonomic computing [RAC]

1. The most basic level requires of the administrator to configure, supervise and keep manual control over the system.

2. The second level or *managed level* takes advantage of personal tools to analyze system components, using the results for making decisions.

3. The third level or *predictive level* adds new structures to the supervision tools. Now, the system can suggest recommendations being the administrator responsible of approving and making the needed actions.

4. In the fourth level or *adaptive level*, the system is taking more responsibilities about decision making. The administrator is in charge of setting the policies that the system must obey.

5. The *autonomic level* is the fifth level. The system must take into account the high-level policies adapting itself in the specified way to the environment.

Many autonomic projects have appeared in the last decade, as first attempts to test some of the characteristics of autonomic computing. Developing complete autonomic solutions is, however, an extremely complicated task, with many different areas of computer science involved. The involvement of powerful entities (leading private companies and important research institutions) is required in order to fully develop the autonomic computing potential. Nowadays, IBM [IBM03], HP and Microsoft are strongly involved in the development of this technology, creating new products for all levels of the system.

3.3 Autonomic projects

In April 2003, IBM, widely considered as the father of autonomic computing, announced the first project to develop new management tools in order to design autonomic systems capable of control complex computing environments [GC03].

To extend the autonomic computing approach within the industry and scientific community, it is indispensable to develop basic tools and common standards. In its seminal project, IBM proposes a set of technical guides that should be implemented in order to make the joint work possible with efficiency. The easiest way of reaching this goal is by using open specifications. The IBM project developed a common terminology for autonomic systems. Furthermore, it tried to use common standards in other fields, such as Open Grid Services Architecture (OGSA) in grid computing (see Section 2.2), in order to define the autonomic management of complex systems.

Besides this project, IBM has developed some tools in order to help in the creation of autonomic systems, namely:

- Log & Trace Tool is designed to facilitate the understanding of failure reasons in the system. It translates data from different system components to a common format that is accessible in a subsequent monitoring phase. The tool helps the administrator to identify the cause of a given problem. In addition, it can make easy the development of self-healing skills.

- Agent Building and Learning Environment (ABLE) is an autonomic technology based on Java for the creation of intelligent agents. Due to the adaptation of agents to heterogeneous environments, ABLE is very useful to develop autonomic system features.

- Autonomic Monitoring Engine is designed to reduce the complexity of heterogeneous infrastructures. It can detect failures and potential problems before they affect the system behavior. It has self-protecting skills making possible for the system to automatically solve critical situations.

- Business Workload Management is a tool that helps to avoid bottlenecks by measuring different parameters like reaction times. Later, it adjusts the resources configuration to achieve its goals.

Nevertheless, although there are tools to develop autonomic systems, the techniques applied to provide autonomic capabilities are not standard and they depend on the domain and the tackled problem.

Nowadays, there are several initiatives and projects with the aim of providing autonomic skills and testing autonomic concepts. IBM and Sun are two of the most active companies that support the autonomic development with some of their projects, such as z/Os, WebSphere, DB2, Tivoli and N1. Meanwhile, different universities and researching centers are using the autonomic concepts to built its own infrastructures, like VIOLIN [JX04, RMX05], Autonomic Virtualized Environments [MB06], Self-* Storage system [GSK03], Kendra [McC03] and Application Performance Prediction and Autonomic Computing [MK04]. More autonomic projects are listed in [ACo].

3.3.1 VIOLIN

Since current network infrastructures are slowly adapted to changes, virtual networks can be designed as service-oriented added value networks. These networks provide efficiency and flexibility, although they have a hard use because the accumulation of both network and service functions. VIOLIN [JX04, RMX05] is a project of the Purdue University published in 2004 that proposes a Virtual Internet working on OverLay INfrastructure (VIOLIN) to solve the overload of the application level and its adaptation to dynamic changes.

VIOLIN creates virtual networks that operate as an overlay of real infrastructures, such as PlanetLab [pla]. They are composed of software-based virtual routers, switches and end-hosts. Its main characteristics are:

1. Each VIOLIN network is a virtual world. Thus, its communications are limited to this network.

2. All network elements can be created and deleted on demand, being automatically adapted to the corresponding circumstance.

3.3.2 Autonomic virtualized environments

According to the autonomic computing group of the George Mason University, autonomic techniques can be applied to decide which computing resources must be allocated to any virtual machines as the system workload changes [MB06].

The objective of this project is to find an optimum usefulness function that helps to make these assignments in virtual environments, being CPU resources considered as the main value to share. It proposes two ways of modeling the problem. First, all tasks belonging to the same virtual machine have the same priority. Thus, the scheduler only takes into account the different priorities among virtual machines. On the other hand, the system can be modeled by means of virtual CPUs, allocating a virtual CPU to each virtual machine. The assignment is not carried out by means of priorities

but fitting the time that every virtual CPU can run in the real CPU. Both models are valid, though the second one obtains improved performance taking advantage of the available CPU time in a better way.

Nowadays this is a very important research line in distributed systems, mainly because virtualized environments have become one of the key aspects of grid computing and specially Cloud computing.

3.3.3 Self-* storage system

Self-* storage system [GSK03] is a project developed in the Carnegie Mellon University. It designs a cluster-oriented storage file system that provides self-organizing, self-managing, self-healing and self-configuring capabilities. It is based on both artificial intelligence and system control. The low cost of cluster components and resources quality benefits the system reliability and availability.

Self-* storage system is in charge of storing files in the corresponding cluster node. Files are split in chunks storing it in an adapted way to the operation of every *storage brick*. This adaptation is mainly directed to obtain fault-tolerance, although it is possible to define other goals.

3.3.4 Kendra

Kendra [McC03] is based on the idea that metadata allows Internet searches to be more efficient and provides useful information for data distribution and decision systems.

Kendra's decision making module supervises the Internet operation and makes decisions trying to optimize the information delivery according to the available resources at each moment. In order to facilitate its adaptability, a set of adaptable components can be run on demand. These components enable optimistic and pessimistic configurations. This project was tested with audio servers creating the Kendra initiative to promote a distribution market of open digital contents.

3.3.5 Application performance prediction and autonomic computing

This project [MK04] from Clemson University refers the use of prediction as an integral part of an autonomic system that monitors, analyzes and controls. A self-managed system can use predictions to develop an analysis of the system to determine the ideal resource configuration dynamically.

In order to prove this approach, they predicted a web application performance taking into account an end-to-end model for each client-server including the network influence. The prediction was based on an stochastic model about the operation of the TCP protocol. The inputs were the sending time and the lost time by the TCP protocol and the output was the performance obtained by means of the

throughput achieved by a TCP flow in steady state. Finally, they built a model that described the application behavior according to the predicted workload of network and server.

3.3.6 z/OS

z/OS is the IBM eServer zSeries's operating system, and it is the successor of the IBM mainframe operating system OS/390. z/OS incorporates autonomic computing features. First of all, it uses *msys*, a system management infrastructure that offers self-configuring skills. *msys* significantly simplifies the task management for software setup. Besides *msys* simplifies the daily operations of a parallel system z/OS. This cause that the operation complexity is reduced, minimizing the number of errors and the workload.

The self-optimizing skills provided by the Workload Manager (WLM) and the Intelligent Resource Director (IRD) allows the system to manage unforeseeable workloads with not much human intervention.

3.3.6.1 WebSphere Application Server

WebSphere Application Server (WAS) makes autonomic computing skills possible to use both for z/OS and for multi-platforms. Thus, it can be used in diverse work environments, such as grid computing and Web services. In general, WAS can be seen as a Web service compatible with Java that provides self-management.

3.3.6.2 DB2 Universal Database

DB2 for z/OS is constructed to be a suitable e-business infrastructure. In this environment, it is necessary to provide database self-management capabilities. DB2 for z/OS can take advantage of the autonomic features of the own operating system. *DB2* provides the needed functionality to simplify the database parameters, and therefore it makes easier the self-configuring procedures.

Besides an improvement based on the search of minimizing the cost of request in the database is used. The system tests the performance of different operations, comparing different factors such as table accesses using different techniques. The least costly operation is used in the following request.

To improve the requests, Materialized Query Tables (MQT) are also used. They accelerate the process storing a summary of the contained data. The system can automatically reformulate the request to use the information summarized in the MQT. Using the summarized information it is possible to improve requests.

3.3.7 Tivoli

The software of system management is the central point of many functions of autonomic computing, especially due to the need to supervise conditions inside systems. Tivoli indicates the steps that are necessary to give to make progress in the different levels of an autonomic system.

Tivoli tries to reduce the complexity and provides the self-management capacity. For doing it there are some different programs based on Tivoli:

- Tivoli Identity Manager is a self-protecting software. It gives automatically the users' rights to access to resources, increasing the security.
- Tivoli Privacy Manager makes automatic the use of policies. It offers self-protecting according to high-level policies.
- Tivoli Storage Resource Manager tries to manage the data storage in an efficient way. It reduces the total cost of the data storage.
- Tivoli Service Level Advisor stores data in order to make data predictions, providing self-optimization features. It manages resources taking into account their availability and System Level Agreements (SLA).
- Tivoli Configuration Manager can make the process of installing and distributing software in an autonomic way.
- Tivoli Business Systems Manager simplifies the management of critical systems managing problems in real time according to high-level policies.

3.3.8 Sun's N1

N1 is a long term vision and an architectural model to reduce the complexity of the data centers. Thanks to N1 the systems can manage its own complexity. N1 includes hardware, storage and different kinds of services. Thus, N1 is an integrated system, making difficult its installation on any device.

The idea of N1 is unifying the resources, both storage and computing resources, in a similar way the grid computing works. The platform makes possible to turn a workcenter into a single system.

N1 has three functionality targets:

1. Infrastructure provisioning.

2. Application level provisioning.

3. Service level automation.

Besides Sun is working in a dynamic resource provisioning.

All these ideas have been implemented in the following software programs:

- Sun N1 System Manager: it simplifies the development cycle of an infrastructure for the servers Sun Fire x64 and SPARC.
- Sun Management Center: It reduces the support cost and provides monitoring and self-management of established systems.
- Sun N1 Grid Engine: it distributes the workload of the applications that need more resources in an grid environment taking control of the used resources and providing the necessary security.

These programs reduce the complexity of new computing infrastructures.

Chapter 4

Statistical and knowledge discovery techniques

Autonomic management should be based on deep knowledge of the system's behavior. Understanding this behavior means to carefully studying it, observing and analyzing its evolution. During this process, the information obtained must be processed through several phases, aimed at extracting useful, non-trivial knowledge that throws new light over issues such as performance, dependability, security, etc. One of the most common ways of developing these phases is the use of statistical and data mining tools and methodologies. These techniques allow us to handle the large volumes of data that observation of a large scale system such as a grid can generate, identifying underlying knowledge and behavior patterns.

Modeling and characterizing the behavior of large scale distributed systems by means of statistical and knowledge discovery techniques have been approached in several other contexts. The most basic approach is benchmarking [DG93], which enables analysis of the system behavior under different workloads. Other approaches describe the system formally using Colored Petri Nets (CPN) [BvdAST08] or Abstract State Machines (ASM) [Gur91] in order to reason about behavior.

Rood and Lewis [RL07, RL08a, RL08b] propose a multi-state model and several analysis techniques in order to forecast the resources availability, aiming at improving scheduler efficiency.

Li et al. [LGW07] present an Instance Based Learning technique to forecast response times of

jobs in large scale systems by means of historical performance data mining. In a similar way, Smith et al. [SFT04] analyze the run times of parallel applications from past executions of similar applications. Cho et al. [CKL08] describe a user demand approach, which employs historical user demands in order to efficiently manage system resources. All these contributions are focused on user jobs or user demands.

Barham et al. [BDIM04] propose Magpie, a toolchain for automatically extracting a system's workload under realistic operating conditions. Magpie is based on low-overhead instrumentation, incorporated to monitor the system and record fine-grained events generated by kernel, middleware and application components. In a more machine learning based approach, Cohen et al. [CZG+05] present a method for automatically extracting from running systems an indexable signature that distills the essential characteristics of the system state. Finally, in the same line Pan et al. [PKT+09] propose a tool for black-box diagnosis of MapReduce systems, aimed at discovering problems and bottlenecks.

As can be seen, current state of the art in these matters shows that many different approaches can be taken regarding this issue. In this chapter, detailed descriptions of the most relevant statistical and knowledge discovery techniques that can be applied to grid behavior modeling are presented.

4.1 Information representation and attribute analysis

In order to achieve a successful data analysis, information should previously be arranged and correctly represented. Selecting a correct, meaningful representation format is a key first step in any knowledge extraction process, providing useful insight on aspects such as data structure, information variability, parameter dependences, and so on. The way information is represented influences all the subsequent process, leading, if correctly performed, to successful results. It could be said that the main objective of information representation is to format the data in a way that relevant facts can be more clearly distinguished, hiding or eliminating irrelevant information that could lead to imprecision in further analysis. However, it can contribute in many other aspects, such as providing the ideal information structure to improve the automated mathematical analysis performance, even reducing algorithms complexity in some cases.

4.1.1 Principal Component Analysis (PCA)

Principal Component Analysis (PCA) [Pea01] is a mathematical procedure designed to transform a number of possibly correlated variables into a possibly smaller number of uncorrelated variables, called **principal components**. These components are generated in a specific order, making the first

one to account for as much of the variability in the data as possible. Following the given order, each succeeding component accounts for another part of the remaining variability, in decreasing sizes. The total sum of the amount of variability represented by all components is always 100%, guaranteeing no loss of information.

PCA is mostly used as a tool in exploratory data analysis and for making predictive models. PCA involves the calculation of the eigenvalue decomposition of a data covariance matrix or singular value decomposition of a data matrix, usually after mean centering the data for each attribute.

PCA is the simplest of the true eigenvector-based multivariate analyses. Often, its operation can be thought of as revealing the internal structure of the data in a way which best explains the variance in the data. If a multivariate dataset is visualized as a set of coordinates in a high-dimensional data space (1 axis per variable), PCA supplies the user with a lower-dimensional picture, a *shadow* of this object when viewed from its (in some sense) most informative viewpoint.

4.1.2 Virtual representation of information systems

The role of visualization techniques in the knowledge discovery process is well known. The increasing complexity of the data analysis procedures makes it more difficult for the user to extract useful information out of the results generated by the various techniques. This makes graphical representation directly appealing. Data and patterns should be considered in a broad sense. The increasing high rates of data generation emerging from the grid require the development of procedures facilitating the understanding of the structure of this kind of data rapidly, intuitively and integrated within a monitoring tool.

Virtual reality (VR) is a suitable paradigm for visual data mining. It is flexible: allows the choice of different ways how to represent the objects according to the differences in human perception. VR allows immersion: the user can navigate inside the data and interact with the objects in the world. One of the steps in the construction of a VR space for data representation is the transformation of the original set of attributes describing the objects under study, in the present case grid related events characterized by several monitored features, into another space of small dimension (typically 2-3) with intuitive metric (e.g. Euclidean). The operation usually involves a non-linear transformation; implying some information loss. There are basically three kinds of spaces sought: *i)* spaces preserving the structure of the objects as determined by the original set of attributes, *ii*) spaces preserving the distribution of an existing class defined over the set of objects and *iii)* spaces representing a trade-off between the previous two. Since in many cases the set of descriptor attributes does not necessarily relate well with the decision attribute, different types of spaces are usually conflicting. Moreover,

they may be created by different non-linear transformations.

A visual, data mining technique based on virtual reality oriented to general relational structures (information systems) was introduced by J. J. Valdés [Val02b, Val03]. It is oriented to the understanding of large heterogeneous, incomplete and imprecise data, as well as symbolic knowledge. Such a structure $U =< O, A >$ is given by a finite collection of objects O, described in terms of a finite collection of properties A (maybe large). These are described by the so called source sets, constructed according to the nature of the information to represent. Source sets also account for imprecise/incomplete information.

A *virtual reality space* VR is given by a finite collection of objects $\hat{O}$ with associated *i) geometries* representing the different objects and relations, *ii) behaviors* which the objects may exhibit in the world, *iii) location* in the VR space which typically is a subset $\Re^m$ of a low cardinality cartesian product of the reals $\mathbb{R}^m$ ($\Re^m \subset \mathbb{R}^m$ of dimension $m \in \{1, 2, 3\}$ and Euclidean metric) and *iv)* functions assigning geometries, behavior and location to the set of studied objects.

If the objects in U are in a heterogeneous space described by n properties, $\varphi : \hat{\mathcal{H}}^n \rightarrow \Re^m$ is the function mapping the objects O from U to those $\hat{O} \in VR$ (i.e. $\hat{o} = \varphi(o)$, where $o \in O$ and $\hat{o} \in \hat{O}$). Several desiderata can be considered for building a transformed space either for constructing visual representations or as new generated features for pattern recognition purposes. According to the the property that the objects in the VR space must satisfy, the mapping can be:

- *Unsupervised*: The location of the objects in the space should preserve some structural property of the data, dependent only on the set of descriptor attributes. Any class information is ignored. The space sought should have minimal distortion.
- *Supervised*: The goal is to produce a space where the objects are maximally discriminated w.r.t. a class distribution. The preservation of any structural property of the data is ignored, and the space can be distorted as much as required in order to maximize class discrimination.
- *Mixed*: A space compromising the two goals is sought. Some amount of distortion is allowed in order to exhibit class differentiation and the object distribution should retain in a degree the structural property defined by the descriptor attributes. Very often these two goals are conflicting.

From the point of view of their mathematical nature, the mappings can be:

- *Implicit*: the images of the transformed objects are computed directly and the algorithm does not provide a function representation.

- *Explicit*: the function performing the mapping is found by the procedure and the images of the objects are obtained by applying the function. Two sub-types are:
  - *analytical functions*: for example, as an algebraic representation.
  - *general function approximators*: for example, as neural networks, fuzzy systems, or others.

Explicit mappings can be constructed in the form of analytical functions (e.g. via genetic programming), or using general function approximators like neural networks or fuzzy systems. An explicit transform φ is useful for both practical and theoretical reasons. On the one hand, in dynamic data sets (e.g. systems being monitored or incremental data bases) an explicit transform φ will speed up the update of the VR information space. On the other hand, it can give semantics to the attributes of the VR space, thus acting as a general dimensionality reducer.

4.1.2.1 The unsupervised perspective: Structure preservation

Data structure is one of the most important elements to consider and this is the case when the location and adjacency relationships between the objects O in U should give an indication of the *similarity relationships* [CP81], [Bor87] between the objects in $\hat{\mathcal{H}}^n$, as given by the set of original attributes A [Val02a]. φ can be constructed to maximize some metric/non-metric structure preservation criteria as has been done for decades in multidimensional scaling [Kru64], [Bor87], or to minimize some error measure of information loss [Sam69]. If δ_{ij} is a dissimilarity measure between any two objects $i, j \in O$ coded by natural numbers ($i, j \in [1, N]$, where N is the number of objects), and $\zeta_{i^v j^v}$ is another dissimilarity measure defined on objects $i^v, j^v \in \hat{O}$ from VR ($i^v = \varphi(i), j^v = \varphi(j)$, examples of error measures frequently used are:

$$\text{S stress} = \sqrt{\frac{\sum_{i<j} (\delta_{ij}^2 - \zeta_{ij}^2)^2}{\sum_{i<j} \delta_{ij}^4}}, \tag{4.1}$$

$$\text{Sammon error} = \frac{1}{\sum_{i<j} \delta_{ij}} \sum_{i<j} \frac{(\delta_{ij} - \zeta_{ij})^2}{\delta_{ij}} \tag{4.2}$$

$$\text{Quadratic Loss} = \sum_{i<j} (\delta_{ij} - \zeta_{ij})^2 \tag{4.3}$$

Classical deterministic algorithms have been used for directly optimizing these measures, like Steepest descent, Conjugate gradient, Fletcher-Reeves, Powell, Levenberg-Marquardt, and others. Computational intelligence (CI) techniques like neural networks [JM92], evolution strategies, genetic algorithms, particle swarm optimization and hybrid deterministic-CI methods have been used as well [Val04, VB05].

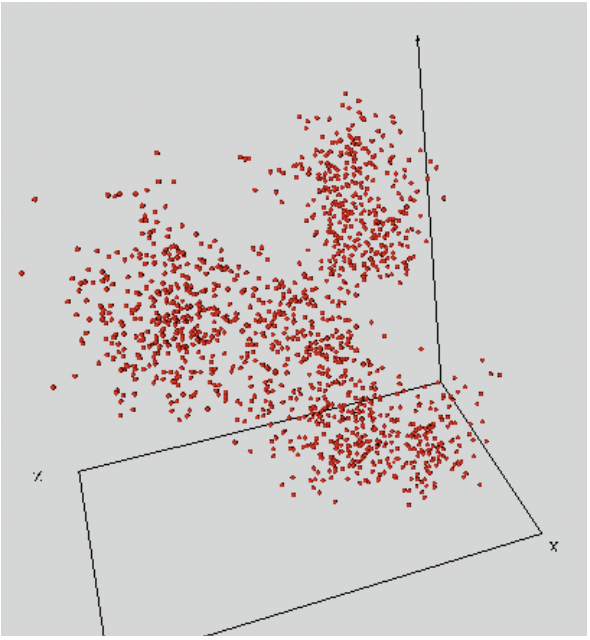


Figure 4.1: Example of tree-dimensional representation using unsupervised VR spaces

The number of different similarity, dissimilarity and distance functions definable for the different kinds of source sets is immense. Moreover, similarities and distances can be transformed into dissimilarities according to a wide variety of schemes, thus providing a rich framework.

4.2 Unsupervised classification: clustering

Cluster analysis [Try39] tries to divide a dataset, creating groups of individuals with certain similarities among them. Therefore the degree of association or similarity between two individuals would be high if they belong to the same group (*cluster*) and low otherwise. Clustering can be used in many different problems, from creating association of related products in a department store to identifying new neuron types while studying the mysteries of the human brain. In this sense, there are many scientific research areas where clustering can help to organize large datasets of individuals, such as stars, animal species or patient typologies.

Clustering techniques can be separated in divisive and agglomerative methods. Divisive methods, on the one hand, follow an iterative sequence, at the beginning of which all elements belong to the same cluster. The clustering method proceeds then dividing this initial cluster in smaller ones, until the final solution is achieved. Agglomerative techniques, on the other hand, begin creating one inde-

pendent cluster for each element, and gradually merging them. In both approaches, the suitable final number of cluster is selected using expert knowledge, in some cases automatically inferred by the clustering technique, in others provided as input by the user. Techniques such as Hierarchical Clustering, K-Means, Expectation-Maximization, Quality Threshold and DBSCAN can be distinguished among the most common clustering methods.

4.2.1 Hierarchical Clustering

Hierarchical Clustering is an agglomerative clustering technique. The algorithm starts joining the closest objects depending on some measure of distance and repeating the operation successively obtaining larger clusters. Furthermore, all objects are joined together in the last step. This creates a hierarchy of clusters, representing them as a tree, called dendogram (see Figure 4.2), with individual elements at one end and a single cluster containing every element at the other. Each level of the tree provides a different set of clusters, based on the maximum distance between points. The distance measure is the key factor in this method. The distance between two elements indicates the similarity between both and thus it is the factor for grouping elements. Several kinds of distance can be used, like the Euclidean distance and the Ward's method [War63]. The Euclidean distance is the geometric distance between two points in any space, whereas the Ward's method uses a variance analysis, minimizing the sum of squares between two clusters at each step. In order to select the final number of clusters is required to inspect the dendogram. This must made by an expert in the analyzed field who interprets the created groups. The reliance on this method depends on the expert and it must be cautiously used [AB84].

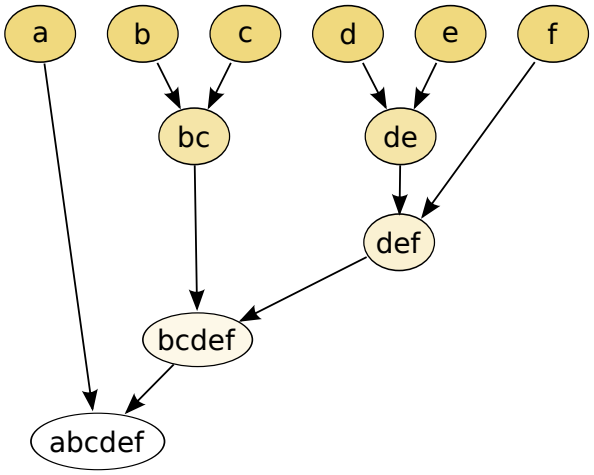


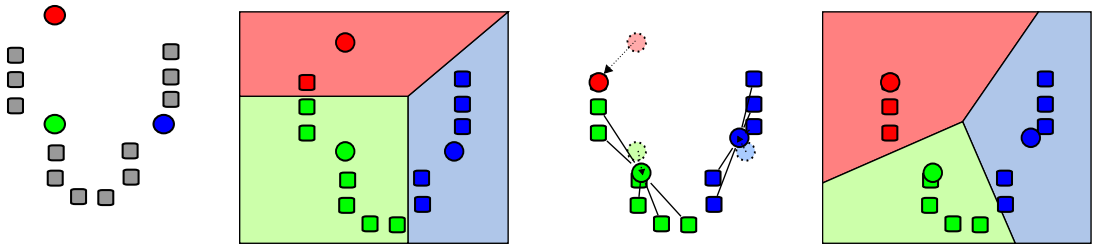
Figure 4.2: An example of dendogram [den]

4.2.2 K-Means

K-Means clustering [Mac67] aims at partitioning n observations into k clusters in which each observation belongs to the cluster with the nearest mean. The number k of clusters must be supplied by the user. Instead of constructing a dendogram, data is initially randomly assigned to the different clusters and them moved among them, reducing the dissimilarities and maximizing variability between clusters. As before mentioned, cluster variability is measured with respect to the means of the classifying variables. The algorithm uses an iterative refinement technique. Its basic phases are described bellow, and also shown as an example in Figure 4.3.

K-means steps:

- **Step 1:** k initial "means" are randomly selected from the data set. This is the initialization step (Figure 4.3(a)).
- **Step 2:** k clusters are created by associating every observation with the nearest mean. This is also called the **assignment step** (Figure 4.3(b)).
- **Step 3:** The centroid of each cluster is calculated, and it becomes the new mean. This is also called the **update step** (Figure 4.3(c)).
- Steps 2 and 3 are repeated until convergence has been reached (Figure 4.3(d)).



(a) Step 1 (b) Step 2 (c) Step 3 (d) Final result

Figure 4.3: K-Means example ($k = 3$) [kme]

4.2.3 Expectation-Maximization (EM)

The expectation-maximization (EM) algorithm [DLR77] is used for finding maximum likelihood. It estimates parameters in probabilistic models, where the model depends on unobserved latent variables. Used as a clustering technique, EM attempts to identify clusters by finding groups of individuals whose distances follow a given probability distribution (normally a Gauss distribution, but

not necessarily). EM is an iterative method which alternates between performing an **expectation** (E) phase and a **maximization** (M) phase. During the E phase the algorithm computes an expectation of the log likelihood with respect to the current estimate of the distribution for the latent variables. In the M phase the parameters which maximize the expected log likelihood found on the E step are computed. These parameters are then used to determine the distribution of the latent variables in the next E step. This algorithm presents some basic similarities with K-Means, but instead of calculating mean centroids EM estimates probability distribution parameters. Also, instead of defining cluster membership by distance (as K-Means does), it calculates it using likelihood of belonging to a given probability distribution.

4.2.4 Quality Threshold (QT)

Quality Threshold (QT) [HKY99] clustering is an algorithm that groups individuals into high quality clusters. It is based on distance between individuals and a specific definition of *clustering quality*. Quality is ensured by finding large cluster whose diameter does not exceed a given user-defined diameter threshold. This method prevents dissimilar individuals from being forced under the same cluster and ensures that only good quality clusters will be formed. It was originally designed for gene clustering.

The *QT* algorithm follows these steps:

- The user chooses a maximum diameter for clusters.
- The algorithm builds a candidate cluster for each point by including the closest point, the next closest, and so on, until the diameter of the cluster surpasses the threshold.
- Then it saves the candidate cluster with the most points as the first true cluster, and remove all points in the cluster from further consideration.
- The algorithm then recurses with the reduced set of points.

The distance between a point and a group of points is computed using complete linkage, i.e. as the maximum distance from the point to any member of the group.

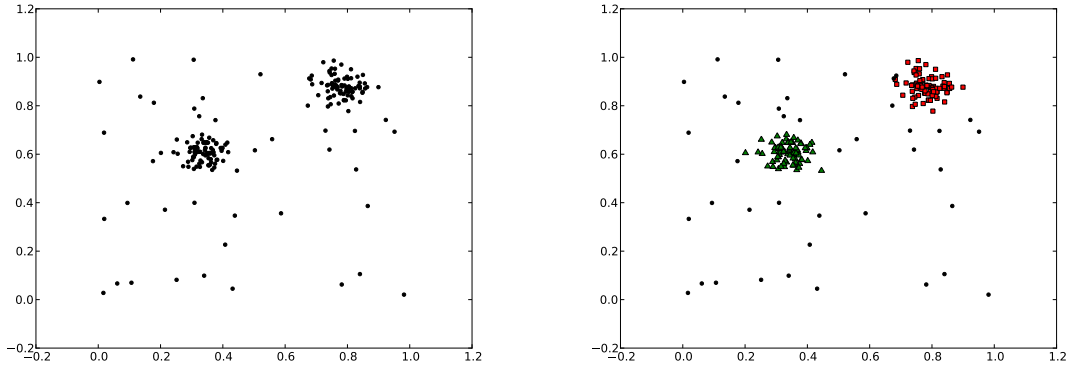
4.2.5 DBSCAN

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) [EKJX96] is a density based clustering algorithm. Density based clustering is based on the idea that the Euclidean space can be divided into sets of connected components. The implementation of this idea for partitioning of a finite set of points requires concepts of density, connectivity and boundary. They are closely related

to a point's nearest neighbors. A cluster, defined as a connected dense component, grows in any direction that density leads. Therefore, density-based algorithms are capable of discovering clusters of arbitrary shapes. Also this provides a natural protection against outliers.

DBSCAN requires two parameters: the distance ε and the minimum number of points required to form a cluster (minPts). It starts with an arbitrary starting point that has not been visited. This point's ε-neighborhood is retrieved, and if it contains sufficiently many points, a cluster is started. Otherwise, the point is labeled as *noise*. Note that this point might later be found in a sufficiently sized ε-environment of a different point and hence be made part of a cluster.

If a point is found to be part of a cluster, its ε-neighborhood is also part of that cluster. Hence, all points that are found within the ε-neighborhood are added, as is their own ε-neighborhood. This process continues until the cluster is completely found. Then, a new unvisited point is retrieved and processed, leading to the discovery of a further cluster or *noise*.



(a) An example of two-dimensional dataset.

(b) Dataset clustered using DBSCAN. Clusters distinguished by dot shape and color. Black dots are *noise*.

Figure 4.4: An example of density based clustering

4.2.6 Other clustering techniques

Other sophisticated clustering methods that can be used for data partition are Self-Organizing Map (SOM) [Koh90], GRIDCLUST [Eri93], or SINICC (Simulation of Near-optima for Internal Clustering Criteria) [BH90]. These techniques are not dicussed here because they are out fo the scope of this thesis.

4.3 Supervised classification

Statistical classification (normally referred simply as *classification* in the data mining field) is a supervised machine learning procedure in which individual items from a dataset are separated into different groups. This classification is based on quantitative information on one or more characteristics inherent in the individuals (normally referred to as traits, variables, characters, etc) and, which is more important and differentiated this approach from clustering, it is based on a training set of previously labeled items. The basic idea behind supervised classification is to mathematically extract a pattern for a set of previously classified individuals, enabling to automatically determine the class of a new one using that knowledge.

Classification algorithms performance always depends greatly on the characteristics of the data to be classified, and there is no single classifier that produces optimal results for any given problem (a phenomenon that may be explained by the No-free-lunch theorem [Wol96] [1]). Various empirical tests have been performed to compare classifier performance and to find the characteristics of data that determine classifier performance. However determining the most suitable classifier for a given problem still requires a thorough data analysis and a high level of expertise in the field.

