Riding Out the Storm: How to Deal with the Complexity of Grid and Cloud Management

Jesús Montes · Alberto Sánchez · María S. Pérez

Abstract Over the last decade, Grid computing paved the way for a new level of large scale distributed systems. This infrastructure made 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. In time, cloud computing emerged as a new type of large scale distributed system, inheriting and expanding the expertise and knowledge that have been obtained so far. Some of the main characteristics of Grids naturally evolved into clouds, others were modified and adapted and others were simply discarded or postponed. Regardless of these technical specifics, both Grids and clouds together can be considered as one of the most important advances in large scale distributed computing of the past ten years; however, this step in distributed computing has came along with a completely new level of complexity. Grid and cloud management mechanisms play a key role, and correct analysis and understanding of the system behavior are needed. Large scale distributed systems must be able to self-manage, incorporating autonomic features capable of controlling and optimizing all 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 and cloud computing, the vast complexity of these systems can make this task extremely complicated. But large scale distributed systems complexity could only be a matter of perspective. It could be possible to understand the Grid or cloud behavior as a single entity, instead of a set of resources. This abstraction could provide a different understanding of the system, describing large scale behavior and global events that probably would not be detected analyzing each resource separately. In this work we define a theoretical framework that combines both ideas, multiple resources and single entity, to develop large scale distributed systems management techniques aimed at system performance optimization, increased dependability and Quality of Service (QoS). The resulting synergy could be the key

to address the most important difficulties of Grid and cloud management.

Keywords Grid computing · Cloud computing · Autonomic computing · Theoretical models

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 the combination of 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 (81.4 % according to the 06/2012 list at TOP500 Supercomputing Site [1]).

With the emergence of the Internet and global interconnection, new forms of large scale distributed computing appeared. Resources could not only be combined within local, private networks, but also geographically dispersed ones, enabling access to a potentially unlimited pool of computational power. Several initiatives have attempted this, but it was in the late 1990s when the idea was deeply explored and developed with the appearance of *Grid computing* [2], trying to address all possible related issues. Cloud computing [3] has continued in this same path, exploring new alternatives for large scale distributed computation.

The Grid is frequently seen as a massive pool of heterogeneous and geographically distributed computational resources. These resources are coordinated, but not subject to a centralized control. They use standard, open and general-purpose protocols and interfaces to interact and, finally, the resulting system delivers non-trivial qualities of service [4]. The Grid allows 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.

Cloud computing shares some of these characteristics, but has also successfully explored and developed a much more market-oriented perspective. Cloud infrastructures are devoted to provide reliable services, delivered through next-generation data centers, and built over virtualized computing and storage technologies. The idea is to enable users (also called *consumers*) 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. Clouds can provide an enormous amount of computational power that many advanced applications can benefit from.

Nevertheless, 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 de-centralized 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 managing this complexity enables to provide scalability, dependability, quality of service, etc.

Apparently, cloud computing addresses some of these issues, or at least reduces its impact. 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. Not only Grid and cloud, but other similar large scale distributed computing concepts such as utility computing and Internet computing were also involved in this debate. 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" [5].

Regardless of their specific differences, most Grid and cloud problems are still the same. Both need to be able to manage large scale (yet somehow different) infrastructures. 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 distributed computations that execute on those resources. Complexity is the source of most Grid and cloud issues.

One of the most common strategies to handle complexity in large scale distributed systems is the incorporation of *autonomic computing* features. Autonomic computing [6] 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 systems multiple management tasks are performed automatically and transparently, providing (among many other features) reliability, dependability and quality of service. The different aspects related to autonomic computing have proved to be beneficial for Grid and cloud computing in many ways, making it possible to achieve some of its most ambitious goals. Incorporating autonomic features into system management mechanisms strongly facilitates the system administrator task, which in a large scale system would be otherwise overwhelming. Proof of this is that current Grid and cloud management techniques include autonomic characteristics [7–12] dealing with each independent resource's separately and automatically optimizing its behavior in order to achieve an improvement in global performance. This approach has been successful so far, enabling the creation of very large distributed infrastructures that have played a key role in important scientific advances in the last decade, such as OSG [13], TeraGrid [14] or the EGEE [15] project and their successors XSEDE [16] and EGI-InSPIRE [17].

This system management approach focused on individual resources seems to be only a part of what it is traditionally done when managing computing systems. Less complex systems such as individual machines or clusters are usually regarded and analyzed also 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 Grids, clouds and other systems, several questions arise, regarding its autonomic management: Why is large scale distributed systems management different? If the concepts of **Grid** and **cloud** exist from theoretical point of view, why is there no translation of them into management terms? Can both visions (multiple resources and single entity) be combined, as it is done in other systems, to improve system management techniques? What are the implications? This work attempts to answer these questions, trying to define the theoretical foundations necessary to study and develop new and better Grid and cloud management techniques.

With this aim we define a theoretical framework and a set of formalisms to provide the necessary basis to fully analyze and understand a large scale distributed system such as a Grid or a cloud. The knowledge obtained from this analysis can be

used in conjunction with an autonomic approach to develop effective and efficient system management techniques that address the main issues of these modern distributed infrastructures. Our vision serves as well as the theoretical basis to develop a single entity abstraction of the system that is both complete and useful. The paper is organized as follows: Section 2 describes related work; Section 3 discusses the issues related to the application of autonomic computing techniques to large scale distributed systems; Section 4 describes the proposed theoretical model and its specific application for the cases of Grid and cloud computing; Section 5 describes a proposed framework for application of this theoretical model and continuous loop of system management improvement; Section 6 describes some cases of study of real Grid and cloud projects that share ideas and are strongly related with the proposed theoretical model; finally, Section 7 presents the final conclusions and discussion motivated by this work.

2 Related Work

There are different research works related to the characterization of Grids and clouds behavior. Benchmarks [18] are mainly used for characterizing the performance behavior of a system under representative workloads. Several Grid benchmarking [19] approaches have arisen. NAS Grid Benchmark (NGB) [20] and GridBench [21] are some examples. Nevertheless, Grid benchmarking is not enough for modeling the dynamic behavior of Grids. Benchmarks provide only pre-defined, static system workloads and their analysis is based on their results. Furthermore, these techniques depend on the accuracy of benchmarks and the suitable selection of inputs and configuration parameters. It is not always easy to select a realistic workload.

