GMonE: a Complete Approach to Cloud Monitoring

Jesús Montes[a,*], Alberto Sánchez[b], Bunjamin Memishi[c], María S. Pérez[c], Gabriel Antoniu[d]

[a] *CeSViMa, Universidad Politécnica de Madrid, Parque Tecnológico UPM, Pozuelo de Alarcón, Madrid, Spain*
[b] *E.T.S. de Ingeniería Informática, Universidad Rey Juan Carlos, Campus de Móstoles, Móstoles, Madrid, Spain*
[c] *Facultad de Informática, Universidad Politécnica de Madrid, Campus de Montegancedo, Boadilla del Monte, Madrid, Spain*
[d] *INRIA Rennes Bretagne Atlantique Rennes, France*

Abstract

The inherent complexity of modern cloud infrastructures has created the need for innovative monitoring approaches, as state-of-the-art solutions used for other large-scale environments do not address specific cloud features. Although cloud monitoring is nowadays an active research field, a comprehensive study covering all its aspects has not been presented yet. This paper provides a deep insight into cloud monitoring. It proposes a unified cloud monitoring taxonomy, based on which it defines a layered cloud monitoring architecture. To illustrate it, we have implemented GMonE, a general-purpose cloud monitoring tool which covers all aspects of cloud monitoring by specifically addressing the needs of modern cloud infrastructures. Furthermore, we have evaluated the performance, scalability and overhead of GMonE with Yahoo Cloud Serving Benchmark (YCSB), by using the OpenNebula cloud middleware on the Grid'5000 experimental testbed. The results of this evaluation demonstrate the benefits of our approach, surpassing the monitoring performance and capabilities of cloud monitoring alternatives such as those present in state-of-the-art systems such as Amazon EC2 and OpenNebula.

Keywords: Cloud computing, monitoring

*Corresponding author
Email addresses: jmontes@cesvima.upm.es (Jesús Montes), alberto.sanchez@urjc.es (Alberto Sánchez), bmemishi@fi.upm.es (Bunjamin Memishi), mperez@fi.upm.es (María S. Pérez), gabriel.antoniu@inria.fr (Gabriel Antoniu)

1. Introduction

Clouds are inherently complex, due to the large number of resources involved and the need to fulfill the SLAs (Service Level Agreements)[1] of different users among other reasons. One of the key features of clouds is elasticity, which enables the adaptation of resources to existing tasks at a given moment. Thus, allocating resources to tasks must be agile and dynamic, which makes this activity even more complex. Among the approaches addressing this complexity, autonomic computing is one of the best known. In fact, the combination of cloud and autonomic computing fits very well, because of the synergies between these two areas [6, 29].

One of the main stages of the autonomic computing process is monitoring. Indeed, the rest of stages of the MAPE (Monitoring, Analysis, Planning and Execution) loop [20] largely depends on this first step. According to the granularity and type of monitored information, the efficiency and scope obtained by the application of autonomic computing differ.

Compared to other large scale environments, clouds have some specific features such as the use of SLAs, elasticity or virtualization. The monitoring needs of the cloud user differ from those of the CSP (Cloud Service Provider). Moreover, monitoring virtual systems and monitoring more traditional physical systems usually have different requirements. Cloud monitoring strongly depends on the following aspects, among others:

- The cloud service model, that is, the kind of services to be provided, i.e, Infrastructure as a Service (*IaaS*), Platform as a Service (*PaaS*) or Software as a Service (*SaaS*).
- The intended use of the information produced, e.g. client side feedback, internal system management, service provisioning accounting, etc.
- The initial source of monitoring, e.g. physical resources, virtual machines, software applications, etc.

To address these cloud-specific features, in recent years different cloud monitoring tools have been presented. These tools either adapt traditional distributed monitoring techniques [44], extend existing cloud platforms [2, 35] or propose new alternatives [25, 32]. However, each of these cloud monitoring tools is focused on certain specific aspects of cloud operation, providing only a partial solution for the cloud monitoring problem. In consequence, covering all aspects of cloud monitoring would require a combination of several monitoring tools, often leading to undesired redundancy and system overhead.

The main contribution of this paper is presenting a general-purpose cloud monitoring framework, covering all different cloud monitoring scenarios. This is an important

[1] A SLA is the part of a service contract where the specific characteristics of the service being provided are formally defined. In Cloud systems, a SLA determines the service requirements, i.e the service level, that have to be guaranteed by the cloud service provider in order to fulfill a client's cloud service provisioning contract.

step forward in cloud monitoring, setting the basis for an optimal cloud monitoring solution. To fulfill this objective we have performed the following steps:

1. We have designed a comprehensive cloud monitoring taxonomy. This allows us to determine what cloud monitoring scenarios exist, and what are their characteristics. The development of this taxonomy is based on the study of previous cloud monitoring works, trying to combine all existing ideas in a unified proposal.

2. According to this taxonomy, we have defined a general-purpose, complete cloud monitoring architecture. This serves as the architectural basis for the development of advanced cloud monitoring tools, capable of addressing the needs of modern cloud environments.

3. Finally, based on the defined architecture, we have developed a general-purpose cloud monitoring tool called GMonE (Global Monitoring systEm), applicable to all areas of cloud monitoring. The experimental validation, using Grid'5000 experimental testbed [1], proves the benefits of GMonE in a large-scale cloud environment, including high performance, low overhead, scalability and elasticity. Moreover, these results show that GMonE performs significantly better than commonly used alternatives, such as Amazon EC2 [3] and OpenNebula [36] monitoring tools [2, 35], in terms of monitoring flexibility and time resolution.

The subsequent sections of this paper are organized as follows: Section 2 discusses related work. Section 3 represents our cloud monitoring taxonomy, discussing possible types of cloud monitoring and their requirements. Section 4 introduces a layered cloud monitoring architecture that fulfills the requirements detected in the previous section. Section 5 describes the implementation of this architecture, called GMonE. Section 6 presents the evaluation of our proposal. Finally, Section 7 summarizes our conclusions and describes future work.

2. Related Work

Several studies have attempted to analyze and define the basic concepts related to large scale distributed systems monitoring in general, and cloud monitoring in particular. Spring [47, 48] proposes a top-down analysis of cloud monitoring: it distinguishes seven cloud layers and identifies their respective monitoring requirements. However, the analysis is presented mainly from the point of view of the CSP, without considering the specific needs of cloud clients. González, Muñoz and Maña [17] propose a multi-layer monitoring architecture for clouds based on a similar analysis. Their study is more focused on the virtualization aspects of common cloud systems and less concerned with physical and low-level software resources and client requirements.

One of the most important uses of system monitoring in cloud environments is SLA supervision. In most commercial cloud infrastructures service level agreements determine the relationship between client and CSP, and system behavior has to be continuously monitored in order to assure these agreements are fulfilled. Several initiatives have tried to define the mechanisms by which these agreements are established, such as WS-Agreement [5] and WSLA [26]. The SLA@SOI project [46] has taken this a

step further, trying to research, engineer and demonstrate technologies that can embed SLA-aware infrastructures into the service economy. When establishing and guaranteeing SLAs, monitoring tools play the key role of translating system-specific, often low-level behavior metrics (CPU load, network traffic, storage usage and so on) into SLA-related terms, i.e. information that can be understood in the same terms the SLA is defined. Trying to bride this gap, Emeakaroha et al. [12] present a framework for managing the mappings of the Low-level resource Metrics to High-level SLAs (LoM2HiS framework). Also addressing the issue of SLA-oriented monitoring, in [11, 13] an application-side cloud monitoring architecture is defined, designed to detect SLA violations.

From a technological point of view, monitoring information in distributed systems has successfully been addressed through different approaches and tools. For instance, Ganglia [28] is a widely used monitoring system for high-performance computing systems, such as heterogeneous clusters or grids, thanks to its robustness and successful deployment on various combinations of hardware architectures and operating systems. Ganglia is, however, not intended to be used for monitoring virtual resources, which is a major limitation in the cloud context. Nevertheless, it can be combined with other tools to this purpose. As an example, sFlow [44] provides an industry standard technology for monitoring large scale data centers, including cloud environments with virtualization features. sFlow has used Ganglia for monitoring both Java virtual machines and virtual machine pools. Although such a setting is possible, the main goal of sFlow is to support monitoring for high-speed switched networks.

Nagios [32] is an integral solution for monitoring an entire IT infrastructure. Although its goal is to provide monitoring information for large-scale systems, this approach does not deal with the dynamism of virtual environments. The same limitation is exhibited by other well known systems, such as MonALISA [33] or GridICE [4]. Both approaches provide good results on grid platforms, but do not address the difficulties raised by virtual resources. The TIMACS project [52] aims at reducing the complexity of the manual administration of computing systems by realizing a framework for management of very large computing systems, which includes efficient tools for scalable low level system monitoring. This project incorporates data aggregation and analysis, improving system behavior understanding, but it is focused on low level resources and therefore is not designed to address high-level, cloud-specific monitoring issues.

Some commercial tools have been widely used in large scale environments. Among the most famous tools are IBM Tivoli Monitoring [22] and HP OpenView [19]. These solutions are oriented to optimize the performance and availability of IT infrastructures, focusing again only on the physical resources.

Closer to our approach is Lattice [8], a monitoring framework for virtual networks. This framework uses data sources and probes to collect various monitoring data both for physical and virtual machines. In our approach we also provide such a feature by enabling the user to define plug-ins. However, we take a step further by providing a double vision including both the client's and the cloud provider's respective visions.

Commercial cloud solutions often make use of their own monitoring systems. However, in general these systems have limited functionality, providing only a fraction of the available information to cloud users. Examples of these monitoring systems are

Amazon CloudWatch [2] or OpenNebula Monitoring System [35]. A few other monitoring tools are described in *http://www.monitortools.com/cloud/*.

In contrast to these state-of-the-art approaches mostly focusing on specific issues in cloud monitoring, our main goal is to present a general-purpose cloud monitoring framework, covering all different cloud monitoring scenarios. After setting the theoretical basis of our problem analysis and proposal in Section 3, Section 6.1 (as a part of our system evaluation) presents a thorough feature study where our proposed system is compared to the above-mentioned cloud monitoring tools.

3. Cloud Monitoring

There are two main ways how system monitoring and monitoring information can be studied: *i)* what is being monitored (i.e. what part of the system) and *ii)* what is the monitoring information intended for. The former determines what the monitoring information obtained describes (e.g. system load, application usage, etc.) and the latter the way this information is provided (i.e. what specific parameters and how they are presented). Concerning cloud monitoring, we call the first one **monitoring level** and the second one **monitoring vision**. Our proposed cloud monitoring taxonomy combines both monitoring level and vision in a unified, generic model. The different aspects of this model are described in detail in the following Subsections.

