Monitoring and Metering in the Cloud

Eduardo Oliveros
Telefónica Investigación y Desarrollo SAU, Spain

Tommaso Cucinotta
Scuola Superiore Sant'Anna, Italy

Stephen C Phillips, Xiaoyu Yang, Stuart Middleton
University of Southampton, IT Innovation Centre, UK

Thomas Voith
Alcatel-Lucent, Germany

ABSTRACT

Monitoring and Metering are essential activities for Service Oriented Infrastructures (SOI) and Cloud services. The information collected through monitoring is necessary to ensure the correct execution of the applications in the Cloud and the monitoring of the SLA compliance.
This chapter will present the reasons and difficulties for monitoring and metering on Cloud infrastructures. The approaches for monitoring of the execution environment and the network on virtualised infrastructures will be described together with the existing monitoring tools present on different commercial and research platforms.

INTRODUCTION

Cloud Computing Overview

Cloud computing has become a new computing paradigm as it can offer dynamic IT infrastructure, configurable service and QoS guaranteed computing environment (Wang et al., 2008)1.1.1.7. Cloud computing can be illustrated from the following aspects:
SPI model - Cloud computing originates from the concept "Hardware as a Service" (HaaS), "Software as a Service" (SaaS). Cloud now advances from SaaS to "Platform as a Service" (PaaS) and "Infrastructure as a Service" (IaaS), known as SPI[i] model. In Cloud computing, customers can avoid capital expenditure on hardware and software by renting the usage from service provider of third party, rather than owning the physical infrastructure by themselves. The hardware and software are rendered to customers as IT services.
Scalability / elasticity - Klems and Gaw (in Geelan, 2008) claim that automatic scale of infrastructure for load balancing is a key element in Cloud computing. The delivered services can elastically / dynamically grow its capacity on an as-needed basis so that the Quality of Service (QoS) can be guaranteed: "on-demand services are all cloud computing based" (de Haaff, 2008).
"Pay-per-use" / "Pay-as-you-go" / "Utility computing" - There is also a vision that Cloud computing is more like a business revolution, rather than a technology evolution. Business model, or we call "pay-per-use", "pay-as-you-go", and "utility computing" is another feature of Cloud computing (Geelan, 2008; Watson, Lord, Gibson et al., 2008); Buyya, Yeo, Venugopal,

2009; McFedries, 2008). The usage of the resource will be metered and service customers will pay bill to service provider for the actual resource usage.
Data centre - Another view of Cloud presents the data centre as the basic unit of the Cloud infrastructure (Vaquero, Rodero-Merino, Caceres and Lindner, 2009). Data centre can offer huge amount of computing power and data storage. The capacity of the data centre can change dynamically when handling a task. According to Vaquero, Rodero-Merino, Caceres and Lindner (2009), this is associated with the concept "massive data scalability" proposed by Hand (2007).
Virtualisation - Cloud computing can also be regarded as a "virtualised hardware and software" (Sheynkman in Geelan, 2008). This perspective emphasizes the use of virtualisation technology in the Cloud computing. Virtualization technologies multiplex hardware and have made the flexible and scalable provision of resource as hardware and software on demand easier. Virtual machine techniques, such as VMware (http://www.vmware.com) and Xen (http://www.xen.org), offer virtualized IT-infrastructures on demand. Virtual network advances, such as Virtual Private Network (VPN), support users with a customized network environment to access Cloud resources.

Why does Cloud need Metering and Monitoring

Monitoring tasks comprise a fundamental functionality in every distributed computing system. Every service should be monitored in order to check its performance and allow for corrective actions in case of failure. Monitoring data represents an operational snapshot of the system behaviour along the time axis. Such information is fundamental in determining the origin of the problems or for tuning different system components. For instance, fault detection and recovery mechanisms need a monitoring component to decide whether a particular subsystem or server should be restarted due to the information collected by the monitoring system (Litke et al., 2005). Metering tasks are necessary for checking the disk space, network and memory usage from the machines of the platform. This information is vital to allocate services under conditions of optimum performance.
Metering and monitoring play an important role in Cloud computing, which can be attributed to the following reasons:
From Cloud computing SPI model perspective – Customer consumes services provided by a service provider and service provider outsources the service hosting to the dedicated infrastructure providers. Service Level Agreement (SLA) is usually employed to serve as a bilateral contract between two parties to specify the requirements, quality of service, responsibilities and obligations. SLA can contain a variety of service performance metrics with corresponding Service Level Objectives (SLO). Therefore we need to meter values of associated metrics defined in the SLA at the usage stage to monitor whether the specified service level objectives are achieved or not.
From Cloud computing "Pay-per-use" / "Pay-as-you-go" / "Utility computing" perspective - Cloud service provider delivers QoS-assured services and other commitments in exchange for financial commitments based on an agreed schedule of prices and payments. This requires the service / resource usage to be metered, based on which the bill can then be calculated.
From Cloud computing scalability / elasticity and data centre perspective - These two perspectives have a feature in common, that is, capacity on-demand or called on-demand resources provisioning. The service and resource usage need to be metered and monitored to support this dynamic scale feature on an as-needed basis. In the report "The Future of Cloud Computing" (Schubert, Jeffery, Neidecker-Lutz, et al., 2010) produced by the EC with the

support of a group of external experts, it has been identified that Cloud computing is facing following challenges. From the technical aspects, the challenging issues are mainly related to: (i) scalability and elasticity, (ii) trust, security and privacy, (iii) handling data in Clouds is still complicated, especially large amount of data and heterogeneous data, and (iv) programming model are currently not aligned to highly scalable applications and thus cannot exploit the capabilities of Clouds. The Cloud scalability and elasticity issue which is directly related to monitoring and metering is listed at the top of the challenges that need to be addressed.

The difficulties of performing monitoring and metering in the Cloud

There are different ways of collecting monitoring data, for instance:

1. Obtain information from the Operating System or the Virtual Machine Unit (VMU) in relation to the CPU usage, the network interfaces and the storage connected to the machine.

2. Simple Network Management Protocol (SNMP) is used mostly on network system, and can be used to monitor the status of the network and detect error events (SNMP traps enable an agent to notify the management station of significant events by way of an unsolicited SNMP message). Besides SNMP there are other protocols provided by network vendors to monitor and gather usage statistics, for instance NetFlow is a network protocol developed by Cisco Systems to run on Cisco IOS-enabled equipment for collecting IP traffic information.

3. Extract the information from the application log files, searching for specific patterns that provide information about interesting events in the application, for instance: error in the interaction with the client (this also could be related to denial of service (DoS) attacks) or number of client interactions performed.

4. Specific ad-hoc monitoring mechanism. Applications that do not provide standard mechanisms (like SNMP) to deliver monitoring data can implement a private API for monitoring. This mechanism could be a set of proprietary functions or more generically a SOAP interface with an XML specific message format.

The following steps are required for monitoring and link the raw monitoring data to the behaviour of the application (Heroix, 2006):

1. **Collect and correlate to reveal service performance**. One of the problems to perform correctly this correlation of information from the different sources is time synchronization. On a virtualized environment where the physical resources are shared by different applications and the infrastructure has the ability of reallocate the resources depending on the needs in each specific moment, one portion of a network or one server is not longer associated to a single service or application. The monitoring system will be unable to associate the corresponding data of the different systems to the application unless a precise time synchronization mechanism is in place. Global Positioning System (GPS) (http://www.gps.gov) afford one way to achieve synchronization within several

tens of μsec. Ordinary application of NTP (http://www.ntp.org/) may allow synchronization within several msec. The different granularities of those time synchronization mechanism determine the precision of the monitoring system at a whole (e.g. it is not possible to detect network delays of the order of milliseconds or less if the time difference of the clocks is also of this order).