Ogura et al. [22] perform a study of the behavior of virtualization software on multi-core platforms in a cloud running scientific and transactional application. The main goal of these experiments is to analyze how the virtual machine configuration affects the performance of applications. Authors use the NPB NAS [23] and TCP-H [24] benchmarks. Although the results obtained are useful for tuning these environments under different workloads, this study does not address the complexity of the use of large clouds.

One step further than benchmarking is modeling. Bratosin et al. [25] provide a formal description of Grids by means of Colored Petri Nets (CPN), which can be used for simulation. Our proposal is not a simulation but an abstract model of a large scale environment (both Grid and cloud), which makes easier the application of more efficient management techniques.

Chan et al. define in [26] a theoretical graph-based model of computing clouds and model-based testing criteria for assuring the quality of applications running on top of the cloud. This work is oriented to small clouds and not to large clouds. It is mainly used for the dynamic composition of clouds, such as merging clouds, splitting clouds, etc. Our purpose is to provide a theoretical framework for making easier the management of large scale environments (both Grids and clouds), hiding their complexity.

Finally, there are some methodologies for building Grid and cloud-based software development projects. Ostberg et al. present in [27] a methodology for Service-Oriented Architecture (SOA) design intended to Grid and cloud developments. The objective of this methodology is to increase flexibility and reduce complexity. Unlike this work, our approach is design model-agnostic. Furthermore, we find differences in the way of dealing with the complexity of Grid and cloud environments. Ostberg et al. methodology does not distinguish Grids from clouds. Finally, only Grid case studies are presented in the above-mentioned paper.

3 Autonomic Management of Large Scale Distributed Systems

As it has been discussed, autonomic computing can provide a practical solution to the problem of managing highly complex systems, such as Grids and clouds. In order to do so, a thorough analysis of this problem is presented in this section, analyzing the autonomic management issues that can be identified in Grids and clouds and developing the basic ideas to efficiently address them.

3.1 Autonomic Management Issues in Grid and Cloud Computing

Over the past decades, as global networking became reality, several different incarnations and definitions of what could be called Grid systems appeared [28]. The most recent cloud computing paradigm has followed an analogous path [29, 30]. 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 [5]. Despite the differences between Grids and clouds, the following five main characteristics can be observed to some extent in most of them:

1. **Distributed:** The system is composed of a set of resources that are logically and sometimes 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 system are simultaneously used by multiple entities (clients, applications, external users, etc.).
3. **Service-oriented:** The system is designed to provide a function or functions in the form of a set of services. In some cases (specially clouds) this evolves into a market-oriented model in which the relation condition of the services provided is controlled by more sophisticated procedures such as Service Level Agreements (SLAs).
4. **Heterogeneous:** In many large scale distributed systems (specially Grids) the computing resources involved are clearly different. Typical examples of this diversity are different architectures, operating systems or network protocols. Clouds tend to be much more homogeneous infrastructures than Grids and this feature is not normally present. Only advanced cloud systems, such as hybrid clouds, can present this characteristic in a way that could be compared to Grids.
5. **Non-centralized:** Even though most large scale distributed systems have global infrastructures that allow their different elements to cooperate (such as Globus [31] and GLite [32] in Grids), sometimes resources actually belong to different owners that keep a high degree of control over their property. This includes from typical resource-sharing Internet projects (like the Seti@home project [33]) to modern Grids and hybrid clouds. The degree of administrative decentralization depends on the type of system (e.g. it is not the same for the Grid5000 [34] platform as for a typical hybrid cloud combining a local Eucalyptus [35] infrastructure with Amazon EC2 [36]).

Most large scale distributed systems are, in consequence, not only distributed in nature, but sometimes also heterogeneous, non-centralized and in most cases composed of non-dedicated or shared resources. Incorporating autonomic features to such complex infrastructures is not a simple task. These properties, added to the fact that these 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. As it has been already explained, features such as service orientation, heterogeneity and non-centralized control can be present only up to a certain degree, depending on the specific system studied. Our approach attempts to analyze and model a scenario as generic as possible and, therefore, all possible characteristics are considered. It seems reasonable to assume that a management system capable of dealing correctly and efficiently with all five characteristics would perform in a similar way on a less complex infrastructure.

When adopting an autonomic computing approach, the system complexity has direct impact in its four main areas: self-configuration, self-healing, self-optimization and self-protection.

3.1.1 Self-Configuration Issues

Most traditional distributed approaches (cluster computing, centralized client-server architectures, etc.) very often present desirable characteristics such as stability, homogeneity or simple and clear behavioral patterns. In these systems, reconfiguration 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.

Clouds present yet another scenario. Compared to Grids, typical clouds can be seen as relatively more stable infrastructures, generally composed of more homogeneous resources and presenting a centralized administration. The desired elasticity of cloud services requires the system to dynamically adapt its configuration to the changes in the use of the services being provided. The capability of the system to adapt to these changes cannot be simply based on local, resource-centered policies. In addition, the responsibility of addressing this issues is divided between the system and the user application, depending on the cloud service provisioning model (IaaS, PaaS or SaaS). The possible coexistence of many different applications and clients on the same cloud elevates the complexity of this problem and makes clear the need of an efficient autonomic approach.

3.1.2 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. In clouds the inherent shared nature of the infrastructure, with different types of services, applications and SLAs in place can experience analogous kinds of resource faults, specially if the system is not efficiently managed. The lesser degree of cohesion of Grids and clouds compared with traditional systems dilutes the concept of failure based on the loss or degradation of resources. Grids and clouds 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 the quality of the services provided instead of the state of its internal resources.

3.1.3 Self-Optimization Issues

A deep system's behavior understanding enables to develop advanced management policies and strategies, designed to make the most of the system resources available. In traditional distributed computing the 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 creates a completely new and different framework, forcing performance optimization techniques to be adapted to these new conditions. Clouds are somehow half-way between traditional clusters and Grids. In this case the physical resources that compose the infrastructure are generally more controlled and under the same management policies. As in the case of *self-configuration*, cloud complexity in this area is caused by the diversity of service-level agreements for applications and clients sharing the system.

3.1.4 Self-Protection Issues

Given the distributed, heterogeneous and decentralized nature of Grids, proactive identification and protection from external attacks are crucial aspects. 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. The massive resource interaction present in Grid systems can, however, render these techniques insufficient, creating the need for protection mechanisms focused also on global aspects of the system. This is true in clouds as well, where different applications share the same pool of computational resources, expecting a secure environment. In this sense, virtualization techniques are an important advance, creating *sandbox* environments that can effectively isolate distributed applications. The extreme complexity of the system (specially in the case of Grids) makes this task difficult, requiring to study the system as a whole and a deep analysis of the resources internal and external interactions.