3.1. Cloud monitoring level

Most cloud definitions include a series of system levels [14, 30, 53]. The exact number of levels varies from definition to definition (usually between 3 and 5), but all cloud models share the same basic characteristics. A typical cloud architecture would include the following levels:

- **Server**: These are 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.
- **Infrastructure**: Cloud infrastructure services or *Infrastructure as a Service* (*IaaS*) deliver computer infrastructure, typically a platform virtualization environment, as a service.
- **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.
- **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.

On the one hand, the lowest level (*server*) contains the machines, network links and other devices that the cloud is composed of. It contains the physical elements, and

it can be considered as the cloud's **physical system**. On the other hand, the remaining three levels (*infrastructure*, *platform* and *application*) contain the virtual resources being provided to the user, and they can be considered as the cloud's **virtual system**. When considering monitoring, this differentiation between the cloud's physical and virtual systems is crucial, since the monitoring techniques required in each case are radically different. Cloud physical systems are usually general purpose data centers made of clusters, and therefore they can be monitored using traditional distributed system techniques. Virtual systems are, however, a completely different environment. Since they are mostly based on virtualization and software abstraction, they require advanced software monitoring techniques, code instrumentation, etc. Therefore physical system monitoring is basically a hardware and low-level software (operating system) monitoring problem, and virtual system monitoring is a high-level software monitoring issue. Additionally, each virtual system level (*infrastructure*, *platform* and *application*) presents different characteristics, so monitoring techniques have to be adapted for each case.

3.2. Cloud monitoring vision

Cloud monitoring can provide information about aspects of system performance, behavior, evolution, etc. The way this information is understood, analyzed and used depends not only on what level of the system is being monitored (*server*, *infrastructure*, *platform* or *application*) but also who is obtaining this information and to what purpose. For instance, in a typical *IaaS* cloud such as Amazon EC2, clients can monitor the state of their virtual machine instances in order to know about system load, memory usage and performance. In the same *IaaS* infrastructure, the CSP would need to monitor all VM instances, continuously making sure SLA restrictions are satisfied. The CSP would also require monitoring information from the *server* level, in order to effectively control overall system load, VM allocation and migration, etc. Therefore, the point of view of the entity that obtains the monitoring information (client, management system, CSP, etc.) and its role in the system determine what kind of information has to be provided. Different entities require different monitoring data and have different visions of the cloud. From a general perspective, two main cloud monitoring visions can be distinguished:

- **Client-side monitoring vision**: From this point of view, the cloud is regarded as an abstract entity, capable of providing a specific set of computational services (the typical opaque *cloud* seen in most cloud computing illustrations). Monitoring information of this type provides an abstract description of the cloud service, expressed in the same terms as the service provisioning relationship is established between the client and the cloud (SLAs, contracts, etc.). This monitoring information helps the client to understand the characteristics of the services received and optimize their use.

- **Cloud-service-provider-side monitoring vision**: From this point of view, the cloud is regarded as a complex distributed infrastructure, with many hardware and software elements combined together to provide a specific set of services. Monitoring information of this type gives the CSP knowledge about the internal

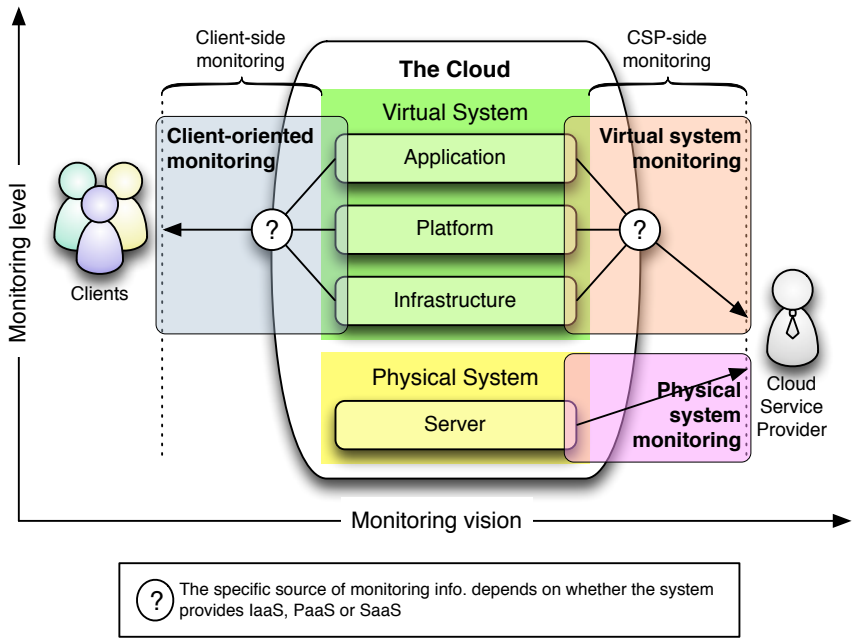


Figure 1: Cloud monitoring: level and vision.

functioning of the different cloud elements, its state, performance, etc. This information serves as an internal status control in order to guarantee SLAs and other service restrictions. It can be also used as behavior and performance log, to optimize system management and use of resources.

These two complementary visions address different cloud monitoring requirements, creating differentiated views of the system behavior and evolution. In order to clarify the difference between these two visions, we can consider the following scenario: we have a typical hybrid cloud, combining the use of a private cloud (for example an OpenNebula-based cloud) together with a public cloud (for example an Amazon EC2-based cloud). As cloud administrators, we need to monitor the entire infrastructure. Monitoring the private cloud is performed from a cloud-services-provider-side monitoring perspective, since we have total control of the private cloud. However, it is not possible to monitor the public cloud in the same way. We can only monitor it as clients, since we are just Amazon EC2 clients and we are limited to the use of the Amazon monitoring tool. This tool provides only client-oriented monitoring information, and therefore a client-side monitoring vision.

3.3. Combining cloud monitoring level and vision

As we have mentioned in previous sections, there are many aspects to consider when you monitor a cloud. Usually cloud monitoring tools are not aware of these dif-

ferent types of monitoring than can be performed on a cloud system. Indeed, one of our objectives is to provide an insight into cloud monitoring, defining a model combining both monitoring level and vision. Figure 1 shows our proposed cloud monitoring model, combining these two aspects in a generic cloud monitoring scenario and including all possible types and sources of monitoring information. In this model we have identified three basic types of cloud monitoring, depending on the specific cloud level and vision being considered: *client-oriented monitoring*, *virtual system monitoring* and *physical system monitoring*.

First, ***client-oriented monitoring*** refers to all the monitoring information provided to the cloud's users. This is directly linked to the client-side monitoring vision, since it is concerned with all aspects related to the client operation of the system. From a monitoring level perspective, it relates to the virtual system levels, since these are the virtual cloud resources being provided to the client, in the form of infrastructure (*IaaS*), platform (*PaaS*) or application (*SaaS*). The specific virtual system level concerned depends on the specific service being provided by the cloud, and this finally determines the specific nature of the monitoring information generated. Examples of *client-oriented monitoring* can be:

- In an *IaaS* cloud, clients can obtain information about the status (for instance CPU or memory) of their VM instances, consumed time and cost per instance, etc.
- In a *PaaS* cloud, clients can obtain information about the usage of platform resources available (for instance hosting space, network traffic or development tools), the impact of this usage in costs and billing, etc.
- In a *SaaS* cloud, clients can obtain information about the status and usage of the cloud applications and associated resources, services costs, etc. The nature of these parameters varies significantly depending on the actual software being provided as a service. Some examples are: in a virtual disk service (e.g. Dropbox [10]), clients can monitor storage capacity, file status, availability and synchronization. In a collaborative office suite (e.g. Google Docs [18]), clients can monitor file properties, authorship, access history, concurrent/sequential modifications and so on. In an on-line survey application (e.g. SurveyMonkey [51]), clients can monitor their survey evolution, temporary results, etc.

Second, ***virtual system monitoring*** refers to all monitoring information provided to the CSP related to the behavior, performance and evolution of the virtual system. In terms of monitoring vision, this is cloud-service-provider-side monitoring, since it deals with the internal information of the service being provided. From a cloud level point of view, it refers to the virtual system levels. *Virtual system monitoring* and *client-oriented monitoring* are strongly related, since they both refer to the behavior, status and evolution of the cloud services. The main difference between them is the monitoring vision, that is, what the monitoring information is intended for, determining what kind of information is generated in each case. Examples of *virtual system monitoring* can be:

- In an *IaaS* cloud, the system manager can monitor the status of every VM instance in the cloud and its internal resources.
- In a *PaaS* cloud, the system manager can monitor the use of platform resources, such as hosting space used, simultaneous network connections, etc.
- In a *SaaS* cloud, the system manager can monitor the application usage patterns, the resources sharing among applications, etc.

Regardless of the specific characteristics of a cloud, *virtual system monitoring* is a source of critical information for the CSP. *Virtual system monitoring* provides information in terms of the virtual system characteristics and cloud services, and therefore it is the key element to control Quality of Service (QoS) and guarantee SLAs. The CSP uses *virtual system monitoring* to determine the exact terms in which the cloud services are being provided and calculate service costs and billing. Therefore, having a comprehensive and efficient source of *virtual system monitoring* is one of the key requirements of a successful cloud business model.

Finally, ***physical system monitoring*** refers to all monitoring information provided to the CSP related to the behavior, performance and evolution of the physical system. In terms of monitoring vision, this is cloud-service-provider-side monitoring, since it deals with the internal information of the cloud resources. From a cloud level point of view, it refers to the server level. *Physical system monitoring* is the basis for cloud system management, as it is related to the physical computing infrastructure upon which the cloud itself is built. It is strongly related to *virtual system monitoring*, since the behavior of the virtual system has a direct impact on the physical resources and *vice versa*. However, the parameters monitored are different in both types of monitoring.

3.4. Monitoring metrics, SLAs and Quality of Service

As it has been explained, each of the three main types of cloud monitoring produces different kinds of monitoring information, intended for different uses and obtained from different hardware and/or software elements within the cloud. Regardless of the specific scenario and type of monitoring being performed, the cloud computing model is always strongly based on guaranteeing QoS (as specified in the SLAs), and therefore useful monitoring data will almost always be related to the terms in which the SLAs are specified. From a CSP-side vision, both physical and virtual systems have to be monitored in order to detect, or even anticipate, system behavior changes that could have an impact on QoS. From a client-side vision, meaningful monitoring information must be provided, in order to let the client know the QoS being received and its relationship with the established SLA. Therefore defining the appropriate monitoring metrics for each type of cloud monitoring is a crucial aspect. There are, however, several issues that need to be considered when defining these metrics.