2. **Interpret the business impact**. It is not trivial to deduce from the monitoring data collected from the different systems the performance of the application, and infer if the application is performing correctly. From one side, a complete outage in one segment of the network affecting some servers inside the application architecture could have no effect in the performance perceived by the user, for instance during low usage period where the rest of the service platform is able to provide the requested performance to the user. On the other side, a completely functional infrastructure could be insufficient to provide the QoS requested by the users. Accordingly, the monitoring system should gather information from the infrastructure and from the application components themselves to have a complete view of the execution and performance of the application.

3. **Resolve quickly; prevent when possible**. The ideal situation is the prevision of the user demand so that the provider can anticipate the requirements and adapt the application dynamically to support this demand. This is the promise of the cloud, the easy adaptation of the application to the changing environment that affects it. To support this, the application must be designed in advance to allow the dynamic deployment of application components, and the reconfiguration of the application to support these changes in the internal architecture of the application.
   When an unexpected error occurs that affects the performance of the application, the cloud provider should provide the mechanism to detect those events and react to them to minimize as possible the negative effects on the clients. From an operational point of view the desired situation is when the monitoring application is able to suggest corrective actions to errors and events that can affect the application performance.

APPROACHES, IMPLEMENTATIONS & COMPARISONS

There are several approaches in the design and development of monitoring and metering systems. This section presents briefly the techniques used to monitor the execution environment and the network on virtualised infrastructures. Finally, different monitoring tools of existing grid and cloud platforms are described, including commercial platforms like Amazon Web Services and Google App Engine; and monitoring systems used on open source and academic solutions like GRIA, Akogrimo, Edutain@Grid, Globus Toolkit and gLite.

Monitoring CPU usage in virtualized SOIs

Monitoring of actual resources usage in SOIs is of paramount importance due to the value that the corresponding information flow possesses in the overall resource allocation and management strategies. In fact, monitoring data may be used by resource providers for a variety of purposes, including:

- foreseeing the future loads for an optimum allocation of the available physical infrastructures;
- identifying possible bottlenecks within the infrastructure leading to either temporary or permanent performance degradations;
- fine-tuning their price lists and business model policies.

More specifically, monitoring data about the actual usage of *computing power* in a *distributed virtualized infrastructure* is particularly critical because not only the availability of CPU power is at the basis of providing compute-intensive services to the final users, but also for I/O intensive ones. In fact, retrieving selected portions/cuts of large amounts of data from remote disks and routing them to the final data consumers may require significant amounts of computing power for such tasks as:

- streaming the data on the network;
- selecting the requested data and/or aggregating or transforming it as requested (think of database applications);
- routing them among possibly virtualized gateway nodes.

Depending on the involved Operating Systems, the virtualization technologies in use, and specifically on the level of para-virtualization of the I/O and networking layers, the computing requirements due to a given data throughput may largely vary. Furthermore, when more VMUs run concurrently on the same physical node, a computing power shortage due for example to a CPU-intensive VMU may result in a degraded network throughput for another data-intensive VMU and vice-versa. Such situations need to be properly detected on-time and compensated (e.g., by live-migrating VMUs for load-balancing purposes), and monitoring plays a key role in this context.

However, monitoring of the CPU usage is not as easy as it is for other resources like network and memory, where objective metrics may be used for monitoring purposes, such as the number of transferred bytes. In fact, there are various factors that need to be properly considered in a monitoring process:

- the accuracy of the used sensors, in terms of time-precision by which they are capable of reporting CPU usage;
- the granularity at which the monitored data is collected, for example one may collect per-thread, per-process, per-VMU or per-node CPU usage data;
- CPU power-saving capability of the monitored physical nodes.

The accuracy of the monitoring data depends on the precision of the hypervisor and involved Operating Systems in accounting the processes load. In this section the focus is on the Linux Operating System and the kvm (“KVM, Kernel-based Virtualization Driver, Whitepaper”, 2006) virtualization solution, both products are open source which allows a major customisation of the software. Using kvm, the Virtual Machine Units run as processes of the host Linux instance.

Accuracy of monitoring of the CPU usage by individual VMUs depends on the accuracy of Linux in tracking the CPU usage of individual processes. Historically, the Linux kernel used to perform all of its time-related book-keeping activities, comprising measuring time and firing timers, at a precision depending on the frequency of the system tick, whose period is statically configured into the kernel to one of a few values between 1ms and 10ms. Even worse, the per-process monitoring data reported by the kernel used to be collected based on what observed at each system tick, independently of the actual CPU usage of a task between ticks, rather than on the actual time instants at which tasks where scheduled on the CPUs by the kernel. As a consequence, it was also possible for a properly designed unprivileged process in the system to consume nearly all of the available CPU power, still having the system report that the system was almost idle (Tsafrir, Etsion & Feitelson, 2007)1.1.1.7.
In order to mitigate such problems, it was once necessary to patch the kernel applying the well-known *high-resolution timers* patch, which introduced a much more fine-grained notion of time within the Operating System. Since the 2.6.16 version, Linux integrates (Corbet, 2006) such mechanism in the mainstream kernel, and in recent kernel versions the tasks accounting mechanism is done based on the actual CPU schedule. However, even though the kernel had high-resolution timers, it was not using them for accounting the CPU time consumed by processes. For this reason, in order to get a reliable CPU accounting information, it was usually necessary to introduce a custom mechanism flanking the scheduler. This was done, for example in the AQuoSA real-time scheduler for Linux (Palopoli, Cucinotta, Marzario & Lipari, 2009).
Also, another important issue to take into consideration is the possible CPU load due to system OS elements whose accounting is troublesome. One typical example is constituted by the CPU time consumed by *interrupt drivers*. These occur asynchronously with respect to the running processes and possible VMUs, and usually the OS accounts "randomly" the duration of the interrupt handlers onto the interrupted processes, what may produce additional undesired noise on the measured data. Such phenomenon may at least be controlled at a finer grain level by using the peempt-rt patchset (http://www.kernel.org/pub/linux/kernel/projects/rt)(Arthur, Emde & Mc Guire, 2007) by Ingo Molnar available on recent Linux kernel series. This introduces into the kernel, among others, an infrastructure for dealing with interrupt handlers into dedicated kernel threads. Basically, interrupt handling code formerly executing in interrupt context now is split into a demultiplexing logic which executes in response to the hardware interrupt, and the actual driver running into a kernel thread on its own and which is woken up by the interrupt driver. This allows not only for a finer control on the way interrupt drivers are scheduled with respect to other activities in the system (Manica, Abeni & Palopoli, 2010), but also for a better monitoring of the elements that take up CPU power into a system.
Concerning the interfaces by which monitoring data is made available from the kernel to user-space processes, Linux offers various alternatives. First, it is possible to rely on POSIX (2004) compliant time measurement system calls, such as the clock_getcpuclockid(), clock_gettime() and pthread_getcpuclockid(). Using the special CLOCK_PROCESS_CPUTIME_ID and CLOCK_THREAD_CPUTIME_ID timers, it is possible to track the CPU power consumed by individual processes and threads, respectively. However, the implementation of such mechanisms in the Linux kernel is far from being stable, and the actual way it is done is still sensitive to the kernel version that is being used. The advantage of relying on standard system calls is of course portability of the software components across different Operating Systems.
On the other hand, custom extensions were proposed to get accurate CPU accounting information from the kernel. For example, in the AQuoSA real-time scheduler mentioned above