3.2 Single Entity vs. Multiple Entities

One of the most puzzling aspects of Grid and cloud systems is that they are considered as single elements in theory but, when it comes to practice, in management related issues they are treated as a set of independent, sometimes 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 and clouds, it becomes clear that the proper tools for making this abstraction are yet to be established.

3.3 Large Scale Distributed Systems Management: *Resource-Level* vs. *Service-Level*

Distinguished by their point of view, autonomic management techniques in Grid and cloud 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 quite straightforward, but a detailed analysis reveals that most of the typical characteristics of a large scale distributed system limit its efficiency. The shared, heterogeneous and non-dedicated nature of the system increase complexity, but it is the non-centralized aspect the one that becomes the great difficulty (specially in Grids and hybrid clouds). 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 performance optimization mechanisms capable of migrating jobs throughout the system, detailed security directives designed to protect against complex coordinated attacks, etc. The high level of resource control usually required in order to apply those techniques, however, would make it extremely hard to deploy them all over the system in

an unified way. Therefore advanced *resource-level* management will in most cases only be applied locally (limited to corporative networks, parts of a hybrid cloud, 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 cloud infrastructures, where the quality of service is the key factor; however, as the management policies have to deal with the whole system, 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. *Service-level* management can benefit from a general representation of the system global state, specially if it is service oriented like clouds and most Grids.

4 A *Service-Level* System Management Model

Service-level management could strongly benefit from a single entity point of view. To achieve this, it is first necessary to formally define the system behavior theoretical model to be used in this case. In this section a *service-level* management model is presented, inspired by the single entity perspective. This model is based on the basic concepts and taxonomy of dependable systems by Avizienis et al. [37]. Figure 1 presents the general case of this model.

Following the single entity point of view, the Grid or cloud can be considered as a single **system**. From a theoretical perspective a Grid or a cloud, 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 or cloud **resources** (computing and storage servers, network nodes, etc). The function or functions are what the system 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 being provided. This functionality is presented by the **service** or set of services. The interaction between the external clients and the system is made through the **service interface**.

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

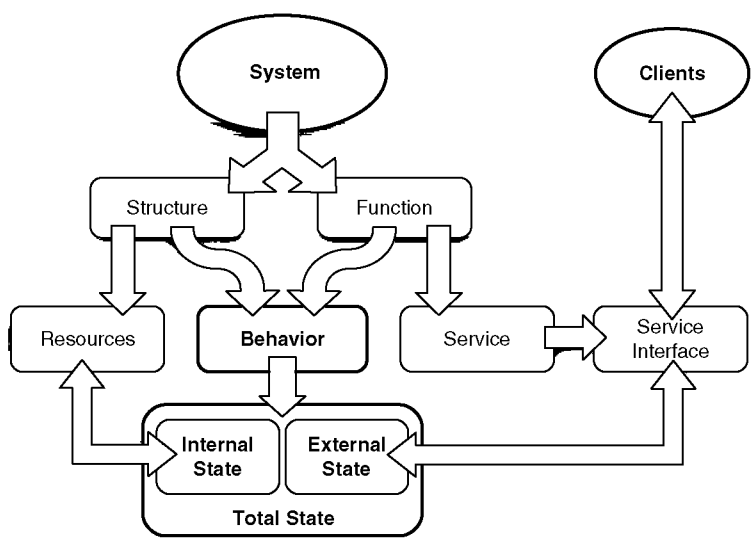

Fig. 1 Structure, function and state: general case. The system is divided into structure (resources) and function (services), but both are responsible for the system behavior, which is expressed by its total state.

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 system structure situation. The combination of both internal and external state produces the system **total state**, that describes the system **behavior** in a complete way.

Resource-level management focuses on aspects related to the system 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. This is a general model that can be applied to any form of large scale distributed system presenting the characteristics previously described. The cases of Grids and clouds are studied in detail in the following subsections.

4.1 *Service-Level* Total State Model Case I: The Grid

When analyzing the specific scenario of a Grid infrastructure, it is clear that its main source of complexity is located in the system resources, i.e. the machines and network links that the system is built on top of. This resource infrastructure is usually divided into virtual organizations or other similar kind of administrative domains. These are designed to facilitate management tasks such as privacy and coexistence of multiple simultaneous security policies and not focused on performance or dependability. Grid resources are shared between partners and usually there is no superior organization with full system management rights over the entire infrastructure.

Figure 2 shows a more detailed version of the *Service-level* total state model for the specific case of Grid systems. Autonomic management tools need to be focused on this internal complexity and also how it affects the external state observed through the service interface.

4.2 *Service-Level* Total State Model Case II: The Cloud

When studying the sources of system complexity, clouds can be seen as the opposite to Grids. To avoid all possible problems and conflicts that the non-centralized nature of the Grid can cause,