In the case of *client-oriented monitoring*, metrics have to provide information in the same terms the SLA is specified, i.e. using the same parameters. For example, in a SaaS system that offers a cloud data storage, if the SLA indicates the maximum storage space available for one client the system must provide up-to-date information about the amount of space being used by each client, so that clients could monitor the provided QoS.

Virtual system monitoring metrics have to be defined in a similar way, but probably including additional internal information. Again, this is strongly dependent on the terms in which SLAs are defined, and therefore strongly varies from one cloud service to other. In the previous example about cloud data storage, the CSP needs also to know, for instance, the total space being used by all clients. This data is specific to virtual system monitoring because it shows the status of the entire virtual service (cloud data storage) regardless of the physical back-end. Additionally, it is concerned with information required only for internal management purposes, and therefore intended for the CSP only and not the clients.

The case of *physical system monitoring* is somehow different, since it is related to the low-level infrastructure that sustains the entire cloud. This means that most monitoring metrics required are those used in regular distributed systems, such as hardware metrics (CPU, memory, physical storage, network traffic, etc.), operating system metrics (system load, virtual memory, etc.) and so on. This information is essentially used for system management purposes. In this case, the connection with SLAs is rooted in the inevitable relationship between the low-level physical system behavior and the high-level virtual system behavior that it supports. Bridging the gap between these two levels is never a trivial task, but it is crucial to effectively managing the cloud infrastructure. A physical system problem, for instance, can be more quickly detected through *physical system monitoring*, but sometimes its specific impact in the virtual system behavior is not clear. Finding the appropriate way to translate this information from one level to another enables to improve performance, dependability, elasticity and QoS.

Let us stress again that this paper aims to introduce general taxonomy of cloud monitoring in order to provide a complete vision of the problem. Defining the appropriate monitoring metrics is an important aspect of this problem, because of its direct relationship with SLAs and QoS. While some common metrics should probably always be present in every cloud (e.g. low-level metrics related to the operating system, network, etc.), they need to be related to the virtual system behavior. This process strongly depends on the type of services being provided. Defining such specific system aspects is out of the scope of this work.

3.5. Security considerations

Clouds propose an abstract service layer between clients and CSPs. The cloud provider is responsible for the service it offers. Clients must establish trust relationships with the CSP and understand risk in terms of how the provider implements, deploys, and manages security on their behalf [21, 57]. The CSP must address the fundamentals of security and privacy, such as identity management, access control, data control, network access, protected communication, and so on, agreed with the client by means of the corresponding SLA as part of the service. Therefore, security and data privacy monitoring must be provided in the same way CSP offers any other services: our approach does not change this client-CSP interaction model. Of course, CSPs can additionally use virtual and physical monitoring data in a private way. Clients can only access monitoring data through the service interface in the way specified by the SLA. Furthermore, clients can implement their own monitoring infrastructure on top of the service as a part of the *client-oriented monitoring* to know the service behavior.

4. Cloud Monitoring Architecture

The section above has presented an in-depth analysis of the requirements and visions of cloud monitoring. This section presents a generic layered cloud monitoring architecture adapted to the needs of any cloud infrastructure based on the previously identified types of cloud monitoring. Two important needs related to cloud monitoring have to be covered, namely:

- Efficient access to information, considering response times and data size sent.
- Coordination of the monitored information from the whole cloud (client-oriented, virtual and physical system monitoring). Both clients and CSPs have to access monitoring data according to its different vision of the system.

To achieve these goals, the architecture shown in Figure 2 is proposed. The architecture is divided into two main components: the first one, regarding the monitoring access and the second one, regarding the data gathering and managing.

- *Monitor Access* provides an independent way to access the monitored information from each different element of the cloud, both services and servers. It is an abstraction layer that provides a unique view of both physical and virtual cloud elements, regardless of its different objective, composition and structure. This layer accesses both the virtual and physical system, monitoring their elements.
- *Data gathering* provides storage for the monitoring information obtained in the previous layer. It includes an historical archive of the evolution of the different cloud elements. Since each user has its own cloud vision and monitoring level, this layer does not only store monitoring data but it coordinates the user queries. This implies that every user has its own monitoring infrastructure providing a distinguished vision. This way, each user can access the monitoring level and vision required.

5. Cloud Monitoring Implementation: GMonE

GMonE is a cloud monitoring system that implements the proposed monitoring architecture. Because the system is designed to provide information from the whole cloud, its services are spread throughout the entire infrastructure. They have been designed to cooperate in order to obtain and manage the monitored data from the whole cloud.

Figure 3 illustrates the whole GMonE architecture. From a general perspective, GMonE is composed of four distinct elements:

- The monitoring element **GMonEMon**: This module is present in every cloud element that needs to be monitored. GMonEMon performs the functions of the *Monitor Access* layer. Its function is to obtain the monitoring measurements at regular time intervals and send them to one or more monitoring managers, hiding the complexity of the monitoring stage itself.

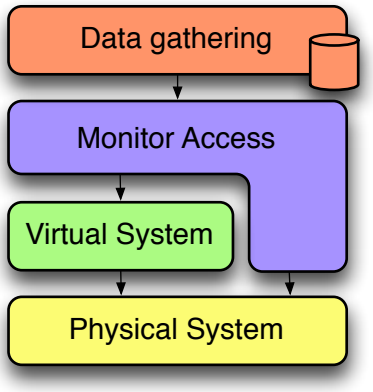


Figure 2: Proposed Cloud Monitoring Architecture.

- The **monitoring Plug-ins**: These are a set of specific software modules inside GMonEMon in charge of the actual monitoring task.
- The monitoring manager **GMonEDB**: This program acts as manager and monitoring archive of monitoring data covering the functions of the *Data Gathering* layer. It stores the monitoring information generated by GMonEMon in its own database. GMonEDB is responsible for providing the monitoring level and vision of each single user.
- The data provider **GMonEAccess**: This library provides monitoring data to the user in an easy way.

GMonE architecture is based on a typical publisher-subscriber paradigm. The GMonEMon instances act as publishers, sending monitoring information to the GMonEDB subscribers at regular time intervals. At any moment there are one or more GMonEDB instances running in the system. Each instance can be subscribed to a different set of GMonEMon instances, allowing different managers specialized in specific subsets of cloud elements. Each user can be focused on its own interests, specializing its own GMonEDB according to its own vision and monitoring level required. The monitoring information obtained from each element can also be extensively customized. The GMonEMon elements are plug-in-based, enabling to extend its monitoring capabilities with personalized plug-ins. All these features aim at providing a monitoring framework as much flexible as possible, which can be adapted to the specific needs of any modern cloud infrastructure. With this in mind, the entire GMonE suite has been developed in the Java programming language and it is distributed as a single *jar* file, in order to maximize its portability. Each of the GMonE modules and services shown here are described in detail in the following Subsections.

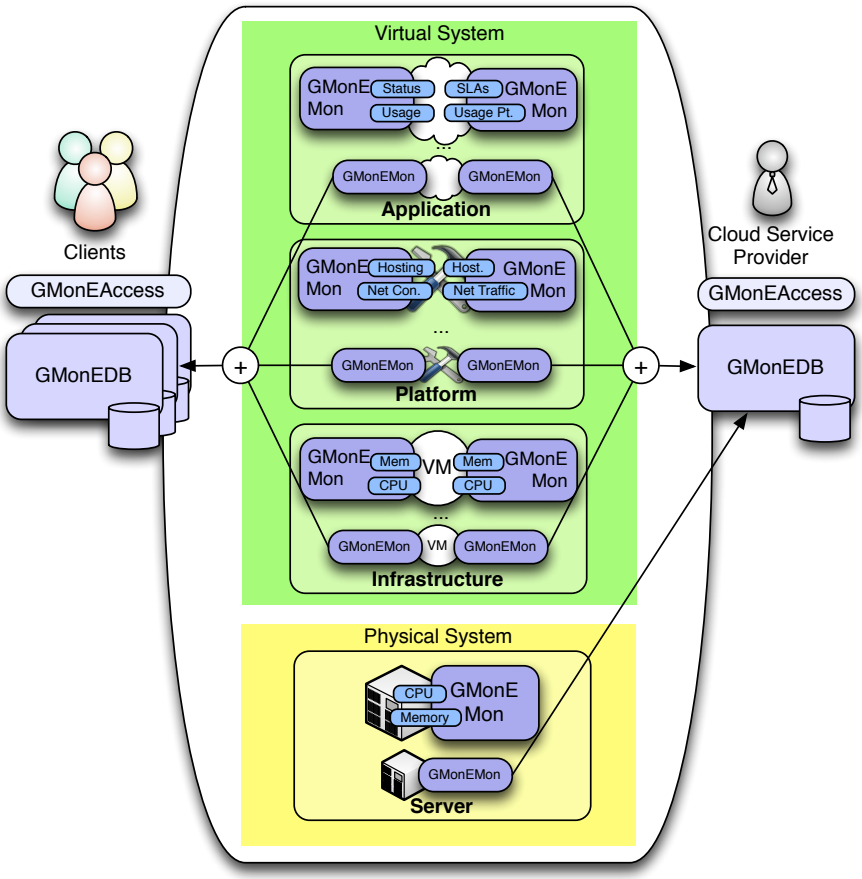


Figure 3: GMonE architecture.

5.1. GMonEMon

The monitoring system must be capable of being adapted to different kinds of resources, services and monitoring parameters. Thus, GMonE performs the monitoring of each element by means of *GMonEMon*, which abstracts the type of resource (virtual or physical). GMonEMon service monitors the required parameters and then it communicates automatically with the monitoring manager (GMonEDB) to send it the monitored data. This communication is done through a standard Java *Remote Method Invocation* (RMI) process.

As a summary, the main features of GMonEMon are:

- It provides a scalable and customizable structure, based on monitoring plug-ins.
- It publishes monitored data to the monitoring managers. This is performed at

regular time intervals. The time spent by GMonEMon in obtaining the monitoring measurements and sending them to the monitoring managers is called *publish time*. The publish time period can be configured by both clients and CSPs, each one according to their necessities.

Since GMonEMon has a modular structure, the set of monitored parameters is not statically defined, and new plug-ins can be developed in order to fit specific requirements. Service providers can use monitoring plug-ins to get information about the status of the VMs, simultaneous network connections, application usage patterns, etc. Clients can use them to obtain information about consumed time and other parameters of their own VM instances, hosting space, network traffic, service costs, etc. Its modular structure is the key to provide the flexibility and adaptation required in clouds. To satisfy this modularity, GMonEMon provides a simple Java interface that can be implemented by developers in order to monitor specific parameters. Several plug-ins can be simultaneously used, allowing the system to offer a fully customizable set of monitored parameters. Developing a custom GMonEMon monitoring plug-in requires just the creation of a Java class that implements the following two methods:

- `parameter_list : get_parameters()`: A simple method that returns the list of names of the parameters monitored by the plug-in.
- `value_list : get_values(parameter)`: A method that returns a list of monitored values (if any) for the specified parameter. Each element of this list is a *monitored value object* that includes also additional information (monitored value, measurement units, host name, parameter name and time stamp).