The measures precision and recall [OD08] are popular metrics used to evaluate the quality of a classification system. More recently, receiver operating characteristic (ROC) curves [GS66] have been used to evaluate the trade-off between true and false-positive rates of classification algorithms.

4.3.1 Logistic regression

In statistics, logistic regression [HL00] (sometimes called the logistic model or logic model) is used for prediction of the probability of occurrence of an event by fitting data to a logistic curve. It is a generalized linear model used for binomial regression. Like many forms of regression analysis, it makes use of several predictor variables that may be either numerical or categorical. The aim of a regression analysis [Lin87] is to know the statistical relation existing between a dependent variable and one or more independent variables. In this sense, a functional relation between the variables must be postulated. In this case, data are fit to a logistic curve. Logistic regression is used extensively in the medical and social sciences as well as marketing applications such as prediction of a customer's propensity to purchase a product or cease a subscription.

[1] The Wolpert and Macready *no free lunch* theorem states that "any two learning algorithms are equivalent when their performance is averaged across all possible problems".

4.3.2 Decision trees

A decision tree (or tree diagram) is a decision support tool that uses a tree-like graph or model of decisions and their possible consequences, including chance event outcomes, resource costs, and utility. Decision trees are commonly used in operations research, specifically in decision analysis, to help identify a strategy most likely to reach a goal. In data mining and machine learning, a decision tree can be used as a predictive model, usually called classification trees or regression trees. These models map observations about an item to conclusions about the item's target value. In these tree structures, leaves represent classifications and branches represent conjunctions of features that lead to those classifications. In data mining, a decision tree describes data but not decisions; the resulting classification tree is used as a model for decision making.

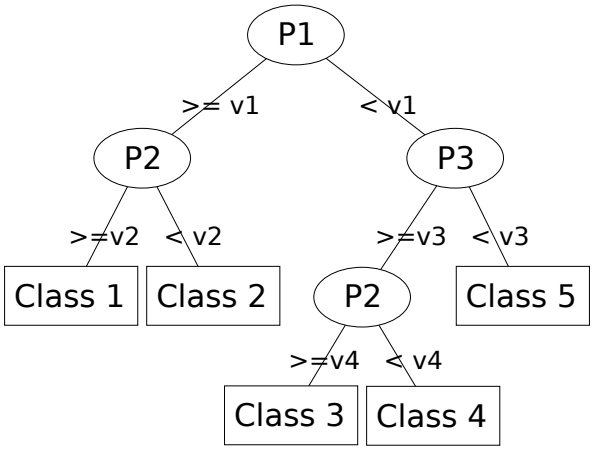


Figure 4.5: An example of classification tree

One of the most commonly used decision tree learning algorithms is C4.5 [Qui93]. This is a statistical classifier of the ID3 family of algorithms [Qui86] that can generate a model in the form of a classification tree or a set of equivalent classification rules. The leaf nodes of the decision tree contain the class name, whereas any non-leaf node is a decision node. The decision nodes represents attribute tests, with each branch (to another decision tree) being a possible value of the attribute. The C4.5 algorithm extends ID3 providing mechanisms to deal with continuous and missing values.

4.3.3 K-Nearest Neighbors

The K-Nearest Neighbors algorithm (KNN) [Das91] is a classifier algorithm based on agreement. KNN is a type of instance-based learning, or lazy learning where the function is only approximated locally and all computation is deferred until classification. The KNN algorithm is amongst the sim-

plest of all machine learning algorithms: an object is classified by a majority vote of its neighbors, with the object being assigned to the class most common amongst its k nearest neighbors (k musty be a positive integer, typically small). If $k = 1$, then the object is simply assigned to the class of its nearest neighbor.

A basic example of KNN can be seen in Figure 4.6. The test sample (green circle) should be classified either to the first class of blue squares or to the second class of red triangles. If $k = 3$ it is classified to the second class because there are 2 triangles and only 1 square inside the inner circle. If $k = 5$ it is classified to first class (3 squares vs. 2 triangles inside the outer circle) [knn].

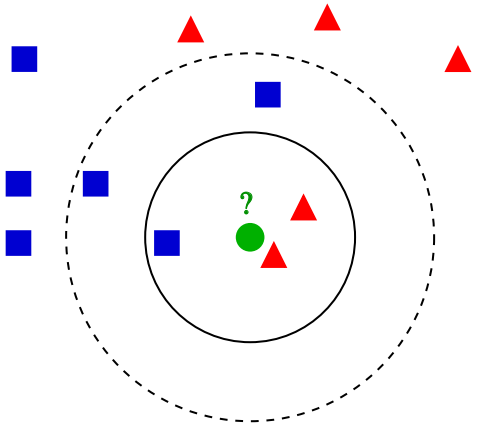


Figure 4.6: An example of KNN classification [knn]

4.3.4 Naïve Bayes classifier

Naïve Bayes [LIT92, Zha04] is based on applying Bayes' theorem. This classifier is a model of conditional independence of predictor attributes, ensuring an optimal classification if explicit assumptions are met. A naive Bayes classifier assumes that the presence (or absence) of a particular feature of a class is unrelated to the presence (or absence) of any other feature.

Depending on the precise nature of the probability model, naive Bayes classifiers can be trained very efficiently in a supervised learning setting. In many practical applications, parameter estimation for naive Bayes models uses the method of maximum likelihood.

An advantage of the naive Bayes classifier is that it requires a small amount of training data to estimate the parameters (means and variances of the variables) necessary for classification. Because independent variables are assumed, only the variances of the variables for each class need to be

determined and not the entire covariance matrix.

4.3.5 Multi-Layer Perceptron

A Multi-Layer Perceptron (MLP) [Hay94] is an artificial neural network model that selects the corresponding output for the specific input data. The MLP extends the standard linear perceptron using several more layers of neurons (nodes) with nonlinear activation functions, and is more powerful than the perceptron in that it can distinguish data that is not linearly separable.

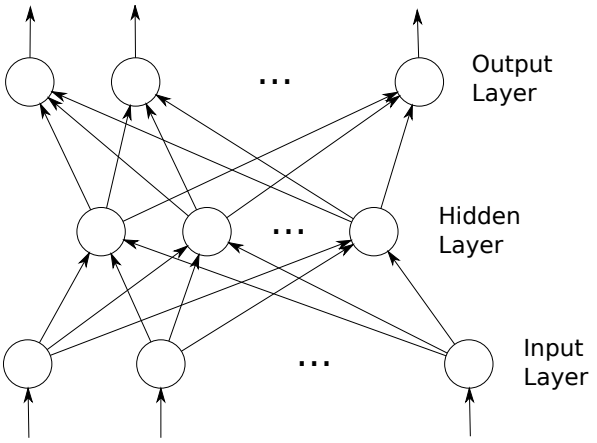


Figure 4.7: Generic MLP model

Multilayer perceptron networks using a back-propagation algorithm are one of the most commonly used techniques in supervised-learning pattern recognition and the subject of ongoing research in computational neuroscience and parallel distributed processing. They are useful in research because of their ability to stochastically solve problems. This often enables to get approximate solutions for extremely complex problems.

MLPs are commonly used in speech recognition, image recognition, and machine translation software. Above all, their most important application has been in the field of artificial intelligence, although the multilayer perceptron does not have connections with biological neural networks as initial neural based networks have.

4.3.6 Support Vector Machines

Support vector machines (SVMs) are a set of related supervised learning methods used for classification and regression. Given a set of training individuals, each marked as belonging to one of two

categories, an SVM training algorithm builds a model that predicts whether a new example falls into one category or the other. Intuitively an SVM model can be seen as a representation of the individuals as points in space, mapped so that the individuals of the separate categories are divided by a clear gap that is as wide as possible. New individuals are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall on. A basic example of how SVMs work can be seen in Figure 4.8. H3 (green) doesn't separate the 2 classes. H1 (blue) does, with a small margin and H2 (red) with the maximum margin [svm].

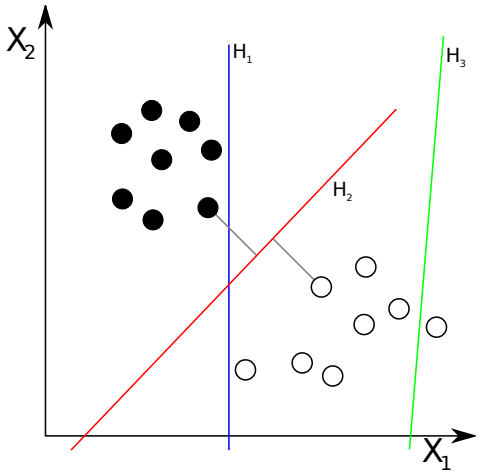


Figure 4.8: A basic example of SVM [svm]

A support vector machine constructs a hyperplane or set of hyperplanes in a high or infinite dimensional space, which can be used for classification, regression or other tasks. Intuitively, a good separation is achieved by the hyperplane that has the largest distance to the nearest training data points of any class (so-called functional margin), since in general the larger the margin the lower the generalization error of the classifier.

Part III

PROBLEM STATEMENT AND PROPOSAL

Chapter 5

Autonomic management of grid systems

Large scale distributed systems have paved the way to face complex, technical and scientific challenges. Most of these new challenges can not be solved with traditional systems, due to their enormous computing and/or storage requirements. Initiatives such as BOINC [BOI], PlanetLab [pla] or TeraGrid [TeG] and, more generally speaking, grid [Fos02] or the recent cloud computing [Clo] provide computing and storage resources that can be scaled to a level difficult to imagine elsewhere. Nevertheless, in spite of their potential, the complexity of these systems turns their management into an extremely difficult task.

Autonomic computing could be a theoretical solution to this problem, providing the system with the necessary mechanisms to manage itself, and leaving only high level decisions to the system administrator. Incorporating autonomic management to these systems usually requires a deep knowledge about the behavior of each single component. However, the large number of different resources involved makes it almost impossible to analyze and implement efficient policies on every one. Most of traditional and current grid management techniques are based on this approach [BAG00, KBM02, SF05, Sán08], dealing with each independent resource's behavior separately. A good alternative could be simplifying the understanding of the whole system, studying it as a single entity instead of the set of elements that together form it. This abstraction would describe how the system globally works and simplify its management.

This approach combines the use of self-adaptive techniques with a *single entity* vision of the grid in order to provide autonomic management and increase dependability. As it has been said, related research has focused on resource-related management, while this work's approach is unique in that it uses this *single entity* vision to focus on service-related global aspects. This approach involves a new starting point for grid management, considering aspects related to the whole system behavior instead of each independent resource.

5.1 Autonomic management issues in grid computing

Over the last decade, as global networking becomes reality, several different incarnations and definitions of what could be called grid systems have appeared [Sto07], from he BOINC infrastructure and the Condor resource pool [Con] to modern grids. Despite their differences, the following four main characteristics can be observed in most of them:

1. **Distributed:** A grid is composed of a set of resources that are logically and physically distributed over a wide-area communications network (WAN). The network is, in consequence, another resource of the system.

2. **Non-dedicated:** In most cases, the resources that compose the grid are simultaneously being used by external entities. The kind of use obviously depends on each element, but typical cases are general purpose networks (such as the Internet or corporative nets) and desktop computers or data center clusters that are also being independently used by their owners.

3. **Heterogeneous:** Also in most large scale distributed systems the computing resources involved are clearly different between them. Typical examples of this diversity are different architectures, operating systems or network protocols.

4. **Non-centralized:** Even though most large scale distributed systems have global infrastructures that allow all their different elements to cooperate (such as the Globus [GlA] and GLite [Gli] middlewares in grid computing), most resources actually belong to different owners that keep a high degree of control over their properties. For instance, in the Seti@home project [SAH] the system has no means of controlling when computing resources will be available, and nothing avoids that a specific user turns off his computer in the middle of a job execution. The degree of administrative decentralization depends on the type of system (e.g. it is not the same for the EGEE [ege] and Grid5000 [JLL+06] platforms), but nevertheless it is an important aspect that is always present to some extent.

Most grids are, in consequence, not only distributed in nature, but also heterogeneous, non-centralized and in most cases composed of non-dedicated resources. Incorporating autonomic features to such complex environments is not a simple task. These properties, added to the fact that grids are large scale systems (and therefore they have a large number of resources), bring the problem to a new level, and it does not seem a matter of simply adapting existing distributed computing techniques.

The inherent complexity of grid systems makes the direct application of traditional management techniques very difficult. In grid, heterogeneity, variability and decentralization are considered, in most cases, as system features. These special characteristics require a different perspective in system management, and traditional behavior patterns can not be directly adapted. When adopting an autonomic computing approach, the grid complexity has direct impact in its four main areas: self-configuration, self-healing, self-optimization and self-protection.

5.1.1 Self-Configuration issues

The heterogeneity and variability of most grids make resource configuration a non-trivial problem. Most traditional distributed approaches (cluster computing, centralized client-server architectures, etc) very often present desirable characteristics such as stability, homogeneity or simple and clear behavior patterns. This makes system configuration almost exclusively a design problem. Examples of this scenario are most of modern high performance computing clusters, where there are dedicate nodes for computing, data storage and so on, in a fixed setup. In these systems, reconfiguration only occurs when service requirements change or upgrades and structure modifications take place (all of which are rare, localized events). The reconfiguration process is usually performed in an off-line or semi-off-line operation mode and it frequently requires certain degree of redesign of the system's structure.

In grids the situation is clearly different. Resources are not only heterogeneous in nature (something that already increases the complexity of the configuration process) but also decentralized and unpredictable, joining and leaving the system at a high rate, and sometimes with variable availability and reliability. Under these conditions it seems clear that, in most cases, a fixed setup would not be completely effective. These large scale systems require a flexible and adaptable configuration in order to correctly take advantage of the available resources. Finally, the complexity of creating a highly adaptable configuration strategy increases even more if we also consider the high rate of resource variability.

5.1.2 Self-Healing issues

As a consequence of the grid natural characteristics, resources can unpredictably appear and disappear, network links can be temporarily or permanently interrupted, parts of the system can be overloaded without any control from the global system administrators and so on. These events are normally considered faults in traditional distributed systems, but in grid computing they are part of the environment's typical behavior. Therefore, is not so clear if these events should be regarded as faults or not, even though they might have a direct impact on its dependability. The lesser degree of cohesion of grids dilutes the concept of failure based on the loss or degradation of resources. Grids are commonly seen as an immense set of resources that provide a series of services. Therefore their proper operation should be understood in terms of quality of the services provided instead of the state of its internal resources.

5.1.3 Self-Optimization issues

Understanding the system behavior is the basis for improving its performance. From an abstract perspective, optimizing the provided services implies modeling the system's function and identifying situations that limit performance (such as bottlenecks). A deep system's behavior understanding enables also to develop even more advanced management policies and strategies, designed to make the most of the system resources available. In traditional distributed computing the usually stable, dedicated and highly fault-tolerant systems nowadays available (such as most modern computational and storage clusters) facilitate this task, allowing to design adaptable and scalable optimization techniques. These optimization techniques usually rely on homogeneous, dependable, high-performance resources (computing nodes, storage and network).

In grid computing, however, the situation is radically different. The massive amount of heterogeneous, non-dedicated and unpredictable resources that interact during the system's operation create a completely new and different framework, forcing performance optimization techniques to be adapted to these new conditions. The main difficulty of grid performance optimization is no other than understanding the system's behavior itself. Behavior pattern extraction and bottleneck identification are very complicated tasks, as synergies, dependencies and possible deadlocks between resources are obscured by the sheer complexity of the large-scale distributed system itself.

5.1.4 Self-Protection issues

Given the distributed, heterogeneous and decentralized nature of grids, proactive identification and protection from external attacks is a crucial aspect. In this sense, protecting each independent resource (computing machine, network element, etc) is the necessary first step. This can be done

incorporating traditional, well tested techniques to defend it from malicious usage and other security threats. However, the massive resource interaction present in grid systems can render these techniques insufficient, creating the need for protection mechanisms focused also on global aspects of the system. Again the grid's extreme complexity makes this task difficult, requiring to study the system as a whole and a deep analysis of the resources internal and external interactions.

5.2 Single entity vs. multiple entities

One of the most puzzling aspects of grid systems is that they are considered as single elements in theory but, when it comes to practice (specially in management related issues), they are treated as a set of independent, loosely related, elements. It might be argued that these systems are no simple ones and their great complexity makes necessary to look after every one of its parts. However, it could simply be a matter of perspective.

To illustrate this idea, it is interesting first to analyze the case of a single desktop computer. This apparently much simpler system is commonly regarded and managed as a single device but, in fact, it is composed of a large set of sophisticated elements that cooperate. Elements like CPUs, memory and its controllers, video cards, hard drives, network interfaces and so on have distinctive functionalities and are technologically complex, but are seen as parts of a single entity, instead of a set of heterogeneous resources. The secret behind this change of perspective is the use of high-level tools (basically the operating system) that provide an abstraction layer between the real, heterogeneous and complex hardware and the user. Several generic parameters are defined, such as CPU load or network usage, in order to express the system state in a standard manner. Even though this abstraction carries some loss of information, it enables the managing techniques to be standardized, regarding all desktop computers by the same parameters.

If this concept is applied to grids, it becomes clear that the proper tools for making this abstraction are yet to be established. Grids are still considered as a set of parts, instead of the sum of them. In consequence, the management tools inherit the complexity of the system.

5.3 Grid systems management: *Resource-level* vs. *Service-level*

Distinguished by their point of view, autonomic management techniques in grid systems can be split into two categories: *Resource-level* and *Service-level*. In order to optimize performance and

increase system dependability the correct combination of these two types of techniques should be applied. However, some important aspects must be considered.

Resource-level management involves the application of standard techniques in each and every one of the resources in the system. This might seem pretty straightforward, but a detailed analysis reveals that most of the typical characteristics of a grid limit its efficiency. The heterogeneous and non-dedicated nature of the system increase complexity, but it is the non-centralized aspect the one that becomes the great difficulty. In many cases, the global management system has so limited control of each resource that there is only a small set of suitable solutions available, such as general directives and coarse-grain strategies. To improve service dependability on a computational grid, for example, each job can be simultaneously executed in several resources, hoping that at least one of them finishes it (basic redundancy). Advanced *resource-level* management strategies (most of them directly inherited from traditional distributed computing) can of course be implemented as well, such as complicated optimization mechanisms, detailed security directives, etc. However, the high level of resource control usually required in order to apply those techniques would make it extremely hard to deploy them all over the grid in an unified way. The main reason for this is that, as it has been said, the grid non-centralized nature prevents any administrator from having full control of the whole system. Therefore advanced *resource-level* management will in most cases only be applied locally (limited to corporative networks, specific VOs, etc).

Service-level management, on the other hand, deals with system-wide policies aiming to increase performance, dependability and quality of the services provided. This is particularly important in utility computing environments, where the quality-of-service (QoS) is the key factor. However, as the management policies have to deal with the whole environment, it is important to find ways to efficiently handle this complexity. It is also important to understand that, as the nature of the system is different from *resource-level* management, the terms in which this management is expressed will certainly differ.

In *resource-level* management basic concepts can be directly inherited from traditional distributed systems. Performance, dependability, security and protection could be defined in those traditional terms, therefore adopting a related traditional distributed systems management approach. This means that, for instance, events such as a machine turning unexpectedly off or the temporary loss of a network link would be clearly regarded as faults. Therefore the different fault tolerance techniques used should be directly aimed to prevent these faults from causing failures. But in a non-dedicated, non-centralized distributed environment like a grid, each partner that shares resources keeps full control over its property (computing nodes, storage elements, network links, etc). Resource providers can

change the state of its own resources, without consent from the grid global management. Some machines could be turned off, originating an event that would be probably considered a fault in traditional distributed systems management. But in grid systems these events are by no means considered as undesirable or unexpected. They are more likely accepted situations that not only may, but will occur as part of the natural evolution of the grid. Therefore *service-level* management can never regard them as faults. Analogous situations can be observed in the areas of system protection and performance optimization. *Resource-level* management techniques in those areas should be focused on individual issues, making the most of the available resources and protecting each and everyone of them from malicious attacks. However, resource-focused protection and optimization does not necessary means grid global security and performance improvement. *Service-level* management should focus on these global aspects, handling resource interaction and developing synergies.

Generally speaking, *service-level* management should focus on QoS issues and global behavior. It can benefit from a general representation of the grid global state, specially if it is service oriented. This would create a grid global behavior model based on the system's service relevant states instead of the multiple specifics of each resource. This representation would not only be ideal for *service-level* management, but also would provide and abstraction layer like the one above mentioned. With such kind of model, grid management tools could finally have a real single entity perspective, incorporating the environment complexity without being overwhelmed by it. This could also take *service-level* management a step further, better understanding and improving the systems behavior, performance and dependability.

5.4 Contribution

In spite of all the *service-level* autonomic management issues here described, there has not been yet a scientific advance aimed at providing a genereric frame of reference for grid total state behavior modeling. As it has been aready explained, and will be extensively discussed in the following chapters, such a model could strongly benefit autonomic management, as well as become a significative step towards grid *single sistem image*. It will also provide a complete representation of the grid state, including internal, resource related aspects as well as external, service and user related ones. This work's contribution is to propose and develop that necessary frame of reference to model grid global behavior, providing the required abstraction to regard and manage it as a sigle system.

The first step it to formalize the basic theoretical concepts on which the global behavior model should be constructed. This means discussing what is considered to be *service-level* behavior, what does the global grid state represents and how it can benefit autonomic management. Chapter 6 ad-

dresses these issues, further extending the autonomic computing discussion and formalizing the basic behavior modeling concepts, from a grid *service-level* perspective.

Once the theoretical aspects are well stablished, the nest step is to formaly define a practical methodology to create a *service-level* model of the grid behavior. Chapter 7 presents a methodology focused on this, detailing a new set of procedures designed to observe and analyze the grid global behavior, constructing, as a result, an abstract model capable of explain the system's evolution as a single entity. This chapter elaborates even more on the subject, providing also a detailed study of the proposed methodology properties and presenting experimental results to validate the approach.

The next step is to empirically illustrate how grid autonomic management can take advantage of *service-level* behavior modeling. In Chapter 8 the full process is described, showing how global behavior analysis provides the necessary knowledge to desing and incorporate *service-level* autonomic management features into a real grid infraestructure. The process is explained in detail through a real scenario, based on a high performance distributed data access service. Finally, the same Chapter 8 presents also a generic framework for *service-level* grid autonomic management, developed as part of this works and based on the global behavior modeling methodology previously described.

Finally, Chapter 9 explores an advanced extension to the *service-level* global grid model presented in Chapter 7, incorporating behavior prediction capabilities.

Chapter 6

A *service-level* grid management model

As it has been previously said, *service-level* management could strongly benefit from a single entity point of view. To achieve this, it is first necessary to formally define the grid behavior theoretical model to be used in this case. In this chapter a *service-level* grid management model is presented, inspired by the single entity perspective. This model is based on the basic concepts and taxonomy of dependable systems presented by A. Avizienis, et al. in [ALRL04].

Following the single entity point of view, the grid can be considered as a unique **system**. From a theoretical perspective a grid, being a system, presents a **structure** and a **function** (or functions). The structure is the set of components that bound together to form the system, in this case the grid resources (computing and storage servers, network nodes, etc). The function or functions are nothing less than what the grid is intended for, and should be described in its **functional specification**. The part of the function that is related to the interaction with external entities (such as clients) can be seen as the functionality **provided** by the grid. This functionality is presented by the **service** or set of services. The interaction between the external clients and the grid is made through the **service interface**.

The analysis of the behavior presented by the grid internal structure (its resources) provides the **internal state** of the system. It could describe events happening in specific machines or network links, but gives little information about how this affects the grid global function. On the other hand, the behavior observed through the services interface (what the clients *see*) can be analyzed to deter-

mine the system's **external state**. This can provide information about how the system's function is being provided, but naturally lacks the capabilities to give a more detailed insight on the grid structure situation. The combination of both internal and external state produces the grid **total state**, that describes the system's behavior in a more complete way.

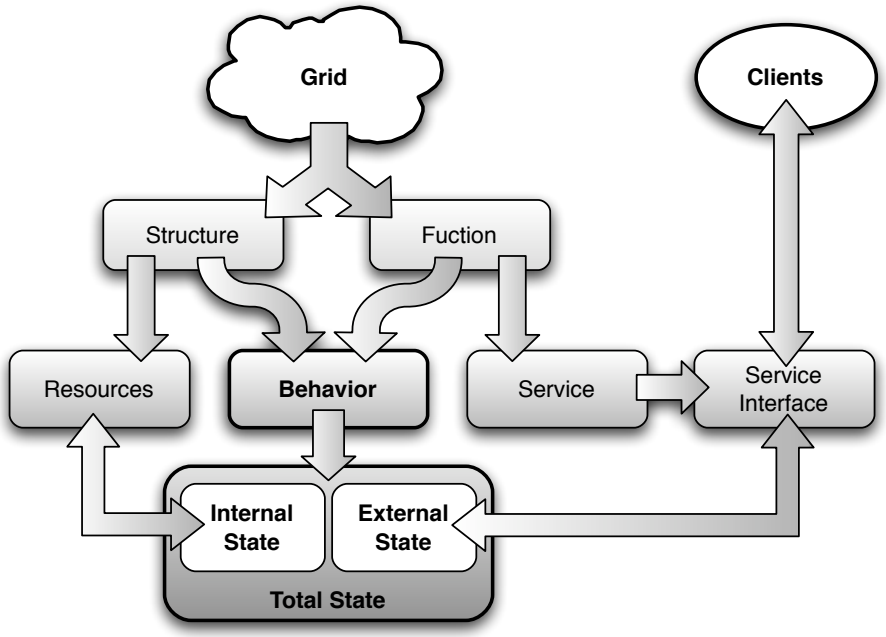


Figure 6.1: The grid: structure, function and state

Resource-level management focuses on aspects related to the grid structure (resources) and, therefore, affects only the internal behavior and state. In order to achieve *service-level* capabilities the whole total state must be considered, incorporating also the external state and its service related information.

6.1 *Service-level* autonomic management

Taking the previous model as a basis, we can analyze now how the different autonomic computing areas can benefit from a total state, service-level approach.

Self-configuration focuses on resource deployment and machine specific configuration. It deals with issues such as what software should be installed and where, in order to achieve the grid service

expectations. It is, therefore, intensively related to the system structure and internal state. In this case a *resource-level* approach is clearly indicated, since it contains all the resource-related relevant information needed.

Self-protection focuses on resource weak points as well as global security threats. A *service-level* approach would strongly benefit the protection against external, global attacks, specially those aiming at complex resource interaction vulnerabilities. Nevertheless, it would lack the necessary information to protect the system against more localized attacks, specially those aimed at single resources. An hybrid approach, combining elements of both *resource-level* and *service-level* management should be advisable in this case.

As explained in Chapter 5, *self-healing* is probably the autonomic computing area most affected by the grid systems special characteristics. Grid services fault tolerance issues are directly related to the system global state and require a *service-level* management approach in order to be handled properly. The grid unique features require basic fault tolerance concepts to be redefined, eliminating the possibility of directly inherit them from traditional distributed systems. This is explained in more detail below.

Finally, *self-optimizing* is focused on performance and quality of service issues. These can be equally related to the system internal or external state. The grid performance is directly dependent on the system resources, but also on the service usage patterns. Additionally, the grid complexity once again becomes an issue, extremely complicating any attempt of self-optimizing autonomic management from a *resource-level* point of view. *Service-level* management seems to be the ideal approach here, as it handles grid complexity without being overwhelmed bi it, focusing at the same time in the system's total state.

As it can be seen, both *self-healing* and *self-optimizing* would strongly benefit from a *service-level*, single entity approach. Therefore, this work was specifically focused on them, trying to provided the optimal basis to develop efficient autonomic management mechanisms in these two areas. Nevertheless, some specific basic theoretical aspects have to be discussed before continuing. *Self-configuration* and *self-protection* areas require the incorporation of *resource-level* elements and, therefore, are out fo the scope of this Ph.D. thesis.

6.1.1 Self-healing: Failures, errors and faults

It can be said that the grid delivers **correct service** when the behavior observed through the service interface follows the original functional specification. The total state associated to correct service is

called **correct state**. When the observed behavior deviates from the functional specification the system moves to an **incorrect state**. The transition from a correct state to an incorrect state is called a **service failure**, often abbreviated simply as a **failure**.

An **error** is the part of the total state of the system that may lead to a subsequent service failure. An error is not an incorrect state itself, but may possibly lead to an incorrect one and therefore is a potentially dangerous situation. The event that causes an **error** in the system's total state is called a **fault**.

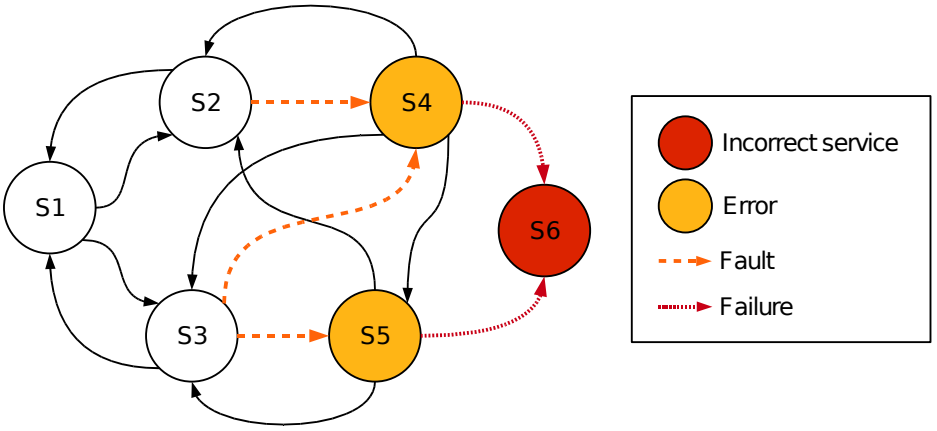


Figure 6.2: An example of a system's total state model

Figure 6.2 illustrates an example of the total state of a system, represented in the form of a finite state machine [HMU00, CL89]. The system presents a total of 6 possible states ($S1$ to $S6$), with one of them ($S6$) providing incorrect service. The possible set of events in the system is represented by the state transitions displayed in the figure and, at it can be seen, two of them are considered failures, because they make the system move to an incorrect service state. There are also two *dangerous* states ($S4$ and $S5$) that, even though they are correct in terms of the service provided, might lead into the incorrect service state ($S6$). For this reason these two states are considered errors, and the events that make the system transit to them are faults.

The use of this fault model allows to model single entity grid failures and provides the basic tools to build fault tolerance mechanisms. The use of the system's total state ensures a global but comprehensive understanding of the grid's behavior, not focusing only in resource-related aspects but also incorporating service-relevant ones.

6.1.2 Self-optimizing: performance and quality of service

Usual grid management mechanisms try to improve performance based on the individual analysis of every component on the system. Then they try to adjust the configuration or predict the behavior of each independent element. This approach may seem reasonable considering the complexity of the grid. However it could possibly fail to achieve optimal performance, because in most cases it lacks the capability to understand the effects that different elements have on each other when they work together. From a more theoretical point of view, if a grid is considered as an individual entity (with its computational power, storage capacity and so on), it seems logical to analyze it as such, instead of composed of a huge set of individual resources.

System performance is usually determined by the amount of useful work accomplished compared to the time and resources devoted to it. In order to optimize this ratio, the system must manage the available resources wisely. In large, heterogeneous and dynamic system such as grids, resource interaction becomes a critical issue, and effectively managing each component separately does not guarantee and improved overall performance. As in other less complex distributed and non-distributed scenarios (clusters, regular computers...), a global, *service-level* perspective is required in order to identify system bottleneck and other performance issues.

Quality of service means not only to achieve higher performance, but to sustain and guarantee a certain level (or levels) of it. This introduces the ideas of maintaining a specific, constant performance through time and providing the system with the necessary tools to ensure it. As this is directly related to performance, the previous reasoning applies directly also in this case, making clear the need for a *service-level*, single entity approach. Additionally, guaranteeing system performance (and therefore providing quality of service) creates the need for self-adapting techniques, in a very similar way to the *self-healing* area.