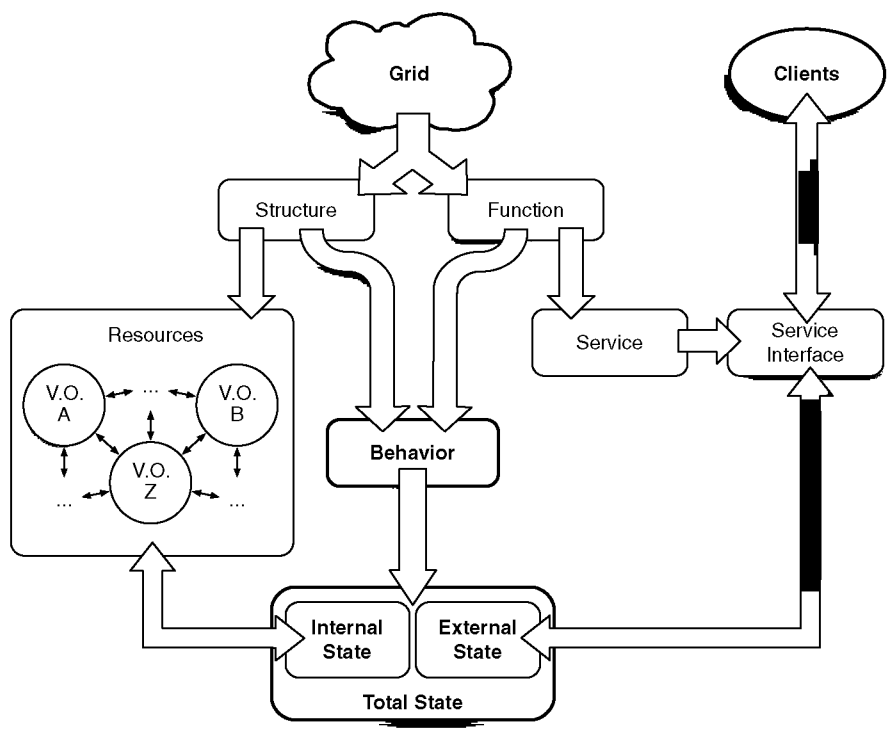

Fig. 2 Structure, function and state: the Grid case. Here the complexity of the system structure is detailed, since it is its main source of complexity

typical clouds are constructed over a more traditional distributed systems infrastructure, on a typical cluster-like data center. This simplifies resource management, as usually cloud resources are more or less homogeneous and with centralized administrative control. This more controlled infrastructure enables to develop and provide a much more sophisticated set of services, and this is where the main source of cloud complexity appears. Common cloud services incorporate advanced characteristics such as complex SLAs and system abstraction through virtualization. Service complexity can be present in two ways:

- Complexity within the service: This is the most typical source of complexity in cloud systems. It is caused by the coexistence of several clients/consumers making use of the same service. The system must provide expected quality of service for all consumers regardless of the specific conditions and use pattern of each one of them. A common example of this can be seen in any commercial cloud providing IaaS or PaaS web hosting services (e.g. Amazon EC2 or Google App Engine [38]). In this scenario many different cloud consumers could be using the cloud resources to run different web applications with different configurations (number of VMs, etc) and SLAs (maximum number of simultaneous connections, expected latency, etc). The cloud management system has to handle these situations.
- Complexity between services: It refers to the coexistence and interaction of different types of services in the same cloud (e.g. a web application service and a data storage service).

Figure 3 shows a more detailed version of the *Service-level* total state model for the specific case of cloud systems. Autonomic management tools need to be focused on this external complexity and also how it is related to the state of the system resources.

4.3 *Service-Level* Autonomic Management

Taking the presented theoretical model as a basis, we can analyze now how the different autonomic

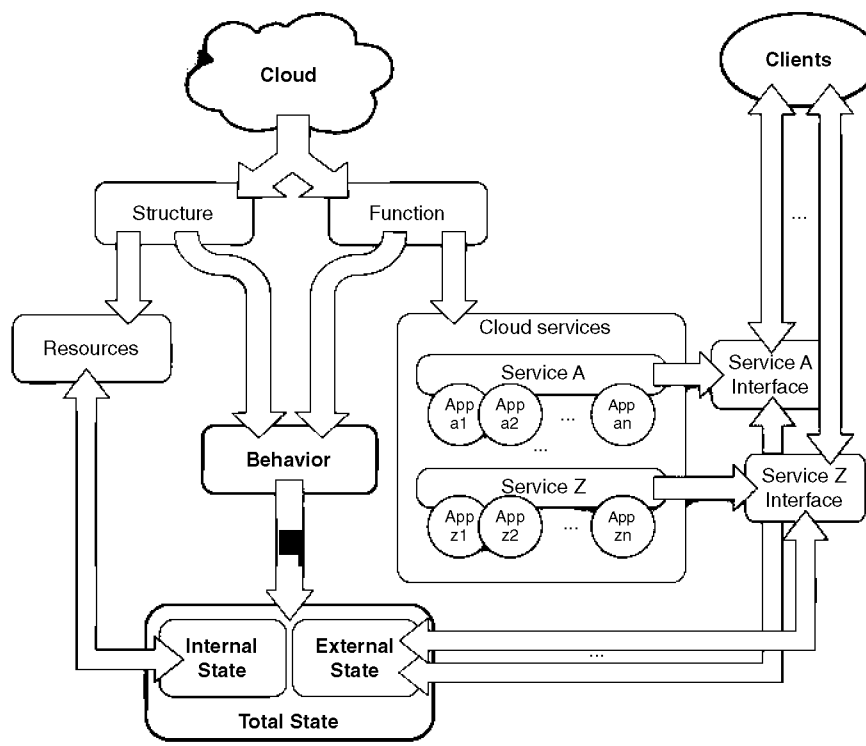

Fig. 3 Structure, function and state: the cloud case. In this type of systems the main source of complexity is usually the advanced service or set of services provided. The system must handle multiple applications and service interfaces

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 should be installed, where and the deployment order, in order to achieve the system 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 protection against external, global attacks, specially those aiming at complex resource and service 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 Section 3, *self-healing* is probably the autonomic computing area most affected by large scale distributed systems special characteristics. Grid and cloud services fault tolerance issues are directly related to the system global state and require a *service-level* management approach in order to be properly handled. The system unique features require basic fault tolerance concepts to be redefined, eliminating the possibility of directly inheriting 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 or cloud performance is directly dependent on the system resources, but also on the services usage patterns. Additionally, the system complexity once again becomes an issue, making any attempt of self-optimizing autonomic management from a *resource-level* point of view extremely complicated. *Service-level* management seems to be the ideal approach here, as it handles system complexity without being overwhelmed by it, focusing at the same time in the system's total state.

As *self-healing* and *self-optimizing* are the two autonomic computing areas that can benefit mostly from a service-level, total state mode, they are studied in more detail in the following subsections.

4.3.1 Self-Healing: Failures, Errors and Faults

It can be said that a large scale distributed system 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**.

This initial set of basic definitions can serve as a starting point for development of *self-healing* capabilities. Its is important to remember that most modern large scale distributed systems (specially clouds) deal with sophisticated service provisioning models. The concept of quality of service and the different SLAs on which cloud services can be provided has to be incorporated in the model. If we analyze this from a single-entity, total state point of view, it is clear that each system state has to be associated with a specific QoS. Depending on how this is defined and measured, more than one state could share the same QoS, creating subsets of system states with equivalent qualities of service, that could be associated with specific SLAs.