(Palopoli, Cucinotta, Marzario & Lipari, 2009), a special API call was introduced for getting at once the overall CPU time consumed by the set of tasks attached to a real-time reservation. Alternatively, on recent kernel versions, the Linux-specific cgroups (Menage, n.d.) virtual filesystem may be exploited for defining groups of processes in the system and attaching them a specific *controller*, which is capable of tracking the overall CPU consumption of all of the tasks in the group. This way it is possible to track the CPU consumption of an entire VMU (constituted by a set of threads composing a VMU process) or a group of VMUs. This method may be easily applied, for example, when using the IRMOS real-time scheduler (Checconi, Cucinotta, Faggioli & Lipari.2009) for achieving temporal isolation across multiple concurrently running VMUs (Cucinotta, Anastasi & Abeni. 2008), which already leverages the cgroups special filesystem for configuring real-time reservations and the tasks associated to them. In fact, when configuring (via the mount operation) the cgroups filesystem, it is possible to attach a CPU Accounting Controller to the same filesystem. This way, a special filesystem entry may be used in order to read the overall time spent by the tasks attached to the reservation in user-mode and kernel-mode. For example, this is extremely useful for the benchmarking (Boniface et al. 2010) phase while deploying real-time Virtual Machines (Cucinotta, Anastasi & Abeni. 2009) in the best possible (ideally optimum) way (Kostanteli et al. 2009; Cucinotta, Konstanteli & Varvarigou. 2009).
Power management usually done by modern CPUs by frequency switching and dynamic voltage scaling constitutes yet another source of uncertainty and unreliability, for a monitoring infrastructure. In fact, a host may figure out as being quite loaded simply because the power management system daemon is running the CPU at a reduced frequency. For example, the powernowd (http://www.deater.net/john/powernowd.html) daemon implements a simple utilization-control loop that aims to keep each CPU saturation level between the configurable thresholds, defaulting to 20% and 80%. If power management is active on a system, then CPU usage monitoring data needs to be accurately complemented by the data about the actual speed of the physical CPUs, in order to gather reliable and meaningful measurements. However, this may not be always done precisely, due to the fact that the monitoring infrastructure usually runs asynchronously with respect to the power management control logic. In such cases, reliable results may be obtained by implementing a custom power-management logic that is tightly integrated with the virtualized infrastructure management services. For example, this needs to be done anyway if one wants to provide real-time guarantees and/or stable QoS levels to the hosted virtualized applications.
Finally, another possibility for taking into account frequency switching into monitoring data is to recur to cycle-counters as available on many CPUs, e.g., the RDTSC instruction on Intel 386 architectures. This way one would base the load information on the number of CPU cycles. However, while such data may easily be gathered for the entire system, differentiating the monitoring on a per-process basis may not be as easy.

Network monitoring in the Cloud

Currently, in most cloud solutions the network is considered as granted and usually information about the network performance and its status inside the cloud infrastructure is not provided (Oberle et al., 2009). This is a major problem to support interactive real-time applications where the characteristics of the network are a key factor for maintaining the performance of the application.

In several IETF documents IP performance metrics have been specified (Paxson, Almes, Mahdavi & Mathis, 1998):

- The metrics must be concrete and well-defined.
- A methodology for a metric should have the property that it is repeatable, i.e. if the methodology is used multiple times under identical conditions, the same measurements should result in the same measurement results.
- The metrics must exhibit no bias for IP clouds implemented with identical technology.
- The metrics must exhibit well-understood and fair bias for IP clouds implemented with non-identical technology.
- The metrics must be useful to users and providers in understanding the performance they experience or provide.

For a given set of well-defined metrics, a number of distinct measurement methodologies may exist. Some examples are:

- **Direct** measurement of a performance metric using injected test traffic.
  Example: measurement of the round-trip delay of an IP packet of a given size over a given route at a given time.
- **Projection** of a metric from lower-level measurements.
  Example: given accurate measurements of propagation delay and bandwidth for each step along a path, projection of the complete delay for the path for an IP packet of a given size.
- **Estimation** of a constituent metric from a set of more aggregated measurements. Example: given accurate measurements of delay for a given one-hop path for IP packets of different sizes, estimation of propagation delay for the link of that one-hop path.
- **Estimation** of a given metric at one time from a set of related metrics at other times. Example: given an accurate measurement of flow capacity at a past time, together with a set of accurate delay measurements for that past time and the current time, and given a model of flow dynamics, estimate the flow capacity that would be observed at the current time.

When a quantity is quantitatively specified, we term the quantity a metric. Each metric will be defined in terms of standard units of measurement. The international metric system will be used (meters, seconds...). Appropriate related units based on thousands or thousandths of units are acceptable (e.g. km or ms, but not cm). The unit of information is the bit. When metric prefixes are used with bits or with combinations including bits, those prefixes will have their metric meaning (related to decimal 1000), and not the meaning conventional with computer storage (related to decimal 1024). When a time is given, it will be expressed in Universal Time Coordinated (UTC).

Metrics are specified but measurement methodologies are not formally standardized.
A methodology for a metric should have the property that it is repeatable: if the methodology is used multiple times under identical conditions, it should result in consistent measurements. In practice, as conditions usually change over time, it is enough to ask for “continuity”, to describe a property of a given methodology: a methodology for a given metric exhibits continuity if, for small variations in conditions, it results in small variations in the resulting measurements.
Network measurement methods can broadly be classified into passive methods that rely on data collected at e.g. routers, and active methods based on observations of actively-injected probe packets. Network operators use active measurements because they are easy to conduct, have low

overhead and, in contrast to passive data collection methods, measure exactly what normal data packets experience. One of the main disadvantages of active measurements is their limited accuracy due to the need to be non-intrusive, thus leaving the measured systems uninfluenced by the observation, fundamentally affecting accuracy (Machiraju, 2006).
<u>Active techniques</u>, in which traffic is injected into the network. The overall link can be characterized and they are not aware of application protocols. This is also known as intrusive measurements.
<u>Passive techniques</u>, in which existing traffic is recorded and analyzed. They may deal with connections (pair of IP addresses and ports) and they may distinguish between different protocol streams. This is also known as non-intrusive measurements.
For active monitoring, it is clear that as the number of resource pairs increases, the injected traffic incurs a significant disruption in the network, so usually such measurements are performed sequentially, measuring one or a few paths at a time. In contrast, a passive monitoring approach can provide an instant estimation across different paths, independently of their number. It is recommended in a cloud to follow a strategy of doing passive monitoring rather than active monitoring. If active monitoring is unavoidable then it is preferred to do representative measurement (e.g. just one per network path) to be as minimal intrusive as possible.