GMonEMon uses the first method (*get_parameters*) to identify the list of parameters provided by the plug-in and the second method (*get_values*) to read those parameters at regular time intervals. The way the actual monitoring is performed depends on the type of monitoring information being obtained, and it is carried out inside the plug-in. This simple interface guarantees that the complexity of developing new monitoring capabilities will always be related solely to the monitoring information that needs to be measured. The GMonE suite remains as flexible as possible, providing the ideal framework for developing custom made monitoring infrastructures.

Additionally, service providers can request monitoring information not only based on each specific user, server, service, etc. of the cloud system but also an aggregated information about its operation. For instance, it can be useful an aggregated information about the storage, workload, the compound use of the services, etc. of the whole system instead of the workload, storage, use of services of each single user. When monitoring compound systems, most monitoring tools provide a set of values as information related to each specific monitored parameter, e.g. the storage usage of 200 clients would be a 200-element-long list instead of a single value. GMonEMon reduces the data sent, aggregating monitoring data, regarding the needs of CSPs and clients. These values can be aggregated in different ways, depending on parameter meaning, cloud vision and monitoring level.

GMonEMon provides a mechanism that allows the system to specify the way this aggregation is performed, depending on parameter characteristics: an aggrega-

tion function can be specified for each monitored parameter, which can be changed and reset during execution time. The syntax of this function is the following:

- Basic arithmetic operators: `+`, `-`, `*`, `/`
- Real numerical constants: `1`, `0`, `26`, `-367.2`, etc.
- Special operators: they represent mathematical functions to apply to a set of values, like:
  - `S`: sum of all the values.
  - `P`: multiplication of all the values.
  - `M`: maximum value.
  - `m`: minimum value.
  - `n`: number of values.
  - `F`: aggregation is not required.

Using this syntax, common statistical descriptors can be easily defined (e.g. the arithmetical mean would simply be `S/n`). The aggregation function is invoked by the GMonEMon core after requesting he monitoring information from the plug-in (through the *get_values* method). By means of this customized aggregation, both cloud services and servers can be abstracted according to user vision and monitoring level, regardless of their internal characteristics.

5.2. GMonEDB

Once the monitored information has been obtained using the GMonEMon service, it is necessary to gather and manage it. Each GMonEDB service collects monitored information storing it in its own database. Each user, either client or CSP, has its own GMonEDB for storing the required information regarding its needs. The CSP maintains information about virtual and physical system whereas clients store only client-oriented data. GMonEDB monitoring database relies on MySQL [31] as default database management system. In addition, GMonEDB offers archiving flexibility, having the possibility to rely on other storage back-end technologies such as SQLite [49], RRD files [42] (through the rrd4j library [41]) or a key-value store (Apache Cassandra [24]). All these alternatives are meant to be a choice of usage depending on the particular monitoring scenario needs.

The main features of the GMonEDB service are:

- It provides monitored information from a set of GMonEMon monitoring elements regarding the user needs.
- It stores data from the different elements in its own database. This gives a fast access to monitored information; it also provides data about the cloud's past behavior. This makes it possible to observe the cloud operation evolution. The time spent by GMonEDB for storing the information in the database is called *archiving time*.

- It manages relevant data. GMonEDB can be configured to subscribe to, store and manage only the necessary information in each particular case. If at any time the needed data should change, the service can be easily reconfigured on-line to start obtaining the new required information.

5.3. GMonEAccess

The last part of the GMonE system is used to obtain monitored information from the GMonEDB service. This final module is called GmonEAccess and works as a programming library with the following features:

- It provides a common interface to access the GMonE system.
- It can be used to obtain monitored data, configure and manage the GMonE infrastructure on-line. Clients can manage their own virtual monitoring infrastructures without depending on service providers, whereas service providers can manage the whole infrastructure, both virtual and physical system. Each one can access their own monitoring data according to their necessities.

6. Evaluation of GMonE

We have performed a series of analytical and experimental studies, in order to validate our proposal. The first is a qualitative study, considering the different types of cloud monitoring and comparing GMonE features and capabilities with the most relevant state-of-the-art cloud monitoring alternatives. The rest are a set of experiments, performed to asses GMonE's performance, scalability, elasticity and the amount of overhead it introduces in the system. All these studies are described in detail in the following Subsections.

6.1. Cloud monitoring features

Nowadays there is a significant variety of state-of-the-art cloud monitoring tools, either as ongoing scientific research projects or as commercial applications. The most relevant are described in Section 2. As detailed in that section, some of them have been developed based on previously existing distributed systems monitoring tools; others have been created from scratch. In both cases the objective behind this development is to address the specific needs of modern cloud environments. An effective way to study all these alternatives, and compare them with our proposed tool GMonE, is to analyze their capabilities in terms of the three basic types of cloud monitoring we have identified in Section 3. Table 1 compares several monitoring approaches according to these different aspects, namely:

- If the monitoring tool provides a solution for the different cloud deployment models, i.e., SaaS, PaaS or IaaS.
- If the monitoring tool provides monitoring capabilities to clients and/or CSP.
- Focusing on the CSP, if the monitoring tool can provide information about the virtual and/or physical resources.

Systems/ Functionality	Client-oriented			Virtual System			Physical System
	SaaS	PaaS	IaaS	SaaS	PaaS	IaaS	
Ganglia [15]							✓
Ganglia+ sFlow [44]			✓				✓
Nagios [32]							✓
MonALISA [33]							✓
GridICE [4]							✓
Lattice [8]			✓				✓
TIMACS [52]							✓
IBM Tivoli Monitoring [22]							✓
HP OpenView [19]							✓
Amazon CloudWatch [2]			✓				
OpenNebula Monitoring System [35]			✓				
OPNET [37]	✓		✓				✓
GFI MAX Remote Management [16]							✓
Intermapper Cloud Monitor [23]			✓			✓	✓
Logic Monitor [25]				✓	✓	✓	✓
NMS [34]				✓	✓	✓	✓
PacketTrap Perspective [38]			✓			✓	✓
Site24x7 [45]	✓						
GMonE	✓	✓	✓	✓	✓	✓	✓

Table 1: Comparison of different cloud monitoring alternatives. **SaaS**: monitoring information about applications. **PaaS**: monitoring information about platform. **IaaS**: monitoring information about infrastructure.

Typical monitoring tools such as Ganglia [15], MonALISA [33] or Tivoli [22] are focused exclusively on the physical level, and are not capable of providing information about the virtual system. Modern commercial alternatives such as Logic Monitor [25] and NMS [34] extend this functionality, but only from a CSP perspective. User interfaces of most cloud infrastructures provide some client-side vision information, such as Amazon CloudWatch [2] and the OpenNebula monitoring system [35]. Recent research proposals such as Lattice [8] try to combine both visions, incorporating some aspects of both client-side and CSP-side monitoring. To the best of our knowledge, GMonE is so far the only monitoring alternative capable of providing monitoring information from all possible monitoring levels and visions. The key to this flexibility is the capability to adapt the cloud monitoring element (the GMonEMon publisher) and the topology of the cloud infrastructure itself (the relationships between different GMonEMon publishers and GMonEDB consumers) to the specific needs of each type of cloud monitoring. The possibility of extending GMonEMon functionality by means of easily developed plug-ins guarantees that GMonE can be successfully used in every possible cloud monitoring scenario.

Additionally, after analysing the most relevant architectures coming from academia and industry, we can conclude that the proposed cloud monitoring architecture in this paper is the most general-purpose, complete and representative at the same time. While the monitoring layer of the architecture (see Figure 2) is being adjusted to physical or virtual system based on the monitoring needs, the data gathering layer acts not only as a storage information provider coming from the monitoring layer; moreover it coordinates user queries while providing different vision to different user/provider requests. This flexibility enables GMonE to provide more useful and rich monitoring data, depending on the specific scenario. This flexibility is not found in other alternatives, as we have seen in Table 1.

A set of experiments have been performed to validate GMonE's features. These are described thoroughly in the following Subsections.

6.2. Experimental testbed

In order to validate GMonE, we need to perform its experimental testing in a cloud environment as much realistic as possible. We carried out our experiments on Grid'5000. We used 45 nodes from the *suno* cluster on Sophia site. These nodes acted as cloud physical resources, on top of which a series of cloud software layers were deployed, in order to recreate a complete IaaS cloud system. The complete deployment is depicted in Figure 4, and it presents the following elements:

- Physical resources: 45 cluster nodes outfitted with Debian GNU/Linux *Lenny* with kernel 2.6.32, x86_64 Intel Xeon E5520 2.26Ghz CPUs, 32 GB of RAM and 2 Gigabit Ethernet (Broadcom NetXtremeII BCM5716) network interfaces.

- Cloud Infrastructure-as-a-Service: OpenNebula [36] installation, providing IaaS capabilities in the form of VMs. OpenNebula is an ideal solution for a private cloud, given the high level of centralization and customization it provides for CSPs and end users. It incorporates a shared file system, which is NFS [39] by default. Using its default configuration with NFS, OpenNebula acts in a highly

centralized manner. We have chosen OpenNebula because of its flexibility and suitability for research works [43].

- Virtual resources: From an IaaS cloud client perspective, a total of 80 virtual machines were deployed, each of them outfitted with Debian GNU/Linux *Lenny* with kernel 2.6.32, a single virtual x86_64 CPU and 1GB of RAM. These VMs were interconnected using a standard OpenNebula Ethernet virtual network.

- Client-side deployment: As an example of use of IaaS resources, Cassandra [24] was deployed on the 80 VMs. Cassandra is a storage system developed as part of the Apache project, designed to handle very large amounts of data, spread out across multiple servers. The objective of Cassandra is to provide a highly available data storage/access service for modern cloud-like applications. It is a *key-value store* solution (not a traditional relational database) that it was initially developed by Facebook and powered their Inbox Search feature until late 2010. Facebook's Cassandra was based on the combination of well known and proven techniques. Tables are very important in Cassandra; a table represents a distributed multi-dimensional map indexed by a key. While raw operations are atomic, columns are grouped into sets called column families (Simple or Super). Super column families can be visualized as a column family within a column family; in principle, every column may be sorted either by name or time. Cassandra is an ideal example of a state-of-the-art application that can be deployed on modern cloud computing infrastructures.