Figure 4 illustrates an example of the total state of a system, represented in the form of a finite state machine [39, 40]. States are grouped in three subsets that indicate distinguishable differences in the quality of the services provided (QoS_1, QoS_2 and QoS_3). These subsets could be associated with different SLAs, at the discretion of the cloud services provider. The incorrect state

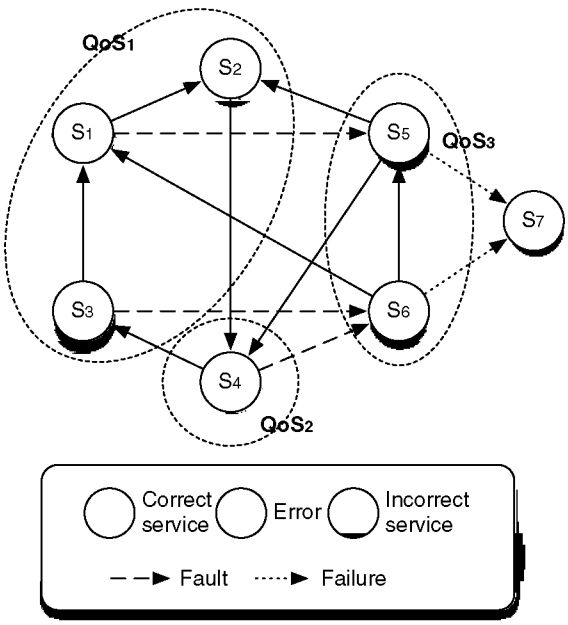

Fig. 4 An example of total state model including different system states, levels of QoS, possible faults, errors and failures

(S_7) is excluded from these QoS subsets because it represents a system state where the QoS is not acceptable by any standards (maybe the service is not even being provided).

The use of this fault model allows to model single entity system failures and qualities of service simultaneously and provides the basic tools to build fault tolerance and QoS mechanisms.

4.3.2 Self-Optimizing: Performance and Quality of Service

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 and clouds, 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.

5 The Behavioral Modeling Cycle

A *behavioral modeling* process enables to create a total state model of a large scale distributed system. This process should present the following properties:

1. Based on observation. In order to identify the different aspects that have an impact on the system behavior, the process has to be at least partially based on data gathered from resource monitoring, service accounting, performance logs and other information sources. This ensures the total state modeled is consistent with the system operation.
2. Completeness. The model generated must provide information of both internal and external states. Since the whole total state of the system has to be described, the behavioral model produced has to describe relevant aspects of both structure (resources) and function (services) of the Grid or cloud.
3. Relevance to service: The model generated must be relevant to the service being provided. Since the ultimate goal of Grids and clouds is providing a service, the behavioral model generated by the process has to incorporate information about how this service is being provided in terms that can be related to QoSs, SLAs and so on.

Aside from these three characteristics, the *behavioral modeling* process can be based on

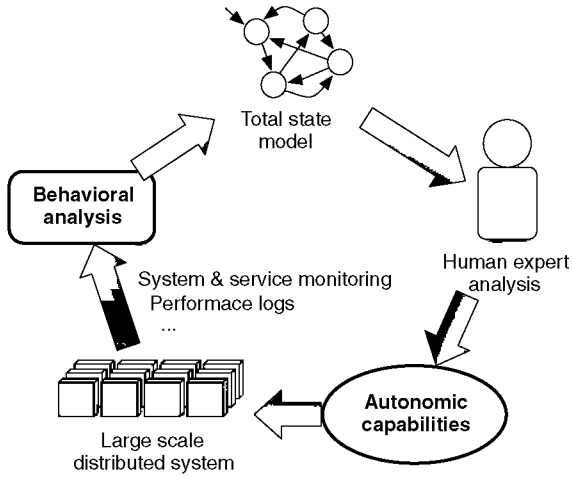

Fig. 5 The behavioral modeling cycle

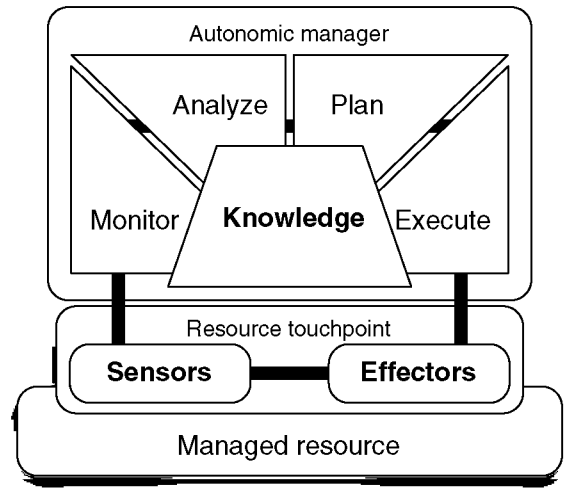

Fig. 6 Autonomic components and the *MAPE* loop: *Monitor, Analize, Plan, Execute.* The loop (and the *Knowledge* element) is part of the autonomic manager that operates the managed resource through a resource touch-point. The sensors allow the autonomic element to observe the resource and the effectors to dynamically control it

analytical models, statistical analysis, *ad hoc* architectures, data mining, or any other formalism desired.

The system behavior can be observed and modeled. The result of this analysis can be used to develop autonomic capabilities and in consequence improve the system performance, dependability, QoS, etc. This process can be performed as a loop, improving the system autonomic capabilities. In each iteration of this *behavioral modeling cycle* (depicted in Fig. 5) the *behavioral modeling* stage can be focused on the particular area of autonomic computing that needs to be developed (*self-optimization*, *self-healing*, etc.), generating new useful understanding of the system each time.

5.1 Relation to the *MAPE* Loop

The *MAPE* loop (*Monitor, Analize, Plan, Execute*) is the general process followed by autonomic system elements [6]. It can be seen in Fig. 6, along with all the other main autonomic components. There is a strong relation between the *MAPE* loop and the *behavioral modeling cycle*. The first describes how an autonomic component works and the second what needs to be done to create it. The key of this relation is the *Knowledge* element of the autonomic component. This *Knowledge* serves as the *heart* of the component, organizing the four loop stages and giving them purpose. The *behavioral modeling cycle* makes it possible to obtain this knowledge in a way that is based on observed behavior, complete and relevant to the service being provided (the three main characteristics of any behavioral model). The connection of these two linked processes is the basis for developing autonomic capabilities in large scale distributed systems.