Monitoring in Cloud Platforms

1.1.1.1. Amazon Web Services

One relevant area of monitoring is the so called 'Cloud' approaches for online access to virtualised and distributed resources and applications. Amazon's Web Service suite (http://aws.amazon.com/) is a good example. Amazon's Web Services include S3 for storage, EC2 for compute resource, SQS for queuing and SimpleDB for a database. Various vendors are adapting and porting their applications to run on the platform, including J2EE, Solaris and mySQL. Relevant to this section is the obvious need for consumers of 'cloud' services to monitor the performance and availability they are getting from the services, especially when using these services to deliver business critical applications.
Amazon CloudWatch (http://aws.amazon.com/cloudwatch/) is a web service that provides monitoring for AWS cloud resources, starting with Amazon EC2. It provides customers with visibility into resource utilization, operational performance, and overall demand patterns—including metrics such as CPU utilization, disk reads and writes, and network traffic. To use Amazon CloudWatch, clients simply have to select the Amazon EC2 instances they want to monitor; Amazon CloudWatch will begin aggregating and storing monitoring data that can be accessed using the AWS Management Console, web service APIs or Command Line Tools. Amazon CloudWatch provides two APIs: Query and SOAP API to collect programmatically monitoring information. The set of metrics that can be collected from the EC2 service are shown in the following table ("Amazon CloudWatch Development Guide", 2009):

Metric	**Description**
CPUUtilization	The percentage of allocated EC2 compute units that are currently in use on the instance. This metric identifies the processing power required to run an application upon a selected instance. Units: *Percent*
NetworkIn	The number of bytes received on all network interfaces by the instance. This metric identifies the volume of incoming network traffic to an

	application on a single instance. Units: *Bytes*
NetworkOut	The number of bytes sent out on all network interfaces by the instance. This metric identifies the volume of outgoing network traffic to an application on a single instance. Units: *Bytes*
DiskWriteOps	Completed write operations to all hard disks available to the instance. This metric identifies the rate at which an application writes to a hard disk. This can be used to determine the speed in which an application saves data to a hard disk. Units: *Count*
DiskReadBytes	Bytes read from all disks available to the instance. This metric is used to determine the volume of the data the application reads from the hard disk of the instance. This can be used to determine the speed of the application for the customer. Units: *Bytes*
DiskReadOps	Completed read operations from all disks available to the instances. This metric identifies the rate at which an application reads a disk. This can be used to determine the speed in which an application reads data from a hard disk. Units: *Count*
DiskWriteBytes	Bytes written to all disks available to the instance. This metric is used to determine the volume of the data the application writes onto the hard disk of the instance. This can be used to determine the speed of the application for the customer. Units: *Bytes*

These interfaces are now driving the emergence of specialised monitoring services, for example CloudStatus that provides online and live feeds on the status of Amazon's Web Services (http://www.cloudstatus.com) as well as longer term statistics on past performance (for example bandwidth for S3). CloudStatus is implemented using an open source monitoring and management infrastructure called Hyperic HQ (http://www.hyperic.com) which provides an extensible approach for application monitoring on SOIs.

1.1.1.2. Google App Engine monitoring tools

This section presents three monitoring tools available to monitor the activity of application running on Google App Engine (http://code.google.com/appengine/): Appstats library, application log files and the App Engine Scaffold plug-in system.
Appstats (Google, 2010) is a Python and Java library that is linked to the application to allow monitoring of the Remote Procedure Calls (RPC) of the application (van Rossum, 2010). This tool is included into the SDK provided by Google. Appstats integrates with the applicatoin using a servlet filter to record events, and provides a web-based administrative interface for browsing statistics.
The Appstats servlet filter adds itself to the remote procedure call framework that underlies the App Engine service APIs. It records statistics for all API calls made during the request handler,

then stores the data in memcache ("Using Memcache", 2010). Appstats retains statistics for the most recent 1,000 requests (approximately). The data includes summary records, about 200 bytes each, and detail records, which can be up to 100 KB each.

Besides this tool, App Engine also maintains a log of messages emitted by the application. App Engine records each request in the log. This log file can be browsed using the Admin Console. If a developer wants to perform more detailed analysis of the application's logs, he or she can download the log data to a file on their computer.

Finally, the App Engine Scaffold (http://support.hyperic.com/display/hypcomm/App+Engine+Plugin+Tutorial)Error: Reference source not found is another tool that allows developers to monitor an application running on Google App Engine using Hyperic HQ. The scaffold is a set of files, a plugin descriptor, some Python scripts, and a shell script, that the developer can use as a starting point for building a special type HQ plugin that enables HQ components to accept and manage metrics generated by an application running on App Engine

Together, Hyperic HQ and an HQ App Engine Plugin allow developers to:

- Chart and trend the application metrics over time, to answer questions like *"How many blog posts did I have in June?"*
- Execute and apply alerts to synthetic transactions. For instance, a developer can define an alert on the time it takes to create a new user, and be notified when the "create user" transaction takes longer than 2 seconds.
- Apply alerts to metrics, including those that indicate application usage. For instance, given a game application, it is possible to calculate a "games_played_today" metric, push it to HQ, and define an alert so that clients will be notified if the value is lower than a predefined threshold.
- Compare application metrics from different deployment environments. For instance, developers can compare how long it takes to run a particular transaction during development, QA, and production environments.

1.1.1.3. Akogrimo's Monitoring System

Akogrimo (http://www.akogrimo.org/) was an EU research project partially funded by the EC under the FP6-IST programme aiming at the definition and realization of a Mobile Grid Architecture ensuring the viability of this concept for the different stakeholders in the value chain by developing new business models for this commercial platform (Wesner, Järnert & Aránzazu).
Within the Akogrimo's framework an Execution Management System was developed on the basis of the OGSA and WSRF specifications. This system was therefore responsible for finding execution candidate locations, selecting the most suitable execution location, preparing, initiating and managing/monitoring the execution of services. In order to address these tasks and at the same time ensure continuous conformation to the terms of the SLA contract, EMS works closely together and involves numerous interactions with a Monitoring group of services.
The Akogrimo Monitoring group of services consists of the following three Grid services, developed on the GT4 platform:

- Metering service, which maintains run-time information on the performance parameters related to the business service execution, defined in the SLA. In particular, it measures low level performance parameters such as memory, CPU and disk usage. This low level information is communicated to the Monitoring service through a notification mechanism established between the two services. The notification mechanism was developed on the basis of the WS-BaseNotification (Graham, Tibco & Murray, 2006), a specification that belongs to the WS-Notification family of specifications (OASIS Web Services Notification TC, 2006) that enables the use of the notification design pattern with Grid Services. In more detail, the Metering service has two main functionalities:
    - To notify the Monitoring service about the changes in the following low level parameters related to the execution of a service:
        - CPU usage.
        - Memory usage.
        - Disk usage.
        - Wall clock time (time that elapsed while the business service was running).
    - An aggregated version of the information that is calculated by the Metering service is communicated to the accounting A4C[ii] system and used for accounting purposes at the end of the service execution.
- QoS Broker service, which is a bandwidth broker that can provide three bundles of network services, each one of them corresponding to a specific usage profile: audio, video or data. It handles QoS network requests, keeps records of information related to the client, such as network QoS levels the client is allowed to use according to the SLA contract and is also responsible for monitoring network parameters, such as network bandwidth throughout the execution of a service.
- Monitoring service, which is the link between the Metering group of services and the SLA Enforcement group of services. It receives notifications about the values of the QoS parameters related to the execution of the services from the Metering and QoS Broker services and notifies the SLA Enforcement group of services. In more detail, the Monitoring component can be seen as a set of four components that provide the following functionalities:
    - Remote control concerning enabling/disabling of the monitoring process.
    - Reception of low level parameters from the Akogrimo producer services (Metering service and QoS Broker service).
    - Sending of QoS object to the SLA enforcement components.
    - Storage of monitoring information about the different SLA/services being monitored.