6.2 Modeling the total state of the grid

As it can be seen, the key to provide *service-level* management in grids and other kinds of large scale distributed systems is to have a deep knowledge about the system's **total state**. Many different techniques and representations can be used to this purpose, but there is always a set of basic characteristics that any grid total state model should present:

- **Specific state definition:** State characteristics and transition conditions should be unambiguously specified. The number of states should also be finite, in order to provide a useful model. A typical model representation that fits with this characteristics is a finite state machine [Arb69].

- **Stability:** The resulting model must be considerably consistent with the environment behavior over time. As these environments are naturally changing, it seems unrealistic to hope for stationarity and to try to find a definitive model for them. However, for a model to be useful, it must present at least a certain degree of stability. A model that needs to be regenerated every time an event occurs in the system is simply unusable.

- **Simplicity:** The resulting model should be easily understandable and provide basic and meaningful information about the systems behavior. A very complex model might be very precise, but it would be extremely complicated to use.

- **Relevance to service:** The model states should be related to the system services. This ensures that the observed behavior can be explained in terms of how these services are being provided. This makes it possible to determine if the conditions are acceptable and if the service provided meets expectations.

These characteristics define a descriptive model that provides a different outlook on system management, focusing on global aspects and events from a *single entity* perspective (instead of considering separately the multiple entities that compose it). A behavior model like this would be very useful in complex distributed environments, specially in system management tasks such as improving quality of service, dependability and fault tolerance. In these kinds of scenarios the global state of the grid is much more important than the state of every independent resource (given the usual amount of redundancy and distribution), and therefore a model focused on a global perspective proves to be more efficient, and much simpler. A technique or set of techniques capable of generating this model would be the key to understand the grid's total state and its global behavior. To this purpose an analysis methodology has been developed. This methodology is introduced and described in Chapter 7.

Chapter 7

A global behavior model for understanding the grid

As it has been explained in detail in previous chapters, two different ways of of understanding the grid can be distinguished. First of all, the *multiple entity* point of view, common in most grid management techniques, where the system is controled analyzing each resource independently. On the other hand, there is also the *single entity* point of view, in which the grid is regarded as a single system (*the* grid). This is, as explained in Chapter 5, similar as how computers are regarded as individual entities, even though they are made of several electronic components of different nature, or how clusters are most times considered as single machines, when in fact they are composed by many computers. It is, finally, a matter of abstraction, and a global behavior model of a grid would provide the necessary abstraction layer that finally makes the single entity point of view possible.

A model capable of providing single entity point of view becomes the basis to develop *service-level* autonomic management strategies. It is specially indicated in the areas of *self-healing* and *self-optimyzing* (as prevously explained in chapter 6). In the following sections a methodology for creating this kind of model is presented, called GloBeM (**Glo**bal **Be**havior **M**odeling). This methodology is strongly based on knowledge discovery techniques, and divided into the following three steps:

1. **Observing the grid:** The system is observed using large scale distributed systems monitoring techniques. At this point, every resource is monitored and the information is gathered. In the same way the operating system of a desktop computer monitors every hardware element, each

resource must be observed as a start point to build the abstraction. After or even simultaneously with the monitoring, the information obtained is represented in a more global way. The volume of montoring information generated could be enormous, due to the high number of different resources present in a grid (from hundreds to millions). Therefore the use of statistical tools (mean, standard deviation, statistical tests, etc) and data mining techniques (visual representation, clustering, etc) is required, in order to provide a correct and useful information representation.

2. **Analyzing the data:** Once the monitoring information is properly formatted, again data mining techniques (machine learning) are applied in order to extract useful knowledge and state related information about the system's behavior.

3. **Building the model:** Finally, a behavior model is constructed (a finite state machine exended with adittional statistical information), providing meaningful states and global, *service-level* information.

The resulting model produced by GloBeM becomes the abstraction layer on top of the grid. This model expresses in a simple and usable way the behavior of the system, and allows us to focus on a single entity view of the environment.

Finally, the whole GloBeM process is made in a transparent way, that is, autonomically.The proposed autonomic management helps to reduce the complexity and drawbacks of grid environments by managing their heterogeneity and complexity.

7.1 Grid global behavior model construction

In the previous section the global behavior model has been introduced and generally described. In order to have a deep understanding of the model capabilities it is necessary to know in detail how it is built. The complete three phases of GloBeM process with all their inner steps are shown in Figure 7.1. The following subsections will describe in detail every stage.

7.1.1 Stage 1: Observing the grid

As in any other autonomic computing process, the first step is observing the environment, that is, monitoring. A correct observation of the relevant parameters is crucial to witness the real grid behavior and to identify its states. However, as the grid monitoring information is very often massive (due to the actual size and heterogeneity of the system), it is also important to find a good way to represent this data, in order to be properly analyzed.

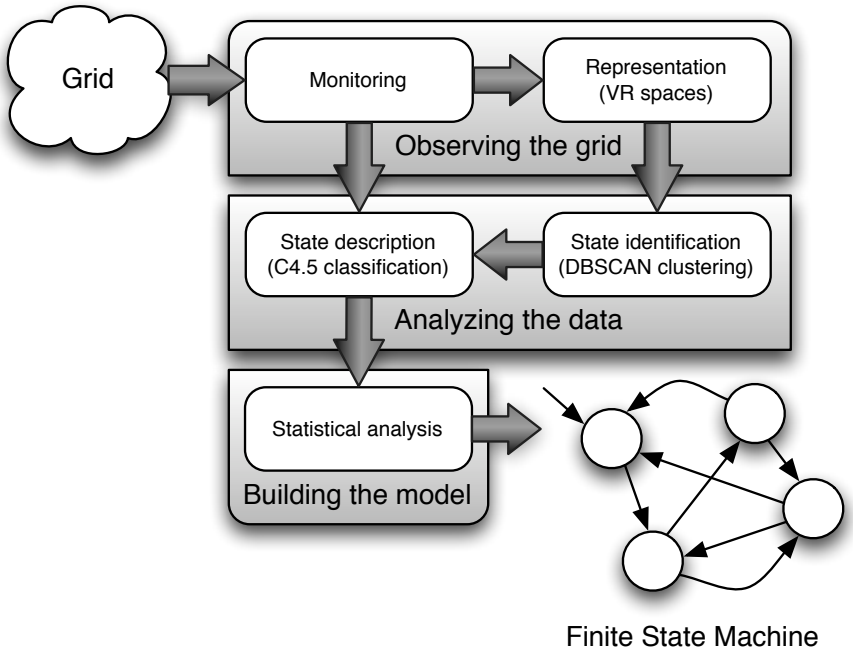


Figure 7.1: GloBeM phases

Therefore the *observing the grid* stage is divided in two steps: System monitoring, and information representation.

7.1.1.1 System monitoring

This first step is basically to gather monitoring information from every grid resource. Many commonly used monitoring tools can be used or adapted to this purpose, such as GMonE [gmo, Sán08], MonALISA [mal, NLB01, NLG+03], NWS [WSH99, Wol03, nws], Ganglia [gan], MDS [SDM+05], etc. The objective is to obtain a set of general observations of diverse parameters that can be monitored in each grid resource and then aggregate them to obtain *grid scale* values. This aggregation can be as complicated as it is desired, but generally common statistic descriptors such as average values and standard deviations are sufficient. The use of GMonE instead of other monitoring systems is recomended, since it performs this aggregation automatically, so no extra software layer is required.

There is an important detail that has to be discussed at this point, and this is the parameter selection. Intuitively it may seem reasonable to think that field experts in the services the grid provides could have a better understanding of the system behavior and therefore they could be able to make a

good monitoring parameter selection, focusing only on relevant information and discarding the rest. In the performed experiments this *human expert factor* proved to be very limited and generally inefficient. This seems understandable considering the vast complexity of the environment, which makes unlikely for any expert to have a complete knowledge of all possible influences on system behavior. On the contrary, a fully automatic approach is proposed here, based on the mathematical analysis of each variable in order to determine its importance. In this proposal, the parameter selection is done by the process itself (as described in this and the following sections), and it emerges naturally as a consequence on the methodology. Instead of providing a set of selected metrics, the administrator just needs to input as many different parameters as possible.

7.1.1.2 Information representation

As important as the monitoring information itself is the way it is represented. A proper analysis can not be performed if data is not correctly organized.

Although, as it has said above, the monitoring information should be aggregated in order to obtain *grid scale* values, this is usually not enough. If the parameter selection was exhaustive the monitoring data set obtained would still have so many variables, making difficult further analysis. To alleviate this problem virtual representation of information systems is used [Val02b, Val03] (see Section 4.1.2 for details).

In GloBeM unsupervised VR spaces are used for representing the grid, as the states are unknown in nature, moreover with a possible time dependent number and composition. This approach is more convenient than other classical techniques like Principal Components Analysis (PCA) [Pea01] for several reasons: *i)* PCA is a linear technique, whereas unsupervised VR spaces are obtained with nonlinear methods, more adequate to describe the complex relationships existing in large masses of monitored objects in an uncontrolled time-varying environment, *ii)* monitored processes are prone to contain missing information (this difficulty can be dealt with using nonlinear VR spaces, but seriously affects PCA), *iii)* unsupervised VR spaces focusses the attention in preserving the similarity relations between every pair of monitored objects in the best possible way, whereas PCA only seeks a transformation which creates a monotonically decreasing distribution of the variance (not necessarily the property of interest when trying to identify states within an unknown system).

GloBeM takes advantage of unsupervised VR spaces in order to identify grid states. Figure 7.2 illustrates a typical tree-dimensional data set represented with this technique. As it can be seen, the similar points appear closer, forming *clouds*[1].

[1] Do not confuse these clouds with this term in cloud computing.

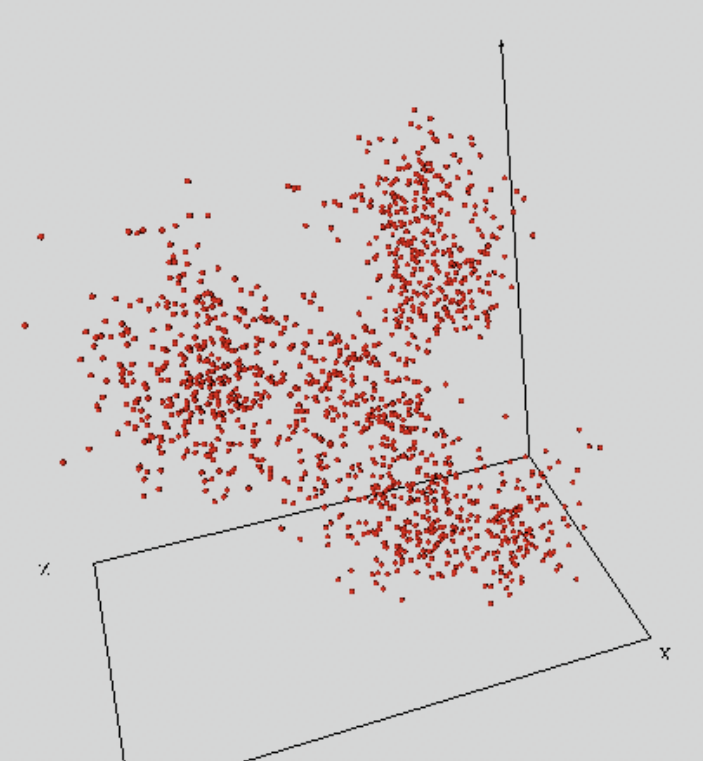


Figure 7.2: Example of tree-dimensional representation

At the end of *stage 1* GloBeM methodology has produced two outputs. The first one is the aggregated monitoring data, which contains the actual observed information. The second one is the three-dimensional virtual representation of that information, created to represent the degree of similarity among observations in an easy to handle way.

7.1.2 Stage 2: Analyzing the data

The aim of this stage is to identify and describe grid states.The final objective of these procedures is to create a finite state machine that models the system behavior, so the states themselves are the main element to identify.

The generated visual representation carries information regarding the degree of similarity among the observed monitoring values. Similar individuals in the data set should appear close in the representation, presumably forming a sort of *clouds*, separated by relatively empty spaces. As all individuals in each *cloud* have similar monitoring values, it is reasonable to presume that they have a close relation among what can be represented as belonging to the same group. Therefore different *clouds* will

represent different states, and it is necessary to analyze and characterize them.

This process is divided as well in two steps. The first one is actually separating the *clouds* in the three-dimensional representation. The second one is to, once each *cloud* has been separated, to study them in order to characterize each state.

7.1.2.1 State identification

In order to divide the original monitoring data set in different groups that will become states, the *clouds* within the tree-dimensional representation have to be separated. This could be done manually, by a visual analysis but, the aim of this proposal is to do it automatically. This is where data mining techniques [HK00] are the key to provide the appropriate solution.

The state identification can be seen as a clustering problem. Basically clustering is the assignment of objects into groups (*clusters*) so that objects of the same *cluster* are more similar to each other than objects from different ones. There are many clustering techniques that can be used, depending on the type of data and information about it available. During the development of this methodology several different clustering algorithms were tested in order to find the most adequate (a detailed description of all the algorithms here mentioned can be found in Section 4.2):

- **K-Means** [Mac67] and derived algorithms were the first to be tested, as they are one of the most commonly used clustering techniques. The results were diverse, showing that the clustering quality depended greatly on the data set used. Also, the K-Means algorithms require to provide the number of clusters as an input, which in this case is the number of states, but this value is unknown at this point. Anyway the performed experiments showed K-Means algorithm did not clustered correctly many of the three-dimensional datasets generated in the previous phase, so these techniques were discarded.

- **Hierarchical clustering** [Joh67, D'A78] was tested then, also with unsuccessful results. The main drawback of this technique is that the tree-dimensional representation contains *noise* that interferes with the hierarchical construction. This *noise* is a relatively small set of values that are not really close to any *cloud*, but resting in the almost empty areas between them. At this point it was clear that the clustering technique chosen should deal with this *noise*, therefore it should be a density based approach.

- **Expectation-Maximization (EM)** [DLR77] techniques were tried then, also with uneven results. At this point nothing can be assumed about the grid states, so limiting the search to a expected probability (as EM does) distribution seems premature.

- **Quality Threshold (QT)** [HKY99] clustering was tested then, with better result in this case. This algorithm allows us to identify very diverse kinds of clusters, without assuming any density probability distribution either specifying the number of clusters *a priori*. It requires more computing power than K-Means and it was originally designed for gene clustering. Although it provided better results than the other techniques, it still presented some problems dealing with *noise* values. Most of these problems basically resulted in the algorithm not being able to correctly separate clusters, grouping together sets of points clearly separated in the three dimensional space, but possibly conected by small groups of noise points. The main reason behind these problems could be that QT was originally designed to work in gene clustering. Generally the type of data sets used in gene clustering are very different from GloBeM three-dimensional representation, usually presenting many dimensions (much more than three) but not so many points.

- Finally **DBSCAN** [EKJX96] family algorithms were tested. DBSCAN techniques are specifically designed to identify density variations in a data set, specially in those with a low number of dimensions (which is the case) and they manage *noise* in a better way. It was finally decided to use this technique as it proved to be the most efficient and stable one for GloBeM's problem, providing a reasonably good clustering in almost every test.

The result of the state identification step is the clustering produced by the DBSCAN algorithm. The clustering information calculated is then incorporated to the monitoring information to perform its analysis.

7.1.2.2 State characterization

As each point in the tree-dimensional visualization represents one individual on the original data set, the clustering can be directly incorporated to the initial readings, and the resulting groups will be still consistent. Once the clustering is finished and its results incorporated to the original data set it is necessary to use a technique that explains the clusters in terms of the original variables.

This is a typical *supervised classification* problem. In this kind of problems the main objective is to create a model that explains a given classification of the individuals of a data set. In this case the classification is the clustering generated on the previous step, and the classification model generated must fulfill two requirements:

1. The model must be understandable by human means. As the main objective of this methodology is to create a behavior model of the grid based on a FSM, it is important that the states are defined in a way a human expert can understand. Therefore the model must be expressed

as a decision tree, decision rules or some other kind of understandable representation. This is important at the time of choosing the right classification algorithm because some techniques (i.e. Artificial Neural Networks [Hay94]) can generate very precise classification models that can not be explained.

2. The model must be simple. For the same reason as the previous point, the generated model must be easy enough for a human expert to understand.

Different classification algorithms that fulfill the previously mentioned two requirements were tested, and finally the C4.5 algorithm [Qui93] was selected. This is a statistical classifier that generates a decision tree as classification model. This tree can be easily translated into a set of rules, each one describing the conditions to determine which class an individual belongs to. The basic nature of the decision tree guarantees no ambiguity, allowing each element to be part of one group only. This algorithm can also be automatically adjusted to make sure that the generated tree is not too complex (although very simple models will probably generate worse classification models). In any case the C4.5 algorithm parameters can be automatically adjusted to generate a model that is understandable and useful.

At the end of this second stage the states have been identified and characterized. The next step is to statistically analyze the monitored data including this new information, in order to obtain the FSM model.

7.1.3 Stage 3: Building the model

The two previous stages provide a set of states and its description, but still more knowledge can be obtained from the monitoring data originally gathered. Transition between states are difficult to define automatically but, as it has been said before, the generated model is not ambiguous (it is represented in the form of a decision tree), so no situation where two states happen at the same time is possible. Therefore the transition between states can be determined by the classification model itself, reviewing the conditions of each one.

A simple statistical analysis of the monitored data, anyway, can provide with some other relevant information:

- **State probability:** The statistical study of each state in the monitoring information data set can offer a good estimation of the probability of each grid state. Therefore the most common states can be identified and separated from the rare ones. This provides a deep understanding of the grid behavior, knowing not only what conditions are possible (the states themselves) but which

of them are expected to be more often (and therefore it is important to focus further efforts on them).

- **Transition probabilities:** In a similar way to the state probabilities, transit on probabilities can be determined by a simple analysis of the monitored data. This complements the previous point, making possible to know not only which are the most frequent states but also which are the most probable transitions.

The resulting model generated by GloBeM is basically a **finite state machine** (FSM), but also includes statistical information that provides a deeper understanding of the grid behavior. As it has been said previously the resulting states are meaningful and easy to understand, providing crucial information on the monitored grid behavior.

7.1.4 Model stability

The three steps described above enable the construction of a FSM that models the grid behavior for the monitored time period. Each identified state is associated with a specific set of conditions which can be explained and are significantly different from the rest. Anyway there are some additional questions that can arise and should be answered in order to understand completely the GloBeM process.

Maybe the most important issue is related to the generated model usefulness and stability. The FSM obtained explains how the grid behaved along the monitored time period, but this behavior must be also observed outside the given time. Otherwise the model could be useless. From this point of view it is very important to find a suitable way to deal with the system variability.

The grid is an environment in constant change. From a theoretical point of view (related to the FSM that this technique aims to build) three types of changes can be identified:

- **Minor changes** are subtle variations related to the specific situation of the system. In most cases these changes are too detailed to be of any use from a global point of view, as they depend largely on the specific moment when they occur.
- **Major changes** are clear variations of the system's behavior, where some basic conditions change. This usually indicates changes of state and the FSM generated should be able to model them.
- **Radical changes** are normally related to drastic variations of the grid behavior, probably originated by great modifications on the system architecture, services provided or global usage

patterns. In these changes most basic grid conditions change, usually rendering the FSM model generated useless.

As it can be seen, what is considered a *minor change*, a *major change* and a *radical change* depends on how detailed the behavior analysis is made. If it is assumed that the FSM generated identifies *major changes*, where *the lines are drawn* to separate these categories depends on the size of the monitoring data set used to create the model.

A very small monitoring data set will generate very specific states, and the model will be generally over-fitted to this data. Therefore very soon as time goes by the behavior described by it will not match the observed one and the model will turn into useless.

On the other hand, if the monitoring data set is too big, the resulting model will be too generic, identifying only very big changes in the system's behavior as changes of state. This kind of model would be definitely much more stable along the time than the previous one, but also useless from a grid management point of view.

The key factor is, in consequence, to find the right monitoring data set size so that the FSM generated usefully models the grid behavior and also has an acceptable stability. It will always be vulnerable to *radical changes* but if the data set size is determined correctly this will not happen frequently.

7.1.4.1 Determining model stability and monitoring data set size

In order to select the proper monitoring data set size to generate the FSM a model stability study must be made. An experimental example of this study can be seen in Section 7.2, but a detailed description of the generic procedure is described here.

The basic concept behind this study is to determine how long a generated model can be used given a monitoring data set size. To express the following parameters should be taken into account:

- t is the instant where the model is generated.
- w is he monitoring window size. This is basically the size of the monitoring data set used to generate the model.
- $C(t, w)$ is the set of clusters observed at time t, in the first step of the *Analyzing the data* stage described above. These clusters are provided as the result of the clustering algorithm used:

DBSCAN. They are generated using monitoring information from interval $(t-w, t]$.

- $M(t, w)$ is the classification model generated at time t at the end of the *Analyzing the data* stage described above. This classification is provided by the classification algorithm used: C4.5. It is based on the clusters identified by $C(t, w)$.

Therefore, at any time t a new $M(t, w)$ model could be generated, or a previously calculated on time $t-d$ model $M(t-d, w)$ could be used. In order to determine which option is the best, it is necessary to find out if $M(t-d, w)$ is still valid at time t.

Using the classification model $M(t-d, w)$ over the current $(t-w, t]$ monitoring data set will provide with a a set of **predicted states** for that interval. Also, the calculation of $C(t, w)$ will provide a set of **observed states**, yet to be explained. The comparison between predicted and observed states is the key to determine whether or not $M(t-d, w)$ is still valid at time t.

The function ***F(t,w,d)*** is defined as the **level of agreement** between predicted (by $M(t-d, w)$) and observed (by $C(t, w)$) state values in the $(tw, t]$ monitoring data set interval. The value d is actually the distance between the data set interval $(t-d-w, t-d]$ used to generate $M(t-d, w)$ and the interval $(t-w, t]$ used to calculate $C(t, w)$. The study of this function for different t, w and d is the key to determine the right combination of values in order to achieve a good model stability.

7.1.4.2 Calculating the F(t,w,d) function

As it has been said, the $F(t, w, d)$ function should measure the level of agreement between the predicted and observed states, that is, "how close" the classification given by model $M(t-d, w)$ and the clustering $C(t, w)$ are. In order to compare them the *confusion matrix* can be used:

Given a set of observed states $O = \{o_1, o_2, ..., o_R\}$ and predicted states $P = \{p_1, p_2, ..., p_S\}$ on the same data set, the confusion matrix can be represented as follows:

	p_1	p_2	...	p_S	Sums
o_1	n_{11}	n_{12}	...	n_{1S}	$n_{1.}$
o_2	n_{21}	n_{22}	...	n_{2S}	$n_{2.}$
...	...	...		...	...
o_R	n_{R1}	n_{R2}	...	n_{RS}	$n_{R.}$
Sums	$n_{.1}$	$n_{.2}$	...	$n_{.S}$	$n_{..} = n$

Where n is the number of points in the data set and n_{ij} is the number of points that are in o_i and p_j.

Several comparison indexes can be calculated based on the confusion matrix, in order to determine how well the predicted states fit in the observed ones. The most simple is the *degree of similarity* (S) between O and P, and can be expressed as follows:

$$S = \frac{\sum_i [max(n_{ij})] + \sum_j [max(n_{ij})]}{2n} \tag{7.1}$$

This produces a value in the $[0, 1]$ interval, where 1 indicates that both O and P are a perfect match and 0 that are completely different. This S index seems like a good first way of calculating the $F(t, w, d)$ function, but other more complex metrics can be also used.

The *Rand* index [Ran71] attempts to be an improved metric for clustering comparision. It is defined by the following equation:

$$Rand = \frac{a + d}{a + b + c + d} \tag{7.2}$$

where

$$a = \sum_i \sum_j \binom{n_{ij}}{2}, \quad b = \sum_i \binom{n_{i.}}{2} - a,$$
$$c = \sum_j \binom{n_{.j}}{2} - a, \quad d = \binom{n}{2} - a - b - c$$

and

$$a + b + c + d = \binom{n}{2}$$

As in the case of S, the $Rand$ index produces a value in the $[0, 1]$ interval, where 1 indicates that both O and P are a perfect match and 0 that are completely different. Anyway this value is generally more accurate than the one provided by the simpler S.

The Fowlkes-Mallows measure of agreement [FM83] is another index that can be used to estimate the value of $F(t, w, d)$. It is defined as follows:

$$B_k = \frac{\sum_i \sum_j n_{ij}^2 - n}{\sqrt{[\sum_i (\sum_j n_{ij})^2 - n][\sum_j (\sum_i n_{ij})^2 - n]}} \tag{7.3}$$

Again the resulting B_k value falls in the $[0, 1]$ interval, providing the same information the S and $Rand$ indexes do. The Fowlkes-Mallows measure of agreement is generally considered to provide high quality measures, similar to those provided by $Rand$.

There are many other clustering comparison indexes that can be used to estimate $F(t, w, d)$. S, $Rand$ and B_k are commonly used and accepted as valid, and therefore have been selected as

GloBeM's basic estimators of the $F(t, w, d)$ function. In in the following section a real example of how they can be used to estimate the global behavior modeling stability is presented.

7.2 Experimental validation

In order to validate the methodology presented in this chapter two different types of experiments have been performed. The first type is based on a simulated grid environment, and its main purpose is to illustrate with a practical example the detailed procedure of constructing a global behavior model. The second one is based on real monitoring information gathered from the PlanetLab [pla] infrastructure. The main objectives of this second experiment are to illustrate how the global behavior modeling can be applied to a real large scale distributed environment and also to make an assessment of the behavior model stability validation techniques described in this chapter.

The detailed characteristics of each scenario, along with the experimental results obtained are described in the following subsections.

7.2.1 First experiment: Case of study

7.2.1.1 Scenario characteristics

In order to produce an as much realistic as possible simulation of a real grid environment, performance statistics and job accounting information from the EGEE project [ege] were used [fNRa, fNRb, fNRc, fNRd]. The EGEE is a project funded by the European Commission's Sixth Framework Programme. It connects more than 70 institutions in 27 European countries to construct a multi-science grid infrastructure for the European Research Area. The experiment simulation was conducted using GridSim [BM02b]. The GridSim toolkit allows modeling and simulation of entities in parallel and distributed computing (PDC) systems-users, applications, resources, and resource brokers (schedulers) for design and evaluation of grid-based applications. Nowadays it is one of the most commonly used and accepted simulation tools specfically designed for grid environments.

The designed scenario had the following defined characteristics:

- **Randomized resources:** Computing resources were slightly randomized to obtain certain heterogeneity. The number of resources was fixed to 100, but the computing power of each of them was randomly generated. Each resource may have one or two machines, each of them with one or two processing elements (CPUs). The power of each processing element was randomized

between 1000 and 5000 MIPS.

- **Randomized clients:** The scenario had a number of 70 clients. These clients randomly generated different types of load (basically CPU load and network load) in order to simulate the uncontrollable changes in the system.
- **Resource failures:** Each resource had a random chance of failure. These were isolated failures that temporarily disconnected the resource from the system randomly affecting its composition. The failure parameters (probability of failure and duration of failure) were adjusted to fit real failure rates observed on the EGEE, according to the reports above cited.
- **Job dispatcher:** In each scenario there was a job dispatcher that represented the grid service. It had a queue of randomly generated jobs. Each job had three randomly generated parameters: the job computing size (between 100 and 100000 millions of instructions) data input size (between 0 and 50 MB) and data output size (also between 0 and 50 MB). These are the three basic job parameters established by GridSim.

7.2.1.2 Experiment development and results

As it has been said, the purpose of this first experiment was to provide a detailed description and a practical example of the global behavior model construction process. In order to achieve it a simulated GMonE monitoring system was deployed along the GridSim simulation, configured to gather monitoring information and produce a monitoring data set. This information was aggregated to produce global values, finally producing the following four parameters:

- Resource CPU load average value.
- Resource CPU load standard deviation.
- Effective Network Bandwidth average value.
- Effective Network Bandwidth standard deviation.

As it can be seen, this is a very simplistic data set, but its simplicity is perfect to illustrate how the global behavior modeling is actually achieved. A total of 30 days of simulated time were generated, gathering a monitoring data set of all this time.

Then the monitoring data was represented using visual representation of information systems. This technique produced a three-dimensional representation of the monitoring data, where the distance between points is proportional to their degree of dissimilarity. The result can bee seen in Figure 7.3. Clearly three different “clouds” of points can be observed, each one presumably related to a

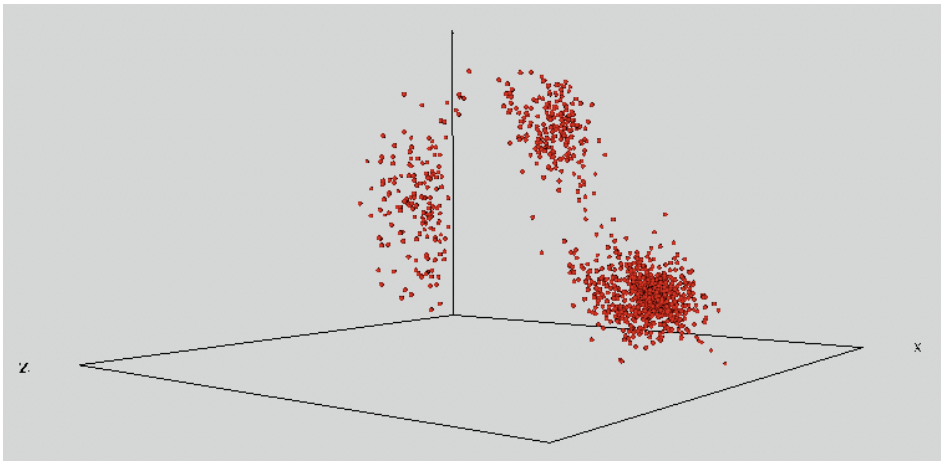


Figure 7.3: Three-dimensional visualization of the first experiment data set

different state. It is also clear that not all "clouds" are equally dense, probably indicating that not all three states are equally probable. Finally some "noise" points can be seen between "clouds", making the task of separating them more complicated.

Entering on the second stage of the global behavior modeling technique described (*analyzing the data*), the next step is to identify each state, by means of the DBSCAN clustering algorithm. The results of the DBSCAN execution can be seen in Figure 7.4. The three intuitively seen "clouds" were clearly identified as different clusters (the cluster membership is represented in the figure using a different color for each cluster). Also a significant amount of *noise* was found, represented in the figure as black dots. The first cluster, identified as *cluster1* (plotted in red in the figure) grouped the 10% of the monitoring observations in the data set. The second one, *cluster2* (plotted in green in the figure), grouped the 16%. The last one, *cluster3* (plotted in blue in the figure), grouped the 50%. The remaining 24% was labeled as *noise*.

At this point it is important to understand what is the real meaning of the *noise* label provided by DBSCAN. An important fraction of the total data set was determined to be *noise*, almost one fourth, so it is necessary to know what are their implications.