6 Cases of Study

In Sections 3, 4 and 5 our proposed autonomic management theoretical model is presented, described and analyzed. Since this is a new contribution, it does not exist yet a real Grid or cloud management project that specifically and/or completely follows our theoretical model. Our model has been derived and developed from the study of existing techniques, trying to define and formalize a basic common theoretical framework. Therefore there are many existing Grid and cloud management that share some of these ideas. Most of these initiatives have been presented in Section 2. There are some among them that can be studied very clearly in terms of our model, although they only cover some areas of it and could be considered

more specific or limited to certain scenarios. This section studies these examples of known Grid and cloud projects in order to (i) illustrate how the ideas behind our theoretical model can be successfully applied in real Grid and cloud scenarios (they have been partially applied already) and (ii) analyze how the application of the full extent of our theoretical model can benefit even more large scale distributed systems management.

6.1 Claudia

Most of cloud systems developed so far follow an IaaS approach. Many well-known cloud initiatives, such as Amazon EC2, Eucalyptus, OpenNebula [41] or Nimbus [42], are infrastructures that can be used to execute HPC applications. IaaS allows consumers to manually allocate VMs in terms of the use of their own applications. Thus, these IaaS systems can be seen as an utility where consumers run their applications. Nevertheless, whereas consumers are interested in aspects of the application life-cycle like application deployment or scalability, IaaS cloud providers are focused on the internal aspects of the infrastructure. The resulting complexity can have an impact on management tasks such as VMs allocation, which can have then an impact on the final QoS the consumer perceives.

To reduce complexity and improve cloud management, cloud platforms have to evolve from a basic infrastructure administration to autonomic service management. The development of Claudia [43], as the IaaS Cloud Service Manager of the EU-funded RESERVOIR project [44], is a first step in this direction. Claudia is located on top of the IaaS infrastructure and it controls the service life-cycle, dynamically allocating VMs regarding SLAs and other business policies. It relies on a service manifest where the service components, requirements, QoS and business policies are declared. Claudia developers have identified four basic goals [43], which clearly fit with some aspects of our proposed autonomic management model:

1. Service abstraction level oriented to define and manage services including relations between components to provide a deployment order. This is a starting point to automatically deploy applications in a self-configuring way.
2. Automatic Scalability. How the service scales is a key factor of the QoS provided to the consumer. In Claudia consumers define the best way the application has to be scaled regarding their own experience and knowledge.
3. Smart Scaling. This refers to the specification of the rules that constrain the automatic scaling. The application of self-optimizing techniques based on these rules is advisable since there are more factors that can impact on its performance.
4. Avoiding Cloud Vendor Lock-in. Nowadays each cloud provider has its own non-standard interface. This makes it difficult to share data and services between different cloud providers. It would be interesting to seemingly be able to use different providers according to business policies. When this effective interoperability occurs,[1] the selection of the suitable cloud providers would be made in an autonomic way.

Claudia works as an abstraction layer of the IaaS infrastructure to control services as a whole, and therefore is a clear example of the single-entity vision we are proposing. The idea is to change the usual way consumers interact with clouds. Instead of dealing with VMs management on different cloud platforms (somehow similar to *resource-level* management), an abstract vision of the cloud is used for application management. In this case, this vision is related to the application life-cycle, which is the part consumers are interested in. Nevertheless, in order to represent the total state of the service, both its external and internal state are required. Claudia is placed on top of the IaaS infrastructure and therefore it has knowledge only of the external state (through the service interface). The internal state of the IaaS infrastructure (how the service is working) is hidden inside the RESERVOIR project architecture (in fact, Open Nebula is used as internal Virtual Execution Environment Manager). Thus,

[1]The group Distributed Management Task Force is currently working on it.

applying our proposed theoretical model to this case study could help to better understand the system and improve its management. In any case, Claudia stands as a promising starting point for cloud autonomic management.

6.2 GloBeM

In 2010 we presented Global Behavior Modeling (GloBeM) [45], a methodology designed to identify and explain regularities in global Grid behavior. Its main objective is to build an abstract, descriptive model of the global system state of the Grid. This enables the model to implicitly describe the interactions between entities, which has the potential to unveil non-trivial dependencies and other significant behavioral aspects. These unique features make GloBeM particularly useful in Grid management, especially because it provides the means to capture complex interactions among components in a simple yet comprehensive finite state machine (FSM) behavioral model. These states can be directly seen as distinctly identified behavioral patterns.

GloBeM follows a set of procedures in order to build such a model, starting from monitoring information that corresponds to the observed behavior. These basic monitoring data are then aggregated into *global monitoring parameters*, representative of the global Grid behavior instead of each Grid resource separately. This aggregation can be performed in different ways, but it normally consists in calculating global statistic descriptors (mean, standard deviation, skewness, kurtosis, etc.) values of each basic monitoring parameter for all Grid resources present. This ensures that global monitoring metrics are still understandable from a human perspective. Global information undergoes a complex analysis process in order to produce a global behavior representation. This process is strongly based on machine learning and other knowledge discovery techniques, such as virtual representation of information systems [46, 47]. In [48], the techniques needed to create global behavior prediction models for Grid systems were defined. Global behavior prediction benefits Grid management, specially in areas such as fault tolerance or job scheduling.

Although GloBeM has been designed for Grid systems, it has been successfully used to analyze clouds too. For instance, in [49] it is shown how GloBeM can be used to obtain a technique to address a stable throughput for each individual access to the cloud storage service BlobSeer [50]. Substantial improvements in the stability of individual data read accesses under MapReduce workloads were achieved.

From the perspective of our contribution, GloBeM can be used as a specific technique for *behavioral modeling*. Since GloBeM uses information from monitoring both internal resources (system structure) and system services (system function), its model can serve as a representation of the total state of the system. The use of knowledge discovery techniques to create it reduces the possible impact of a biased analysis manually performed by a system administrator.

GloBeM models are tailored for a specific configuration of resources and services, and cannot be transported from one system to another. This enables them to precisely model the characteristics of each systems, but at the expense of becoming less general. The GloBeM methodology could be successfully applied to many different systems, but it would generate a completely different model for each one of them.

7 Conclusions

Nowadays the use of large scale distributed systems enables users to execute demanding computing and data-intensive applications, renting resources suitable for their application needs. In general, Grid and cloud systems provide to some extent user-friendly interfaces that improve the interaction with the user, hiding behind them the inner complexity of the system. From the point of view of resource providers, however, the management of such a complex infrastructure is sometimes overwhelming. This complicates achieving the desired levels of dependability, quality of service, etc.