More information on their functionality and their implementation can be found in (Litke et al., 2005, 2007; Inácio et al., 2005).

1.1.1.4. GRIA Monitoring System

GRIA (http://www.gria.org/) is a service-oriented infrastructure designed to support B2B collaborations through service provision across organisational boundaries in a secure, interoperable and flexible manner. GRIA uses Web Service protocols based on key interoperability specifications. GRIA is available free and open source (most of the software is LGPL).

The GRIA SLA Management Service Error: Reference source not foundwas developed in the SIMDAT project (http://www.simdat.org/) to monitor and manage a wide range of services. The main objective of the SIMDAT project was to test and enhance grid technology for product development and production process design as well as to develop federated versions of problem-solving environments by leveraging enhanced Grid services.
The SLA Service in GRIA takes responsibility for pulling usage reports from managed services, evaluating the usage of the services against the SLAs in force and enacting the necessary management actions to maintain the service provider's commitments.
Each managed GRIA service is instrumented to place usage reports on a WS-BaseNotification (Graham, Tibco & Murray, 2006) pull-point which the SLA Service periodically polls. The usage reports are defined in terms of "metrics", loosely defined as "something measurable" such as the number of CPUs, amount of data or number of sessions. Each metric is represented by its own unique URI. These metrics are also used in to define the service's capacity, the quality of service offered in an SLA and the cost of usage.

The use of metrics is reported and recorded in terms of "instantaneous" and "cumulative" measurements. The cumulative usage is the numerical integration of the instantaneous measurements over time. For some metrics, data-transfer for example, the instantaneous measurement is best thought of as a rate (bytes per second) and the cumulative usage (the time integral, which in this case is the amount of data transferred) has no time dimension (bytes). For other metrics, such as CPU, the instantaneous measurement is just the quantity in use at the time (e.g. 3 CPUs) and it is the cumulative usage that has the time dimension, e.g. 180 CPU.seconds. By using some basic assumptions, the SLA service can convert between the two. For instance, if the rate of data transfer is reported periodically then, with the assumption that the rate of a metric remains constant unless otherwise notified, the SLA service can compute the numerical time integral of this rate to give an approximation of the total data transfer (see Figure 1). Of course, there are easier ways to find the total data transferred, this example just demonstrates the concept of inferring one value of a metric from others.

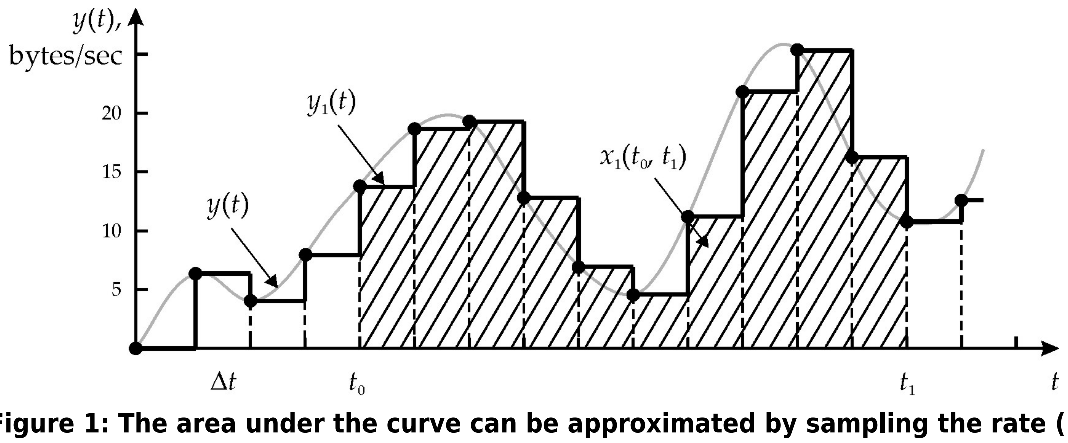

Figure 1: The area under the curve can be approximated by sampling the rate (black dots) and performing numerical integration (the hatched area).

Another example is CPU time. If a job runs on 1 CPU for 5 minutes then the SLA service will be notified that the instantaneous measurement of CPU usage went to 1 at the start and then to 0 five minutes later. The SLA service can infer that 300 seconds of CPU time have been used (1*5*60 = 300 CPU seconds). Alternatively, if a service reported that it had used 120 units of a

resource in a 1 minute period, the SLA service would infer that the average instantaneous measurement (rate of usage) had been 2 units/s.
All metrics have both instantaneous measurements and cumulative usage which may be recorded or inferred from each other. For some metrics one or other concept will not be useful, but the SLA manager has no idea of what it is counting, restricting or billing for in each metric, and so can cope with either type of measurement and can always infer the other from it.
A managed GRIA service can report usage of a metric in two main ways:

1. An instantaneous rate report: specifying the rate of usage of a metric at a particular instant in time. For instance, “the number of CPUs in use at 12:53 was 2.2”.

2. A cumulative usage report: specifying the time-integral of a metric between two times. For instance, “the number of CPU seconds used between 10:20 and 10:30 was 1320”.

For instance, the standard GRIA job service uses the following metrics:

- http://www.gria.org/sla/metric/activity/current-activities

  This is set to one when a job is created and to zero when it is destroyed.
- http://www.gria.org/sla/metric/activity/job

  This is set to one when a job is created and to zero when it is destroyed.
- http://www.gria.org/sla/metric/resource/cpu

  As a job executes, the number of CPU.seconds used by the process as reported by the host operating system is repeatedly reported as cumulative reports.

The standard GRIA data service uses the following metrics:

- http://www.gria.org/sla/metric/activity/current-activities

  This is set to one when a job is created and to zero when it is destroyed.
- http://www.gria.org/sla/metric/activity/data-stager

  This is set to one when a data-stager is created and to zero when it is destroyed.
- http://www.gria.org/sla/metric/resource/disc

  When data is stored in a data-stager, the disc space usage is recorded by setting this to the file size.
- http://www.gria.org/sla/metric/resource/data-transfer

  When data is transferred to or from a data-stager, the transfer is recorded by setting the cumulative usage of this metric to the file size.

The information from these reports is stored in a database at the SLA Service and the individual data points or the inferred points in between can be queried. The data, along with the mathematical model encoded in the service, provide for flexible SLA terms. For instance:

- Constraining the total disc space permitted.

- Pricing disc space usage per GB/month (as Amazon S3).

- Pricing disc usage according to the maximum capacity (as many ISPs do).

- Constraining and pricing total data transfer.
- Monitoring (and potentially constraining) data transfer rate.
- Charging per job at a flat rate and/or according to the CPU time.
- Constraining the CPU time over a period.
- Constraining the maximum number of CPUs permitted at any one time.

All this is achieved without the SLA Service having any specific understanding of the metrics being monitored. When services are adapted to report usage to a GRIA SLA Service the developer chooses metrics appropriate to the service. Usage of these metrics is reported and the same variety of functionality is automatically available.