As it has been said in previous sections, DBSCAN is a density-based clustering algorithm. This means that it tries to find areas of the data set space where the density of points is considerably higher than the rest. Therefore the difference of density is the key factor while identifying clusters. When the density changes abruptly, as happens in the case of *cluster1* o *cluster2* on the figure, DBSCAN is able to identify almost the entire "cloud" as member of one cluster. When the density changes grad-

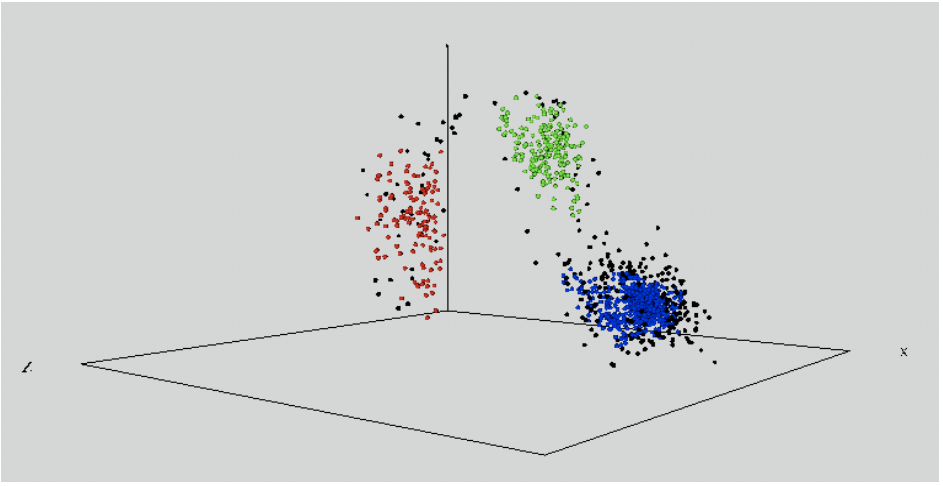


Figure 7.4: Three-dimensional clustering of the first experiment data set

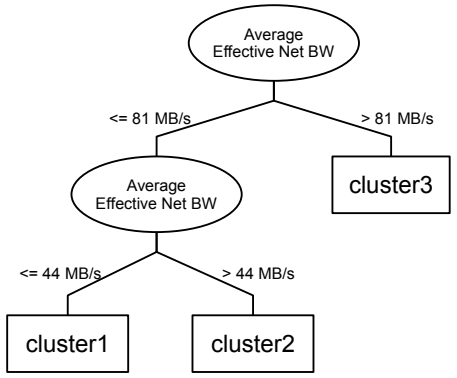
ually, as in the case of *cluster3*, it is much more difficult for DBSCAN to "draw the frontier of the cluster", loosing points in the process. These lost points are labeled as *noise*, as they interfere in the clustering procedure and make it more complicated. Anyway, ***noise* points should not be considered as outliers of any kind**, as are a result not of the data set itself but of the DBSCAN characteristics. It could be said that what DBSCAN provides is, in fact, each cluster "core" points, meaning those points that are in the most dense areas and therefore are those how better represent the cluster characteristics.

Once the clustering is done, the next step is to perform the state characterization, by means of the C4.5 classification algorithm. The use of this algorithm is very simple and the resulting decision tree can be seen in Figure 7.5.

Some additional values regarding the generated classification model are also shown in Figure 7.5. These values were obtained as the result of a stratified tenfold cross-validation of the model over the monitoring data and provide a better understanding of its quality. As it can be seen, the model is simple but accurate (less than a 2% of incorrectly classified instances) and the FSM can be easily constructed from it.

Finally, the last remaining step is to actually build the FSM, based on the classification model generated. Figure 7.6 shows the resulting FSM, displaying each state and the transition conditions. A close analysis of the decision tree generated in the previous stage and the FSM displayed here can provide an understandable explanation of each state:

- **State 1:** It is characterized by a low average network bandwidth (below 44 MB/s), probably



Correctly Classified Instances	901 (98.4699%)
Incorrectly Classified Instances	14 (1.5301%)
Mean absolute error	0.0184
Relative absolute error	5.3423%
Total Number of Instances	915

Figure 7.5: Decision tree generated by the C4.5 algorithm and cross-validation results

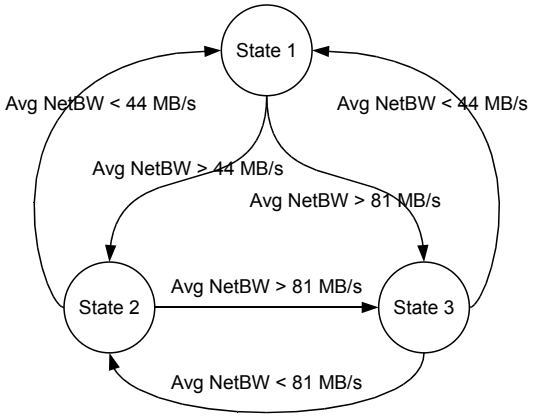


Figure 7.6: Finite State Machine generated at the end of the first experiment

mostly due to network overload. It corresponds to the cluster *cluster1* observed.

- **State 2:** It is characterized by a medium average network bandwidth (between 44 and 81 MB/s). It seems to represent the medium load state of the grid. It corresponds to the cluster *cluster2* observed.

- **State 3:** It is characterized by a high average network bandwidth (over 81 MB/s). This represents a barely loaded grid, where the network can be used at full capacity. It corresponds to the cluster *cluster3* observed.

As it can be seen the resulting model is not only simple and useful, but also makes perfect sense given the environment. It is important to remember that it has been generated without any "human guidance", therefore the emerging FSM is only the result of the natural grid behavior, and not influenced by any previous conception or assumption.

This is, nevertheless, a very small example of the potential of this technique. The second experiment is focused on a "real-life" scenario, displaying how global behavior modeling applies to real environments and further developing its characteristics and advantages.

7.2.2 Second experiment: Real scenario

7.2.2.1 Scenario characteristics

For this second part of the experimental validation real monitoring data from PlanetLab [pla] was used. PlanetLab is a global scientific research network, used by researchers at top academic institutions and industrial research labs to develop new technologies for distributed storage, network mapping, peer-to-peer systems, distributed hash tables, and query processing. PlanetLab currently consists of 991 nodes at 485 sites, scattered all over the world. It presents all the heterogeneity, complexity and variability expected from any real grid computing infrastructure, and therefore it is an excellent scenario for testing the global behavior modeling techniques presented here.

PlanetLab provides free access to a monitoring tool called CoMon [com], capable of presenting detailed information about the current state of each active node in the system. Many different parameters are monitored, including CPU usage, memory usage, network traffic, architecture characteristics, I/O operations, an so on. Information from this tool is beeing gathered in order to create a comprehensive monitoring database of the historical evolution of PlanetLab. The information contained on this database has been used in order to test the global behavior modeling techniques in a "real-life" scenario. The basic objective of this experiment is to create a useful global behavior model of PlanetLab and validate its quality and stability.

7.2.2.2 Experiment objectives

Using the monitoring information gathered from PlanetLab, a new series of advanced experiment was performed. Its objective was the empirical validation of the following GloBeM intended features:

- **Model quality:** The generated model should be able to describe accurately the observed behavior, specially in terms of the states identified.
- **Model usefulness:** The generated model should be understandable from a human perspective, in order to be useful for management purposes.
- **Model stability:** The generated model should present at least some stability along the time, in order to be usable.

7.2.2.3 Experiment development

For this experiment a total of 22 weeks of PlanetLab monitoring data were used. As it has been said before, this data contained information related to each node independently, so first of all it had to be aggregated to provide global values. Prior to this aggregation, the CoMon monitoring tool provided the following 10 parameters:

1. Busy CPU percentage
2. Last 5 minutes CPU load
3. Memory size
4. Free memory
5. Hard disk input traffic
6. Hard disk output traffic
7. Hard disk size
8. Hard disk use
9. Network input traffic
10. Network output traffic

These are generic parameters related to the PlanetLab resources and not selected by any human expert regarding any specific requirements (services provided, etc). The global behavior modeling process will automatically identify those that are really relevant to the system's behavior.

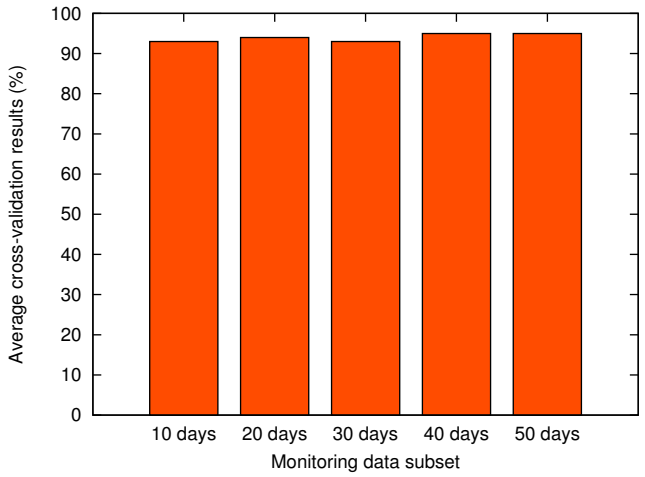


Figure 7.7: Average cross-validation results

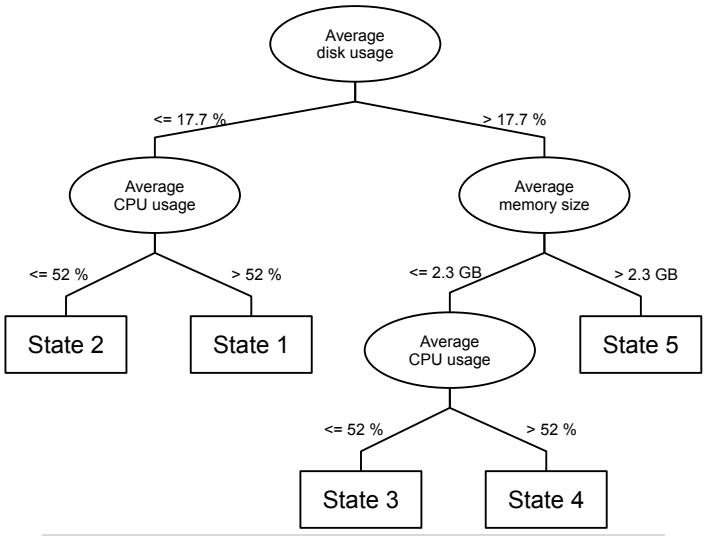
The data set was then fragmented in 1 hour monitoring intervals and aggregated values of average and standard deviation were calculated for each of these 10 parameters, resulting a total of 20 global monitoring parameters.

The data set was then divided in in many subsets, in order to generate different models in different instants of time. The length of these subsets was also variable, with a fixed minimum distance between monitoring subsets of 5 days. Subset sizes of 10, 20, 30, 40 and 50 days were selected generating a total of 105 subsets (21 of each size), with different degrees of overlapping between them. All these subsets were used to generate different global behavior models, and then compare between them.

7.2.2.4 Model quality

Using the global behavior modeling procedure described in this chapter a behavior model for each one of the 105 data subset was generated. As in the case of the simulated scenario presented before, the state characterization step included a stratified tenfold cross-validation, in order to provide a measure of how the observed states were correctly identified by the classification model. As this classification model is the basis for the finite state machine finally constructed, the result of this cross-validation indicates how well the behavior model describes the observed global behavior. Specifically, this test indicates the percentage of correctly classified instances in the data subset.

Figure 7.7 shows the average correctly classified instances percentage obtained, separated by monitoring subset size. As can be seen the value slightly differs between subsets, but in no case it is below 90%. The lowest average value is obtained by the 10 days subsets, to be precise 93.44%. The



Correctly Classified Instances	995 (99.1036%)
Incorrectly Classified Instances	9 (0.9864%)
Mean absolute error	0.006
Relative absolute error	2.0052%
Total Number of Instances	1004

Figure 7.8: Sample decision tree generated by the C4.5 algorithm using PlanetLab monitoring data and cross-validation results

highest value is obtained by the 40 days subsets, specifically 95.34%. As it can be seen, the difference is not so significant (less than 2%), making possible to affirm that the model capability to correctly identify each observed state is very good, in all cases.

7.2.2.5 Model usefulness

A set of previously described requirements have to be fulfilled (understandability, simplicity, etc) to obtain a useful behavior model. The PlanetLab used monitoring data produced a total of 105 different behavior models, each one generated using one of the monitoring subsets. Presenting here all of them would be clearly excessive and by no means necessary, specially considering that (as has will be explained in detail in the following section 7.2.2.6) the system's partial stability would make most models very similar. Instead of that, one example will be described and analyze, in order to determine its particular characteristics and suggest possible applications. The model selected was generated using a 50 days subset of the PlanetLab monitoring data. All the global behavior modeling stages were performed as described before and the resulting finite state machine was generated.

Figure 7.8 shows the decision tree generated by the C4.5 algorithm and the specific cross-validation results for this model. As can be seen five different states have been identified, and the analysis of

this tree provides the qualitative description of them:

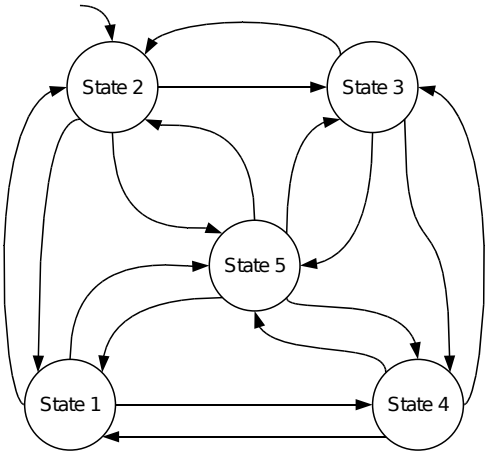
- **State 1** indicates that the average disk usage of the system is low (below 17.7%) but the CPU is considerably loaded (higher than 52%). This can be summarized as a situation where the grid is being used only for calculations or other CPU demanding operations.
- **State 2** presents a system with low disk usage and also low CPU usage. This probably indicates a generally low loaded system. This is also the most frequent state in the monitoring subset.
- **State 3** presents a system with more disk usage than state 1 and 2 (over 17.7%) but, not a high CPU load. Another parameter is introduced here, also indicating that average available memory is bellow $2.3GB$. This indicates a low loaded grid, but with not very much available memory and storage capacities.
- **State 4** is similar to state 3, but in this case the CPU load is higher than 52%. This seems to be the most heavy loaded state, as the disk usage could be high and there is not much memory available.
- **State 5** indicates a system also with more disk usage than states 1 and 2 (over 17.7%) but with a higher memory size than states 3 and 4.

Using this information a finite state machine was built, as displayed in Figure 7.9. The transitions have been indicated in a separated table to prevent the graph from being difficult to read.

As it can be seen the resulting states and state transitions are very easy to understand and work with, and provide valuable behavior information about the whole system. This information can now be used to design specific grid management mechanisms adapted to the most frequent situations (represented by the five identified states) on the system.

Job scheduling can be pointed out as a possible example of how this model can be applied to grid management. The five identified states clearly determine five different system conditions where some jobs might be more suitable for execution. A CPU demanding job, for instance, would be more easily and rapidly dispatched during states 2 and 3 than 1 and 4. Incorporating this information to the job scheduler would provide significant improvement of the use of system resources, scheduling jobs according to the system state. But not only CPU usage is present on the model, so other aspects such as memory and disk availability could be considered, more effectively scheduling jobs depending on their memory and storage requirements.

Generally speaking the incorporation of the global behavior model into the job scheduler could strongly improve its effectiveness. The identified states came from observation of the system instead



Transitions

State	Event	Transits to
State 1	CPU use $\leq 52\%$	State 2
	Disk use $> 17.7\%$ and Mem size $\leq 2.3GB$	State 4
	Disk use $> 17.7\%$ and Mem size $> 2.3GB$	State 5
State 2	CPU use $> 52\%$	State 1
	Disk use $> 17.7\%$ and Mem size $\leq 2.3GB$	State 3
	Disk use $> 17.7\%$ and Mem size $> 2.3GB$	State 5
State 3	Disk use $\leq 17.7\%$	State 2
	CPU use $> 52\%$	State 4
	Mem size $> 2.3GB$	State 5
State 4	Disk use $\leq 17.7\%$	State 1
	CPU use $\leq 52\%$	State 3
	Mem size $> 2.3GB$	State 5
State 5	Disk use $\leq 17.7\%$ and CPU use $> 52\%$	State 1
	Disk use $\leq 17.7\%$ and CPU use $\leq 52\%$	State 2
	Mem size $\leq 2.3GB$ and CPU use $\leq 52\%$	State 3
	Mem size $\leq 2.3GB$ and CPU use $> 52\%$	State 4

Figure 7.9: Sample finite state machine generated using PlanetLab monitoring data

of a human expert subjective advise, therefore they are more likely to fit into the real behavior, focusing only on observed events.

Anyway this is just a simple example of how this model could be used. Other applications such as fault tolerance, storage allocation, load balancing and many more can benefit from a global behavior modeling approach. From a more general perspective, this model could be used tu incorporate a variety of autonomic management techniques, specially in the areas of *self-healing* and *self-optimyzing*.

7.2.2.6 Model stability

In order to determine the global behavior model stability, the three clustering comparison indexes described in previous sections were used (degree of similarity, Rand index and Fowlkes-Mallows level of agreement). The classification model obtained from each monitoring subset was compared to the observed states of the rest, separated by sizes (the '10 days' models were compared with other '10 days' models, '20 days' models with other '20 days' models and so on).

To better understand how this comparison was made, it is necessary to go back to the definition of the $F(t, w, d)$ function and see how it was calculated in the experiment:

1. First a starting monitoring subset $(t - d - w, t - d]$ was selected. The size of this subset (10, 20, 30, 40 or 50 days) determined the value of w. This subset also corresponded to a specific moment in time, which is $t - d$.

2. Using this monitoring subset, the corresponding global behavior model was generated. One of the last steps of this construction produced the classification model $M(t - d, w)$ (classification model at time $t - d$ with subset size w).

3. Then a new monitoring subset $(t - w, t]$ was selected, of the same size of the previous one, and therefore with the same w value. Calculating the time diference between this new subset and the first one the values t (the time instant of the new subset) and d (the distance between both subsets) are obtained.

4. The clustering $C(t, w)$ was then calculated, using the new monitoring subset. The resulting clusters are the **observed states** for interval $(t - w, t]$.

5. The next step was to apply model $M(t - d, w)$ to the new $(t - w, t]$ monitoring subset. This produces a set of **predicted states**.

6. Finally one of the clustering comparison indexes was used to compare observed and predicted states. The resulting value can be used as an estimator of $F(t, w, d)$.

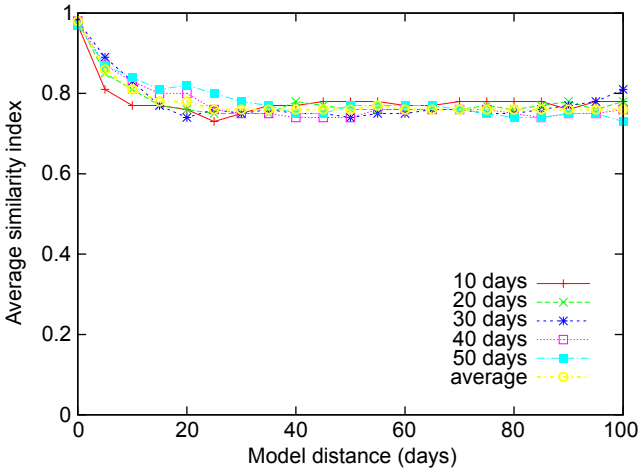


Figure 7.10: Average S index values

Repeating this operation for all possible combinations of t, w and d (in the monitoring data set) generates an estimation of the $F(t, w, d)$ function, based on the clustering comparison index selected. As three different indexes were used, three different estimations of this function are presented, each one based on one of them.

Figure 7.10 shows the estimation of the $F(t, w, d)$ function using the *degree of similarity* (S) index. The X axis indicates the distance d between the two subsets compared and the Y axis measures the S index value obtained. Instead of plotting 105 curves (one for every subset), the obtained values have been grouped (using average values) by w value, displaying five curves for 10, 20, 30, 40 and 50 days monitoring subsets. An additional sixth curve is also plotted, displaying the total average values of the other five ones.

The first thing that can be seen in Figure 7.10 is that all plotted curves have a S index value very close to 1 at distance $d = 0$. This is perfectly understandable and expected, as at distance 0 the two compared subset are in fact the same one, and, as it has been previously said, the cross-validation performed on the classification models generated presented very good results (around 95%). After that the curves decrease at different speeds, basically depending on the size of w. To understand this is important to remember that when the distance is small there might be overlapped data between the two subsets compared. For instance, a 30 days subset compared with another that is only 10 days away will have 20 days of information shared with it. This overlapped data makes the two subsets very similar (as they positively share some values) and therefore the classification model will naturally predict the states better. These first values are considered unrealistic for stability measurement procedures, as the real intention is to compare completely different subsets. Therefore the most im-

portant information displayed on Figure 7.10 begins when the distance d is higher than 50 (so there is no overlapped data in any case).

Since that moment the most significant information is revealed: the similarity index values seem to stabilize around 0.78. This is important for two main reasons:

- First of all this seems to indicate that there is real behavior stability within the system. Therefore a behavior model generated at a given time can predict the grid behavior at any other time, with a certain level of error. The fact that the $F(t, w, d)$ function estimator seems to be stable along the time (once cleared of the "overlapping effect") indicates that, even though there is real variability on the system, there is very possibly also a basic almost stable behavior. A global behavior model such a finite state machine seems, in consequence, to be a suitable representation of this basic behavior, in order to improve long-term management and performance prediction. Global behavior prediction issues are dicussed in detail in Chapter 9.
- The second reason why the stabilization of the $F(t, w, d)$ function is important is because the value is high enough (0.78 in this case). This means that there is an important amount of behavior information that keeps constant along the time, which is the basic behavior above mentioned. It also indicates that the prediction error of a generated model will probably be within acceptable limits, as it is able to correctly identify a great part of the grid behavior.

Besides, Figure 7.10 shows that, once the $F(t, w, d)$ function is stabilized, there is not a big difference between different values of w. All models seem to present very close similarity values. This seems to indicate that the size of w is not as important as was originally believed, as 10 days models seem to provide as good results as 50 days ones.

Even though the results presented in Figure 7.10 provide very significant information, it is important to use more than one comparison index in order to really validate them.

Figure 7.11 shows the same results commented above, but using the Rand index as clustering comparison tool. The data is still grouped by w value, the X axis indicates the distance between compared subsets and the Y axis shows the average value of the Rand index for each case.

As it can be seen the basic patterns observed in the previous case are still present, but the different index used incorporates new information to the graph.

The improved agreement due to overlapped data (the "overlapping effect" described above) is much more evident in this case, specially when comparing the '10 days' and '50 days' case. The

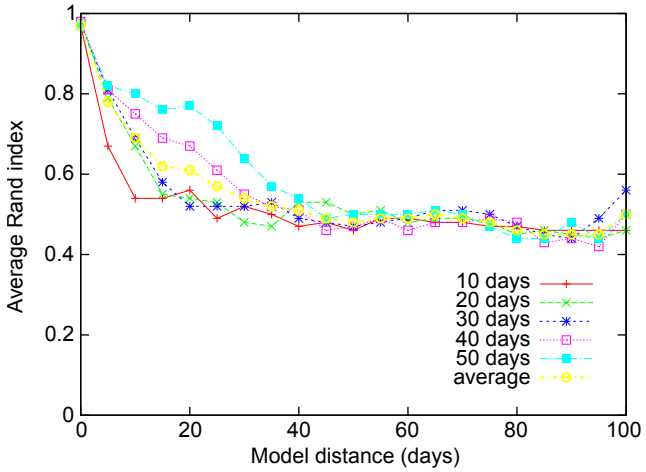


Figure 7.11: Average Rand index values

stabilization of the $F(t, w, d)$ function is present here as well, but less clearly than in the previous case. All curves show a very similar behavior (after the model distance is higher than 50) and all tend to stability, but there are still some variations present. Anyway the tendency to stability can be seen, happening in this case around a value of 0.5 in the Rand index.

Generally speaking, the Rand index seems much more sensitive to data variations, providing reasonably good values but lacking the clearness of the previous case. The use of another index is required to clarify these results, indicating if the conclusions obtained with the first index are valid. To solve these doubts the more accurate Fowlkes-Mallows level of agreement (B_k) index was used. The results obtained, represented in the same way as the two previous cases, can be seen in Figure 7.12.

The Fowlkes-Mallows index (B_k) clearly identifies the "overlapping effect" when d is small, as the S index and more clearly the Rand index did. The Fowlkes-Mallows index evolution also displays the clear model stability observed in the first case. The $F(t, w, d)$ function is stabilized now around 0.63.

The results provided by this third index clearly support the conclusions extracted from the case of the S index, showing a clear behavior stability and a great amount of constant behavior information that can be identified with any model generated.

At this point, it is important to consider the three stability values obtained for the $F(t, w, d)$ function using the three cluster comparison indexes. The first one provided 0.78, the second one

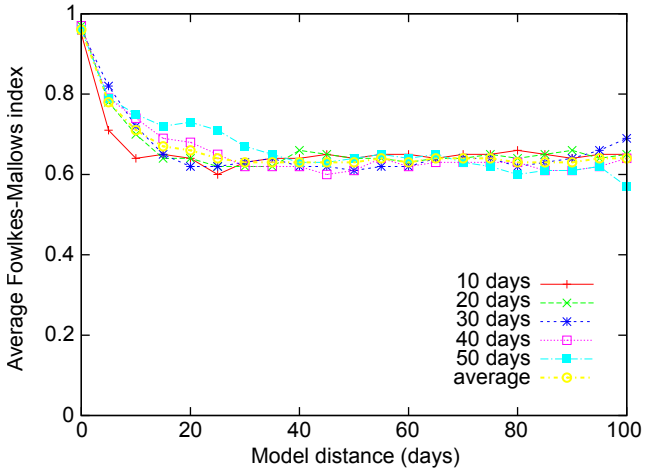


Figure 7.12: Average Fowlkes-Mallows level of agreement (B_k) values

0.5 and the third one 0.63. **These values can not be numerically compared**, as they came from different indexes. What can be compared is the qualitative meaning of these values, as all three indexes are designed to provide information regarding the same thing. From this point of view, all three indexes results indicate that there is a reasonably good agreement between the clustering compared, supporting the conclusion that the system behavior has certain stability thoughtout time and also there is an important amount of its behavior that is constant and can be globally modeled and predicted.

Chapter 8

Service-level autonomic management based on global behavior modeling

The GloBeM methodology described in Chapter 7 generates a simple, descriptive model of the grid global behavior. This model provides insight on the system's total state, specially convenient in *service-level* autonomic management and other related global aspects. In this chapter, the usefullness of the GloBeM approach is further studied, providing experimental results based on global behavior modeling in the autonomic computing areas of *self-optimizing* and *self-healing*. Firstly, a *self-optimizing* case of study is presented and described in detail, focusing on grid data storage quality of service. Secondly, a specially designed global autonomic management framework is introduced, presenting a simple *self-healing* use case to provide computational grid global fault tolerance.

8.1 Incorporating self-optimizing capabilities in grid data storage services

As distributed, global-scale, data-intensive applications are becoming more and more common, an increasing pressure is being put on the underlying distributed data services. As such services need to support massively concurrent, largely distributed accesses to huge shared datasets, the stability and scalability of their performance are critical. More specifically, the ability to sustain a stable high throughput is a very desirable property, as it strongly impacts the quality of service of the data storage

system and thereby the overall application performance. Handling quality of service in a grid system is however a very difficult task, as a very large number of factors are involved: the data access patterns, the status of a huge number of physical components, etc. In this Section an approach to self-optimizing autonomic management is explored, based on global behavior modeling combined with client-side quality of service feedback. This case of study main objective is to automate the process of identifying dangerous behavior patterns in storage services. To demonstrate our approach, global behavior knowlledge provided by GloBeM is used to improve quality of service in BlobSeer [NAB09a, NAB09b], a distributed storage service for large-scale data-intensive applications. BlobSeer is specifically designed to sustain high throughput under heavy access concurrency. This improvement is evaluated through extensive experimentation on the Grid'5000 testbed using hard experimental conditions: highly-concurrent data access patterns for long periods of service uptime, while supporting failures of the physical storage components. Results show substantial progress in sustaining a higher and more stable data access throughput.

8.1.1 Case of study: BlobSeer

For this case of study, the BlobSeer distributed data service was chosen as an experimental playground for investigating the expected benefits of the GloBeM approach. BlobSeer has specifically been designed to deal with the requirements of large-scale data-intensive distributed applications that process raw unstructured data.

BlobSeer consists of a series of distributed communicating processes, as shown in Figure 8.1. In BlobSeer, data is stored as unstructured sequences of bytes called BLOBs (Binary Large OBject). Each BLOB is made up of fixed-size *chunks* that are distributed among *data providers*. *Clients* read, write and append data to/from BLOBs. Data-intensive distributed applications typically employ a large number of processing elements that access the BLOB concurrently. *Metadata* facilitates access to a range (*offset, size*) for any existing version of a BLOB, by associating such a range with the *data providers* where the corresponding chunks are located. Metadata is stored and managed on *metadata providers* through a decentralized, DHT-based infrastructure (see details below). A central *version manager*, is in charge of assigning versions and of exposing newly created versions to the readers in such way as to ensure consistency. Finally, a *provider manager* decides which chunks are stored on which data providers when writes or appends are issued by the clients.

The goal of BlobSeer is to sustain high throughput under heavy access concurrency in reading, writing and appending. Three key design factors enabling BlobSeer to address these requirements are *data striping*, *distributed metadata management* and *versioning-based concurrency control*.

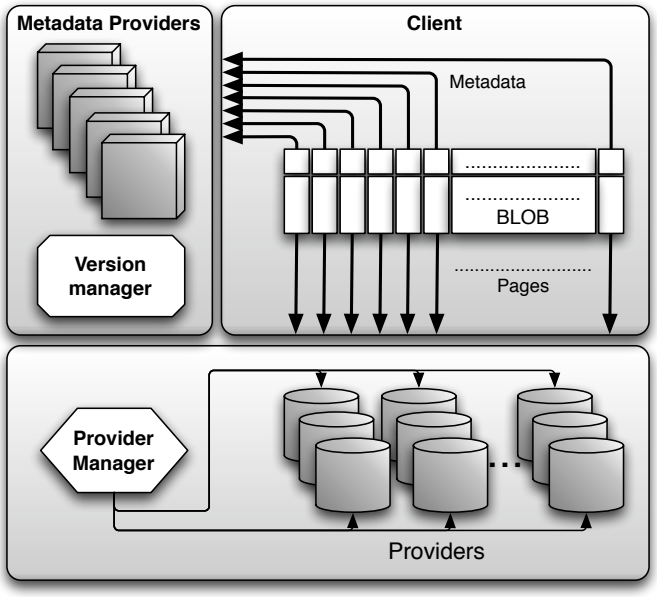


Figure 8.1: Architecture of BlobSeer

8.1.2 Improving QoS in BlobSeer using GloBeM

The high complexity of storage services in large-scale data-intensive settings makes it difficult to analyze the behavior of the system by explicitly considering every individual resource separately. The use of the global behavior modeling techniques described in Chapter 7 can substantially help by providing a valuable insight into the system's performance and evolution. GloBeM can automatically construct a descriptive global behavior model of BlobSeer's operation that can be used to identify bottlenecks and improve the overall performance and quality of service.

8.1.2.1 Experimental methodology

To achieve the indicated goal the following methodology was defined. Synthetic data-intensive workloads were applied to BlobSeer instances running in a grid setting for long enough time so that relevant behavior can be observed. To provide a realistic scenario, BlobSeer running instances were subjected to a failure-scenario generated by a failure-injection framework specially implemented, which was responsible for simulating data providers going down and becoming available again according to typical failure patterns observed in such large-scale setups. BlobSeer was then monitored throughout its run time and the monitoring information was fed to GloBeM in order to automatically

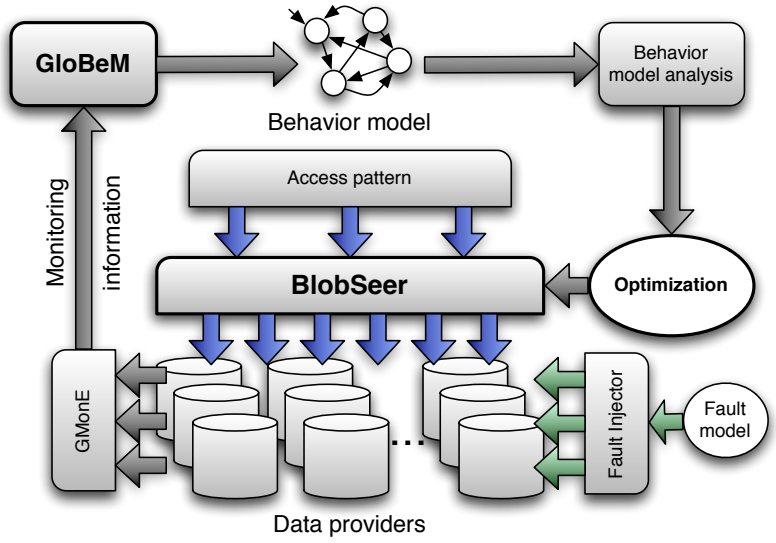


Figure 8.2: Improving QoS in BlobSeer by means of GloBeM

characterize BlobSeer's behavior. This in turn allowed to identify potential adjustments to BlobSeer that could be used to improve its performance and increase its quality of service. Fig. 8.2 shows a graphical description of the whole process.