- Benchmarking: To generate a realistic workload for this infrastructure and provide a complete cloud computing scenario, the Cassandra virtual deployment was subjected to the Yahoo! Cloud Serving Benchmark (YCSB) [9]. One of the main advantages of this benchmark is an extensible workload generator, the YCSB Client, which can be used to load datasets and execute workloads across a variety of data serving systems. All the core package workloads use the same dataset, so it is possible to load the database once and then run all the workloads. In the last version of YCSB (0.1.4) [56], there are six workload versions. Table 2 describes each one of these workloads in detail. Using these standard workloads allows us to test our cloud infrastructure against a variety of access patterns and client uses. This provides a comprehensive scenario where we can effectively test the capabilities of our flexible monitoring system, GMonE.

This experimental testbed is just an example of a cloud configuration, and therefore does not cover all possible scenarios. Our aim is to demonstrate GMonE's monitoring capabilities on all three major types of cloud monitoring. To that effect, we performed the following tasks:

- Physical system monitoring: GMonE was deployed on the Grid'5000 physical resources. Each node was monitored using the default GMonEMon Linux plug-in, which provides information about CPU usage, memory usage and system load. The monitoring information generated was gathered in a central GMonEDB node, called the *CSP monitoring manager*.

Workload	Description	Applications examples
A	Read/update ratio: 50/50	Session store recording recent actions.
B	Read/update ratio: 95/5	Photo tagging; add a tag is an update, but most operations are to read tags.
C	Read/update ratio: 100/0	User profile cache, where profiles are constructed elsewhere (e.g., Hadoop).
D	Read/update/insert ratio: 95/0/5 (read the latest)	User status updates; people want to read the latest.
E	Scan/insert ratio: 95/5	Threaded conversations, where each scan is for the posts in a given thread (assumed to be clustered by thread id).
F	Read/read-modify-write ratio: 50/50	User database, where user records are read and modified by the user or to record user activity.

Table 2: YCSB workloads. Each workload generates a total of 100,000 operations, accessing a total of 1000 Cassandra records. These records can be different in each execution.

- Virtual system monitoring: An additional instance of GMonEMon was deployed on the OpenNebula front-end node, to gather information about the OpenNebula evolution from a CSP-side vision. To this particular purpose a custom GMonEMon monitoring plug-in was developed, called *OpenNebulaPlugin*. This custom plug-in takes advantage of the OpenNebula XMLRPC interface to provide monitoring information about the total number of VMs in the system, physical node CPU and memory usage of each VM, transferred/received network traffic of each VM and number of VM disk images registered in the system. The plug-in consists of a single Java 1.6 class, implemented in a single file with less than 250 lines of code. This is a perfect example of how easy it is to extend and customize the GMonEMon capabilities. The monitoring information obtained using this GMonEMon instance was published to the *CSP monitoring manager*.

- Client-oriented monitoring: All 80 VMs created for this testbed were also monitored using GMonE. This provides the cloud users with live, detailed information about the state of their resources. Each VM included an instance of GMonMon, monitoring its CPU and memory usage by means of the GMonEMon basic Linux plug-in. Additionally, a custom GMonEMon plug-in was developed, in order to obtain more detailed information about Cassandra. This custom *CassandraPlugin* measured the amount of disk space used by Cassandra in each virtual node. As in the previous case, the entire plug-in is a single Java 1.6 class implemented with less than 80 lines of code.

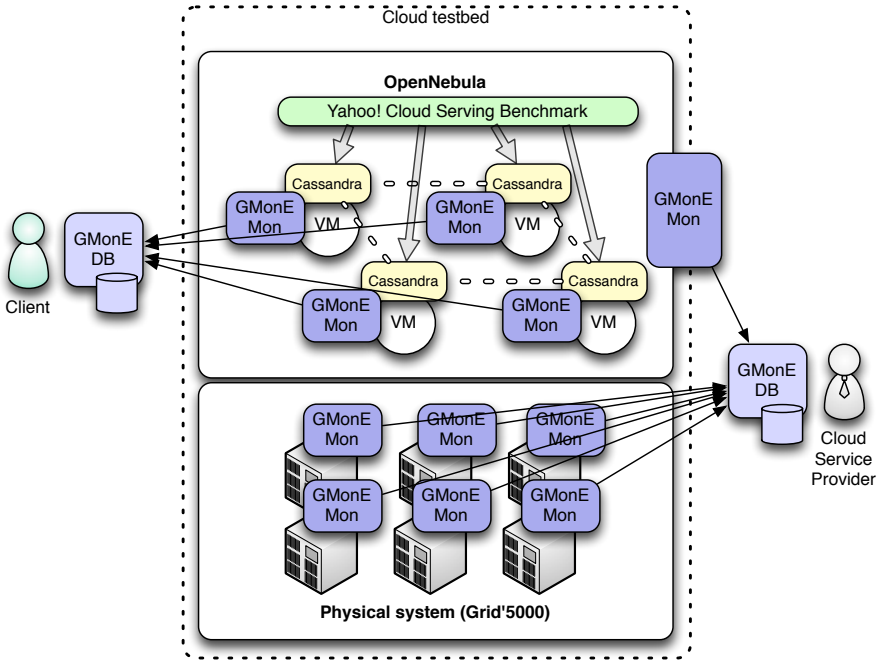


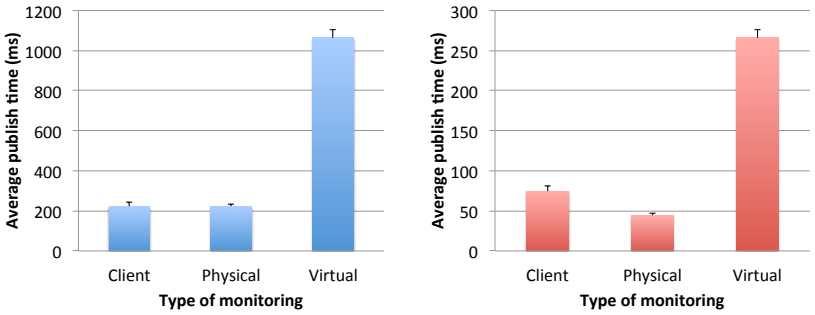
Figure 4: Experimental configuration.

Figure 4 depicts the entire experimental testbed, including all main hardware and software layers and monitoring elements.

6.3. Monitoring performance

Using the cloud testbed described in Subsection 6.2, we performed a series of experiments to demonstrate GMonE's capabilities. The first of these tests was designed to measure GMonE's performance, that is, GMonE's capability of efficiently providing monitoring information in real time. GMonEMon and GmonEDB elements were deployed as previously explained, and the 6 YCSB workloads were executed in a sequenced, looped, round-robin fashion. This guaranteed a continuous and diverse workload that occupied the system while the monitoring was being performed. During the experiment the publish time of all GMonEMon elements was measured. All GMonEMon elements were configured to publish their information every 5 seconds. Average publish time results are shown in Figure 5. Results show high performance values, especially in the cases of client-oriented monitoring and physical system monitoring.

Client-oriented monitoring presented an average value of 223ms. Given that a total of 3 monitoring parameters were published (CPU usage, memory usage and Cassandra disk space), this leaves us with an average of 74.3ms per parameter. It is important to remember that this value includes both measurement and publishing (sending the value



(a) Aggregated publish time

(b) Publish time per parameter

Figure 5: Monitoring performance. Column height indicates the average publish time observed (in ms). Error bars show the standard deviation for each case. The *client-oriented monitoring* columns show the results obtained by the VMs GMonEMon elements. The *Physical monitoring* columns show the results obtained by the Grid'5000 nodes GMonEMon elements. The *virtual monitoring* columns show the results obtained by the OpenNebula front-end GMonEMon instance.

to the GMonEDB element). This result shows that GMonE is capable of efficiently monitoring and propagating results.

Physical system monitoring presented almost identically good results, showing that GMonE is an ideal solution for this type of monitoring as well. In this case the average value observed was of 222ms, and the GMonEMon elements published 5 parameters (CPU usage, memory usage, 1min system load, 5min system load and 15min system load) for an average of 44.4ms per parameter.

Finally, virtual system monitoring presented an average publish time of 1062ms. The GMonEMon element monitored 4 parameters (total number of VMs, CPU usage per VM, memory usage per VM, number of disk images), for an average value of 265.5ms per parameter. The apparent difference with the other two types of monitoring can be easily explained. In this case the GMonEMon plug-in has to access OpenNebula through the XMLRPC interface, request the data and process its response. Additionally, the monitoring parameters provided present more complex data, such as lists of values (CPU usage for each VM, for example). In addition to data gathering and publishing, GMonEMon performs data aggregation of these structures. This additional operation requires execution time, although it reduces the amount of information transferred through the network and the data archived by GMonEDB. Nevertheless, all required monitoring data is produced in approximately 1s.

This series of experiments provide insight on GMonE's publish time and performance. However, assessing GMonE's final capabilities requires performing additional studies concerning archiving times, scalability and elasticity. These studies are presented in the next Subsection. Afterwards, Subsection 6.5 combines all these results to evaluate GMonE in comparison with other cloud monitoring alternatives.

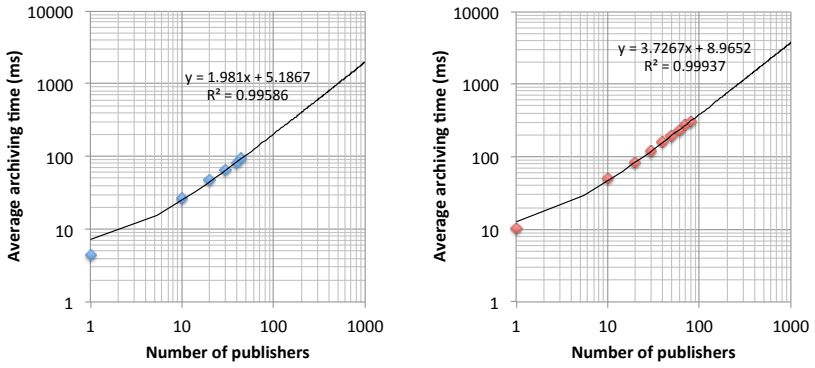
6.4. Monitoring scalability and elasticity

Cloud systems are inherently dynamic. Physical resources can grow, virtual resources can change depending on client use and SLAs, etc. As a general purpose cloud monitoring tool, GMonE has to be able to adapt to these changes, both in terms of scalability and elasticity. In order to asses these desirable features, a second type of experiments was performed in the same cloud testbed. In this case, the objective was to study how GMonE can react to a changing number of physical and virtual resources. As we have seen in the previous experiment, on the one hand GMonE monitoring elements (GMonEMon) present a high performance, and since they care about single resources (virtual or physical) their performance should not change when the number of resources grows. The monitoring managers (GMonEDB), on the other hand, gather information from all monitoring elements, so they are sensitive to changes in the physical and virtual systems. To demonstrate GMonEDB scalability and elasticity, we exposed both the client-oriented monitoring manager and the CSP monitoring manager to an increasingly large number of monitoring elements publishing information. During this process, we measured the time it took for these monitoring managers to process the received data and store it in their respective databases. Figure 6 shows the average data archiving times observed and its standard deviations. It also shows a linear regression fit to the data presented, and the accuracy of this regression using the R^2 coefficient[1].