1.1.1.5. Cloud Metering and Monitoring in Edutain@Grid

Edutain@Grid (http://www.edutaingrid.eu/) is an EU-funded project which aimed to develop a scalable QoS-enabled business Grid environment for multi-user Real-time Online Interactive Applications (ROIA) (Fahringer, Anthes, Arragon, Lipaj et al., 2007). ROIAs (e.g. on-line games) are characterised by the high rate of interaction between users, requiring very fast updates of information being passed from one computer to another (Ploß, Glinka & Gorlatch, 2009). Large numbers of users may participate in a single session and are typically able to join or leave at any time. Thus an ROIA typically has an extremely dynamic distributed workload, making it difficult to host efficiently. In this project, a service-oriented infrastructure with elastic resource provisioning (both up and down) and enhanced security features was developed to support a business model where multiple independent infrastructure providers provide a level of QoS to ROIA service to customers.

The Edutain business model (Middleton, Surridge, Nasser & Yang. 2009) introduces a concept, namely the "Coordinator", which can be regarded as an organisation that plays the role of ROIA application service provider. However, the Coordinator itself may not have any physical infrastructure for running ROIA, instead it outsources the ROIA hosting services to one or more 'Hosters' (i.e. organisations that provide the infrastructure or platform for running ROIA applications). Each Hoster provides a collection of web services required for running the ROIA, QoS monitoring (QoS targets defined in a SLA) and invoicing facilities. The Coordinator has an account and a bipartite Service Level Agreement with each Hoster and the Hoster provides metered services to the Coordinator. When the hosting service finishes, the Hoster sends the bill to the Coordinator for the resource usage.

The combination of each autonomous Hoster and the Coordinator provides a scalable virtual environment for running the ROIA. The capacity of this virtual environment can dynamically grow or shrink through 'zone migration' and 'zone replication' across multiple Hosters during a live gaming session.

The GRIA middleware was employed in the implementation of the business layer of this system. The standard GRIA SLA Management Service adds together metric QoS measurements from across all activities.

In Edutain, six metrics to be monitored in the SLA were defined, namely,

- ServerTickDuration in milliseconds (a measure of server frame rate),

- AveragePacketLoss as a percentage,
- ServerThroughputIn in bytes per second,
- ServerThroughputOut in bytes per second,
- AveragePacketLatency in milliseconds, and
- ClientConnectionCount.

Associated pricing terms such as variable pricing and penalty costs were also defined for each metric. For some metrics such as ServerTickDuration, a penalty is incurred if its peak value exceeds the constraint defined in the SLA, whereas for other metrics such as ServerThroughputIn and ServerThroughputOut, a fee is calculated depending on the accumulated usage. Once an ROIA is in use, it is monitored and QoS measurement data of these metrics are reported back to the SLA service. The SLA service aggregates the data and generates the QoS measurement report on-demand to the Coordinator's SLA monitoring service. If the work load of a particular Edutain local session is predicted to exceed the threshold defined in SLA, Edutain 'zone migration' can be triggered either manually or automatically by to migrate the zone to an idle Edutain Hoster.

1.1.1.6. Globus Toolkit Monitoring

The Globus Toolkit (http://www.globus.org/toolkit/) is an open source middleware used for building Grid applications. The toolkit includes software services and libraries for resource monitoring, discovery, and management, security and file management in a Grid environment (Foster, Kesselman & Tuecke. 2001).
The Globus Toolkit provides a Monitoring and Discovery System (MDS) (http://www.globus.org/toolkit/mds/) which is the information services component and provides information about the available resources on the infrastructure and their status. MDS4 is a WSRF implementation of information services released with Globus Toolkit 4.0.
This version of MDS includes WSRF implementations of the Index Service, a Trigger Service, WebMDS (formerly known as the Web Service Data Browser) and the underlying framework, the Aggregator Framework.
The MDS is a suite of web services to monitor and discover resources and services. This system allows users to discover what resources are considered part of a *Virtual Organization (VO* and to monitor those resources. MDS services provide query and subscription interfaces to arbitrarily detailed resource data and a trigger interface that can be configured to take action when pre-configured trouble conditions are met. The services included in the WS MDS implementation (*MDS4*) acquire their information through an extensible interface which can be used to:

- query WSRF services for resource property information,
- execute a program to acquire data, or
- interface with third-party monitoring systems.

Different computing resources and services can advertise a large amount of data for many different use cases. MDS4 was specifically designed to address the needs of a Grid monitoring system – one that publishes data that is available to multiple people at multiple sites. As such, it is not an event handling system, like NetLogger, or a cluster monitor on its own, but can interface to more detailed monitoring systems and archives, and can publish summary data using standard interfaces.

MDS4 includes two WSRF-based services: an *Index Service* (The Globus Alliance, 2005c)Error: Reference source not found, which collects data from various sources and provides a query/subscription interface to that data, and a *Trigger Service* (The Globus Alliance, 2005a), which collects data from various sources and can be configured to take action based on that data. An *Archive Service*, which will provide access to historic data, is planned for a future release.

The *Index Service* is a registry similar to UDDI, but much more flexible. Indexes collect information and publish that information as *resource properties*. Clients use the standard WSRF resource property query and subscription/notification interfaces to retrieve information from an Index. Indexes can register to each other in a hierarchical fashion in order to aggregate data at several levels. Indexes are "self-cleaning"; each Index entry has a lifetime and will be removed from the Index if it is not refreshed before it expires.

The *Trigger Service* collects information and compares that data against a set of conditions defined in a configuration file. When a condition is met, or triggered, an action takes place, such as emailing a system administrator when the disk space on a server reaches a threshold.

In addition to the services described above, MDS4 includes several additional software components, including an *Aggregator Framework* (The Globus Alliance, 2005b), which provides a unified mechanism used by the Index and Trigger services to collect and aggregate data.

Currently, MDS4 includes the following sources of information:

- *Hawkeye Information Provider, and Ganglia Information Provider,* which include information about basic host data (name, ID), processor information, memory size, OS name and version, file system data, processor load data and other basic cluster data.

- *WS GRAM*: The job submission service component of GT4. This WSRF service publishes information about the local scheduler, including: queue information, number of CPUs available and free, job count information and some memory statistics.

- *Reliable File Transfer Service (RFT)*: The file transfer service component of GT4. This WSRF service publishes: status data of the server, transfer status for a file or set of files, number of active transfers and some status information about the resource running the service.

- And any other WSRF service that publishes resource properties.

Finally, MDS provides a web-based interface to WSRF resource property information called *WebMDS* that is available as a user-friendly front-end to the Index Service. WebMDS uses standard resource property requests to query resource property data and displays the results in various formats.

1.1.1.7. gLite-MON: gLite Monitoring System Collector Server

The gLite Monitoring System is based on R-GMA monitoring system collector server. R-GMA (Relational Grid Monotoring Architecture) (http://www.r-gma.org/)Error: Reference source not found is an information system very similar to a standard relational database.
The R-GMA server is a Java servlet-based web application which provides a service for information, monitoring and logging in a distributed computing environment. R-GMA makes all the information appear like one large relational database that may be queried to find the information required. It consists of Producers which publish information into R-GMA, and Consumers which subscribe to the information.
In a distributed environment, such as a computing Grid, it is important to be able to find information on what resources are available. This may be information on what computers are available on various sites to run jobs, including their current load and what software they have available. Information may also be needed on mass data storage facilities, including the current status and maximum size and number of files that may be stored. It is also important to be able to monitor the progress of jobs, especially when the user will probably not know where the job is going to be executed when the job is submitted.
The Open Grid Forum (OGF) (http://www.ogf.org/) defined a basic architecture for monitoring within the Grid, called the 'Grid Monitoring Architecture' or 'GMA'. This architecture consists of three components: *Consumers*, *Producers* and a directory service (called *Registry*).