8.1.2.2 Generating a data-intensive workload

A typical scenario in grid data-intensive applications consists in continuously acquiring (and possibly updating) huge datasets of unstructured data while performing large-scale computations over the data. For example, crawling the web for new content such as text, audio, video may proceed in parallel with a processing phase for these information to build search index structures, aggregated statistics or discover new knowledge useful for Internet services or scientific applications [Bry07]. For this reason, generating a typical data-intensive workload involves two aspects: i) a write access pattern that corresponds to constant data gathering and maintenance of data in the system and (in parallel) ii) a read access pattern that corresponds to the data processing.

As explained in [GGL03], because managing a very large set of small files is not feasible, data is typically gathered in few but enormous files. Moreover, experience with data-intensive applications has shown that these extremely large files are generated mostly by appending records concurrently and seldom overwriting any record. To reproduce this behavior in BlobSeer, a small number of BLOBs were created and several clients were deployed, generating and writing random data concurrently to the BLOBs. Each client predominantly appended and occasionally overwrited chunks of

64 MB to a randomly selected BLOB at random time intervals, sleeping meanwhile. The frequency of writes corresponded to an average constant pressure of 1MB/s on each of the BlobSeer data providers throughout the experiment duration.

8.1.2.3 Data intensive processing through MapReduce

In order to model the data processing aspect, workloads generated by MapReduce [DG08] applications were considered, which typically scan the whole dataset in parallel and aggregate interesting information about it. This translates into a highly concurrent read access pattern to the same BLOB. Note that writing the final end result is negligible, because most of the time it is a single aggregated value, such as the number of times a certain pattern occurred in the dataset. For this reason, modeling writes of end results was omitted. To reproduce the highly concurrent read access pattern, clients that perform parallel reads of chunks of 64MB from the same BLOB version were implemented. This data computation was simulated simply by keeping the CPU busy. An average I/O time to computation time ratio of 1:7 was kept (which is typical for MapReduce applications). As it is highly likely that computations of chunks from the same BLOB take similar amount of time to complete, we needed to compensate for this effect. To this end reads were adjusted to happen, unlike the case of writes, in bursts that put a more variable pressure on the data providers.

Both data gathering and data processing were executed concurrently in order to simulate a realistic setting where data is constantly analyzed while updates are processed in the background. The clients were implemented in such way as to target an overall write to read ratio of 1:10.

8.1.2.4 Simulating resource failures

Since real grid environments are subject to resource failures, a data provider failure-injection framework was implemented. This framework models resource failure patterns observed in real large-scale systems build from commodity hardware that run for long periods of time. The multi-state resource availability characterization study described in [RL07] was used, in order to generate random resource failure scenarios for these experiments.

In these failure scenarios, each data provider was assigned a predefined behavior in time: when it was unavailable and when it was available. A transition from available to unavailable meant killing the data provider, while a transition from unavailable to available caused the data provider to be restarted. Individual behavior of all providers was generated in such way that the global behavior coresponded to the results obtained in the above mentioned study.

8.1.2.5 Monitoring data providers

A wide range of monitoring parameters were periodically collected. These parameters describe the state of each data provider of the BlobSeer instance. For this task the monitoring framework GMonE [gmo, Sán08] was used.

GMonE runs a process called *resource monitor* on every node to be monitored. Each such node publishes monitoring information to one or more *monitoring archives* at regular time intervals. These monitoring archives act as subscribers and gather the monitoring information in a database, constructing a historical record of the system's evolution.

Resource monitors can be customized with *monitoring plugins*, used to adapt the monitoring process to a specific scenario by selecting relevant monitoring information.A BlobSeer monitoring plug-in was developed, responsible for monitoring each provider and for pushing the following parameters into the GMonE archive: number of read operations, number of write operations, free space available, CPU load and memory usage. These parameters represent the state of the provider at any specific moment in time. Every node running a data provider that is alive (was not rendered unavailable by the failure-injector) published this information each 45 seconds to a single central *monitoring archive* that stored the monitoring information for the whole experiment.

8.1.2.6 Building the global history record

Once the monitoring information was gathered, an aggregation process was undertaken, in order to calculate global values. Mean and standard deviation values were calculated for each of the five previous metrics, producing general descriptors of the system global behavior. Additionally, the number of data providers available was included as an additional parameter. The resulting set of monitoring metrics (mean and standard deviation for each data provider metric plus the number of available data providers) provided a global historical record of the BlobSeer behavior.

8.1.2.7 GloBeM behavior analysis

The historical data above mentioned was then fed into GloBeM in order to model BlobSeer's global system behavior into a finite set of states. Thanks to GloBeM, this process is fully automated and a characterization of the system states was obtained, in terms of the most important parameters that lead to each state. Client-side information was collected as well: what operations were performed, if they were successful and how well the operations did perform in terms of performance

measurements such as observed bandwidth from the client point of view.

8.1.2.8 Behavior model interpretation

Using the client-side information and the BlobSeer states characterization, we can reason about aspects of BlobSeer that can be improved to cope with such data-intensive workloads. More precisely, we classify states into *desirable states* that offer good performance to the clients and *undesirable states* that offer poorer performance to the clients.

8.1.2.9 Improving BlobSeer

Finally, the challenge was to incorporate self-optimizing autonomic capabilities into BlobSeer, improving its behavior in such way as to avoid undesirable states. As this is completely dependent on the characterization of the states obtained by using GloBeM and part of the experimental results, this aspect is described in detail in Section 8.1.3. For now, it is important to stress that modeling the behavior of BlobSeer hard, realistic conditions applied simultaneously to recreate a real-life behavior is an inherently difficult problem. The use of GloBeM helped to address this problem using global state identification. This approach enabled to identify improvements that would have been very difficult to detect by other means.

8.1.3 Experimental evaluation

Section 8.1.2 describes how to apply GloBeM behavior modeling techniques to improve BlobSeer, optimizing it for a data-intensive access pattern. In the following section the experimental procedures and results obtained are described.

8.1.3.1 Experimental setup

Experiments were performed on the Grid'5000 [JLL+06] testbed, a highly configurable and controllable experimental grid platform gathering 9 sites in France. Nodes from the Lille (130 nodes) and Orsay (275 nodes) clusters were used. The nodes are outfitted with x86_64 CPUs, and 2 GB of RAM. Raw buffered reads from the hard drives were measured, observing values at 61.8MB/s on Lille and 53.2 MB/s on Orsay. These measures were obtained using the *hdparm* standard Linux utility. Internode bandwidth is 1 Gbit/s (we measured 117.5 MB/s for TCP end-to-end sockets with MTU of 1500 B) and latency is 0.1 ms.

MapReduce-style computing systems are traditionally running on commodity hardware, collocating computation and storage on the same physical resources. Recent proposals [SRC09] advocate the use of converged networks to decouple the computation from storage in order to enable a more flexible and efficient datacenter design.

We aim at evaluating the benefits of applying global behavior modeling to BlobSeer in both scenarios. For this purpose, the Lille cluster was used to model collocation of computation and storage on the same physical node and the Orsay cluster to model decoupled computation and storage. More specifically, in the case of Lille cluster a version manager, a provider manager and 15 metadata providers were executed on dedicated nodes. In further 110 nodes a data provider and a BlobSeer client that generates the data-intensive workload were codeployed in each resource.

In the case of Orsay, we set up a version manager, a provider manager and 15 metadata providers on dedicated nodes. This time data providers were deployed on 120 dedicated nodes and another 120 dedicated nodes were used to run the clients that generate the workload.

In both scenarios each node that ran a data provider also executed a GMonE resource monitor. This monitor was responsible to collect the monitoring data throughout the experimentation. Further, in each of the clusters a special node was reserved to act as the GMonE monitoring archive that collected the monitoring information from all resource monitors.

From now on the scenario that models collocation of computation and storage on the Lille cluster will be refered simply as setting A and the scenario that models decoupled computation and storage on the Orsay cluster as setting B.

8.1.3.2 Results

To evaluate the benefits of applying global behavior modeling to BlobSeer, a multi-stage experimentation was perfomed, involving:

1. Running and monitoring an original BlobSeer instance under the data-intensive access pattern and failure scenario described in Section 8.1.2.

2. Interpreting the monitoring results by applying GloBeM.

3. Identifying and developing a self-optimizing improvement from BlobSeer.

4. Running an improved BlobSeer instance in the same conditions as the original instance.

5. Comparing the results and proving the improvement hinted by GloBeM was indeed successful in providing self-optimization capabilities, raising the BlobSeer's performance.

This multi-stage experimentation was performed for both settings A and B described in Section 8.1.3.1.

A BlobSeer instance was executed in both settings A and B, while the entire process was monitored using GMonE. Once the historical monitoring records were parsed successfully, GloBeM was applied to generate the global behavior model for each of the two experiments. Both experiments ran for a fixed duration of 10 hours. The data-intensive workload accessed a total of $\simeq 11TB$ of data on setting A, out of which $\simeq 1.3TB$ was written and the rest read. Similarly, a total of $\simeq 17TB$ of data was generated on setting B, out of which $\simeq 1.5TB$ was written and the rest read.

GloBeM analysis produced a set of global states, indicating statistical information that characterizes each one of them. As explained in Chapter 7, the GloBeM process involves the use of a combination of information representation and machine learning techniques. Its objective is to identify similarities in the historical monitoring information and use them to infer behavior patterns as system states. A statistical analysis is then performed in order to obtain representative descriptors for each distinctive behavior that was detected. Tables 8.1 and 8.2 show the average values for the most representative monitoring metrics in each global state, for both settings A and B. As can be seen, GloBeM identified four possible states in the case of setting A and three in the case of setting B.

Table 8.1: Global states - Setting A

parameter	**State 1**	**State 2**	**State 3**	**State 4**
Avg. read ops.	68.9	121.2	60.0	98.7
Read ops stdev.	10.5	15.8	9.9	16.7
Avg. write ops.	43.2	38.4	45.3	38.5
Write ops stdev.	4.9	4.7	5.2	7.4
Free space stdev.	$3.1e7$	$82.1e7$	$84.6e7$	$89.4e7$
Nr. of providers	107.0	102.7	96.4	97.2

These behavior models were combined with additional metrics extracted from the client logs (as explained in Section 8.1.2), in order to identify a relationship between internal global behavior patterns (observed by GloBeM) and service quality and performance. The number of read faults and effective read bandwidth were considered to be relevant metrics from the client point of view.

Table 8.2: Global states - Setting B

parameter	State 1	State 2	State 3
Avg. read ops.	98.6	202.3	125.5
Read ops stdev.	17.7	27.6	21.9
Avg. write ops.	35.2	27.5	33.1
Write ops stdev.	4.5	3.9	4.5
Free space stdev.	$17.2e6$	$13.0e6$	$15.5e6$
Nr. of providers	129.2	126.2	122.0

Average read bandwidths for each of the states are represented in Tables 8.3 and 8.4 for both settings A and B. Figures 8.3(a) and 8.3(b) depict evolution in time of the total number of read faults as observed by the clients for both scenarios. At this point it is important to remember that these are client related data and, therefore, neither read bandwidth nor failure information was available to GloBeM when identifying the states. Nevertheless, the different global patterns identified correspond to clearly different behavior in terms of client metrics, as Tables 8.3 and 8.4 and Figures 8.3(a) and 8.3(b) show.

Table 8.3: Average read bandwith - Setting A

State 1	State 2	State 3	State 4
24.2	20.1	31.5	23.9

units are MB/s

Table 8.4: Average read bandwith - Setting B

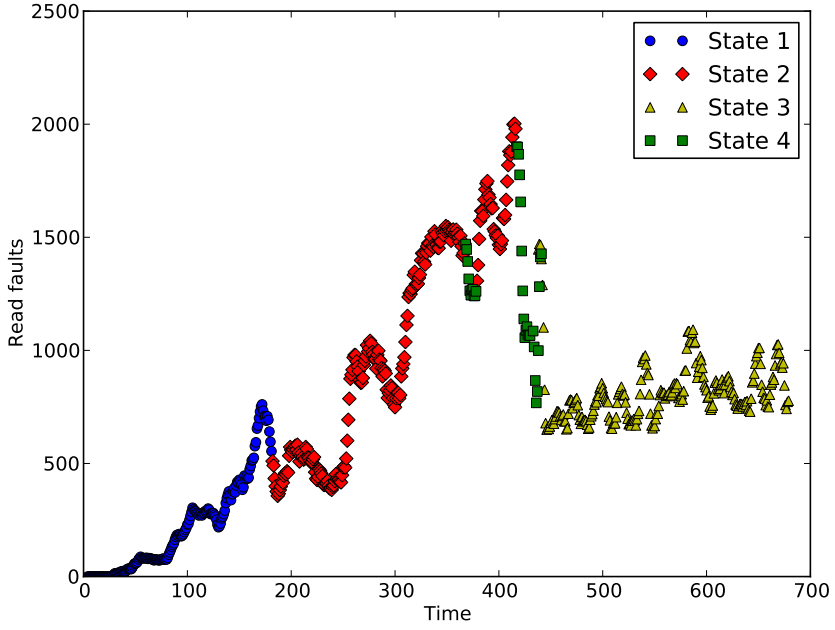
State 1	State 2	State 3
50.7	35.0	47.0

units are MB/s

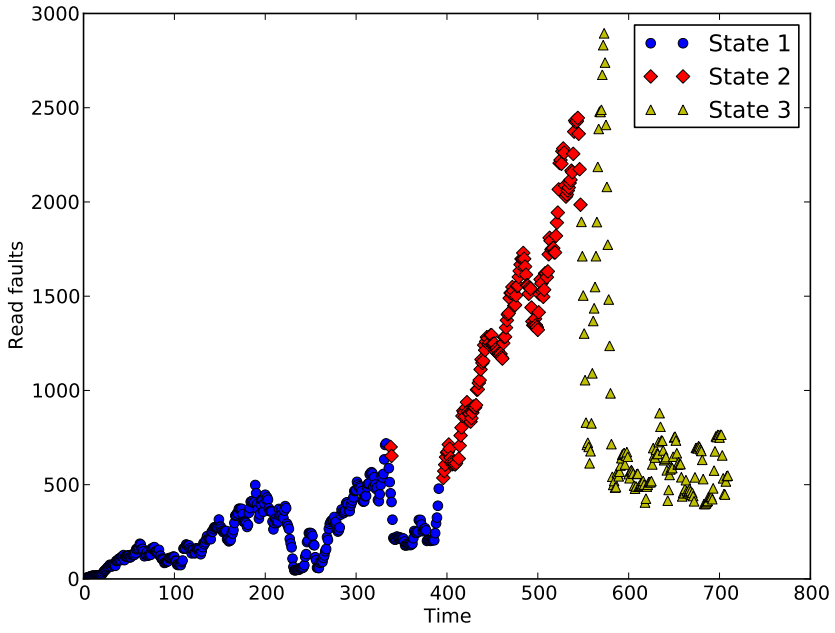
As previously described, the GloBeM analysis generated two global behavior models, each one corresponding to the behavior of BlobSeer in settings A and B. Further analysis was performed using the effective read bandwidth and number of read faults as observed from the client point of view in order to classify the states of the behavior models into *desired states* (where the performance metrics are satisfactory) and *undesired states* (where the results of the performance metrics can be improved).

In the case of setting A, *State 2* presents the lowest average read bandwidth ($\simeq 20MB/s$). It is also the state where most read faults occur, and where the failure pattern is more erratic. A similar situation occurs with setting B. In this case again *State 2* is the one with the lowest average bandwidth ($\simeq 35MB/s$) and the most erratic read fault behavior. We conclude these states (*State 2* in both settings A and B) to be *undesired*, because the worst quality of service is observed from the client point of view.

Considering now the global state characterization provided by GloBeM for both scenarios (Tables 8.1 and 8.2), a distinctive pattern can be identified for these *undesired states*: both have clearly the highest average number of read operations and, in the case of setting B specifically, a high standard



(a) Setting A



(b) Setting B

Figure 8.3: Read faults: states are represented with different point styles

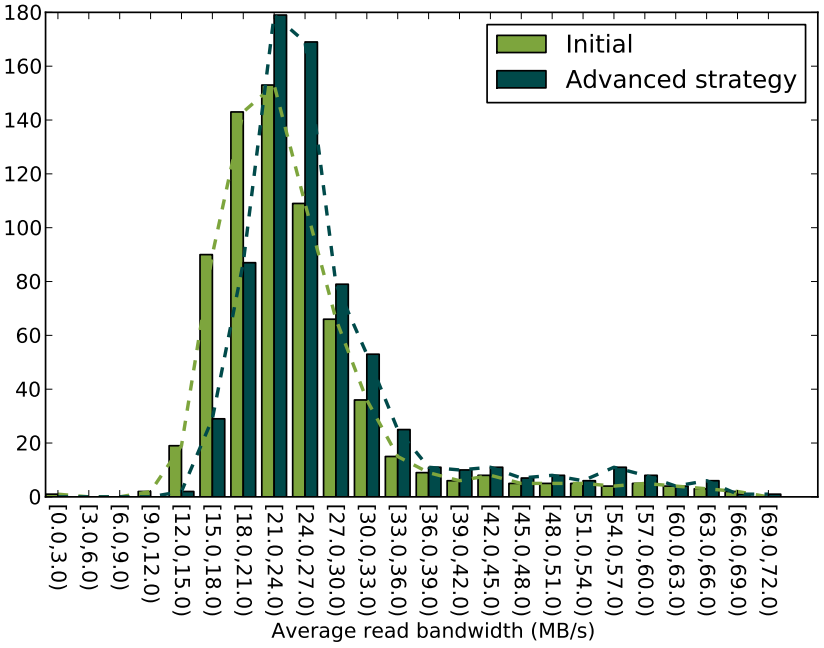
deviation for the number of read operations. This indicates a state where the data providers are under heavy read load (hence the high average value) and the read operation completion times are fluctuating (hence the high standard deviation).

Now that the problem has been identified, we aim at improving BlobSeer by implementing a self-optimizing mechanism that tries to avoid reaching the *undesired states* described above. Since the system is under constant write load in all states for both settings A and B (Tables 8.1 and 8.2) we aim at reducing the total I/O pressure on every data provider by avoiding to allocate providers under heavy read load to store new chunks generated by writers.

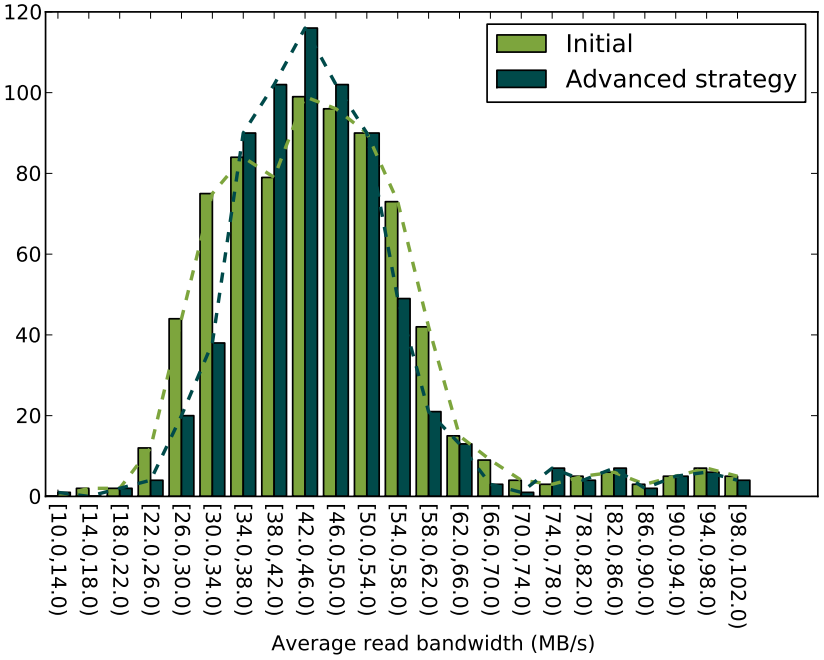
This in turn improves the read throughput but at the cost of a slightly less balanced chunk distribution. This eventually affects the throughput of future read operations on the newly written data. For this reason, avoiding writes on providers with heavy read loads is just an emergency measure to prevent reaching an *undesired state*. During normal functioning with non-critically high read loads, the original load-balancing strategy for writes can be used. The self-optimizing autonomic strategy mut be able to distinguish this situation, selecting a different mechanism to allocate data providers depending on the system global state.

The average read operation characterization provided by GloBeM for *State 2*, which is the *undesired* state (both in settings A and B), is the key threshold (121.2 average total simultaneous read ops. in setting A and 202.3 average total simultaneous read ops. in setting B) to decide when a provider is considered to be under heavy read load and should not store new chunks. This idea was implemented in the chunk allocation strategy of the provider manager. Since data providers report periodically to the provider manager with statistics, it is simply a matter of avoiding choosing providers for which the average number of read operations goes higher than the threshold. Those providers will be obviusly enabled to be choosed again when the number of read operations goes below this threshold. The same experiments were again conducted in the exact same conditions, (for both settings A and B), using in this case the improved self-optimizing BlobSeer chunk allocation strategy. As explained, the purpose of this new strategy is to improve the overall quality of service by avoiding the undesirable states identified by GloBeM (*State 2* in both setting A and setting B).

As final measure of the quality of service improvement, a deeper statistical comparison of the average read bandwidth observed by the clients was done. Figures 8.4(a) and 8.4(b) show the read bandwith distribution for each experimental scenario. In each case, the values of the original and improved BlobSeer version are compared. Additionally, Table 8.5 shows the average and standard deviation observed in each experiment scenario.



(a) Setting A



(b) Setting B

Figure 8.4: Read bandwidth stability: distribution comparison

Table 8.5: Statistical descriptors for read bandwith (MB/s)

Scenario	mean (MB/s)	standard deviation
Setting A - Initial	24.9	9.6
Setting A - Advanced strategy	27.5	7.3
Setting B - Initial	44.7	10.5
Setting B - Advanced strategy	44.7	8.4

The results seem to indicate a clear improvement (especially in setting A). However, in order to eliminate the possibility of reaching this conclusion simply because of different biases in the monitoring samples, we need further statistical assessment. In order to consider differences in values on Figures 8.4(a) and 8.4(b) and Table 8.5 as statistically meaningful, we need to ensure that the different monitoring samples are in fact obtained from different probability distributions. This would certify that the quality of service improvement observed is real, and not a matter of simple bias.

The read bandwidth observations were compared using the Kolmogorov-Smirnov statistical test [Ste74]. This technique works under the null hypothesis that the samples are drawn from the same distribution. A statistically meaningful result indicating that the null hypothesis is false[1] would validate the differences observed with the new BlobSeer strategy. The test results can be seen in Table 8.6. The values obtained clearly indicate that the null hypothesis is false in both cases, which validates the statistical relevance of the differences observed.

Table 8.6: Kolmogorov-Smirnov test results

Scenario	p-value
Setting A	$2.098e{-}14$
Setting B	0.004529

Finally, the results show that the inclusion of the self-optimizing strategy caused a clear quality of service improvement in both settings A and B. In setting A, the average read bandwidth shows a 10% increase and, which is more important, the standard deviation was reduced substantially. This indicates a lesser degree of dispersion in the effective read bandwidth observed, and therefore a much more stable bandwidth (for which the difference between the expected bandwidth (the mean value) and the real bandwidth as measured by the client is lower). As it has been said, these read bandwidth mean and standard deviation improvements indicate a significant increase in the overall data access quality of service.

[1] In statistical hypothesis testing, the p-value is the probability of obtaining a test statistic at least as extreme as the one that was actually observed, assuming that the null hypothesis is true. For our work we have considered that a p-value ≤ 0.01 result in the test is statistically meaningful proof that the null hypothesis is false.

In setting B, the average read bandwidth remained stable, which is understandable given that, as explained in Section 8.1.3.1, we are close to the maximum physical hard drive transfer rate limit of the testbed characteristics and, therefore, achieving a higher value is very difficult. Nevertheless, the read bandwidth standard deviation was again significantly reduced, resulting in a much more stable data access and, therefore, improved data access quality of service.

8.2 A global behavior autonomic management framework

Combining the ideas of single entity view, total state and *service-level* autonomic computing, a management framework called *FIRE*[2] has been developed as part of this work. The basic idea behind FIRE is that, if several states can be distinguished within a grid (the way GloBeM does it), different environment behavior should be obviously expected for each one of them. It seems reasonable to assume that not all management techniques would be optimal for every state. Therefore, if a set of compatible management policies are available, it would be essential to identify which one is most adequate for each state and provide the necessary mechanisms to switch between them when the system transits from one state to another. FIRE's main purpose is exactly that: to serve as a simple but effective automated policy selector, based on a GloBeM's finite state machine model of the system's behavior.

FIRE's most important characteristic are:

- It monitors the system (by means of a monitoring infrastructure) and represents the information in the same way the global behavior modelling procedure (GloBeM) does.
- Based on the represented monitoring information, it determines the current system state, by means of a previously provided finite state machine model.
- It activates the correct policy for the current state. The corresponding policy for each state must also be provided to FIRE as part of its configuration.
- The policies controlled by FIRE can be of any kind. Typical examples of this are a set of interchangeable job scheduling policies or data management and replication policies. FIRE has to communicate with the proper management subsystem (job scheduler, data manager, etc) in order to activate the proper policy.

[2]The acronym *FIRE* stands for "**F**IRE **I**sn't just a **R**eplication **E**nvironment". The reason behind that name is that the system was originally conceived as a data replication policies manager. Nowadays it has grown beyond that, to deal with different types of *service-level* autonomic management.

- FIRE has been designed as a grid computing management framework. Therefore it is directly integrated in this kind of systems. As in the case of the global behavior modelling methodology, the common basic characteristics between grid computing and other large scale distributed systems makes easy to adapt FIRE to other similar environments.

FIRE requires of some initial configuration (providing the finite state model and the corresponding policies) an therefore it is not a completely autonomic system. Anyway, it makes the administration work much easier. Once the finite state machine is automatically generated, the system administrator only has to decide which policy fits better in each state, in order to increase dependability.

8.2.1 Architecture of FIRE

Figure 8.5 illustrates the FIRE's architecture. It has been designed to provide an extensible, adaptive, autonomic framework for grid management. From an autonomic point of view, the system must present the following elements [IBM06]:

- **Sensors** (*the eyes*): These are the elements that gather information about the grid evolution and behavior. To this purpose FIRE takes advantage of a grid monitoring service. More specifically, it uses the GMonE tool mentioned in Chapter 7 and previous Section 8.1.2.

- **Effectors** (*the hands*): These are the elements that perform the actual grid management, following a specific policy or set of policies. The specific characteristics of these effectors change depending on the grid services and applications. In a data grid, for instance the effectors would be those software tools in charge of the management operations such as data allocation, load balancing, etc.

- **Knowledge** (*the brain*): This the autonomic system's core. It contains the necessary information and capabilities to perform four basic tasks:

  1. *Monitor* (by means of the sensors): This makes the system aware of its own state.
  2. *Analyze*: This makes possible to understand the system's state in terms of the behavior model in use. In the case of FIRE, this analysis is based on a global behavior model generated using the GloBeM methodology.
  3. *Plan*: Once the system's behavior has been observed and understood, the appropriate management decisions are made, in order to self-adapt to the current conditions.
  4. *Execute* (by means of the effectors): The planned decisions are executed.

  These four tasks are the base to provide autonomic capabilities to the grid management mechanisms. FIRE is focused on these aspects, providing the *knowledge* element in the grid autonomic management.

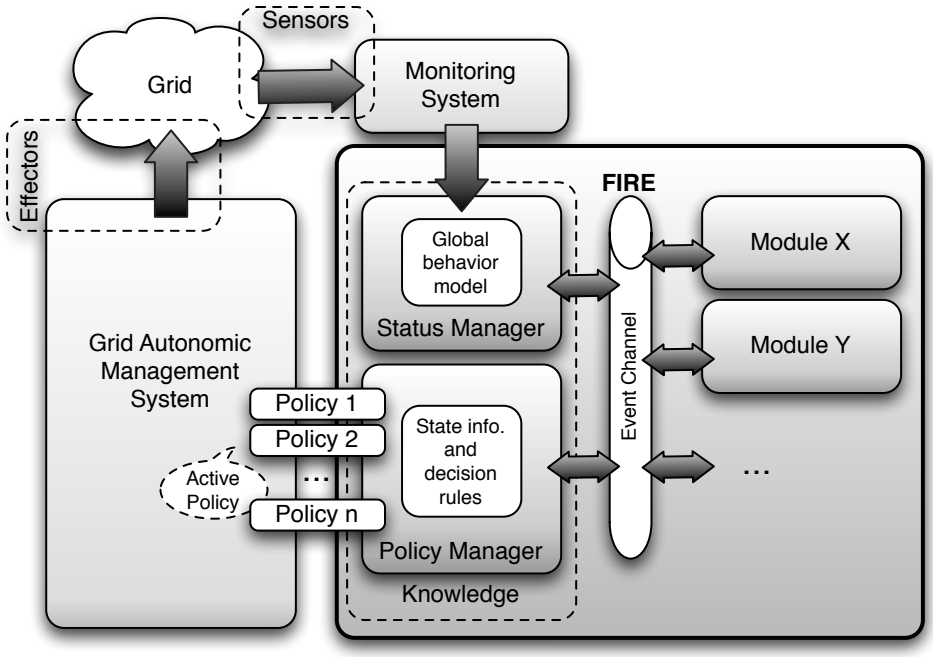


Figure 8.5: FIRE's architecture

At it can be seen in figure 8.5, FIRE itself does not stand as a complete autonomic solution, but as a basic framework to incorporate autonomic capabilities to a grid management system. From an architectural point of view it is designed around an standard **event channel**[3] in order to naturally increase its modularity and simplify its adaptation to different management problems. FIRE has three main elements: the *event channel* and the two main modules, connected through it (the *Status Manager* and the *Policy Manager*). It may contain also some other additional modules, in order to add new functionalities.

The *Status Manager* gathers monitoring information from the system resources. Then, with the use of the GloBeM finite state machine model, determines the current state and notifies it tho the *Policy Manager*. The *Policy Manager* receives the current state and determines which policy is to be activated. It then activates the policy and notifies this fact to the corresponding management subsystem. A more detailed description of FIRE's architecture can be found in Appendix A.

[3]This is a standard software design technique where communication between modules is carried out by a central event manager. A set of events are specified and the different modules can act as event publishers and/or event subscribers. This structure strongly simplifies the introduction of new modules on the system.

8.2.2 Incorporating self-healing capabilities in grid computational services using FIRE

Using FIRE as a basic autonomic framework, system administrators can effectively manage a large scale distributed system without being overwhelmed by the environment complexity. The total state, service oriented behavior model is the key to *see the big picture* and focus on global management.

To easily illustrate this, a simple simulated scenario is going to be described, indicating the FIRE configuration and performace. This initial experiment is based on the simulated grid modeled by GloBeM in Section 7.2.1. In that subsection the detailed GloBeM process was illustrated using a simulated GridSim system, based on statistical information from the EGEE project. The purpose of this basic experiment is not, in any case, to demonstrate FIRE's full potential but to demonstrate its benefits and functionalities in the simplest way possible.

It has been said that FIRE can address many different problems, depending on the management system or systems it is working with and the set of policies provided. One of the most common uses of large scale distributed systems in general and grid computing in particular is job execution (computational grids). The distributed nature of the environment allows to run multiple jobs in different resources, but to efficiently do it, an adequate job scheduler is required. The capabilities of this scheduler can almost entirely establish the system's performance and dependability, and therefore they are of maximum importance.

For this experiment, a job execution service was simulated. Since FIRE addresses *service-level* autonomic management, different autonomic areas can be adressed. In this case the experiment was focused on self-healing capabilities, regarding from a system's total state and global behavior point of view.

To correclty model the system total state, the basic monitoring parameters of the experiment must be not only focused on the grid resources, but also service oriented. In this sense, job failure rate can be used to measure the QoS provided by this service from a *self-healing* point of view. Thus, randomly generated jobs were submitted to the simulated grid through a job scheduler and their failure rate was measured. To determine a job failure, a time deadline was established for each of them, based on its time of submission and job size. A job failure, in consequence, could be originated by a resource crash, a network overload, etc. The objective of this experiment was to show how FIRE, with the appropriate policies, can increase system's dependability by providing self-healing capabilities that lower the job failure rate.