The contribution presented in this paper constitutes an important step in the construction of a theoretical formalism that helps to enhance the management of these infrastructures. Combining

and expanding the ideas gathered from the study of many past and current system management techniques, we have defined a generic theoretical framework that can be used as a basis for the development of efficient large scale distributed systems. This formalism provides an abstract understanding of the underlying large scale system, which unlike other models, allows us to analyze and manage the system as a single entity. The main advantages of our approach are its completeness and simplicity, which makes the application of efficient management techniques easier. Moreover, our approach is independent of the specifics of each kind of system (resources, services, etc.).

One of the main motivations for developing this formalism has been to provide the theoretical foundations of a single entity vision of a Grid or a cloud. So far the abstraction mechanisms being used (specially in cloud computing) provide only a biased, incomplete vision of the system, presenting only information related to the external state of the system. We have identified and explained the need of a total state model that incorporates internal as well as external state information. This model may not be necessary from the point of view of service-client interaction, but it is definitely required from a system management perspective. Nowadays the high-level, external vision and the low-level, resource-based vision are separated. We defend the need of combining them to achieve optimal performance. The structure and function of a system cannot be separated, since they are both parts of its behavior. The conceptual representation of this idea is the proposed *service-level* total state model.

The knowledge obtained from analyzing and understanding the complete system behavior can be used to develop efficient management techniques where autonomic computing play a key role. In this paper we have proposed and justified the use of autonomic computing to manage extremely complex systems such as Grids and clouds. We have analyzed the possible issues that may emerge in the different areas of autonomic computing and how to address them, proposing theoretical procedures and explaining the implications that using a total state model may have. Our analysis suggests that self-configuration requires a resource level approach and self protection a hybrid approach. Self-healing and self-optimizing are the two areas that can benefit most from a service level, total state model. We have also proposed the use of a *behavioral modeling cycle* to continuously improve the system autonomic capabilities. This cycle defines the basic theoretical sequence of procedures that have to be performed in order to develop the desired system features. We have integrated this cycle in the typical autonomic process, detailing its relation to other basic autonomic concepts such as the MAPE loop.

Even though, to the authors' knowledge, no such formal model as the one here presented exists, some past and current large scale distributed management initiatives seem to follow a similar direction. During the development of our model we have studied in detail many of them, and we have analyzed here two examples (Claudia and GloBeM) in order to illustrate how similar basic ideas are already being used in Grids and clouds. None of these initiatives constitutes a theoretical framework for the generic application of manageable and useful management techniques. Our main objective in this paper has been to develop a unified, generic theoretical formalism to define a single entity vision of a Grid or a cloud that can be used for autonomic management purposes.

References

1. TOP500 Supercomputing Sites: http://www.top500.org/ (online). Accessed Jul 2012
2. Kesselman, C., Foster, I. (eds.): L The Grid: Blueprint for a New Computing Infrastructure. Morgan Kaufmann, San Mateo, CA (1998)
3. Weiss, A.: Computing in the clouds. Networker **11**(4), 16 (2007)
4. Foster, I.: What is the Grid? A three point checklist. Grid Today **1**(6). http://www-fp.mcs.anl.gov/~foster/Articles/WhatIsTheGrid.pdf. (2002)
5. Foster, I.T., Zhao, Y., Raicu, I., Lu, S.: Cloud computing and Grid computing 360-degree compared. CoRR abs/0901.0131 (2009)