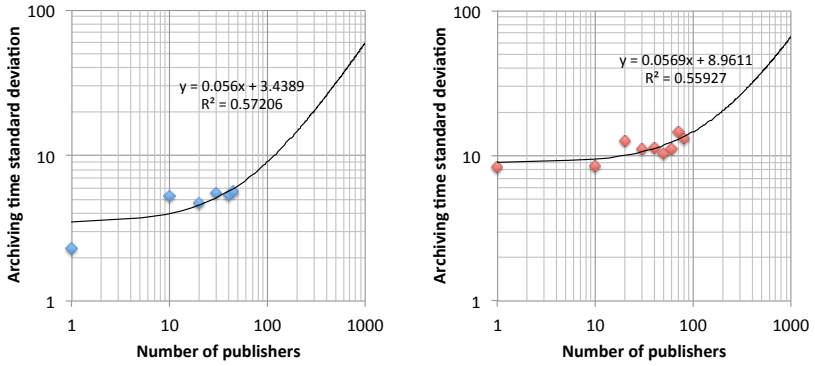
The results show very fast archiving time and a very desirable linear growth in the narrow range of points we explored. In this experiment each GMonEMon monitoring element published its information every 5s, and each GMonEDB element processed the information received every 30s. This implies that, at each data archiving event (every 30s) each GMonEDB element had to process a total of 4 monitoring samples from each GMonEMon. In this context (data archiving of physical resources), monitoring took approximately 1.98ms per node regardless of the number of nodes. Given that each node published its monitoring information 4 times between data archiving events, this gives us a total of approximately 0.5ms per monitoring sample. In the case of VM monitoring, data archiving took approximately 3.73ms per VM, again regardless of the number of VMs. As in the physical system case, each GMonEMon published its information 4 times between data archiving events, given a total of 0.93ms of data archiving time per monitoring sample.

The observed linear growth is very clear and guarantees a very desirable feature: the monitoring data archiving time depends linearly on the number of monitoring elements, therefore it can be estimated before deploying the infrastructure. To further validate this conclusion, we have to consider that the archiving process consists only on a basic information processing stage (performed in constant time per monitoring sample) and database insertion. Since GMonEDB uses MySQL as its default monitoring information storage back-end, this database insertion can be performed in linear time [40], given the number of monitoring samples, and hence the linear growth observed in Figures 6a and 6b. To safely extrapolate the results obtained for very large numbers

[1]The coefficient of determination (R^2) is a measurement of the degree of adjustment of a curve to a data series. Its values range from 0 to 1, where 0 indicates no apparent fit to the data and 1 indicates a perfect fit.



(a) Physical system average archiving time (b) Client-oriented average archiving time



(c) Physical system archiving time standard dev. (d) Client-oriented archiving time standard dev.

Figure 6: Scalability and elasticity study. Dots show the average/standard deviation archiving time observed. The black line shows the approximated linear regression curve, obtained from the shown data. All axes are shown in log_{10} scale.

of monitored physical or virtual resources, we have also to consider the data dispersion observed, that is, the archiving time standard deviation shown in Figures 6c and 6d. This standard deviation can give us an estimation of how much the actual archiving times will deviate from the average value. As it can be seen in the figures, in both cases the standard deviation shows an apparent linear growth, indicating that archiving time dispersion increases linearly with the number of publishers. Assuming that for large numbers of publishers, the archiving time is normally distributed, then more than 95% of the observed values would be in the $[average - 2 * stdev, average + 2 * stdev]$ interval. Using the linear regression models shown in Figure 6, we can extrapolate the average and standard deviation archiving time values for numbers of publishers of increasing orders of magnitude. Table 3 shows several examples of this extrapolation.

This statistical analysis and extrapolation allow the client or CSP to calculate the

Number of publishers	Physical system monitoring archiving time		Client-oriented monitoring archiving time	
	Average (s)	Stdev	Average (s)	Stdev
10	0.025	0.004	0.046	0.010
100	0.203	0.009	0.382	0.017
1000	1.986	0.059	3.737	0.066
10000	19.815	0.563	37.276	0.578
100000	198.105	5.603	372.679	5.699

Table 3: Extrapolated average and standard deviation archiving time for physical system and client oriented monitoring.

minimum database update period GMonE can be configured with, given the number of monitoring elements and its publish period. For instance, in the extreme case of 100000 physical resources being monitored, the average monitoring data archiving time can be estimated to 198.105s, that is 3.3min. If we incorporate the standard deviation model created, assuming that the archiving time data is normally distributed, we can estimate that more than 95% of archiving times will be in the $[3.1, 3.5]$min interval. Considering this is an extreme scenario (100K monitoring elements publishing to one single manager), these results show the very desirable scalable nature of the GMonE monitoring system.

This extrapolation is, however, based on the assumption that the linear regression model constructed can realistically predict GMonE's behavior for very large scenarios. The question is: Can a model generated with data from less than 100 publishers be still valid for 1000 o 10000 publishers? This linearity assumption is motivated, as it has been explained, by the fact that the GMonDB archiving procedure is mainly a database insertion operation. As the number of publishers grows, the size of this insertion operation will grow accordingly. Previous work shows that, for a general purpose DBMS like MySQL in such a scenario, linear growth can be expected for the time of this insertion operation [40]. In addition, linear regression models have already been successfully used in other scalability models in data center and large scale distributed applications [54]. Nevertheless, every system has its own specific characteristics, and GMonE introduces a small layer of software on top of the DBMS, which must be properly analyzed for large scenarios before fully accepting the linearity assumption. In order to perform this analysis, we have carried out another experiment. As it has been explained, as the number of publishers grows GMonEDB archiving process has to deal with larger amounts of monitoring data. We have deployed a GMonEDB element in our physical system experimental testbed, and subjected it to an increasingly large amount of published monitoring information. We have generated this information synthetically, in order to achieve the expected data volume for a system composed of a number of publishers ranging from few tens to several thousands. For the case of 1000 publishers, for instance, the total amount of monitoring information would be (following the same model used in previous experiments): 1000 publishers $\times$ 4 records/(publisher $\times$ update) $\times$ 4 updates/round = 16000 records/round. We have measured the observed GMonEDB archiving time and compared it with the expected time predicted by our

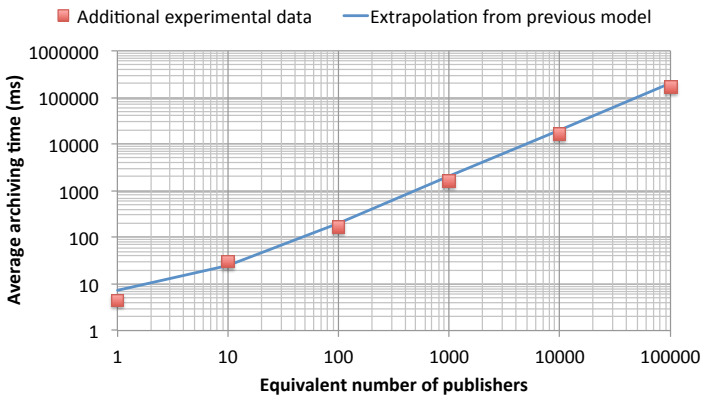


Figure 7: Study of the feasibility of the linear regression model. The expected values predicted by the regression model are compared with experimental data obtained from a real GMonEDB instance being subjected to increasingly large amounts of monitoring information. Both axes are shown in log_{10} scale. The extrapolation curve from the previous model (Figure 6a) fits the new experimental data with $R^2 = 0.961$.

linear regression model. Results can be seen in Figure 7. These results clearly show that our linearity assumption is valid, and the regression model previously generated is a very accurate prediction mechanism to extrapolate GMonE's behavior for very large scenarios.

As it can be seen, in extremely large system the scalability of the underlying storage back-end plays a decisive role. As time passes, the size of the stored monitoring information can grow extremely quickly, becoming a potential cause of bottlenecks and other performance problems. Even though MySQL scales linearly in our experiments, other studies show that its performance can drop, depending on the underlying infrastructure characteristics [7]. Very large storage scenarios often require advanced solutions, in order to achieve optimal scalability. As described in Section 5.2, GMonEDB uses MySQL as default storage back-end, but can be configured to use other efficient alternatives, such as RRD Tool or Cassandra. In terms of storage back-end, the ideal GMonE configuration will depend on the characteristics of the system where it is deployed. MySQL produces excellent results in our experiments, and we believe is best suited for system sizes up to several hundred nodes. Selecting this alternative, however, requires the presence of a MySQL database management system (DBMS) installation and its proper configuration. This alternative is therefore specially adequate for CSP-side monitoring in high performance scenarios where detailed monitoring information is required (small monitoring period). The RRD configuration offers a good alternative to MySQL when a lightweight, system-independent storage back-end is required (no other DBMS or services present), such as small systems and some types of client-side monitoring. Although the RRD files do not facilitate a SQL interface, data is still very easily accessible, especially for real-time queries and graphic representation (the typical RRD Tool plots). The extremely light nature of this alternative makes it also

ideal in very complex and performance-demanding scenarios, such as very large systems (thousands of nodes or more). The main RRD drawback in this latter case, in comparison with the other storage alternatives, is that its circular buffer nature reduces its capabilities to store long-term historical data. Finally, Cassandra's key-value store approach offers a completely different alternative. This type of storage solutions have been developed, among other reasons, to address the above mentioned limitations of relational databases, such as MySQL, in certain scenarios. Cassandra is specifically designed to efficiently operate while handling extremely large amounts of data. In terms of GMonE, this means storing long-term historical information with the best possible detail. It is therefore, best suited for very large systems (thousands of nodes or more) where high monitoring resolution is necessary, as well as a comprehensive monitoring archive.