The experiment characteristics will be very briefly summarized here (for a more detailed explanation please refer to the above mentioned Chapter 7):

- **Randomized resources:** The number of resources was fixed for each scenario, but the computing power of each of them was randomly generated. Each resource may have one or two machines, each of them with one or two processing elements (CPUs). The power of each processing element was randomized between 1000 and 5000 MIPS.
- **Randomized clients:** Each scenario had a different number of clients. These clients function was to randomly generate different types of load (basically CPU load and network load) in order to simulate the uncontrollable changes in the system.
- **Resource failures:** Each resource had a random chance of failure. These were isolated failures that temporarily disconnected the resource from the system randomly affecting its composition. The failure parameters (probability of failure and duration of failure) were adjusted to fit real job failure rates observed on the EGEE (this is explained in more detail below).
- **Job dispatcher:** In each scenario there was a job dispatcher that represented the grid service. It had a queue of randomly generated jobs. Each job had three randomly generated parameters: the job computing size (between 100 and 100000 millions of instructions) data input size (between 0 and 50 MB) and data output size (also between 0 and 50 MB). These are the three basic job parameters established by GridSim.

Table 8.7 shows the different parameters established for each simulation scenario. Tests simulated 30 days of execution of these environments.

Table 8.7: Test scenarios

Scenario name	**20R**	**50R**	**100R**
Num. resources	20	50	100
Num. CPU load clients	10	25	50
Num. Net load clients	10	20	20

Since, in this scenario, FIRE aims at providing fault tolerance (self-healing capabilities), one of the main aspects that must be considered in order to perform a realistic simulation of this grid environment is job failure rate. As EGEE was used as a reference for the simulated scenarios, it was important o reproduce the same failure rates observed in the real environment. The above mentioned references show that this parameter oscillates due to many factors, but it is usually around 16%. For

the simulated scenarios, it was decided to generate a basis job failure rate of 16%, and then show how the use of FIRE lowers this rate. This value includes any kind of job failure, both generated by resource failures and/or network problems.

8.2.2.1 Behavior model

Prior to performing the FIRE tests, an initial configuration was designed for comparative purposes. This configuration used FCFS (First-come, first-served) as the only job scheduler policy, so jobs were always strictly dispatched in the order they arrived. It was executed on all three test scenarios and a behavior model was generated using GloBeM. The detailed process of this construction can be seen in Section 7.2.1.

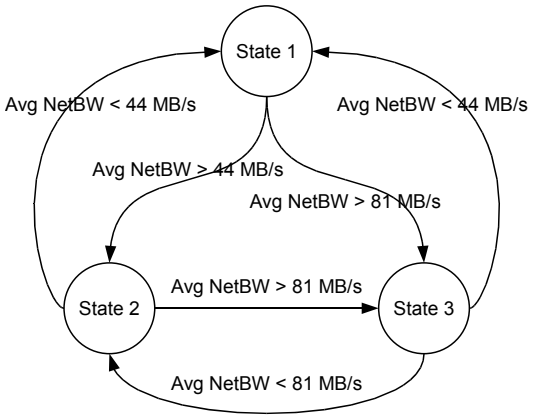


Figure 8.6: Global behavior model of the test scenario

The resulting FSM is again presented in Figure 8.6. It is important to remind that this is an automatically generated model, and the state analysis took place after its construction. This ensures that the resulting states are based only on the behavior information monitored and not on any system administrator's personal assumptions. The three observed states are:

- **State 1:** It is characterized by a low average network bandwidth (below 44 MB/s), mostly due to network overload.
- **State 2:** It is characterized by a medium average network bandwidth (between 44 and 81 MB/s). It seems to represent the medium load state of the grid.
- **State 3:** It is characterized by a high average network bandwidth (over 81 MB/s). This represents a barely loaded grid, where the network can be used at full capacity.

From a service point of view, state 1 seems to be the most problematic, as low network bandwidth can make the data input and output transfer times longer and therefore increase the job's failure probability. State 3, on the other hand, seems like the best one, as the high network bandwidth guarantees fast data transfers. State 2 certainly is in an intermediate point.

In order to increase dependability, FIRE needs a set of policies adapted to each state. In this case, a set of job scheduling policies was configured, aimed to improve system's dependability. To make the example easier to understand, the chosen policies were very simple but still effective:

- **Policy A** gave a higher priority to jobs that had small input and output data. This reduces the chances of job failure when the network is slow. This policy was configured for **state 1**.
- **Policy B** gave a higher priority to jobs that had large input and output data. This was specifically designed to take advantage of times when network bandwidth is very high. The idea is to execute the heavier jobs when their success chance was higher. Obviously this policy was configured for **state 3**.
- **Policy C** dispatched jobs in strict arrival order (FCFS). It was configured for **state 2**.

8.2.2.2 Simulation results

Each scenario (20R, 50R and 100R) was simulated using the basic FCFS scheduling policy and the special multi-policy scheduler controlled by FIRE. 16 simulations (each of them using different random seeds) were performed for each scenario and scheduler system, giving a total number of 480 days of simulated time for each experiment (every execution simulated 30 days).

The average job failure rate results for each experiment can be seen in Figure 8.7, grouped by scenario. As it has been said, the job failure rate for the FCFS configuration was fixed to 0.16, in order to produce a value observed on a real grid environment (EGEE, in this case). The FIRE configuration, as it can be seen, clearly reduces the job failure rate in every experiment.

A more detailed analysis is displayed in Figure 8.8. In this case, every scenario configuration is displayed separately but, in each of them, separated failures rates are displayed for each state. It is clear now that the state where most of job failures occur is state 1. The multi-policy based FIRE configuration succeeded in lowering this state failure rate.

As a curious detail, it can also be seen in Figure 8.8 that the use of the multi-policy FIRE configuration very slightly increases the state 2 failure rate. Although this does not affect the overall result,

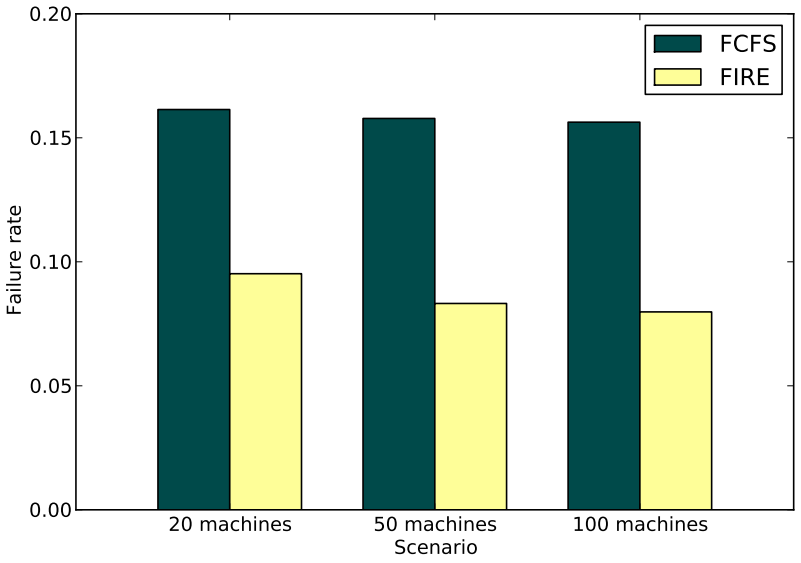


Figure 8.7: Job failure rate for each scenario and policy configuration

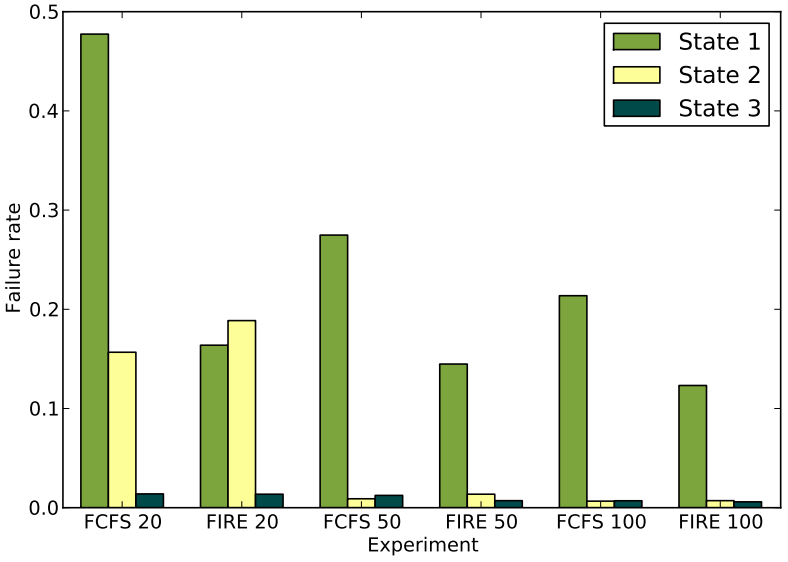


Figure 8.8: Job failure rate for each state in each experiment

it is interesting to provide an explanation for this phenomena. It is important to remember that the associated policy to state 1 (policy A) increases the small jobs priority, making them more likely to be executed. In consequence, this makes that most small jobs are executed while this policy is active, and when the system returns to state 2, the average job size in the queue could very posibly be higher. Therefore the jobs executed during state 2 are generally bigger than in the FCFS configuration and therefore their chance of failure is higher. This slightly increases state 2 failure rate. Even though, this increase has little effect in the overall failure rate.

Chapter 9

Grid global behavior prediction

Global behavior modeling has proved to be very useful in effectively managing grid complexity but, in many cases, deeper knowledge is needed. GloBeM's descriptive model could be greatly improved if extended not only to explain behavior, but also to predict it. Grid management can benefit from global behavior prediction, specially in areas such as fault tolerance or job scheduling. In this chapter a grid global behavior prediction methodology based on GloBeM is presented. Its objective is to define the techniques needed to create global behavior prediction models for grid systems. Experimental results are also presented, obtained in real scenarios in order to provide a proper validation and illustrate the benefits of this approach.

9.1 Predicting global behavior: Initial considerations and a-priori study

GloBeM models provide very useful information about the system behavior, but they are strictly descriptive in nature. They can be used to analyze, understand and optimize a grid, but they provide very little knowledge about the system evolution over time and/or its future state. In order to further increase their usefulness, GloBeM models could be combined with predictive techniques, capable of foreseeing future events. This would enable the management system to act before such events actually occur, avoiding global faults or any other possibly dangerous situation and improving general performance and/or dependability.

In this chapter a set of algorithms are presented, designed to create global state prediction models in terms of GloBeM behavior descriptions. They are based on machine learning and time series analysis techniques. A set of basic elements can be distinguished in all of them:

- The **set of grid states** $S = \{s_1, s_2...s_n\}$.
- The **behavior model** $B(t)$ generated using the GloBeM methodology. It describes the grid states S and the events that cause a transition from one state to another. At any instant t, $B(t) = s_k \mid s_k \in S$, where s_k is the grid state in that instant.
- The **prediction model** $P(t)$ that predicts the futures states indicated by the behavior model. At any instant t, $P(t) = B(t + 1)$.
- The **training data**. This is the historical grid monitoring data set used to create the behavior and prediction models. It contains a log of values of the monitoring parameters used by the behavior model in order to determine the current state and the associated global state.
- The **test data**. This is a different set of historical grid monitoring data. Although it is similar to the training data, it is much larger and it is used to evaluate the prediction model accuracy.

In basic terms, given a set of current monitoring values, the behavior model indicates, among other things, the current grid state, but it provides no information about the future. Given the same set of values (and a history of past ones too) the prediction model will be able to predict the future state. The accuracy of this prediction will depend on the quality of the training data and the algorithm used to generate the prediction model. The test data is used to estimate this accuracy, based on two statistic parameters of any given prediction model $P(t)$:

- Average percentage of correct predictions, $AC(P)$.
- Average percentage of transitions correctly predicted, $AC_T(P)$. This indicates how many actual state transitions were correctly anticipated by the prediction model.

The second value is important because it has been demonstrated that GloBeM's models of these environments tend to be very stable [MSV+10] (GloBeM hides the vast complexity of a large distributed system such as a grid) and state transitions represent major changes in the system behavior. From a general perspective, predicting global state transitions could be the key to improve grid management in many areas (job scheduling, dependability and fault tolerance, etc). In most cases a change of global state will require a change in the management policies, specially to prevent undesirable situations or states where some service requirements are not met (faults and/or failures, decreases in quality of service, etc). A prediction model strongly benefits the management system in these critical situations (transitions), making it possible to anticipate and act ahead of faults and changes. Naturally, the best prediction models would score highly on $AC(P)$ and $AC_T(P)$ values.

9.1.1 *A-priori* study

In order to obtain a basic framework, an *a-priori* study was made. For this study real monitoring data from PlanetLab [pla] were used. As already explained, Planetlab presents all the heterogeneity, complexity and variability expected from any real grid computing infrastructure, and therefore it is an excellent scenario for testing the global behavior prediction techniques presented here.

Again using the PlanetLab monitoring tool CoMon, many different parameters were monitored, including CPU usage, memory usage, network traffic, architecture characteristics, I/O operations, an so on. Information from this tool is being gathered in order to create a comprehensive monitoring database of the historical evolution of PlanetLab. For this study a total of 8 months of PlanetLab monitoring information from that database were used. Data was aggregated in 1 hour monitoring intervals and divided in many subsets, in order to produce an extensive collection of training sets. Subsets sizes ranged from 10 to 110 days, with different degrees of overlapping between them. Altogether a set of 220 training subsets was created.

Using this training data set, different behavior models were produced in order to explain the behavior observed in the whole data set. Then, for each behavior model, statistics about percentage of transitions and state stability were calculated.

Table 9.1: A-priori study results

	mean	**standard deviation**
Stable periods	90.1%	2.51
State transitions	9.9%	2.51
Stable period duration	$18.28h$	8.22

Summary values of the *a-priori* study are presented in Table 9.1. As can be seen, the average number of state transitions is quite low ($\simeq 10\%$), which illustrates the previously stated idea that GloBeM models are very stable, with few but relevant transitions. These transitions, however, are crucial events, representing major changes in the system behavior that normally involve clear modifications in aspects such as performance or dependability. From a management point of view, these are the key situations that need to be anticipated, creating the need of a prediction model capable of foreseeing state transitions.

Furthermore, as part of the *a-priori* study, a basic predictor was constructed, in order to provide a basis for evaluation and comparison. This was called the *naïve* predictor.

The *naïve* predictor

As the simple prediction model reference for the *a-priori* study, the *naïve* predictor $P_N(t)$ was defined in the following terms:

$$P_N(t) = B(t) \tag{9.1}$$

This basically means that $P_N(t)$ will always predict the future state to be the current state, as given by the behavior model. In consequence, the prediction will be correct as long as no state transition occurs. When the transition takes place, the $P_N(t)$ predictor fails, as it always expects the state to remain stable.

The accuracy of this *predictor-that-does-not-predict* will obviously depend on the stability of the system, as it only fails when transitions occur. A study of the $AC(P_N)$ and $AC_T(P_N)$ values would provide a basic frame of reference for prediction models evaluation, defining when predicting is actually better than a simple descriptive approach with no anticipation.

Table 9.2: Naïve predictor accuracy metrics

$AC(P_N)$	90.1%
$AC_T(P_N)$	0.0%

Using the 220 PlanetLab monitoring training sets to generate different behavior models, the accuracy of the *naïve* predictor was evaluated. Table 9.2 shows the predictor accuracy metrics for $P_N(t)$. As it can be seen in table 9.2, even though it is incapable of predicting any transitions ($AC_T(P_N) = 0\%$) the total average is very high ($AC(P_N) \simeq 90\%$). This is consistent with the statistical results presented in Table 9.1. The system is very stable with very few state transitions. Nevertheless, detecting these transitions is our main objective, as these are the relevant events that are identified and give meaning to the GloBeM behavior model. The *naïve* predictor is incapable of doing that, but, even so, it is capable of correctly anticipating the future state 90% of the time. This is worth keeping in mind when evaluating prediction models, because sometimes the best option could be simply not to predict, and just use the $P_N(t)$ instead.

9.2 Global behavior predictors

As was explained, state transitions in GloBeM behavior models indicate crucial events in the system, usually requiring the adaptation of global management policies. In this section two approaches are presented to global state prediction, in order to anticipate future states and state transitions in a

grid system. The first one is a basic, single variable prediction strategy, based on traditional time series analysis techniques and machine learning. The second one is a far more complex, multi-stage approach, introducing some advanced concepts.

9.2.1 Basic predictor

Considering the system's global state as a variable, at a given time t $B(t) = s_t \mid s_t \in S$, and therefore s_{t-1} would be the state at time $t-1$, s_{t-2} the state at time $t-2$, and so on. We can consider the associated time series as:

$$S_t = \{s_t, s_{t-1}, s_{t-2}, s_{t-3}, ...\}$$

For any given instant in time t, S_t will contain the past and present state values of the system, showing its historical evolution.

Using traditional time series analysis techniques, we define our basic predictor model as a function capable of calculating the future state based on the present and past values of the global state time series variable S_t:

$$P_B(t) = f(s_t, s_{t-1}, s_{t-2}, s_{t-3}, ...) \tag{9.2}$$

In practical terms, there is only so many instants in the past that can be considered and therefore we redefine $P_B(t)$ as:

$$P_B(t, w) = f(s_t, s_{t-1}, s_{t-2}, ..., s_{t-w}) \tag{9.3}$$

where w is the number of past values considered in the prediction. We call w the **predictor window**.

The $P_B(t, w)$ algorithm consists of three distinct phases, aimed at creating a prediction model for a GloBeM behavior model. These phases are illustrated in Figure 9.1 and described below:

1. **Training data classification**: Using the behavior model, the training data are classified, in order to determine the state associated to each monitoring instant. The result is an extended version of the training data set, including the state variable along with the monitoring parameters.

2. **Time series selection**: The values from the state variable in the training data set are selected, generating the S_t time series.

3. **Machine learning**: Using a machine learning algorithm, a prediction model is trained using data from the S_t time series. The number of past values the machine learning algorithm can include in its calculations is determined by the w value defined above.

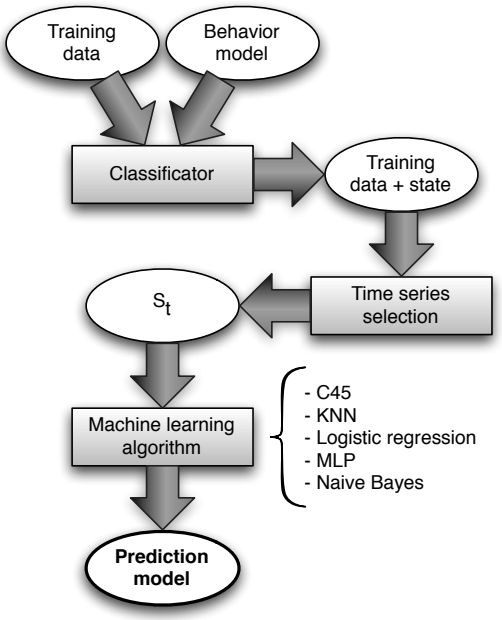


Figure 9.1: Basic predictor phases

The result is a prediction model $P_B(t)$ for the S_t time series. The exact form of this model depends on the machine learning algorithm used. At this point, instead of selecting one specific algorithm, we have proposed the following set of them (see Section 4.3 for details):

- C4.5.
- K-Nearest Neighbors (KNN).
- Logistic regression.
- Multi-Layer Perceptron (MLP).
- Naïve Bayes.

Our objective is to provide an extensive set of machine learning algorithms, in order to present a study as complete as possible. The five selected techniques are well known, widely used and scientifically relevant. In Section 9.3, experimental results are presented to illustrate which machine learning technique is more adequate in our case, and the overall performance of the basic predictor.

9.2.2 Multi-stage predictor

After the basic predictor $P_B(t)$ was developed, the need of a more advanced prediction technique appeared, motivated by several issues.

First, as shown in the *a-priori* study, the amount of state transitions observed in the GloBeM models is quite low. Training data sets are composed mostly of data that represent stable instants where no transition takes place. When this training data sets are used in machine learning algorithms, they usually lead to prediction models that are over-fitted to predict stability and less concerned with transitions. In situations where the disproportion among stable instants and transitions is extreme, the machine learning algorithm basically disregards transitions, as they represent a very uncommon situation. This could lead, in the end, back to a $P_N(t)$ *naïve* predictor.

Second, the behavior state variable (and its associated time series S_t) is clearly dependent on the monitoring parameters, as it is derived from them by the GloBeM model. This information is not included in the $P_B(t)$ model, which limits its efficiency.

In order to deal with these issues, a more complex predictor was developed. We consider again the state time series S_t and the predictor window w. We incorporate also the set of monitoring variables $\{V_1, V_2, ..., V_M\}$ selected by GloBeM (not all monitoring parameters but only the relevant ones accoding to the global behavior analysis) and the associated time series for each one:

$$\begin{aligned} V1_t &= \{v1_t, v1_{t-1}, v1_{t-2}, ...\} \\ V2_t &= \{v2_t, v2_{t-1}, v2_{t-2}, ...\} \\ &... \\ VM_t &= \{vm_t, vm_{t-1}, vm_{t-2}, ...\} \end{aligned}$$

We define the predictor $P_M(t)$ as a function of the values of $S_t, V1_t, ..., VM_t$:

$$P_M(t) = f(s_t, ...s_{t-w}, v1_t, ...v1_{t-w}, ...vm_t, ...vm_{t-w}) \tag{9.4}$$

As can be seen, the first difference between $P_M(t)$ and the previous $P_B(t)$ is that the monitoring parameters are also considered in the prediction, and not just the present and past state. In addition, $P_M(t)$ improves the transition prediction accuracy by means of a multi-stage prediction process. This process structure can be seen in Figure 9.2. The multi-stage predictor is composed of three basic elements:

- The **metapredictor** $MP(t)$ is a prediction model trained not to predict the future state but just state transitions. It is capable of foreseeing whether the system is going to change state, but not the specific state it is going to transit to.

- The ***naïve* predictor** $P_N(t)$, as defined in Section 9.1.1, is used when the metapredictor indicates that no transition is going to happen. This strongly simplifies the prediction process in those cases, as no prediction is really made.
- The **transition predictor** $P_T(t)$ is a prediction model specifically trained to anticipate only state transitions. It is trained using monitoring data only from instants in time when state transitions happen, and therefore it is generated specifically for those situations. The transition predictor is used when the metapredictor anticipates a transition, maximizing the probability of correct prediction in those cases without affecting the general prediction accuracy.

As can be seen in Figure 9.2, the multi-stage predictor uses its metapredictor to determine if transitions are going to happen. In case a global state transition is anticipated, the multi-stage predictor then relies on its transition predictor to determine the future state. In case no transition is foreseen, the multi-stage predictor simply anticipates no change, providing the *naïve* predictor result as its final prediction.

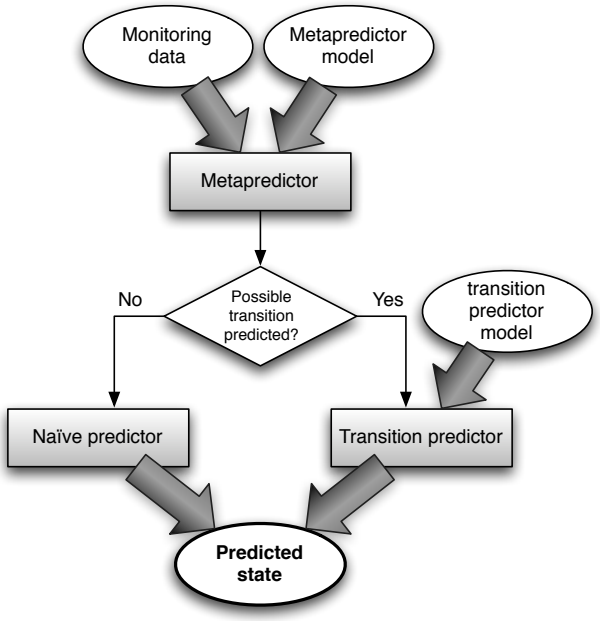


Figure 9.2: Multi-stage predictor

9.2.2.1 The metapredictor

The construction of the metapredictor model is carried out in four phases, as shown in Figure 9.3. The process is similar in essence to the one previously described for the basic predictor, but more complex. The four metapredictor model construction phases are:

1. **Training data classification**: In the same way as for the $P_B(t)$ model, the training data are classified using the behavior model, in order to determine the state associated to each monitoring instant.

2. **Time series selection**: In this case not only the values from the state variable are selected, but also the ones from the monitoring variables, creating the time series set $\{S_t, V1_t, ..., VM_t\}$.

3. **Undersampling and attribute selection**: In order to increase the quality of the time series training set generated, two special refining techniques are used in this phase. These are *undersampling* and *attribute selection* and they are described in detail below.

4. **Machine learning**: Finally the machine learning algorithm is executed in this phase, in a similar way as in the basic predictor. As in that previous case, we selected five possible algorithms to be used: C4.5, KNN, logistic regression, MLP and Naïve Bayes.

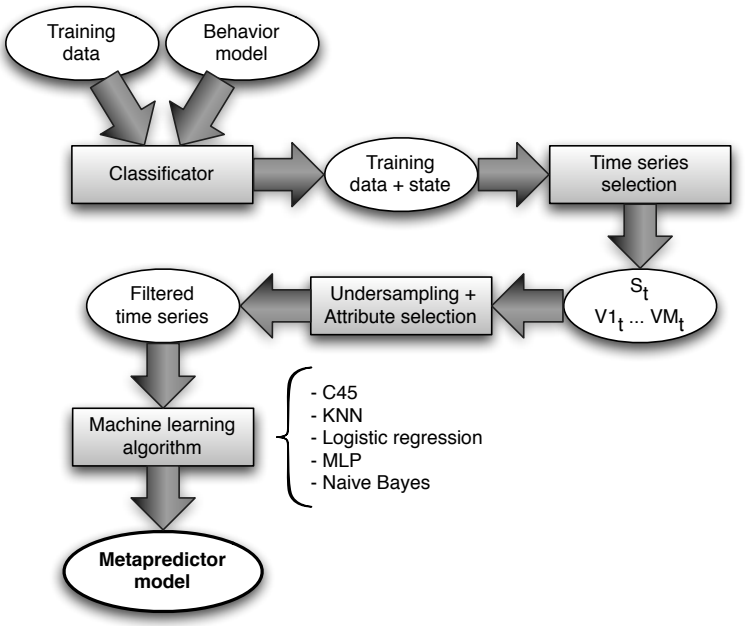


Figure 9.3: Metapredictor model construction phases

Undersampling is a data filtering technique commonly used in machine learning procedures where, given a classification of a training set, the proportions in which each class appears are clearly uneven. In our case, if we divide the data set in *stable instants* and *state transition instants*, we find out most of them belong to the first group. As was explained before, in these cases machine learning algorithms tend to focus only on the majority class (stable instants, in our case), almost completely ignoring the minorities. To avoid this phenomenon, the majority class is reduced to a statistically

significant subset of values, representative of the whole group but of a size similar to the minority groups (or at least not so overwhelmingly larger). This gives the machine learning technique a chance to correctly identify all classes.

To achieve this we used the K-Means clustering algorithm [Mac67] (see Section 4.2.2). K-Means classifies the given data in a specified number of classes, with similar observations assigned to the same class. As a result, it produces a list of representative values, called centroids, one for each class. To undersample the metapredictor training data set, the observations that represent stable instants (much more frequent than the ones representing state transition instants) are firstly separated and then classified using K-Means. The metapredictor algorithm sets a number of classes for the K-Means algorithm to the number of state transitions observed, and takes the resulting centroids as representative observations. This a widely used, generic undersampling methods, capable of providing satisfactory results in may different data analysis problems.

A second training set optimization carried out in the metapredictor construction algorithm is attribute selection. As shown in (9.4), the $P_M(t)$ model is defined from a function of many parameters, basically the present and past values of the system's global state and monitoring parameters, given a certain predictor window w. When the number of monitoring parameters and w is high, this will originate a function with a very large set of parameters. Not all these parameters are statistically relevant for prediction purposes, but nevertheless they increase the training data set size, making the subsequent machine learning process difficult.

In order to select only the statistically representative parameters for the machine learning process, the metapredictor algorithm calculates the autocorrelation coefficients for each input time series.

Autocorrelation coefficients [Cha03] are a commonly used time series analysis tool. They indicate the correlation (usually the Pearson correlation coefficient) between present and past values of a time series, at any given time. For instance, a time series of a variable whose value at any time is dependent only on its last two values will score closer to 1 in its two first autocorrelation coefficients and close to 0 in the rest. Calculating these coefficients will indicate that no other past observations are needed in order to predict the variable value.

In the metapredictor construction, the first w autocorrelation coefficients are calculated for each time series used ($\{S_t, V1_t, ..., VM_t\}$). Then, only the relevant historical values of each series are selected, effectively reducing the number of parameters provided to the machine learning algorithm.

Finally, the machine learning algorithm is configured to generate a model that only predicts whether the system global state is going to remain stable, or a transition will occur. The final model produced is called the **metapredictor model**.

9.2.2.2 The transition predictor

The second part of the multi-stage predictor is the transition predictor $P_T(t)$. The construction of this prediction model takes place in the following six phases, also shown in Figure 9.4:

1. **Training data classification**: In the same way as for the $P_B(t)$ and $MP(t)$ models, the training data are classified using the behavior model.

2. **Time series selection**: Like in the case of $MP(t)$, the state and monitoring variables are selected, creating the time series set $\{S_t, V1_t, ..., VM_t\}$.

3. **Transition selection**: At this point, the time series training set is filtered, in order to select only values related to state transitions. This creates a data set containing only specific information about global changes of state.

4. **Time series differencing**: The time series are differenced in order to remove from them any trend or other unnecessary information that could affect the subsequent machine learning process. This process is explained in detail below.

5. **Attribute selection**: In a similar way as in the case of $MP(t)$, time series data set attributes are selected using autocorrelation coefficients.

6. **Machine learning**: Finally the machine learning algorithm is executed in this phase, in a similar way as in the previous predictors. Again we selected the same five possible algorithms to be used: C4.5, KNN, logistic regression, MLP and Naïve Bayes.

Differencing is a commonly used time series analysis tool. Its objective is to eliminate any possible trend in the series, leaving only relevant information about changes in the variable. From a general perspective, if we consider the time series $X_t = \{x_t, x_{t-1}, ...\}$, first order differencing X_t consists in replacing it with a new series Y_t defined as:

$$Y_t = \{y_t, y_{t-1}, ...\} \quad \forall y_k \in Y_t, y_k = \nabla x_k = x_k - x_{k-1}$$

During construction of the transition predictor, first order differencing is applied to all numeric series in the time series training data set in order to provide only useful information to the machine learning algorithm.

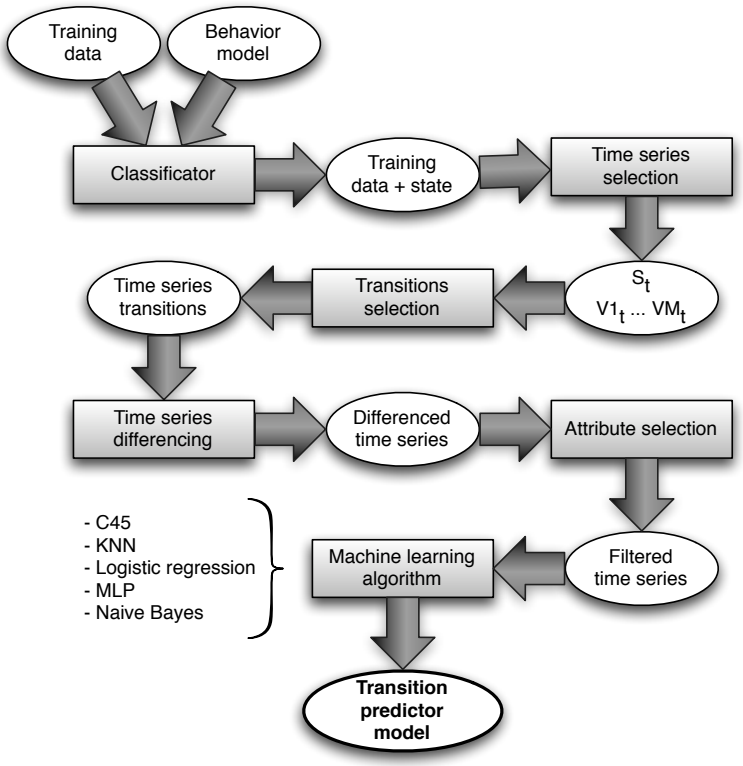


Figure 9.4: Transition predictor model construction phases

Once the six previously explained phases take place, the result obtained is a predictor model specifically trained to detect global state transitions. When it is incorporated inside the multi-stage predictor, it is only used when the metapredictor model indicates a transition will occur. The combined use of $P_N(t)$, $MP(t)$ and $P_T(t)$ carried out by the multi-stage predictor generates a more efficient prediction model than $P_B(t)$, specially anticipating global state transitions. In the following Section 9.3 an experimental study is presented, evaluating and comparing the different approaches.