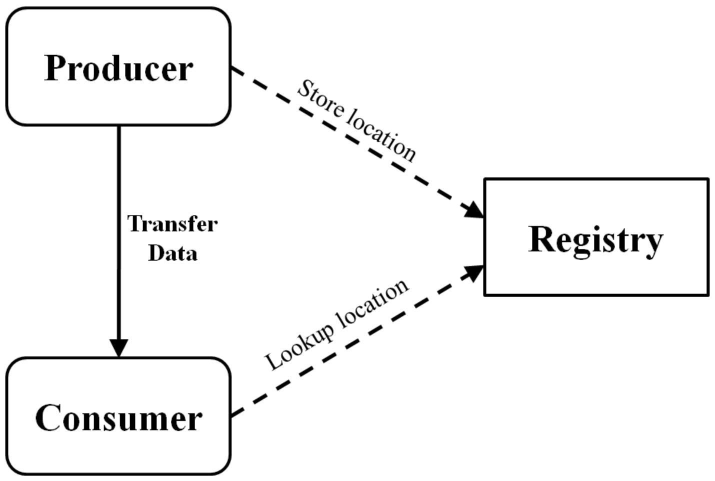

Figure. Grid Monitoring Architecture (GMA) (http://www.r-gma.org)

In the GMA Producers register themselves with the Registry and describe the type and structure of information they want to make available to the Grid. Consumers can query the Registry to find out what type of information is available and locate Producers that provide such information. Once this information is known the Consumer can contact the Producer directly to obtain the relevant data. The Registry communication is shown by a dotted line and the main flow of data by a solid line. The GMA architecture was devised for monitoring but it also makes an excellent basis for combined information and monitoring system.
The *Relational Grid monitoring Architecture* (R-GMA) is an implementation of GMA, with two special properties:

- Anyone supplying or obtaining information from R-GMA does not need to know about the Registry, the Consumer and Producer handle the registry behind the scenes.
- The Information and monitoring system appears like one large relational database, and can be queried as such. Hence the 'R' GMA, a relational implementation of GMA.

Note that R-GMA provides a way of using the relational model in a Grid environment and that it is not a general distributed RDBMS. All the producers of information are quite independent. It is relational in the sense that Producers announce what they have to publish via an SQL CREATE TABLE statement and publish with an SQL INSERT and that Consumers use an SQL SELECT to collect the information they need.
To interact with R-GMA APIs are available in various languages. Currently APIs are available in Java, C, C++, and Python.
A web browser is available, which allows users to browse the status of a Grid. And a command line tool is also available, which helps a user to understand and test out producers and consumers.
The R-GMA usage up to now has been as a Grid Information, Monitoring and Logging tool. When the user submits the job the Job Manager will need to find a suitable resource to run the job. The information System may be used to find the appropriate resource.
The user will also want to track progress of the job - R-GMA can be used to keep track of job progress. In this case the user submitting a job does not need to know about R-GMA, it is hidden from the user. Besides this, any system where a developer wishes to manage information in a distributed system can use R-GMA.
For a more detailed description of R-GMA see the R-GMA architecture (http://www.r-gma.org/arch-virtual.html).

CONCLUSION

There are currently a great number of tools for monitoring the diverse infrastructure resources and the status and performance of applications running on this infrastructure. This chapter has presented some tools that are particularly relevant for virtualized and distributed environments. The existing diversity on monitoring tools and the metrics used makes difficult the integration of monitoring systems as the infrastructure grows. To solve this problem the use of semantic tools could be a solution to allow a coherent data access providing a common understanding of the different metrics used by the different monitoring systems. It is therefore necessary to agree on a common and standardised ontology for monitoring systems. In that sense, the IP traffic Measurement for Industrial Ontology Specification Group (ISG MOI) in ETSI is working in the specification of an ontology standard for IP traffic measurements. This work should be extended to cover the other parts of the infrastructure.

REFERENCES

Amazon (2009). Amazon CloudWatch Development Guide (API Version 2009-05-15). Retrieved June 25, 2010, from http://docs.amazonwebservices.com/AmazonCloudWatch/latest/DeveloperGuide/index.html?arch-AmazonCloudWatch-metricscollected.html

Arthur, S., Emde, C. and Mc Guire, N. (2007). Assessment of the Realtime Preemption Patches (RT-Preempt) and their impact on the general purpose performance of the system , 9th Real-Time Linux Workshop, Linz, Austria

Boniface, M., Nasser, B., Papay, J., Phillips, S.C., Servin, A., Yang, X., Zlatev, Z., Gogouvitis, S. V., Katsaros, G., Konstanteli, K., Kousiouris, G., Menychtas, A. and Kyriazis, D. (2010). Platform-as-a-Service Architecture for Real-Time Quality of Service Management in Clouds, Proceedings of the Fifth International Conference on Internet and Web Applications and Services, pp. 155–160, Barcelona, Spain.

Buyya, R., Yeo, C.S. and Venugopal, S. (2009). "Cloud computing and emerging IT platforms : Vision, hype, and reality for delivering computing as the 5th utility", Future generations computer systems, vol. 25, no. 6, pp.599-616.

Checconi, F., Cucinotta, T., Faggioli, D. and Lipari,G. (2009) "Hierarchical Multiprocessor CPU Reservations for the Linux Kernel," in Proceedings of the 5th International Workshop on Operating Systems Platforms for Embedded Real-Time Applications (OSPERT 2009), Dublin, Ireland

Corbet, J. (2006). The high-resolution timer API, available at http://lwn.net/Articles/167897

"CPU Accounting Controller," available on-line at: http://www.kernel.org/doc/Documentation/cgroups/cpuacct.txt

Cucinotta, T., Anastasi, G. and Abeni, L. (2008)."Real-Time Virtual Machines," in Proceedings of the 29th Real-Time System Symposium (RTSS 2008) -- Work in Progress Session, Barcelona

Cucinotta, T., Anastasi, G. and Abeni, L. (2009). "Respecting temporal constraints in virtualised services," in Proceedings of the 2nd IEEE International Workshop on Real-Time Service-Oriented Architecture and Applications (RTSOAA 2009), Seattle, Washington

Cucinotta, T., Konstanteli, K. and Varvarigou, T. (2009). "Advance Reservations for Distributed Real-TimeWorkflows with Probabilistic Service Guarantees," in Proceedings of the IEEE International Conference on Service-Oriented Computing and Applications (SOCA 2009).

de Haaff, B. (2008). Cloud computing – the jargon is back! Computing Journal. Electronic Magazine, article available at http://cloudcomputing.sys-con.com/node/613070

Fahringer, T., Anthes, C., Arragon, A., Lipaj, A., Müller-Iden, J., Rawlings C. and Prodan, R. (2007). "The Edutain@Grid Project" In: Veit, D.J., Altmann, J. (eds.) GECON 2007. LNCS, vol. 4685, pp. 182–187. Springer, Heidelberg.