In any distributed infrastructure the scalability of the entire system is also generally dependent on the scalability of the network; this means that the network can become a source of bottlenecks, again specially in very large systems. Monitoring tools introduce a slight amount of network load and, as in the case of the storage back-end, they are limited by the hardware and software elements underneath them. These external limitations are not inherent to the monitoring system, but they have to be taken into account when configuring and deploying it, in order to avoid performance and scalability issues. Additionally, modern large scale distributed systems are built using complex techniques such as advanced network topologies and hierarchical organizations. In such kind of environments, the placement of the monitoring and the data storing elements definitely affects its performance and scalability. In large and very large systems specialized GMonEMon elements can be deployed in each system component, using its flexible plug-in design to gather only the required information in each case. This means that dedicated nodes (file servers, computing nodes, external front-ends, etc.) can have specific information being monitored and published at optimized time periods. Analogously, multiple GMonEDB elements can be organized in different ways, allowing hierarchical organization, dedicated monitoring information repositories (for specific parameters, specific system components, etc.), redundancy, etc. This kind of systems usually presents a partitioned network and hierarchical structure. In such a case, GMonE can be deployed with several GMonEDB elements creating different monitoring spaces. GMonE data aggregation and abstraction capabilities would facilitate a tier-like configuration, where only relevant information is propagated from one level of the hierarchy to the next. This would guarantee an efficient use of resources and avoid scalability issues due to network saturation.

6.5. Monitoring resolution

One of the best ways to asses the effectiveness and performance of a given monitoring tool is to study its monitoring resolution, i.e. the level of temporal detail provided or monitoring update period. Amazon CloudWatch, for instance, provides seven preselected metrics updated at 5-minute intervals free of charge, or at 1-minute intervals for an additional fee [2]. As another example, OpenNebula Monitoring Subsystem presents a host monitoring interval of 10 minutes by default, although it is possible to improve this resolution reducing this period down to 30 second intervals [35].

In GMonE the effective monitoring resolution depends on two parameters: the publish period and the archive period. As it has been explained in Subsections 6.3 and 6.4, the publish period depends on the number of published parameters and the archive period on the number of nodes being monitored. In order to generate adequate monitoring information, both tasks (publishing and archiving) have to be performed. These can be done concurrently, since they are performed by different parts of GMonE (GMonEMon publishes the information and GMonEDB archives it). Therefore, the minimum GMonE monitoring period can be estimated using the following equation:

$$mon\_period = max(num\_nodes \times archive\_time\_per\_node, \\ num\_parameters \times publish\_time\_per\_parameter) \quad (1)$$

If we now consider the results shown in Subsections 6.3 and 6.4, we can calculate the best monitoring period for each cloud monitoring type, as shown in Table 4. These results show that GMonE can be a significant improvement over commonly used techniques. Considering the above mentioned monitoring tools update periods of common cloud platforms, such as Amazon EC2 or OpenNebula, we see GMonE as an extremely powerful alternative in this area.

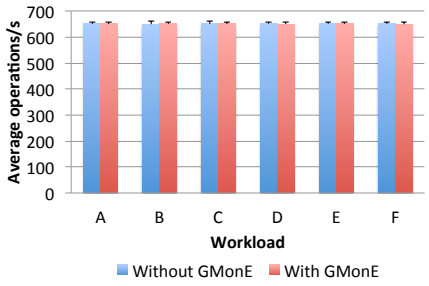
Monitoring type	$num\_nodes \times arch\_time\_per\_node$	$num\_parameters \times publish\_time\_per\_param$	mon_period
Client-oriented	$80 \times 3.73ms = 298.4ms$	$3 \times 74.3ms = 223ms$	$298.4ms$
Physical	$45 \times 1.98ms = 89.1ms$	$5 \times 44.4ms = 222ms$	$222ms$
Virtual	(*)	$4 \times 265.5ms = 1.062s$	$1.062s$

Table 4: Minimum GMonE monitoring periods on the three types of cloud monitoring. (*): Virtual system monitoring is performed globaly, using the OpenNebula XMLRPC interface. Therefore, the *num_nodes* parameter does not make sense in this context, since there is only one OpenNebula system.

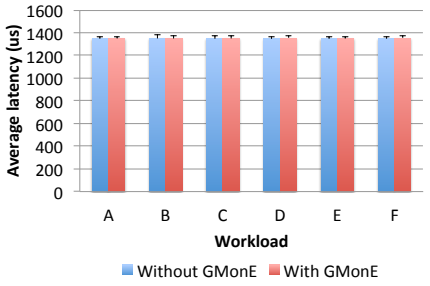
6.6. Monitoring overhead

So far some of the most important features and capabilities of GMonE have been studied. However, another crucial feature every monitoring system must present is its ability to operate transparently, that is, adding as less overhead as possible to the system being monitored. Although absolute transparency is, by principle, not possible (all monitoring systems must consume at least a very small amount of computational resources to operate), an efficient monitoring tool must maintain its overhead below acceptable levels.

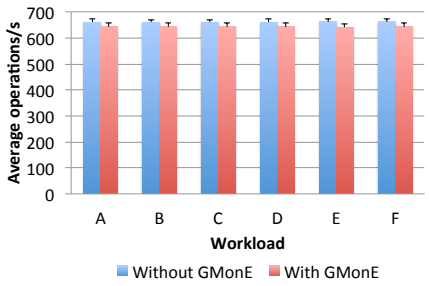
To measure GMonE's overhead we executed a total of 200 repetitions of each YCSB workload (A, B, C, D, E and F) in our cloud testbed, without any GMonE element present. This execution gave us a set of reference measurements. Next, we deployed GMonE at all levels of the infrastructure and we repeated the benchmark series. Using the entire set of YCSB workloads allowed us to measure GMonE overhead



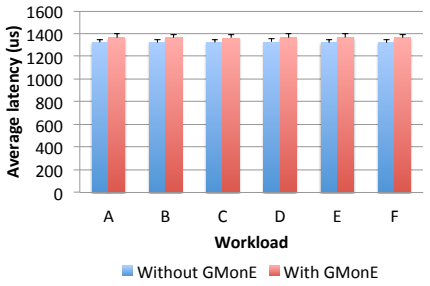
(a) Standard cloud monitoring benchmark throughput



(b) Standard cloud monitoring benchmark latency



(c) Intensive cloud monitoring benchmark throughput



(d) Intensive cloud monitoring benchmark latency

Figure 8: Monitoring overhead experiment. Column height show average values. Error bars show the standard deviation for each case.

under different application conditions. The 200 repetitions of each workload guarantees statistically meaningful results. We performed this experiment using two different GMonE configurations, in order to determine GMonE's overhead when different monitoring requirements are present. These two GMonE configurations were:

- Standard cloud monitoring: this configuration represented typical monitoring processes in cloud infrastructures. GMonEMon elements published their information every 1min and GMonEDB elements archived it every 5min.
- Intensive cloud monitoring: this configuration represented an intensive monitoring scenario where status info was required in short periods of time. GMonEMon elements published their information every 5s and GMonEDB elements processed it every 30s.

Figure 8 shows the YCSB benchmark throughput and latency for each workload and GMonE configuration. Results show no apparent difference in the benchmark operation descriptors (throughput and latency) when using a standard cloud monitoring configuration (Figures 8a and 8b). This seems to indicate that GMonE generates negligible overhead in these conditions. To further validate this conclusion, we have ex-

tended our analysis for this case, performing statistical testing using the Kolmogorov-Smirnov [50] and Wilcoxon [27, 55] statistical tests. These tests essentially take two samples as input and output a *p-value*, looking for statistically significant differences in the two samples. A *p-value* smaller than 0.01 can lead us to conclude that these differences exist. Using these tests we have compared the throughput and latency values obtained using the standard cloud monitoring configuration with the reference values; *p-value* results can be seen in Tables 5a and 5b. As it can be noticed, both statistical tests are able to identify differences only in the case of the latency values of workload D. Differences are found by the Kolmogorov-Smirnov test only in workload D throughput and workload A latency. These results show that GMonE overhead is present (as we said, absolute transparency is impossible), but almost undetectable on most cases. This guarantees effective transparency in standard cloud monitoring configurations.

Workload	**A**	**B**	**C**	**D**	**E**	**F**
Kolmogorov-Smirnov p-value	0.088	0.711	0.393	0.009	0.465	0,964
Wilcoxon p-value	0.289	0.784	0.214	0.020	0.323	0,431

(a) Throughput

Workload	**A**	**B**	**C**	**D**	**E**	**F**
Kolmogorov-Smirnov p-value	0.009	0.178	0.022	0.002	0.068	0.465
Wilcoxon p-value	0,054	0,296	0,012	0,001	0,109	0,176

(b) Latency

Table 5: Standard cloud monitoring statistical tests results.

Figures 8c and 8d show the benchmark operation descriptors obtained for an intensive cloud monitoring scenario. In this case differences can be seen, both in throughput and latency. This means that in intensive monitoring conditions GMonE overhead is more intense, and can be effectively detected. Statistical testing with Kolmogorov-Smirnov and Wilcoxon tests showed a *p-value* less than 0.01 in all cases, identifying statistically significant differences in all workloads. However, results show that even in this intensive conditions benchmark throughput is decreased by less than 3% and latency is increased by less than 3.5% in all cases (workloads A to F). This shows that even in these intensive monitoring conditions GMonE overhead has a very limited impact in the infrastructure operation.

7. Conclusions and Future Work

Modern clouds are highly complex systems designed to efficiently provide infrastructure, platform and applications in the form of elastic, abstract services. In order to control and optimize these services, cloud management systems can make use of detailed monitoring information. Some of this information can be provided as a service as well, presenting clients with a detailed description of their infrastructure/platform/application status and evolution. Cloud monitoring plays a key role, as it provides the means to generate, process and distribute this status information. In this

context, the work described in this paper represents a step forward in the conceptualization and implementation of cloud monitoring systems.

Firstly, we have shown a comprehensive analysis of the different types of cloud monitoring, detailing its characteristics and highlighting the differences between them. The main aspects considered in this case were called *cloud monitoring level* and *cloud monitoring vision*. These two aspects are used as a basis to define a generalized cloud monitoring model, that can be used to understand and study the specific requirements of each cloud monitoring scenario. Based on this model, a generic cloud monitoring architecture is proposed and a specific implementation of it is described.

Secondly, the paper demonstrates the qualitative and quantitative benefits of our approach. As shown in Section 6, GMonE is the only tool which covers all the cloud monitoring types identified. Additionally, the different experiments shown allow us to conclude that GMonE presents a very good behavior according to important metrics: performance, scalability, monitoring resolution and overhead.

As future work, we plan to research the behavior of GMonE in different scenarios including multiple cloud services and applications, and especially in heterogeneous environments as federated clouds. This last point will be of high importance to service providers having private clouds, which in some cases need to play twofold roles: providing resources to their end users, but also using resources from other CSPs. Another work that we intend to explore is the possibility of using GMonE information to detect anomalies and failures in the cloud. The idea is using a failure detector on top of a monitoring system, which requires that the processing data obtained from GMonE should be even more detailed.