9.3 Experimental results and evaluation

Using the same PlanetLab scenario described in the *a-priori* study (see Section 9.1.1), a series of experimental tests were performed. The objective of these tests was to evaluate the accuracy of the different prediction algorithms proposed, using the previously defined metrics $AC(P)$ and $AC_T(P)$. The general characteristics of the test series can be seen on Table 9.3.

As is presented on the table, several different predictor window values were used. Also training sets of different sizes were included in the experiment series, in order to generate results that are as

Table 9.3: Experiment characteristics

Total size of test data	8 months
Data time resolution	1 hour
Size of training data	60, 70, 80, 90 or 100 days
Total number of training models	100 (20 of each size)
Predictor window (w)	10, 20, 30, 40 or 50 hours
Total number of configurations	500

complete as possible. Each experiment was generated using a specific training set with a fixed w value, giving a total number of 500 experimental configurations.

9.3.1 Basic predictor evaluation

The basic predictor $P_B(t)$ was evaluated using the experiment series previously described. Each experiment was performed using the five machine learning algorithms considered (C4.5, KNN, Logistic regression, MLP and Naïve Bayes) and the $AC(P_B)$ and $AC_T(P_B)$ values were calculated.

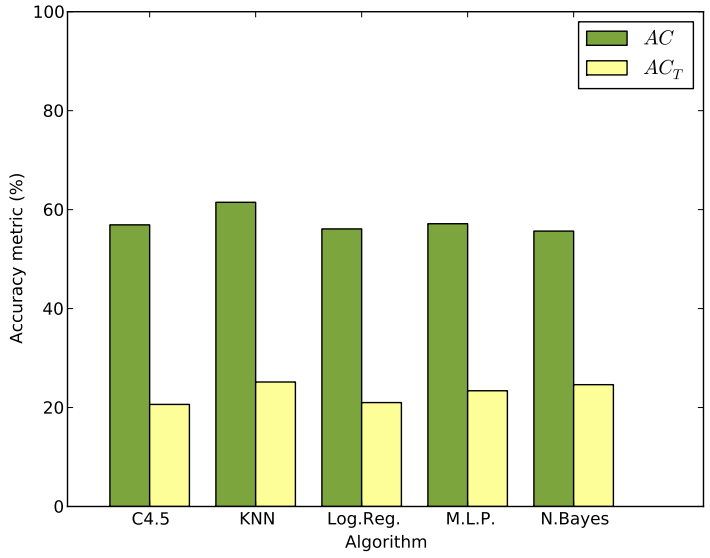


Figure 9.5: Basic predictor results

Figure 9.5 shows the average results obtained, separated by machine learning algorithm used. The issues previously anticipated in Section 9.2.2 can be clearly seen here, causing a reduction in the predictor accuracy. The $AC(P_B)$ value ranges between 55% and 62% and the $AC_T(P_B)$ between 20% and 25%. Even though the predictor is capable of anticipating a few transitions, the total and transition accuracy is too low to be considered acceptable. This results clearly justify the need for a

more complex approach.

9.3.2 Multi-stage predictor evaluation

In a similar fashion to the basic predictor, the multi-stage predictor was evaluated, using the five suggested machine learning algorithms. In this case the two stages have to be considered (metapredictor and transition predictor) and evaluated separately, before calculating the overall accuracy of the multi-stage predictor.

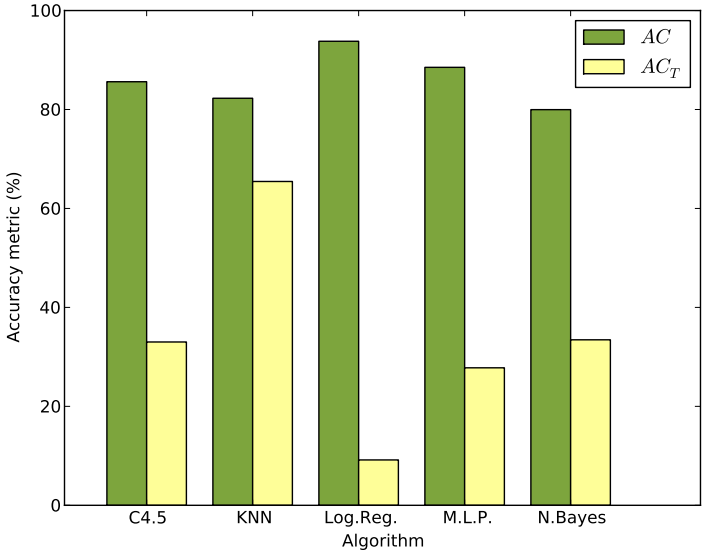


Figure 9.6: Metapredictor results

Figure 9.6 shows the average results obtained for the metapredictor. It is important to remember that this model does not predict the future global state itself, but just determines whether a transition is going to happen or not. Therefore the $AC(MP)$ and $AC_T(MP)$ values have to be understood accordingly.

The histogram shows that not all machine learning algorithms are equally capable of generating good metapredictor models. The total average value $AC(MP)$ is generally very good (between 80% and 94%) but the $AC_T(MP)$ values are uneven. KNN turns out to be the only machine learning algorithm capable of producing models with fairly good values in both $AC(MP)$ and $AC_T(MP)$ metrics. These results strengthen the idea that state transitions are a very rare, difficult to predict event. Providing some capabilities to anticipate them is an extremely complicated task, even with the appropriate combination of tools.

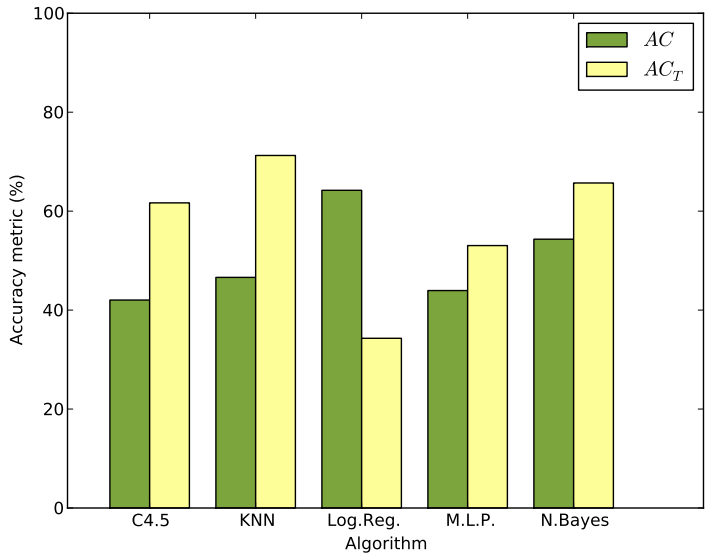


Figure 9.7: Transition predictor results

The evaluation of the transition predictor was performed in a similar way, and the results are summarized in Figure 9.7. $AC(P_T)$ and $AC_T(P_T)$ metrics are displayed, even though only the latter is interesting in this case. It is important to remember that this is a prediction model specifically designed to be used only when transitions are anticipated, and therefore it is adapted only to those events. The histogram shows an improvement in the AC_T metric if compared to its previous values for the $P_B(t)$ and $MP(t)$ models, but not in the cases of all machine learning algorithms. This further strengthens the idea that state transitions are very difficult to predict, because even specialized models are not always good at detecting them. Even so, the C4.5, KNN and Naïve Bayes algorithms produce fairly good results (62%, 71% and 67% respectively) that can be used inside the multi-stage predictor.

The metapredictor and transition predictor evaluations just described show that KNN is the machine learning algorithm that produces better results in both cases. Using this algorithm, the multi-stage predictor was constructed and evaluated using the previously described experiment series. The summarized results can be found in Table 9.4.

Table 9.4: Multi-stage predictor results (KNN)

$AC(P_M)$	87.61%
$AC_T(P_M)$	62.5%

The final multi-stage predictor achieves a very high total average of correct predictions, which guarantees its overall accuracy. Plus, it is capable of correctly anticipating almost two thirds of the global state transitions, which is a fairly good value, given the intrinsic difficulty of anticipating this rare but critical events.

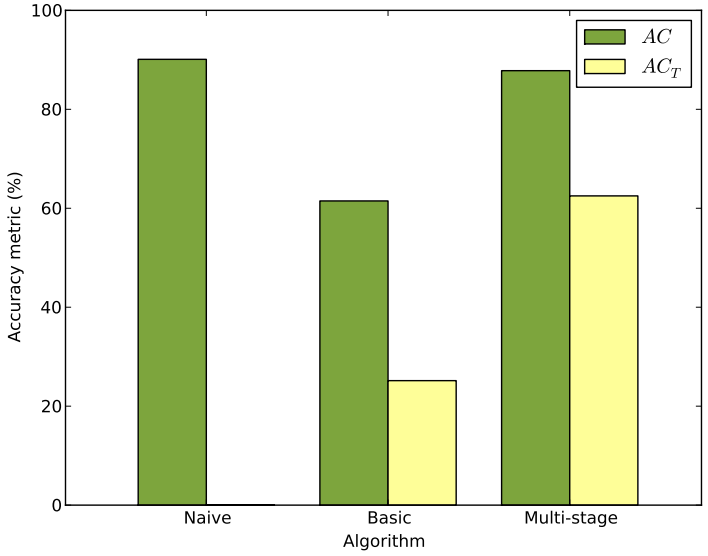


Figure 9.8: Compared predictor results

Finally, Figure 9.8 shows a general comparison of the simplistic *naïve* predictor and the two prediction algorithms presented in this chapter (the KNN results have been selected in the case of the basic predictor). As a curious detail, the *naïve* predictor gets the highest AC value (90.1%), even higher than the multi-stage approach (87.61%). This small difference is regarded as not significant, specially when the AC_T metric is included in the comparison. Given the explained importance of global state transitions in terms of grid management, the improvement shown by the multi-stage predictor in this area (62.5% of correctly anticipated transitions) clearly makes it the best approach for global state prediction.

Part IV

CONCLUSIONS AND FUTURE WORK

Chapter 10

Conclusions

The work presented in this Ph.D. thesis has successfully fulfilled the objectives defined at the beginning of this document (Section 1.2). These accomplishments clearly validate the initial hypothesis, allowing to state that **a high-level, global, unified and service-oriented model of the grid behavior can benefit its management, improving its autonomic capabilities and providing the necessary abstraction to make single-entity vision possible**. In this chapter the most relevant conclusions of this work will be summarized, emphasizing the most important achievements.

10.1 System behavior analysis

One of the main contributions of this Ph.D. thesis is the GloBeM methodology (Chapters 5 to 7). GloBeM combines monitoring techniques and performance metrics with advanced data mining and other knowledge discovery techniques in order to analyze and model the behavior of a large scale distributed system (a grid, in this case). The methodology is designed to produce a service-level, global representation of the system's total state, focusing on an innovative single entity vision. This is, so far, one of the few attempts to model the whole grid behavior in such way, and the only one based on real, monitored performance information and not on theoretical models and generic approaches. GloBeM makes no initial assumptions and simply models what is observed, simplifying the subsequent analysis and optimization. The main advantages of this approach are:

- The model which is built by means of GloBeM's methodology can be easily interpreted by a system manager or administrator.

- The model characteristics simplifies its use within management tools in order to improve the performance of the grid.
- The model is built in an autonomous way. No human interaction is required.

Additionally, GloBeM has been designed as a modular and flexible process. This methodology is more than a closed approach, easily allowing future improvements and adaptations. New knowledge discovery techniques can be incorporated, complementing or replacing the existing ones, other monitoring techniques can also be integrated, etc. Also, the methodology can be easily adapted to other kinds of systems, not limiting itself to grid infrastructures. Other platforms such as clusters or supercomputers could probably benefit from global behavior analysis, in a similar way the grid does.

Chapter 7 describes two use cases (Section 7.2), one in a simulated scenario and another one in a real scenario. An understandable model has been found in both cases. The stability of the models has also been proven. Furthermore, some examples of application of the model have been also shown, emphasizing the importance of this initiative.

Finally, Chapter 9 presents an important extension to the GloBeM model, incorporating behavior prediction capabilities, anticipating crucial changes in system behavior. As was explained in Section 9.1, the process of predicting these crucial changes is not an easy task, since grid systems behave in a stable way, from a global modeling perspective. Given the stability of behavior models, transitions or behavior changes rarely occur and, therefore, are difficult to predict. This makes any basic statistical predictor incapable of finding such changes. The multi-stage predictor proposed in Section 9.2.2 is capable of predicting a high percentage of these transitions, as well as being able to recognize the system stability, as described in Section 9.3. Consequently, the prediction proposed can significantly benefit grid management systems, enabling one to act ahead of system changes and to select suitable management policies to deal with those changes before they occur.

10.2 Grid single entity vision

Also a major contribution of this Ph.D. thesis is the abstraction mechanism that makes the single entity vision of the grid finally possible. This is particularly important from a theoretical perspective, since it answers one of the basic question that raised when developing the initial hypothesis and motivations of this work: **Is the grid really a single system?** if it is, **can it be studied, not only in abstract form, but more practically as a single entity?**.

The new single entity abstraction presented in this thesis clarifies these questions, providing a

solid, useful representation of the whole grid behavior that can be used for analysis, modeling and management purposes. It is also a service-level vision, modeling the system's total state as an extended finite state machine. This provides a familiar representation of the grid operation, simplifying the model's use and revealing only relevant aspects of both system's structure and function.

10.3 Global autonomic management

GloBeM provides the necessary methodology to create a single entity behavior model of the grid. This is an important theoretical achievement but its practical scientific relevance is proven only when its usefulness as an analysis and management tool is demonstrated. The implications that this single entity, service-level model can have in autonomic system management are studied in Chapters 5 and 6, but is in Chapter 8 where its benefits are fully displayed in several experimental scenarios.

Section 8.1 shows how a general grid problem (storage services quality) can be addressed using global behavior modeling. The massively parallel data accesses issued by grid applications place a heavy burden on the storage service which has to react efficiently. To improve the quality of service provided by the storage service, it is required to reason about its behavior in order to identify potential bottlenecks. However, the complexity of the system's behavior makes this problem difficult. A lot of different factors affect the behavior simultaneously: highly-concurrent data access patterns, long periods of service uptime, failures of physical components, the highly distributed nature of the storage service itself, etc.

GloBeM's models allow us to analyze and model the behavior of the grid's storage service, providing the necessary clues to develop efficient performance improvements and eliminate possible bottlenecks. The benefits of this approach are clear, obtaining improvements in service bandwidth both in amount and stability, as the experiments shown in Section 8.1.3.2. It is specially important to remember the benefits of improving the bandwidth stability: a higher quality of service at this level makes the cost of access operations more predictable, improves the efficiency of scheduling algorithms used by data-intensive processing frameworks and helps optimizing the overall application throughput. These series of experiments are a very clear, descriptive example of how global behavior modeling can contribute to grid autonomic management, enabling to introduce self-optimizing capabilities.

In Section 8.2 the contribution to system autonomic management is extended, presenting FIRE, an autonomic framework for system management based on global behavior modeling. FIRE is a general purpose management software that provides a basic infrastructure for building grid autonomic

systems based on the single entity vision presented in this thesis central hypothesis. Its usefulness is also illustrated with an experimental scenario, focused in this case on self-healing related issues. This approach provides fault tolerance based on a global behavior model generated by GloBeM. On the one hand, the use of GloBeM simplifies the decision making tasks over the system. On the other hand, the FIRE framework enables the proper application of management policies, as Section 8.2.2 shows. Consequently, this improves significantly the system's dependability.

10.4 Selected publications

To conclude this chapter, the most relevant publications produced during the development of this Ph.D. thesis are referenced here.

10.4.1 International Journals

- Alberto Sánchez, María S. Pérez, Pierre Gueant, Jesús Montes, Pilar Herrero, and Toni Cortes. Improving GridFTP transfers by means of a multiagent parallel file system. Multiagent and Grid Systems, 3(4):441–451, 2007.

- Alberto Sánchez, María S. Pérez, Jesús Montes, and Toni Cortes. A high performance suite of data services for grids. Future Generation Computer Systems, 26(4):622 – 632, 2010. Current JCR IF: 1,467.

- Jesús Montes, Alberto Sánchez, Julio J. Valdés, María S. Pérez, and Pilar Herrero. Finding order in chaos: a behavior model of the whole grid. Concurrency and Computation: Practice and Experience, page In press., 2009. Current JCR IF: 1,79.

10.4.2 Book Chapters

- Alberto Sánchez, Jesús Montes, Werner Dubitzky, Julio J. Valdés, María S. Pérez, and Pedro de Miguel. Data mining meets grid computing: time to dance? In Werner Dubitzky, editor, Data Mining in Grid Computing Environments, pages 1–16. Wiley Publishing, 2008.

10.4.3 Internartional conferences and other publications

- Alberto Sánchez, María S. Pérez, Pierre Gueant, Jesús Montes, and Pilar Herrero. A parallel data storage interface to GridFTP. In Robert Meersman and Zahir Tari, editors, OTM Conferences (2), volume 4276 of Lecture Notes in Computer Science, pages 1203–1212. Springer, 2006.

- Alberto Sánchez, Toni Cortés, Jesús Montes, Pierre Gueant, and María S. Pérez. Lessons learnt from cluster computing: How they can be applied to grid environments. In 8th Hellenic European Research on Computer Mathematics and its Applications, Athens, Grece, 2007.

- Jesús Montes, Alberto Sánchez, Julio J. Valdés, María S. Pérez, and Pilar Herrero. The grid as a single entity: Towards a behavior model of the whole grid. In Robert Meersman and Zahir Tari, editors, OTM Conferences (1), volume 5331 of Lecture Notes in Computer Science, pages 886–897. Springer, 2008.

- Jesús Montes, Alberto Sánchez, and María S. Pérez. GloBeM: Un modelo global del grid. In XX Jornadas de Paralelismo, A Coruña, Spain, 2009.

- Jesús Montes, Alberto Sánchez, and María S. Pérez. Improving grid fault tolerance by means of global behavior modeling. In 9th International Symposium on Parallel and Distributed Computing (ISPDC) 2010, Accepted (to appear).

Chapter 11

Future work

The work presented in this Ph.D. thesis addresses the areas of grid behavior modeling and autonomic management. It satisfactory fulfills the initial objectives and hypothesis presented in Chapter 1, exceeding its original expectations and producing results of scientific relevance. However, a series of further improvements and new experimentation and analysis could be considered as new research initiatives, inspired by this work.

In this chapter, some of the most relevant future lines and open issues at the end of this work will be enumerated. Some of them deal with the addition of new techniques, whereas others are related to the inclusion of new structural variations in the GloBeM+FIRE framework. The application of the proposed methodology to new problems is always a good idea, as well as the implementation of new quality and analysis measures. The most relevant of these alternatives are described in the following sections.

11.1 Behavior prediction extension and further validation

The basic GloBeM's single entity abstraction is extended in Chapter 9 to incorporate prediction capabilities. This turns the initial **descriptive model**, which represents the observed behavior, highlighting important aspects and relevant events, into a **predictive model**, capable also of foreseeing

system changes before they occur. The experimental results presented in Section 9.3 focus on the statistical properties of the different prediction models shown, describing different options depending on the grid characteristics and the knowledge discovery techniques in use. Following this line of research, several aspects lie open for further study:

- Studying new knowledge discovery techniques, developing comparative studies that show ways to improve the predictor's performance.

- Extend the actual prediction models, in order to incorporate more advanced time series analysis elements.

11.2 Global behavior modeling of other distributed systems

The global behavior methodology presented was originally designed for the grid, but could be easily adapted to model other distributed infrastructures. This application coud benefit these other systems, in a similar way GloBeM's vison benefits the grid. However, some minor adjustements might be required, due to the evident structural diferences. The most relevant alternatives are:

11.2.1 Clusters

Although clusters are typically much less complex than grids, with dedicated resources and high performance, dependable networks, global behavior modeling could help to model its performance and enrich the single system image that, up to a certain level, most of them provide. This is specially relevant in big infrastructures (such as most TOP500 computers [TSS]), where the large amount of nodes (hundredths or even thousands) increases complexity enormously, making system's much less predictable and manageable.

11.2.2 Clouds

The service-oriented nature of these systems make them close relatives of the grid, sharing some of its most important characteristics. The single entity, service-level behavior model generated by GloBeM can be used in clouds in a similar way it is used on grids, providing insight on the system performance and clues to improve its dependability and quality of service.

11.3 Creation of *super-grid* infrastructures

Maybe one of the most interesting applications of GloBeM's single entity vision of the grid is to help creating higher level structures (the name *super-grid* is used here only as a suggested possibility), capable of integrating different grids. The single entity vision enables to regard each grid as an unique, independent resource, developing orchestrating mechanisms capable of integrate all services provided by different grids. There has already been some research in this area [DdABV08], but there are still many posibilities to explore.

Adittionally, if this idea is combined with the previous one, presented in Section 11.2 (modeling other distributed systems), GloBeM could provide a unfied vision for several different distributed systems, including grids, clusters and clouds. Fully regarding these systems as single entities allows to easily combine them, creating these *super-grids*. Without global behavior modeling and single entity abstraction creating a super-structure like that would be extremely complicated, due not only to technical issues but also to global vision and unified management problems.

11.4 Application of GloBeM's analysis methodology to other fields

Although GloBeM has been designed as a distributed systems behavior analysis methodology, it is possible to adapt it (or some part of it) to other scientific fields. The core of GloBeM is a powerful information analysis and knowledge extraction mechanism, and these features can be put to use in other research areas. In most cases the characteristics of the scientific problem involved would require an adaptation of GloBeM methodology to some extent, to fit these specifics. These are a few examples of other possible uses of the GloBeM's analysis methodology, selected for its scientific relevance:

- Behavior analysis of complex biological systems, specially in the fields of bioinformatics and neuroscience. These systems, such as natural neural networks, can be observed and analyzed, acording to several biological parameters. These parameters can be used as a basis for global behavior modeling, trying to extract general patterns and discover advanced interactions among large numbers of interconnected cells. The CesViMa research group has previous experience in this field, as a part of the Cajal Blue Brain Project [cbb, Mar06].

- Also in the field of neuroscience, GloBeM analysis methodology could be benefitial in the problem of neuron classification [RyC99, CPT$^+$00]. The abstraction capability of GloBeM could be adapted to find hidden relations between cells, in a similar way it is capable of identifying

grid states.

- Contribution to improve advanced scientific visualization tecniques. The GloBeM methodology could be adapted to be used to enrich complex information representation techniques, providing information abstraction and highlighting the most relevant aspects of the represented data. These enriched representation could be used in many scientific problems, such as diverse statistical analyses and other scientific areas such as complex phisics, fluid mechanics or the above mentioned bioinformatics and neuroscience.

- In the area of optimization problems and, more specifically, in the complex sub-field of algorithm hybridization [Tal02], the GloBeM analysis methodology could be adapted to study the resulting hybridization patterns and to develop new combination techniques. GloBeM helps to analyze the grid evolution and to create advanced autonomic management mechanisms. In the same way, it could help to understand how the hybridization process works and to provide insight on how to develop improved hybrid algorithms.

Part V

APPENDICES

Appendix A

Detailed architecture of FIRE

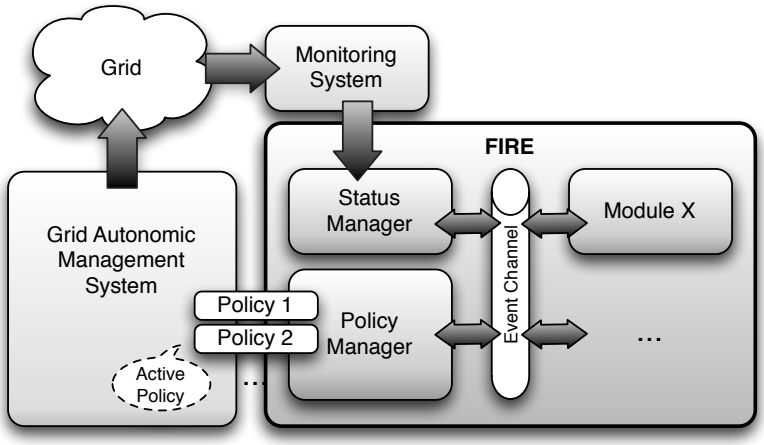


Figure A.1: Architecture of FIRE

In Section 8.2 the FIRE autonomic management framework was introduced. Although a detailed description of its features and functionalities was presented (both theoretical and practical), the system's architecture was only briefly summarized. In this Appendix FIRE architecture is discussed in detail, thoroughly describing its design and operation. Figure A.1 shows a general overview of the system, including all relevant modules. Each of them is going to be described in detail below.

A.1 Event Channel

FIRE's architecture is organized around a standard *event channel*. This software design technique allows to construct a very modular system, in which several different elements gather together. Communication between these modules takes place through this *event channel*, in the form of specific *event notification messages* to which modules can subscribe. These messages provide information about specific situations, and can be created by the modules to fit their specific needs and functionalities.

At any given time, any module connected to the event channel can perform three basic operations:

- **Subscribe** to an event: This indicates that the module is interested in that specific event. From that moment it will receive messages whenever that event is notified.
- **Unsubscribe** to an event: The opposite operation to the first one. It cancels the module subscription to the event, so no new notification messages will be received.
- **Publish** an event notification: The module notifies the occurrence of an event to the system. All modules subscribed to that specific event will receive an event notification message.

FIRE's management capabilities are based on GloBeM's behavior models. As described in Chapter 7, these models are presented in the form of a finite state machine (FSM), that can be understood in the following terms:

- *States* represent different cases of identified behavior, according to certain characteristics.
- *Transitions* between states represent **events** on that make the system's behavior change, moving from one state to another. Detecting these transitions plays a very important role in autonomic management, because it indicates behavior changes that might be relevant to the system's performance and functionalities.

From this perspective it seems fitting to adopt an *event-driven* approach, like the one provided by an *event channel*. In order to provide autonomic features FIRE must also incorporate the necessary modules to identify, analyze and make decisions based on the system's behavior FSM model.

A.2 Status Manager

The **Status Manager** is the first FIRE module. Its mission is to monitor the system's behavior and interpret it in terms of the GloBeM's FSM. Its basic functions are:

Table A.1: Status Manager events

Event	Role	Description
State Request	Subscriber	Indicates that a module has requested to know the current grid state.
Current State	Publisher	This event message contains information about the system's current state and monitoring values.
State Transition	Publisher	Notifies an system's state transition, indicating the new state and transition conditions.

- It uses monitoring information provided by the grid *sensors* to calculate the system's state according to the FSM model. Normally this information is provided by GMonE, but other grid monitoring tools could be adapted to be used in its place.

- Using the event channel, it can inform of the system's current state and detailed monitoring values.

- It detects state transitions and notifies them sending event notification messages into the event channel.

Table A.1 shows the events subscribed to and published by the Status Manager.

A.3 Policy Manager

The second FIRE module is called the **Policy Manager**. This module's main purpose is to provide self-adaptive autonomic capabilities to the grid's management system, by automatically selecting a set of compatible management policies for each case. As can be seen in Figure 8.5, the Policy Manager is directly connected with the grid's management system, and it is capable of automatically changing its configuration in order to autonomically adapt to the system's behavior changes. This is done by selecting the appropriate management policy for each situation (normally each state of the FSM model).

If we consider data replica allocation on a Data Grid, for instance, several different allocation policies could be used, depending on the system state. Some polices could be optimal when the network connections are heavy loaded, others when the CPU usage is very unbalanced, etc. Dynamically selecting the adequate replica allocation policy for each state would strongly improve the overall system's performance (assuming that all policies are compatible among themselves).

Therefore, the Policy Manager main functions are:

Table A.2: Policy Manager events

Event	Role	Description
State Request	Publisher	Indicates that the Policy manager has requested to know the current grid state.
Current State	Subscriber	This event message contains information about the system's current state and monitoring values.
State Transition	Subscriber	Notifies an system's state transition, indicating the new state and transition conditions.

- It contains knowledge about the FSM behavior model, the management policies available and which is more adequate for each state.
- With the help of the Status Manager, it detects behavior changes in the grid and selects the appropriate management policy in each situation, so it can be applied by the management system.

The knowledge about the policies available, the FSM behavior model and the correct combination of policies and states have to be provided to the Policy Manager in an initial setup phase. The FSM model and policies have to be studied prior to the use of FIRE (normally by a human expert) and the correct configuration has to be created and introduced in the Policy Manager. After this setup stage the system can be completely autonomous, not needing any external intervention or system administration supervision.

Table A.2 shows the events subscribed to and published by the Status Manager.

A.4 Other events and modules

The extensibility and flexibility of the event channel architecture allows the inclusion of custom modules and events, represented in figure 8.5 by generic modules *X* and *Y*. The specific nature of a grid system where FIRE would be deployed could make necessary to incorporate new functionalities. FIRE has been designed with this in mind and, therefore, it could be easily accomplished.

In the above mentioned data grid example, for instance, it could be necessary to incorporate information about location of the replicas, load distribution and so on. If that could not be accomplished through the monitoring system (which provides information to the Status Manager), a new module an a set of events could be easily implemented and integrated in FIRE. In the same way the Policy Manager could be very simply customized to subscribe to the new necessary events and take that information into account when making decisions.

Bibliography

[AB84] M. Aldenferder and R. Blashfield. Cluster Analysis. Sage Publications, 1984.

[abi] abiCloud open source cloud platform, accessed Mar 2010 [Online]. Available: *http://www.abiquo.com/en/products/abicloud.html*.

[ACo] Welcome to Autonomic computing.org, accessed Feb 2010 [Online]. Available: *http://www.autonomiccomputing.org*.

[AGl] About the globus toolkit, accessed Feb 2010 [Online]. Available: *http://www.globus.org/toolkit/about.html*.

[ALRL04] A. Avizienis, J. C. Laprie, B. Randell, and C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. Dependable and Secure Computing, IEEE Transactions on, 1(1):11–33, 2004.

[app] Google App Engine, accessed Mar 2010 [Online]. Available: *http://code.google.com/appengine/*.

[Arb69] Michael A Arbib. Theories of abstract automata (Prentice-Hall series in automatic computation). Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1969.

[azu] Windows Azure Platform, accessed Mar 2010 [Online]. Available: *http://www.microsoft.com/windowsazure/*.

[BAG00] R. Buyya, D. Abramson, and J. Giddy. Nimrod/G: an architecture for a resource management and scheduling system in a global computational grid. In Proceedings of The Fourth International Conference/Exhibition on High Performance Computing in the Asia-Pacific Region, volume 1, pages 283–289 vol.1, 2000.

[BBL02] M. Baker, R. Buyya, and D. Laforenza. Grids and grid technologies for wide-area distributed computing. Software-Practice and Experience, 32(15):1437–1466, 2002.

[BCC+02] R. Byrom, B. Coghlan, A. Cooke, R.Cordenonsi, L. Cornwall, A. Datta, A. Djaoui, L. Field, S. Fisher, S. Hicks, S. Kenny, J. Magowan, W. Nutt, D. O Callaghan, M. Oevers, N. Podhorszki, J. Ryan, M. Soni, P. Taylor, A. Wilson, and X Zhu. R-GMA: A

Relational Grid Information and Monitoring System. In 2nd Cracow Grid Workshop, Cracow, Poland, 2002.