Foster, I., Kesselman, C. and Tuecke, S.(2001). The anatomy of the Grid: Enabling scalable virtual organizations. Intl. Journal of Supercomputing Applications, vol. 15, no.3, pp.200. Retrieved June 25, 2010, from http://www.globus.org/alliance/publications/papers/anatomy.pdf.

Geelan, J. (2008). "Twenty-One Experts Define Cloud Computing". Electronic magazine, available: http://cloudcomputing.sys-con.com/node/612375?page=0,1

Google. (2010). Google App Engine. Appstats for Java. Retrieved June 25, 2010, from http://code.google.com/intl/es-ES/appengine/docs/java/tools/appstats.html

Google. (2010). Using Memcache. Retrieved June 25, 2010, from http://code.google.com/intl/en/appengine/docs/python/memcache/usingmemcache.html

Graham, S., Tibco D. and Murray, B.(2006). Web Services Base Notification 1.3 (WS-BaseNotification). Retrieved June 25, 2010, from http://docs.oasis-open.org/wsn/wsn-ws_base_notification-1.3-spec-os.pdf

Hand, E. (2007). “Head in the Clouds”, Nature,(449):963.
Heroix. (2006). “The Best Practices Guide to Developing and Monitoring SLAs” Whitepaper.

IEEE Standard for Information Technology – Portable Operating System Interface (POSIX). Available online at the URL: http://www.opengroup.org/onlinepubs/009695399

Inácio N. et al. (2005). D4.1.1. “Mobile Network Architecture, Design & Implementation”. Retrieved June 25, 2010, from http://www.akogrimo.org/download/Deliverables/D4.1.1.pdf

Kostanteli, K., Kyriazis, D., Varvarigou, T., Cucinotta, T. and Anastasi, G. (2009). "Real-time guarantees in flexible advance reservations," in Proceedings of the 2nd IEEE International Workshop on Real-Time Service-Oriented Architecture and Applications (RTSOAA 2009), Seattle, Washington.

KVM: Kernel-based Virtualization Driver, Whitepaper, Qumranet 2006, available at the URL: http://docs.huihoo.com/kvm/kvm-whitepaper.pdf

Litke, A. et al. (2005). D4.3.1 “Architecture of the Infrastructure Services Layer V1”. Akogrimo consortium. Retrieved June 25, 2010, from http://www.akogrimo.org/modulese73d.pdf?name=UpDownload&req=getit&lid=37

Litke, A. et al. (2007). D4.3.4 “Consolidated Report on the Implementation of the Infrastructure Services Layer Version 1.0”. Akogrimo consortium. Retrieved June 25, 2010, from http://www.akogrimo.org/modulesa3f9.pdf?name=UpDownload&req=getit&lid=121

Machiraju, S. (2006). Theory and Practice of Non-Intrusive Active Network Measurements. Doctoral Thesis. UMI Order Number: AAI3228413., University of California at Berkeley.

Manica, N., Abeni, L. and Palopoli, L. (2010). Reservation-Based Interrupt Scheduling, to appear in Proceedings of the 16th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS 2010), April 2010, Stockholm, Sweden

McFedries, P. (2008). The cloud is the computer. IEEE Spectrum Online. Electronic Magazine, available at http://www.spectrum.ieee.org/aug08/6490

Menage, P. (n.d.). CGROUPS, Official Linux Kernel Documentation, available in current kernel distribution archive as Documentation/cgroups/cgroups.txt, also available at the URL: http://git.kernel.org/?p=linux/kernel/git/torvalds/linux-2.6.git

Middleton, S.E. Surridge, M. Nasser and B.I. Yang, X. (2009). "Bipartite electronic SLA as a business framework to support cross-organization load management of real-time online applications", Real Time Online Interactive Applications on the Grid (ROIA 2009), Euro-Par 2009

OASIS Web Services Notification (WSN) TC website. Retrieved June 25, 2010, from http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsn

Oberle, K., Kessler, M., Stein, M., Voith, T., Lamp, D. and Berger, S. (2009). “Network Virtualization: The missing piece”, 13th International Conference on Intelligence in Next Generation Networks, 2009.

ICIN 2009, pp. 1-6. Available at http://irmosproject.eu/Files/ICIN2009_FinalVersion_NetworkVirtualization.pdf

Palopoli, L., Cucinotta, T., Marzario, L. and Lipari, G. (2009). “AQuoSA - Adaptive Quality of Service Architecture” Software: Practice and Experience, Vol. 39, No. 1, pp. 1–31, doi 10.1002/spe.883

Paxson, V., Almes, G., Mahdavi, J. and Mathis, M. (1998). RFC2330 - Framework for IP Performance Metrics.

Ploß, A., Glinka, F. and Gorlatch, S. (2009). "A case study on using RTF for developing multi-player online games". Euro-Par 2008 Workshops-Parallel Processing. pp. 390–400, Springer.

Schubert, L., Jeffery, K., Neidecker-Lutz, B. et al. (2010). "The future of Cloud Computing. Opportunities for European Cloud Computing beyond 2010”. Retrieved June 25, 2010, from http://cordis.europa.eu/fp7/ict/ssai/docs/cloud-report-final.pdf

The Globus Alliance. (2005). GT 4.0 WS MDS Trigger Service, Retrieved June 25, 2010, from http://www.globus.org/toolkit/docs/4.0/info/trigger/

The Globus Alliance. (2005). GT 4.0: Information Services: Aggregator Framework. Retrieved June 25, 2010, from http://www.globus.org/toolkit/docs/4.0/info/aggregator/

The Globus Alliance. (2005). GT 4.0: Information Services: Index. Retrieved June 25, 2010, from http://www.globus.org/toolkit/docs/4.0/info/index/index.pdf

The Measurement Ontology for IP traffic Industrial Specification Group, MOI ISG website, http://portal.etsi.org/portal/server.pt/community/MOI

Tsafrir, D., Etsion, Y. and Feitelson, D. (2007). Secretly monopolizing the CPU without superuser privileges, in Proceedings of the 16th USENIX Security Symposium, (Boston, MA).

van Rossum, G. (2010) Appstats - RPC instrumentation and optimizations for App Engine, May 2010, Google I/O 2010 event. Video and presentation available at http://code.google.com/intl/es-ES/events/io/2010/sessions/appstatsrpc-appengine.html

Vaquero, L.M., Rodero-Merino, L., Caceres, J. and Lindner, M. (2009). “A Break in the Clouds: Towards a Cloud Definition”, ACM SIGCOMM Computer Communication Review, Volume 39 , Issue 1, pp. 50-55.

Wang, L. et al., (2008). “Scientific Cloud Computing: Early Definition and Experience”, 10th IEEE International Conference on High Performance Computing and Communications, 2008. HPCC '08. 25-27 Sept. 2008 Page(s):825 – 830.

Watson, P., Lord, P., Gibson, F. et al. (2008). “Cloud Computing for e-science with CARMEN”, Proceedings of IBERGRID Conference, pp. 1-5, Porto (Portugal). May 12 – 14.

Wesner, S., Järnert, J. M. and Aránzazu, M. (n.d.). “Mobile Collaborative Business Grids – A short overview of the Akogrimo Project”. Retrieved June 25, 2010, from http://www.akogrimo.org/download/White_Papers_and_Publications/Akogrimo_WhitePaper_Overview.pdf

[i] SPI from SaaS, PaaS and IaaS
[ii] Authentication, Authorization, Accounting, Auditing and Charging