6. IBM: An Architectural Blueprint for Autonomic Computing, 4th edn. IBM Autonomic Computing White Paper (2006)
7. Buyya, R., Abramson, D., Giddy, J.: 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, vol. 1, pp. 283–289 (2000)
8. Krauter, K., Buyya, R., Maheswaran, M.: A taxonomy and survey of Grid resource management systems for distributed computing. Softw. Pract. Exp. **32**(2), 135 (2002)
9. Siddiqui, M., Fahringer, T.: GridARM: Askalon's Grid resource management system. In: Advances in Grid Computing - EGC 2005 - Revised Selected Papers. Lecture Notes in Computer Science, vol. 3470, pp. 122–131. Springer Verlag GmbH, Amsterdam, Netherlands. ISBN 3-540-26918-5 (2005)
10. Sánchez, A., Montes, J., Pérez, M.S., Cortes, T.: An autonomic framework for enhancing the quality of data Grid services. Future Gener. Comput. Syst. **28**(7), 1005 (2012)
11. Maurer, M., Breskovic, I., Emeakaroha, V., Brandic, I.: Revealing the MAPE loop for the autonomic management of cloud infrastructures. In: 2011 IEEE Symposium on Computers and Communications (ISCC), pp. 147–152 (2011)
12. Solomon, B., Ionescu, D., Litoiu, M., Iszlai, G.: Designing autonomic management systems for cloud computing. In: 2010 International Joint Conference on Computational Cybernetics and Technical Informatics (ICCC-CONTI), pp. 631–636 (2010)
13. Open Science Grid: https://www.opensciencegrid.org/bin/view (online). Accessed Jul 2012
14. TeraGrid Archives: https://www.xsede.org/tg-archives (online). Accessed Jul 2012
15. Gagliardi, F., Jones, B., Grey, F., Bgin, M.E., Heikkurinen, M.: Building an infrastructure for scientific Grid computing: status and goals of the egee project. Philos. Trans. R. Soc. A: Math. Phys. Eng. Sci. **363**(1833), 1729 (2005)
16. XSEDE – Home: https://www.xsede.org/ (online). Accessed Jul 2012
17. EGI-InSPIRE: http://www.egi.eu/about/egi-inspire/ (Online). Accessed Jun 2012
18. Dongarra, J.J., Gentzsch, W. (eds.): Computer Benchmarks. Elsevier Science Publishers B. V., Amsterdam, The Netherlands, The Netherlands (1993)
19. Dikaiakos, M.D.: Grid benchmarking: vision, challenges, and current status: research articles. Concurr. Comput.-Pract. Exp. **19**(1), 89 (2007)
20. Frumkin, M., der Wijngaart, R.F.V.: NAS Grid benchmarks: a tool for Grid space exploration. Cluster Comput. **5**(3), 247 (2002)
21. Tsouloupas, G., Dikaiakos, M.D.: Gridbench: a tool for the interactive performance exploration of Grid infrastructures. J. Parallel Distrib. Comput. **67**(9), 1029 (2007)
22. Ogura, D.R., Midorikawa, E.T.: Characterization of scientific and transactional applications under multi-core architectures on cloud computing environment. In: Proceedings of the 2010 13th IEEE International Conference on Computational Science and Engineering. CSE '10, pp. 314–320. IEEE Computer Society, Washington, DC, USA (2010)
23. Bailey, D.H., Barszcz, E., Barton, J.T., Browning, D.S., Carter, R.L., Dagum, L., Fatoohi, R.A., Frederickson, P.O., Lasinski, T.A., Schreiber, R.S., Simon, H.D., Venkatakrishnan, V., Weeratunga, S.K.: The nas parallel benchmarks - summary and preliminary results. In: Proceedings of the 1991 ACM/IEEE Conference on Supercomputing. Supercomputing '91, pp. 158–165. ACM, New York, NY, USA (1991)
24. TPC: TPC-H http://www.tpc.org/tpch/ (Online). Accessed Jul 2012
25. van der Aalst, W.M.P., Bratosin, C., Sidorova, N., Trcka, N.: A reference model for Grid architectures and its validation. Concurr. Comput.-Pract. Exp. **22**(11), 1365 (2010)
26. Chan, W.K., Mei, L., Zhang, Z.: Modeling and testing of cloud applications. In Kirchberg, M., Hung, P.C.K., Carminati, B., Chi, C.H., Kanagasabai, R., Valle, E.D., Lan, K.C., Chen L.J. (eds.) APSCC, pp. 111–118. IEEE (2009)
27. Östberg, P.O., Elmroth, E.: Increasing flexibility and abstracting complexity in service-based Grid and cloud software. In: Proceedings of CLOSER 2011 - International Conference on Cloud Computing and Services Science, pp. 240–249. SciTePress (2011)
28. Stockinger, H.: Defining the Grid: a snapshot on the current view. J. Supercomput. **42**(1), 3 (2007)
29. Twenty-One Experts Define Cloud Computing: http://cloudcomputing.sys-con.com/node/612375/print (online). Accessed Jul 2012
30. Vaquero, L.M., Rodero-Merino, L., Caceres, J., Lindner, M.: A break in the clouds: towards a cloud definition. Comput. Commun. Rev. **39**(1), 50 (2009)
31. The Globus Alliance: http://www.globus.org (online). Accessed Jul 2012
32. gLite: Lightweight Middleware for Grid Computing. http://glite.cern.ch/ (online). Accessed Sept 2011
33. Seti@home: The Search for ExtraTerrestrial Inteligence. http://setiathome.ssl.berkeley.edu (online). Accessed Jul 2012
34. Jégou, Y., Lantéri, S., Leduc, J., Noredine, M., Mornet, G., Namyst, R., Primet, P., Quetier, B., Richard, O., Talbi, E.G., Iréa, T.: Grid'5000: a large scale and highly reconfigurable experimental Grid testbed. Int. J. High Perform. Comput. Appl. **20**(4), 481 (2006)
35. Nurmi, D., Wolski, R., Grzegorczyk, C., Obertelli, G., Soman, S., Youseff, L., Zagorodnov, D.: The eucalyptus open-source cloud-computing system. In: Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2009), pp. 124–131. IEEE Computer Society (2009)
36. Varia, J.: Architecting for the Cloud: Best Practices. Amazon White Paper (2010)
37. Avizienis, A., Laprie, J.C., Randell, B., Landwehr, C.: Basic concepts and taxonomy of dependable and

secure computing. IEEE Trans. Dependable Secure Compu. **1**(1), 11 (2004)
38. Google App Engine - Google Code: http://code.google.com/appengine/ (online). Accessed Jul 2012
39. Hopcroft, J.E., Motwani, R., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation, 2nd edn. Addison Wesley (2000)
40. Carroll, J., Long, D.: Theory of Finite Automata with an Introduction to Formal Languages. Prentice-Hall, Inc., Upper Saddle River, NJ, USA (1989)
41. Sotomayor, B., Montero, R.S., Llorente, I.M., Foster, I.: Virtual infrastructure management in private and hybrid clouds. IEEE Iternet Comput. **13**, 14 (2009)
42. Keahey, K., Figueiredo, R., Fortes, J., Freeman, T., Tsugawa, M.: Science clouds: early experiences in cloud computing for scientific applications. In: Proceedings of the 2008 Cloud Computing and Its Applications 2008 (CCA-08) (2008)
43. Rodero-Merino, L., Vaquero, L.M., Gil, V., Galn, F., Fontn, J., Montero, R.S., Llorente, I.M.: From infrastructure delivery to service management in clouds. Future Gener. Comput. Syst. **26**(8), 1226 (2010)
44. Rochwerger, B., Breitgand, D., Levy, E., Galis, A., Nagin, K., Llorente, I.M., Montero, R., Wolfsthal, Y., Elmroth, E., Cáceres, J., Ben-Yehuda, M., Emmerich, W., Galán, F.: The RESERVOIR model and architecture for open federated cloud computing. IBM J. Res. Develop. **53**, 535 (2009)
45. Montes, J., Sánchez, A., Valdés, J.J., Pérez, M.S., Herrero, P.: Finding order in chaos: a behavior model of the whole Grid. Concurr. Comput.-Pract. Exp. **22**, 1386 (2010)
46. Valdés, J.J.: Similarity-based heterogeneous neurons in the context of general observational models. Neural Netw. World **12**, 499 (2002)
47. Valdés, J.J.: Virtual reality representation of information systems and decision rules. Lect. Notes Artif. Intell. **2639**, 615 (2003)
48. Montes, J., Sánchez, A., Pérez, M.S.: Grid global behavior prediction. In: Proceedings of the 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid 2011), pp. 124–133. IEEE Computer Society (2011)
49. Montes, J., Nicolae, B., Antoniu, G., Sánchez, A., Pérez, M.S.: Using global behavior modeling to improve qos in cloud data storage services. In: 2nd IEEE International Conference on Cloud Computing Technology and Science (CloudCom 2010), pp. 304–311. IEEE Computer Society (2010)
50. Nicolae, B., Antoniu, G., Bougé, L., Moise, D., Carpen-Amarie, A.: Blobseer: next-generation data management for large scale infrastructures. J. Parallel Distrib. Comput. **71**, 169 (2011)