Acknowledgment

This work is partially supported by the Madrid Regional Authority (Comunidad de Madrid) and the Universidad Rey Juan Carlos under the URJC-CM-2010-CET-5185 contract, the Spanish Ministry of Education under contract TIN2010-212889-C02-01 and the Marie Curie Initial Training Network (MCITN) "SCALing by means of Ubiquitous Storage (SCALUS)" under contract 238808. Experiments presented in this paper were carried out using the Grid'5000 experimental testbed, being developed under the INRIA ALADDIN development action with support from CNRS, RENATER and several Universities as well as other funding bodies (see https://www.grid5000.fr). The authors would like to thank also Eduardo Perez, who worked with us in the programming of GMonE.

References

[1] Aladdin-Grid'5000, 2012. `http://www.grid5000.fr`.

[2] Amazon CloudWatch, 2012. *http://aws.amazon.com/es/cloudwatch/*.

[3] Amazon Elastic Compute Cloud (Amazon EC2), 2012. *http://aws.amazon.com/en/ec2/*.

[4] Andreozzi, S., De Bortoli, N., Fantinel, S., Ghiselli, A., Rubini, G. L., Tortone, G., Vistoli, M. C., Apr. 2005. GridICE: a monitoring service for Grid systems. Future Gener. Comput. Syst. 21 (4), 559–571.
URL http://dx.doi.org/10.1016/j.future.2004.10.005

[5] Andrieux, A., Czajkowski, K., Dan, A., Keahey, K., Ludwig, H., Nakata, T., Pruyne, J., Rofrano, J., Tuecke, S., Xu, M., 2007. Web Services Agreement Specification (WS-Agreement).
URL https://forge.gridforum.org/projects/graap-wg/

[6] Buyya, R., Yeo, C. S., Venugopal, S., Broberg, J., Brandic, I., Jun. 2009. Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility.
URL http://dx.doi.org/10.1016/j.future.2008.12.001

[7] Cattell, R., May 2011. Scalable sql and nosql data stores. SIGMOD Rec. 39 (4), 12–27.
URL http://doi.acm.org/10.1145/1978915.1978919

[8] Clayman, S., Galis, A., Mamatas, L., Apr. 2010. Monitoring virtual networks with Lattice. In: Proceedings of Network Operations and Management Symposium Workshops (NOMS Wksps), 2010 IEEE/IFIP.

[9] Cooper, B. F., Silberstein, A., Tam, E., Ramakrishnan, R., Sears, R., 2010. Benchmarking cloud serving systems with YCSB. In: Proceedings of the 1st ACM symposium on Cloud computing. SoCC '10. ACM, New York, NY, USA, pp. 143–154.
URL http://doi.acm.org/10.1145/1807128.1807152

[10] Dropbox - Simplify your life, 2012. *http://www.dropbox.com*.

[11] Emeakaroha, V., Ferreto, T. C., Netto, M. A. S., Brandic, I., Rose, C. A. F. D., 2012. Casvid: Application level monitoring for sla violation detection in clouds. In: IEEE Computer Software and Applications Conference (COMPSAC 2012), July 16th-20th, 2012 in Izmir, Turkey.

[12] Emeakaroha, V. C., Brandic, I., Maurer, M., Dustdar, S., 2010. Low level metrics to high level SLAs - LoM2HiS framework: Bridging the gap between monitored metrics and SLA parameters in cloud environments. In: The 2010 High Performance Computing and Simulation Conference (HPCS 2010) June 28July 2, 2010.

[13] Emeakaroha, V. C., Netto, M. A., Calheiros, R. N., Brandic, I., Buyya, R., Rose, C. A. D., 2012. Towards autonomic detection of sla violations in cloud infrastructures. Future Generation Computer Systems 28 (7), 1017 – 1029, ¡ce:title¿Special section: Quality of Service in Grid and Cloud Computing¡/ce:title¿.
URL http://www.sciencedirect.com/science/article/pii/S0167739X11002184

[14] Foster, I., Zhao, Y., Raicu, I., Lu, S., 2008. Cloud Computing and Grid Computing 360-Degree Compared. 2008 Grid Computing Environments Workshop abs/0901.0 (5), 1–10.
URL http://arxiv.org/abs/0901.0131

[15] Ganglia Monitoring System, 2012. *http://ganglia.sourceforge.net/*.

[16] GFI MAX RemoteManagement, 2012. *http://landmaxrm.gfi.com/remote-server-monitoring/*.

[17] González, J., Muñoz, A., Maña, A., 2011. Multi-layer Monitoring for Cloud Computing.
URL http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=6113910

[18] Google Docs, 2012. *http://docs.google.com*.

[19] HPOpenView, 2012. *http://support.openview.hp.com/*.

[20] Huebscher, M. C., McCann, J. A., Aug. 2008. A survey of autonomic computing degrees, models, and applications. ACM Comput. Surv. 40 (3), 7:1–7:28.
URL http://doi.acm.org/10.1145/1380584.1380585

[21] IBM Global Technology Services, June 2011. Security and high availability in cloud computing environments. Technical White Paper MSW03010-USEN-00, IBM.

[22] IBM Tivoli Monitoring, 2012. *http://www-01.ibm.com/software/tivoli/products/monitor/*.

[23] InterMapper, 2012. *http://www.intermapper.com/*.

[24] Lakshman, A., Malik, P., Apr. 2010. Cassandra: a decentralized structured storage system. SIGOPS Oper. Syst. Rev. 44 (2), 35–40.
URL http://doi.acm.org/10.1145/1773912.1773922

[25] LogicMonitor, 2012. *http://www.logicmonitor.com/*.

[26] Ludwig, H., Keller, A., Dan, A., King, R. P., Franck, R., Jan. 2003. Web Service Level Agreement (WSLA) Language Specification, v1.0.
URL http://www.research.ibm.com/wsla/WSLASpecV1-20030128.pdf

[27] Mann, H. B., Whitney, D. R., 1947. On a test of whether one of two random variables is stochastically larger than the other. The Annals of Mathematical Statistics 18 (1), 50–60.
URL http://www.jstor.org/stable/2236101

[28] Massie, M. L., Chun, B. N., Culler, D. E., 2003. The Ganglia Distributed Monitoring System: Design, Implementation And Experience. Parallel Computing 30, 2004.

[29] Maurer, M., Breskovic, I., Emeakaroha, V., Brandic, I., 28 2011-july 1 2011. Revealing the MAPE loop for the autonomic management of Cloud infrastructures. In: Computers and Communications (ISCC), 2011 IEEE Symposium on. pp. 147 –152.

[30] Mell, P., Grance, T., 2009. The NIST Definition of Cloud Computing. National Institute of Standards and Technology 53 (6), 50.
URL http://csrc.nist.gov/groups/SNS/cloud-computing/cloud-def-v15.doc

[31] mysql.www, 2012. Mysql - the world's most popular open source database, accessed Nov 2012.
URL http://www.mysql.com/

[32] Nagios.org, 2012. *http://www.nagios.org*.

[33] Newman, H. B., Legrand, I. C., Galvez, P., Voicu, R., Cirstoiu, C., Jun. 2003. MonALISA : A Distributed Monitoring Service Architecture. In: Proceedings of CHEP03, La Jolla, California.
URL http://arxiv.org/abs/cs.DC/0306096

[34] Nimsoft Monitor Solution, 2012. *http://www.nimsoft.com/solutions/nimsoft-monitor/cloud*.

[35] Open Nebula Monitoring Subsystem 3.4, 2012. *http://opennebula.org/documentation:rel3.4:img*.

[36] OpenNebula: The Open Source Toolkit for Cloud Computing, 2012. *http://www.opennebula.org/*.

[37] OPNET, 2012. *www.opnet.com*.

[38] PacketTrap, 2012. *http://www.packettrap.com*.

[39] Pawlowski, B., Noveck, D., Robinson, D., Thurlow, R., 2000. The NFS version 4 protocol. In: In Proceedings of the 2nd International System Administration and Networking Conference (SANE 2000).

[40] Ramakrishnan, R., Gehrke, J., 2002. Database Management Systems. McGraw-Hill international editions: Computer science series. McGraw-Hill Education.
URL http://books.google.es/books?id=JSVhe-WLGZ0C

[41] rrd4j.www, 2012. Rrd4j - a high performance data logging and graphing system for time series data. - google project hosting, accessed Nov 2012.
URL http://code.google.com/p/rrd4j/

[42] rrdtool, 2012. Rrdtool, accessed Nov 2012.
URL http://oss.oetiker.ch/rrdtool/

[43] Sempolinski, P., Thain, D., 30 2010-dec. 3 2010. A Comparison and Critique of Eucalyptus, OpenNebula and Nimbus. In: Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on. pp. 417 –426.

[44] sFlow.org, 2012. *http://www.sflow.org*.

[45] Site24x7, 2012. *https://www.site24x7.com/*.

[46] SLA@SOI, 2012. *http://http://sla-at-soi.eu/*.

[47] Spring, J., 2011. Monitoring Cloud Computing by Layer, Part 1. IEEE Security & Privacy 9 (2), 66–68.
URL http://dblp.uni-trier.de/db/journals/ieeesp/ieeesp9.html#Spring11

[48] Spring, J., 2011. Monitoring Cloud Computing by Layer, Part 2. IEEE Security & Privacy 9 (3), 52–55.
URL http://dblp.uni-trier.de/db/journals/ieeesp/ieeesp9.html#Spring11a

[49] sqlite.www, 2012. Sqlite home page, accessed Nov 2012.
URL http://www.sqlite.org/

[50] Stephens, M. A., 1974. EDF Statistics for Goodness of Fit and Some Comparisons. Journal of the American Statistical Association 69 (347), 730–737.

[51] SurveyMonkey: Free online survey software & questionnaire tool, 2012. *www.surveymonkey.com*.

[52] TIMACS: Tools for Inteligent System Management of Very Large Computing Systems, 2012. *http://www.timacs.de*.

[53] Vaquero, L. M., Rodero-Merino, L., Caceres, J., Lindner, M., 2009. A Break in the Clouds : Towards a Cloud Definition. Computer Communication Review 39 (1), 50–55.
URL http://portal.acm.org/citation.cfm?id=1496100

[54] Viswanathan, K., Lakshminarayan, C., Talwar, V., Wang, C., Macdonald, G., Satterfield, W., 2012. Ranking anomalies in data centers. In: NOMS. IEEE, pp. 79–87.

[55] Wilcoxon, F., 1945. Individual comparisons by ranking methods. Biometrics Bulletin 1 (6), 80–83.
URL http://www.jstor.org/stable/3001968

[56] Yahoo! Cloud Serving Benchmark (YCSB), 2012. *https://github.com/brianfrankcooper/YCSB/wiki*.

[57] Zissis, D., Lekkas, D., 2012. Addressing cloud computing security issues. Future Generation Computer Systems 28 (3), 583 – 592.
URL http://www.sciencedirect.com/science/article/pii/S0167739X10002554