[BDIM04] Paul Barham, Austin Donnelly, Rebecca Isaacs, and Richard Mortier. Using magpie for request extraction and workload modelling. In Proceedings of the 6th Symposium on Opearting Systems Design and Implementation, pages 259–272, 2004.

[Beo] Beowulf.org: The Beowulf Cluster Site, accessed Mar 2010 [Online]. Available: *http://www.beowulf.org*.

[BH90] D. E. Brown and C. L. Huntley. A practical application of simulated annealing to clustering. Technical Report IPC-91-03, University of Virginia, 1990.

[BM02a] R. Buyya and M. Murshed. GridSim: a toolkit for the modeling and simulation of distributed resource management and scheduling for Grid computing. Journal of Concurrency and Computation: Practice and Experience (CCPE), 14(13–15):1175–1220, May 2002.

[BM02b] Rajkumar Buyya and M. Manzur Murshed. Gridsim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing. CoRR, cs.DC/0203019, 2002.

[BOI] BOINC, accessed Mar 2010 [Online]. Available: *http://boinc.berkeley.edu/*.

[Bor87] I. Borg. Multidimensional similarity structure analysis. Springer-Verlag New York, Inc., New York, NY, USA, 1987.

[Bry07] Randal E. Bryant. Data-intensive supercomputing: The case for disc. Technical report, CMU, 2007.

[BvdAST08] Carmen Bratosin, Wil M. P. van der Aalst, Natalia Sidorova, and Nikola Trcka. A reference model for grid architectures and its analysis. In Robert Meersman and Zahir Tari, editors, OTM Conferences (1), volume 5331 of Lecture Notes in Computer Science, pages 898–913. Springer, 2008.

[cbb] Cajal Blue Brain Project (CajalBBP), accessed Mar 2010 [Online]. Available: *http://cajalbbp.cesvima.upm.es/*.

[CFF+01] G. Cancio, S. M. Fisher, T. Folkes, F. Giacomini, W. Hoschek, D. Kelsey, and B. L. Tierney. The DataGrid Architecture. Technical Report DataGrid-12-D12.4-333671-3-0, EU DataGrid Project, 2001.

[Cha03] Chris Chatfield. The Analysis of Time Series: An Introduction, Sixth Edition (Texts in Statistical Science). Chapman & Hall/CRC, July 2003.

[CKL08] Kyu Cheol Cho, Tae Young Kim, and Jong Sik Lee. User demand prediction-based resource management model in grid computing environment. In Proceedings of the 2008 International Conference on Convergence and Hybrid Information Technology (ICHIT '08), pages 627–632, Washington, DC, USA, 2008. IEEE Computer Society.

[CL89] John Carroll and Darrell Long. Theory of finite automata with an introduction to formal languages. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1989.

[Clo] Twenty-One Experts Define Cloud Computing, accessed Feb 2010 [Online]. Available: *http://cloudcomputing.sys-con.com/node/612375/print*.

[CoG] Condor-g, accessed Jan 2010 [Online]. Available: *http://www.cs.wisc.edu/condor/condorg*.

[com] CoMon - A Monitoring Infrastructure for PlanetLab, accessed Mar 2010 [Online]. Available: *http://comon.cs.princeton.edu/*.

[Con] The Condor Project, accessed Mar 2010 [Online]. Available: *http://www.cs.wisc.edu/condor*.

[CP81] Jean-Louis Chandon and Suzanne. Pinson. Analyse typologique : theories et applications / par Jean-Louis Chandon, Suzanne Pinson ; preface de E. Diday. Masson, Paris ; New York :, 1981.

[CPT+00] B. Cauli, J. T. Porter, K. Tsuzuki, B. Lambolez, J. Rossier, B. Quenet, and E. Audinat. Classification of fusiform neocortical interneurons based on unsupervised clustering. PNAS, 97(11):6144–6149, 2000.

[CrG] The CrossGrid Project, accessed Mar 2010 [Online]. Available: *http://www.eu-crossgrid.org/*.

[CZG+05] Ira Cohen, Steve Zhang, Moises Goldszmidt, Julie Symons, Terence Kelly, and Armando Fox. Capturing, indexing, clustering, and retrieving system history. SIGOPS Oper. Syst. Rev., 39(5):105–118, 2005.

[D'A78] Roy D'Andrade. U-statistic hierarchical clustering. Psychometrika, 43(1):59–67, March 1978.

[Das91] B. V. Dasarathy. Nearest neighbor (NN) norms : NN pattern classification techniques. IEEE Computer Society Press, 1991.

[DdABV08] Marcos Dias de Assunçao, Rajkumar Buyya, and Srikumar Venugopal. Intergrid: a case for internetworking islands of grids. Concurr. Comput. : Pract. Exper., 20(8):997–1024, 2008.

[den] Dendogram - Wikipedia, the free encyclopedia, accessed Jan 2010 [Online]. Available: *http://en.wikipedia.org/wiki/Dendogram*.

[DG93] Jack J. Dongarra and Wolfgang Gentzsch, editors. Computer benchmarks. Elsevier Science Publishers B. V., Amsterdam, The Netherlands, The Netherlands, 1993.

[DG08] Jeffrey Dean and Sanjay Ghemawat. MapReduce: simplified data processing on large clusters. Communications of the ACM, 51(1):107–113, 2008.

[DLR77] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. Journal of the Royal Statistical Society. Series B (Methodological), 39(1):1–38, 1977.

[ec2] Amazon Elastic Compute Cloud (Amazon EC2), accessed Mar 2010 [Online]. Available: *http://aws.amazon.com/ec2/*.

[ege] EGEE Portal: Enabling Grids for E-sciencE, accessed Febr 2010 [Online]. Available: *http://www.eu-egee.org/*.

[EKJX96] Martin Ester, Hans-Peter Kriegel, S. Jörg, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise, 1996.

[Eri93] Schikuta Erich. Grid-clustering: A fast hierarchical clustering method for very large data sets. Technical report, Center for Research on Parallel Computation, 1993.

[ES01] D. W. Erwin and D. F. Snelling. UNICORE: A Grid computing environment. Lecture Notes in Computer Science, 2150, 2001.

[FdW02] M. Frumkin and R. F. Van der Wijngaart. NAS Grid Benchmarks: A Tool for Grid Space Exploration. Cluster Computing, 5(3):247–255, 2002.

[FK97] Ian Foster and Carl Kesselman. Globus: a metacomputing infrastructure toolkit. International Journal of High Performance Computing Applications, 11(2):115–128, June 1997.

[FKNT02] I. Foster, C. Kesselman, J. Nick, and S. Tuecke. The physiology of the grid: An open grid services architecture for distributed systems integration, 2002.

[FM83] E. B. Fowlkes and C. L. Mallows. A method for comparing two hierarchical clusterings. Journal of the American Statistical Association, 78(383):553–584, 1983.

[fNRa] CERN European Organization for Nuclear Research. EGEE - Alice Job Summary, accessed Feb 2009 [Online]. Available: *http://dashb-alice.cern.ch/dashboard/request.py/jobsummary*.

[fNRb] CERN European Organization for Nuclear Research. EGEE - Atlas Dashboard, accessed Dec 2008 [Online]. Available: *http://dashb-atlas-prodsys-test.cern.ch/dashboard/request.py/summary*.

[fNRc] CERN European Organization for Nuclear Research. EGEE - Atlas Job Summary, accessed Feb 2009 [Online]. Available: *http://dashb-atlas-job.cern.ch/dashboard/request.py/jobsummary*.

[fNRd] CERN European Organization for Nuclear Research. EGEE - CMS Job Summary, accessed Feb 2009 [Online]. Available: *http://lxarda09.cern.ch/dashboard/request.py/jobsummary*.

[Fos01] I. Foster. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. In Proceedings of the 7th International Euro-Par Conference Manchester on Parallel Processing (Euro-Par '01), volume 2150 of Lecture Notes In Computer Science, pages 1–4, London, UK, 2001. Springer-Verlag.

[Fos02] I. Foster. What is the Grid? A Three Point Checklist. Grid Today, 1(6), Jul 2002.

[Fos05] Ian T. Foster. Globus toolkit version 4: Software for service-oriented systems. In Hai Jin, Daniel A. Reed, and Wenbin Jiang, editors, NPC, volume 3779 of Lecture Notes in Computer Science, pages 2–13. Springer, 2005.

[FS03] M. A. Frumkin and L. Shabanov. Arithmetic Data Cube as a Data Intensive Benchmark. Technical Report NAS-03-005, NAS, 2003.

[FSB+06] I. Foster, A. Savva, D. Berry, A. Djaoui, A. Grimshaw, B. Horn, F. Maciel, F. Siebenlist, R. Subramaniam, J. Treadwell, and J. Von Reich. The Open Grid Services Architecture. Technical Report Version 1.5, Global Grid Forum, June 2006.

[FZRL09] Ian T. Foster, Yong Zhao, Ioan Raicu, and Shiyong Lu. Cloud computing and grid computing 360-degree compared. CoRR, abs/0901.0131, 2009.

[gan] Ganglia distributed monitoring and execution system, accessed Mar 2010 [Online]. Available: *http://ganglia.info/*.

[Gbu] The Gridbus Project, accessed Jun 2009 [Online]. Available: *http://www.gridbus.org*.

[GC03] A. G. Ganek and T. A. Corbi. The dawning of the autonomic computing era. IBM Systems Journal, 42(1):5–18, 2003.

[GCB+97] J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, M. Venkatrao, F. Pellow, and H. Pirahesh. Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab, and Sub-Totals. Data Mining and Knowledge Discovery, 1(1):29–53, 1997.

[GGL03] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The google file system. SIGOPS - Operating Systems Review, 37(5):29–43, 2003.

[GKM+06] S. Graham, A. Karmarkar, J. Mischkinsky, I. Robinson, and I. Sedukhin. Web Service Resource Framework. Technical Report Version 1.2, OASIS, 2006.

[GlA] The Globus Alliance, accessed Mar 2010 [Online]. Available: *http://www.globus.org*.

[Gli] EGEE - GLite, accessed Feb 2010 [Online]. Available: *http://glite.web.cern.ch/glite/default.asp*.

[GMA] R-GMA: Relational Grid Monitoring Architecture, accessed Jan 2010 [Online]. Available: *http://www.r-gma.org/*.

[gmo] GMonE: Grid Monitoring Environment, accessed Mar 2010 [Online]. Available: *http://laurel.datsi.fi.upm.es/proyectos/globem/gmone/*.

[GPW] Grid Forum Grid Performance Working Group Home Page, accessed Feb 2010 [Online]. Available: *http://www-didc.lbl.gov/GGF-PERF/GMA-WG*.

[GR3] GRID3, accessed Jan 2007 [Online]. Available: *http://www.ivdgl.org/grid2003/*.

[GS66] D.M. Green and J.M. Swets. Signal detection theory and psychophysics. John Wiley and Sons Inc., New York, 1966.

[GSK03] G. R. Ganger, J. D. Strunk, and A. J. Klosterman. Self-* storage: brickbased storage with automated administration. Technical Report CMU-CS03 -178, Carnegie Mellon University, August 2003.

[GSp] GridSphere, accessed Feb 2010 [Online]. Available: *http://www.gridsphere.org/gridsphere/gridsphere*.

[Gur91] Yuri Gurevich. Evolving Algebras: An Attempt to Discover Semantics. In EATCS Bulletin, volume 43, pages 264–284. European Assoc. for Theor. Computer Science, February 1991.

[GWB+04] M. Gerndt, R. Wismüller, Z. Balaton, G. Gombás, P. Kacsuk, Z. Nemeth, N. Podhorszki, H. Truong, T. Fahringer, M. Bubak, E. Laure, and T. Margalef. Performance tools for the grid: State of the art and future. APART White Paper, 2004.

[GWM] GridWay Metascheduler: Metascheduling Technologies for the Grid, accessed Feb 2010 [Online]. Available: *http://www.gridway.org/*.

[Haw] Hawkeye, accessed Feb 2010 [Online]. Available: *http://www.cs.wisc.edu/condor/hawkeye/*.

[Hay94] S. Haykin. Neural Networks: A Comprehensive Foundation. Macmillan, New York, 1994.

[HHML05] Jose Herrera, Eduardo Huedo, Rubén S. Montero, and Ignacio M. Llorente. Porting of scientific applications to grid computing on GridWay. Scientific Programming, 13(4):317–331, 2005.

[HK00] Jiawei Han and Micheline Kamber. Data Mining: Concepts and Techniques (The Morgan Kaufmann Series in Data Management Systems). Morgan Kaufmann, September 2000.

[HKY99] Laurie J. Heyer, Semyon Kruglyak, and Shibu Yooseph. Exploring expression data: Identification and analysis of coexpressed genes. Genome Res., 9(11):1106–1115, November 1999.

[HL00] David W. Hosmer and Stanley Lemeshow. Applied logistic regression. Wiley Interscience, October 2000.

[HL03] P. Hunt and D. Larson. Addressing IT Challenges with Self-Healing Technology. Technology@Intel Magazine, pages 1–6, October 2003.

[HML05] Eduardo Huedo, Rubén S. Montero, and Ignacio M. Llorente. The GridWay framework for adaptive scheduling and execution on grids. Scalable Computing: Practice and Experience, 6(3):1–8, 2005.

[HMU00] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. Introduction to Automata Theory, Languages, and Computation. Addison Wesley, 2nd edition, November 2000.

[Hoo05] G. Hoolahan. Applying Adaptive Enterprise principles to collaborative business infrastructure-based solution designs. HP White Paper, June 2005.

[Hor01] P. Horn. IBM's Perspective on the state of information technology, accessed Mar 2010 [online]. available: *http://www.research.ibm.com/autonomic/overview/*, 2001.

[IBM03] IBM - International Business Machines Corporation. Delivering the vision: autonomic computing. Technical Report The Mainstream, Issue 3, The IBM eServer zSeries and S/390 software, Aug 2003.

[IBM06] IBM. An architectural blueprint for autonomic computing, accessed Feb 2010 [Online]. Available: *http://www-03.ibm.com/autonomic/pdfs/AC_Blueprint_White_Paper_4th.pdf*. IBM Autonomic Computing White Paper, 2006.

[Isa02] N. Isailovic. An Introspective Approach to Speculative Execution. Technical Report UCB/CSD-02-1219, U.C. Berkeley, December 2002.

[JGN99] W. E. Johnston, D. Gannon, and B. Nitzberg. Grids as production computing environments: The engineering aspects of NASA's Information Power Grid. In Proceedings of the 8th IEEE International Symposium on High Performance Distributed Computing (HPDC '99), page 34. IEEE Computer Society, 1999.

[JLL+06] Yvon Jégou, Stephane Lantéri, Julien Leduc, Melab Noredine, Guillaume Mornet, Raymond Namyst, Pascale Primet, Benjamin Quetier, Olivier Richard, El-Ghazali Talbi, and Touche Iréa. Grid'5000: a large scale and highly reconfigurable experimental grid testbed. International Journal of High Performance Computing Applications, 20(4):481–494, November 2006.

[JM92] A. K. Jain and J. Mao. Artificial neural network for nonlinear projection of multivariate data. In Neural Networks, 1992. IJCNN., International Joint Conference on Neural Networks, volume 3, pages 335–340 vol.3, 1992.

[Joh67] Stephen Johnson. Hierarchical clustering schemes. Psychometrika, 32(3):241–254, September 1967.

[JX04] X. Jiang and D. Xu. VIOLIN: Virtual Internetworking on OverLay INfrastructure. Lecture Notes in Computer Science, 3358:937–946, 2004.

[KBM02] Klaus Krauter, Rajkumar Buyya, and Muthucumaru Maheswaran. A taxonomy and survey of grid resource management systems for distributed computing. Software Practice and Experience, 32(2):135–164, 2002.

[KC03] J. O. Kephart and D. M. Chess. The vision of autonomic computing. Computer, 36(1):41–50, 2003.

[Ker] Kerrighed, accessed Dec 2007 [Online]. Available: *http://www.kerrighed.org*.

[KF98] C. Kesselman and I. Foster, editors. The Grid: Blueprint for a New Computing Infrastructure. Morgan Kaufmann Publishers, November 1998.

[KF08] Kate Keahey and Tim Freeman. Science clouds: Early experiences in cloud computing for scientific applications. In CCA08: Cloud Computing and Applications 2008, Chicago, IL, USA, 2008.

[kme] K-means clustering - Wikipedia, the free encyclopedia, accessed Jan 2010 [Online]. Available: *http://en.wikipedia.org/wiki/K-means_clustering*.

[knn] k-nearest neighbor algorithm - Wikipedia, the free encyclopedia, accessed Jan 2010 [Online]. Available: *http://en.wikipedia.org/wiki/K-nearest_neighbor_algorithm*.

[Koh90] T. Kohonen. The self-organizing map. In Proceedings of the IEEE, volume 78(9), pages 1464–1480, 1990.

[Kru64] J. Kruskal. Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. Psychometrika, 29(1):1–27, March 1964.

[LGW07] Hui Li, David Groep, and Lex Wolters. Mining performance data for metascheduling decision support in the grid. Future Gener. Comput. Syst., 23(1):92–99, 2007.

[Lin87] D. V. Lindley. Regression and correlation analysis. New Palgrave: A Dictionary of Economics, 4:120 – 123, 1987.

[LIT92] Pat Langley, Wayne Iba, and Kevin Thompson. An analysis of bayesian classifiers. In Proceedings of the Tenth National Conference on Artificial Intelligence, pages 223–228. MIT Press, 1992.

[LSS] LinuxSSI - XtreemOS : A Linux-based Operating System to support Virtual Organizations for next generation Grids, accessed Dec 2009 [Online]. Available: *http://www.xtreemos.eu/science-and-research/ploneearticlemultipage.2007-05-03.8942730332/copy_of_linuxssi*.

[Mac67] J. B. MacQueen. Some methods for classification and analysis of multivariate observations. In L. M. Le Cam and J. Neyman, editors, Proc. of the fifth Berkeley Symposium on Mathematical Statistics and Probability, volume 1, pages 281–297. University of California Press, 1967.

[mal] MonALISA - Monitoring the Grid since 2001, accessed Mar 2010 [Online]. Available: *http://monalisa.cacr.caltech.edu/*.

[Mar02] J. Marco. Grids and e-Science. RedIris Bulletin, 61, September 2002.

[Mar06] Henry Markram. The blue brain project. Nat Rev Neurosci, 7(2):153–160, February 2006.

[MB06] D. A. Menasce and M. N. Bennani. Autonomic virtualized environments. In Proceedings of the IEEE International Conference on Autonomic and Autonomous Systems (ICAS '06), page 28, Silicon Valley, USA, July 2006. IEEE Computer Society.

[McC03] J. A. McCann. Adaptivity for improving web streaming application performance. In Nandish V. Patel, editor, Adaptive evolutionary information systems, pages 172–191. IGI Publishing, 2003.

[mds] Globus: Monitoring and Discovery System, accessed Feb 2010 [Online]. Available: *http://www.globus.org/toolkit/mds/*.

[MK04] J. Martin and H. Karlapudi. Web application performance prediction. In Proceedings of the IASTED International Conference on Communication and Computer Networks, pages 281–286, Boston, MA, USA, November 2004.

[Moo65] G. E. Moore. Cramming more components onto integrated circuits. Electronics, 38(8):114–117, April 1965.

[Mor07] Christine Morin. XtreemOS: A Grid Operating System Making your Computer Ready for Participating in Virtual Organizations. In Proceedings of the 10th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC'07), pages 393–402, Los Alamitos, CA, USA, 2007. IEEE Computer Society.

[mos] MOSIX, accessed Dec 2009 [Online]. Available: *http://www.mosix.org*.

[MPI] MPI - the Message Passing Interface standard, accessed Jan 2010 [Online]. Available: *http://www-unix.mcs.anl.gov/mpi/*.

[MSV+10] Jesús Montes, Alberto Sánchez, Julio J. Valdés, María S. Pérez, and Pilar Herrero. Finding order in chaos: a behavior model of the whole grid. Concurrency and Computation: Practice and Experience, page In press., 2010.

[NAB09a] Bogdan Nicolae, Gabriel Antoniu, and Luc Bougé. BlobSeer: How to enable efficient versioning for large object storage under heavy access concurrency. In Proc. 2nd Workshop on Data Management in Peer-to-Peer Systems (DAMAP'2009), Saint Petersburg, Russia, March 2009. Held in conjunction with EDBT'2009.

[NAB09b] Bogdan Nicolae, Gabriel Antoniu, and Luc Bougé. Enabling high data throughput in desktop grids through decentralized data and metadata management: The blobseer approach. In Proc. 15th International Euro-Par Conference on Parallel Processing (Euro-Par '09), volume 5704 of Lect. Notes in Comp. Science, pages 404–416, Delft, The Netherlands, 2009. Springer-Verlag.

[NLB01] H.B. Newman, I.C. Legrand, and J.J. Bunn. A Distributed Agent-based Architecture for Dynamic Services. In CHEP 2001, Beijing, September 2001.

[NLG+03] H B Newman, I C Legrand, P Galvez, R Voicu, and C Cirstoiu. MonALISA : A Distributed Monitoring Service Architecture. In 2003 Computing in High Energy and Nuclear Physics (CHEP03), La Jolla, California, USA, March 2003.

[nws] Network Weather Service: Introduction, accessed Mar 2010 [Online]. Available: *http://nws.cs.ucsb.edu/ewiki/*.

[OD08] D. L. Olson and D. Delen. Advanced Data Mining Techniques. Springer, New York, 2008.

[ogc] The Open Grid Computing Environments Portal and Gateway Toolkit, accessed Feb 2010 [Online]. Available: *http://www.collab-ogce.org/*.

[OGF] Open Grid Forum, accessed Feb 2010 [Online]. Available: *http://www.ogf.org/*.

[ogs] Globus: The Open Grid Service Architecture, accessed Feb 2010 [Online]. Available: http://www.globus.org/ogsa/.

[OMo] openMosix, an Open Source Linux Cluster Project, accessed Jan 2010 [Online]. Available: *http://openmosix.sourceforge.net*.

[ope] OpenNebula: The Open Source Toolkit for Cloud Computing, accessed Mar 2010 [Online]. Available: *http://www.opennebula.org/*.

[PBS] OpenPBS, accessed Jan 2010 [Online]. Available: *www.openpbs.org*.

[Pea01] K. Pearson. On lines and planes of closest fit to systems of points in space. Philosophical Magazine, 2(6):559–572, 1901.

[Pér03] M. S. Pérez. Arquitectura Multiagente para E/S de Alto Rendimiento en Clusters. PhD thesis, Universidad Politécnica de Madrid, 2003.

[PKT+09] Xinghao Pan, Soila Kavulya, Jiaqi Tan, Rajeev Gandhi, and Priya Narasimhan. Ganesha: Black-box diagnosis for mapreduce systems. In Proceedings of the Second Workshop on Hot Topics in Measurement & Modeling of Computer Systems, Seattle, WA, USA, 2009.

[pla] Planetlab - An open platform for developing, deploying and accessing planetary-scale services, accessed Mar 2010 [Online]. Available: *http://www.planet-lab.org/*.

[PVM] PVM: Parallel Virtual Machine, accessed Jan 2010 [Online]. Available: *http://www.csm.ornl.gov/pvm/pvm_home.html*.

[PW03] P. Padala and J. N. Wilson. GridOS: Operating System Services for Grid Architectures. In T. M. Pinkston and V. K. Prasanna, editors, Proceedings of the 10th International Conference on High Performance Computing (HiPC 2003), volume 2913 of Lecture Notes in Computer Science, pages 353–362. Springer, 2003.

[Qui86] J. R. Quinlan. Induction of decision trees. Machine Learning, 1(1):81–106, March 1986.

[Qui93] Ross J. Quinlan. C4.5: Programs for Machine Learning (Morgan Kaufmann Series in Machine Learning). Morgan Kaufmann, January 1993.

[RAC] IBM Research Autonomic Computing, accessed Feb 2010 [online]. available: *http://www.research.ibm.com/autonomic/*.

[Ran71] William M. Rand. Objective criteria for the evaluation of clustering methods. Journal of the American Statistical Association, 66(336):846–850, 1971.

[RBL+09] Benny Rochwerger, David Breitgand, Eliezer Levy, Alex Galis, Kenneth Nagin, Ignacio M. Llorente, Ruben Montero, Yaron Wolfsthal, Erik Elmroth, Juan Caceres, Muli Ben-Yehuda, Wolfgan Emmerich, and Fermin Galan. The RESERVOIR model and architecture for open federated cloud computing. IBM Journal of Research and Development, 53(4), 2009.

[RL07] Brent Rood and Michael J. Lewis. Multi-state grid resource availability characterization. In GRID '07: Proceedings of the 8th IEEE/ACM International Conference on Grid Computing, pages 42–49, Washington, DC, USA, 2007. IEEE Computer Society.

[RL08a] Brent Rood and Michael J. Lewis. Resource availability prediction for improved grid scheduling. In Proceedings of the 2008 Fourth IEEE International Conference on eScience (e-Science 2008), pages 711–718, Washington, DC, USA, 2008. IEEE Computer Society.

[RL08b] Brent Rood and Michael J. Lewis. Scheduling on the grid via multi-state resource availability prediction. In Proceedings of the 9th IEEE/ACM International Conference on Grid Computing (Grid 2008), pages 126–135. IEEE, 2008.

[RMX05] P. Ruth, P. McGachey, and D. Xu. Viocluster: Virtualization for dynamic computational domains. In Proceedings of the IEEE International Conference on Cluster Computing (CLUSTER '05), Boston, September 2005.

[RyC99] S. Ramón y Cajal. Textura del Sistema Nervioso del Hombre y de los Vertebrados. Moya, 1899.

[SAH] Seti@home. The Search for ExtraTerrestrial Inteligence, accessed Mar 2010 [Online]. Available: *http://setiathome.ssl.berkeley.edu*.

[Sam69] J. W. Sammon. A non-linear mapping for data structure analysis. IEEE Transactions on Computers, 18(18):401–408, 1969.

[Sán08] Alberto Sánchez. Autonomic high performance storage for grid environments based on long term prediction. PhD thesis, Universidad Politécnica de Madrid, 2008.

[SDM+05] J. M. Schopf, M. D'Arcy, N. Miller, L. Pearlman, I. Foster, and C. Kesselman. Monitoring and Discovery in a Web Services Framework: Functionality and Performance of the Globus Toolkit's MDS4. Technical Report ANL/MCS-P1248-0405, Argonne National Laboratory, April 2005.

[SF05] Mumtaz Siddiqui and Thomas Fahringer. GridARM: Askalon's Grid Resource Management System. In Advances in Grid Computing - EGC 2005 - Revised Selected Papers, volume 3470 of Lecture Notes in Computer Science, pages 122–131, Amsterdam, Netherlands, June 2005. Springer Verlag GmbH, ISBN 3-540-26918-5.

[SFT00] W. Smith, I. Foster, and V. Taylor. Scheduling with advanced reservations. In Proceedings of the 14th International Symposium on Parallel and Distributed Processing (IPDPS '00), pages 127–132, Washington, DC, USA, 2000. IEEE Computer Society.

[SFT04] Warren Smith, Ian T. Foster, and Valerie E. Taylor. Predicting application run times with historical information. J. Parallel Distrib. Comput., 64(9):1007–1016, 2004.

[SGE] gridengine:Home, accessed Feb 2010 [Online]. Available: *http://gridengine.sunsource.net*.

[Sil87] Joaquim Silvestre. Economies and Diseconomies of Scale. In The New Palgrave: A Dictionary of Economics, volume 2, pages 80–84. Palgrave Macmillan, 1987.

[SMLF08] Borja Sotomayor, Rubén S. Montero, Ignacio M. Llorente, and Ian Foster. Capacity Leasing in Cloud Systems using the OpenNebula Engine. In CCA08: Cloud Computing and Applications, 2008.

[SRC09] Jeffrey Shafer, Scott Rixner, and Alan L. Cox. Datacenter storage architecture for mapreduce applications. In ACLD: Workshop on Architectural Concerns in Large Datacenters, 2009.

[Ste74] M. A. Stephens. Edf statistics for goodness of fit and some comparisons. Journal of the American Statistical Association, 69(347):730–737, 1974.

[Sto07] Heinz Stockinger. Defining the grid: a snapshot on the current view. The Journal of Supercomputing, 42(1):3–17, 2007.

[svm] Support vector machine - Wikipedia, the free encyclopedia, accessed Jan 2010 [Online]. Available: *http://en.wikipedia.org/wiki/Support_vector_machines*.

[SYAD05] K. Seymour, A. YarKhan, S. Agrawal, and J. Dongarra. Netsolve: Grid enabling scientific computing environments. In Lucio Grandinetti, editor, Grid Computing: The New Frontier of High Performance Computing, volume 14 of Advances in Parallel Computing. Elsevier, 2005.

[Tal02] E. G. Talbi. A taxonomy of hybrid metaheuristics. Journal of Heuristics, 8(5):541–564, 2002.

[TD03] G. Tsouloupas and M. Dikaiakos. GridBench: A Tool for Benchmarking Grids. In Proceedings of the 4th International Workshop on Grid Computing (GRID 2003), page 60, Phoenix, Arizona, USA, November 2003.

[TD05] G. Tsouloupas and M. Dikaiakos. Design and Implementation of GridBench. In Advances in Grid Computing - EGC 2005: European Grid Conference, volume 3470 of Lecture Notes in Computer Science, pages 211–225. Springer, 2005.

[TeG] TeraGrid, accessed Feb 2010 [Online]. Available: *http://www.teragrid.org/*.

[Tor04] J. Tordsson. Resource brokering for grid environments. Master's thesis, Umea University, June 2004.

[Try39] R. C. Tryon. Cluster analysis. Edwards Brothers, Inc., 1939.

[TSS] TOP500 Supercomputing Sites, accessed Jan 2010 [Online]. Available: *http://www.top500.org/*.

[Val02a] J. J. Valdés. Similarity-based heterogeneous neurons in the context of general observational models. Neural Network World, 12:499–508, 2002.

[Val02b] J. J. Valdés. Virtual reality representation of relational systems and decision rules. In P. Hajek, editor, Theory and Application of Relational Structures as Knowledge Instruments, Meeting of the COST Action 274, Nov 2002.

[Val03] J. J. Valdés. Virtual reality representation of information systems and decision rules. Lecture Notes in Artificial Intelligence, 2639:615–618, 2003.

[Val04] J. J. Valdés. Building virtual reality spaces for visual data mining with hybrid evolutionary-classical optimization: Application to microarray gene expression data. In IASTED International Joint Conference on Artificial Intelligence and Soft Computing, ASC'2004, pages 161–166. ACTA Press, Anaheim, USA, 2004.

[VB05] J. J. Valdés and A. J. Barton. Virtual reality visual data mining with nonlinear discriminant neural networks: application to leukemia and alzheimer gene expression data. In Proceedings of the 2005 IEEE International Joint Conference on Neural Networks (IJCNN '05), volume 4, pages 2475–2480 vol. 4, 2005.

[War63] J. Ward. Hierarchical grouping to optimize an objective function. Journal American Statistical Association, 58:236–244, 1963.

[Wei07] Aaron Weiss. Computing in the clouds. netWorker, 11(4):16–25, 2007.

[wlc] WLCG - Worldwide LHC Computing Grid, accessed Feb 2010 [Online]. Available: *http://lcg.web.cern.ch/LCG*.

[Wol96] David H. Wolpert. The lack of a priori distinctions between learning algorithms. Neural Comput., 8(7):1341–1390, 1996.

[Wol03] Rich Wolski. Experiences with predicting resource performance on-line in computational grid settings. SIGMETRICS Perform. Eval. Rev., 30(4):41–49, 2003.

[WSH99] Rich Wolski, Neil T. Spring, and Jim Hayes. The network weather service: a distributed resource performance forecasting service for metacomputing. Future Generation Computer Systems, 15(5–6):757–768, 1999.

[Zha04] Harry Zhang. The Optimality of Naive Bayes. In Valerie Barr and Zdravko Markov, editors, Proceedings of the International Conference on Field Laser Applications in Industry and Research (FLAIR '04). AAAI Press, 2004.