J Grid Computing (2016) 14:283–297
DOI 10.1007/s10723-015-9357-4

IaaSMon: Monitoring Architecture for Public Cloud Computing Data Centers

Juan Gutierrez-Aguado · Jose M. Alcaraz Calero · Wladimiro Diaz Villanueva

Received: 2 June 2014 / Accepted: 23 November 2015 / Published online: 4 March 2016

Abstract Monitoring of cloud computing infrastructures is an imperative necessity for cloud providers and administrators to analyze, optimize and discover what is happening in their own infrastructures. Current monitoring solutions do not fit well for this purpose mainly due to the incredible set of new requirements imposed by the particular requirements associated to cloud infrastructures. This paper describes in detail the main reasons why current monitoring solutions do not work well. Also, it provides an innovative monitoring architecture that enables the monitoring of the physical and virtual machines available within a cloud infrastructure in a non-invasive and transparent way making it suitable not only for private cloud computing but also for public cloud computing infrastructures. This architecture has been validated by means of a prototype integrating an existing enterprise-class monitoring solution, Nagios, with the control and data planes of OpenStack, a well-known stack for cloud infrastructures. As a result, our new monitoring architecture is able to extend the exiting Nagios functionalities to fit in the monitoring of cloud infrastructures. The proposed architecture has been designed, implemented and released as open source to the scientific community. The proposal has also been empirically validated in a production-level cloud computing infrastructure running a test bed with up to 128 VMs where overhead and responsiveness has been carefully analyzed.

J. Gutierrez-Aguado · W. Diaz Villanueva
Departament d'Informàtica, Universitat de València, Avda. De la Universitat, s/n 46100 Burjassot, Valencia, Spain

J. Gutierrez-Aguado
e-mail: juan.gutierrez@uv.es

W. Diaz Villanueva
e-mail: wladimiro.diaz@uv.es

J. M. Alcaraz Calero (✉)
School of Engineering and Computing, University of the West of Scotland, Paisley Campus, Paisley, PA1 2BE, Scotland
e-mail: jose.alcaraz-calero@uws.ac.uk

Keywords Cloud computing · Monitoring · Distributed monitoring · Network management · Infrastructure-as-a-service

1 Introduction

Cloud computing is changing radically the way in which businesses, governments, researches and consumers are using computational power. Cloud Computing enables them to make better use of their own computational resources (private cloud) and to rent computational resources to third-parties on-demand (public cloud) to satisfy their constantly changing computational requirements. Public cloud infrastructures are associated to scenarios where cloud users do not have any control over the management of the physical topology of the infrastructures where they are renting virtual machines (aka VMs), i.e. virtual topologies.

However, users have control over their rented VMs and what services are being executed therein.

From the point of view of the cloud provider, it is imperative to analyze, optimize and discover what is happening in their entire infrastructure. To do so, monitoring tools are essential to identify anomalies, analyze behavior, optimize infrastructure resources, provide feedback to consumer, check infrastructure healthy, perform auto-scaling of resources [11], allocate resources in multi-clouds [9, 15], or monitor Service Level Agreement [3]. The vast majority of monitoring tools available in the market do not fit for the purpose of monitoring cloud infrastructures. On the one hand, their architectural design usually imposes the installation of a software agent in the resources to be monitored to extract metrics. However, cloud users are especially reluctant to run any kind of services/software in their rented VMs for providing information to third parties so that this requirement is simply not acceptable in cloud environments. On the other hand, current monitoring tools do not deal efficiently with the new life-cycle associated to virtual topologies. For example, current monitoring tools assume that monitored resources will remain always with the same IP address. Then, if such address is not responding, it is assumed that resources are shutdown or failing. However, this is not true at all in cloud infrastructures where IP addresses are being highly reused and dynamically assigned to different VMs in a matter of seconds. Traditional monitoring solutions will not realize of this re-use of IP addresses and will consider the monitored resource always to be same one even when they are now monitoring a completely different resource. These facts make difficult for cloud providers to implement effective monitoring solutions for their infrastructure and it is required the design of novel non-intrusive monitoring solutions running in a transparent way from the point of view of the cloud users whereas they provide accurate information for the cloud provider. This is exactly the main motivation of this research work. Our contribution is, to the best of our knowledge, the first attempt to integrate both the control and data planes of a cloud computing infrastructure with an existing monitoring tool to fit in the monitoring of the completely new life-cycle associated to virtual cloud infrastructures. This novel integration provides an effective monitoring solution for public cloud infrastructures allowing a transparent monitoring of the customer s VMs, and at same time, an agent-based monitoring of the rest of the resources such as physical machines, hard disks, etc. The following list of requirements summarizes the key and unique features and requirements that make the monitoring of public cloud computing infrastructures a challenge. Some of these items are already available in almost all the current monitoring solutions; however, some others are simply not supported nowadays:

- R1. To perform a transparent monitoring of VMs (no tools installed in the customer s VM)
- R2. To perform an agent-based monitoring of the management VMs
- R3. To perform an agent-based monitoring of the physical machines
- R4. To enable correlation between metrics from VMs and physical machines where they are allocated.
- R5. To quickly adapt against frequent changes in the virtual topology using efficient auto-discovery protocols.
- R6. To quickly adapt against IP address reassignation.
- R7. To quickly adapt against changes in VM state (VM life-cycle).
- R8. To integrate the monitoring architecture with the management plane of the cloud stack to keep continuously synchronized the status of the cloud stack and the monitoring tool.
- R9. To integrate the monitoring architecture with the data plane of the cloud stack to keep continuously synchronized the status of the VMs and the monitoring tool.
- R10. To be high scalable, suitable for monitoring large amount of resources efficiently.

More than 50 different monitoring solutions have been analyzed in this work. None of them meet all these requirements simultaneous. It is not our purpose to provide a completely new monitoring framework designed from scratch. An analysis of existing monitoring tools has been done (lately explained in detail in Section 2) to select a good candidate to be extended and adapted to fulfill all these requirements and thus making it suitable for the monitoring of cloud infrastructures. According to a recent study performed by

Dataloop,[1] Nagios is the dominant monitoring tool for cloud infrastructures in the market even if it has not yet completely being adapted for such purpose. Nagios has been selected as a base monitoring software to be extended in this research work due to its flexibility, world-wide acceptance, suitability for large-scale deployments, due to the incredibly large number of extensions available and specially due to the fact that auto discovery algorithms of new resources is completely customizable.

The proposed architecture described in this contribution provides support for all the above list of requirements. To achieve it, the architecture is based on the integration between the monitoring tool, and its resource discovery protocol, and the control and data planes of the cloud computing infrastructure. This integration is in fact our main contribution. The architecture has been prototypically implemented by means of the integration between OpenStack, a well-known enterprise-class cloud computing stack used for both private and public cloud computing infrastructure and Nagios, an enterprise-class distributed monitoring tool. When users interact with OpenStack, a series of messages are interchanged between the different modules available in the cloud stack. Our architecture intercepts all these messages and extracts topology information to be lately integrated in Nagios. The proposal has been implemented and released as open source project to the community. It has also been validated in a production-level cloud computing infrastructure running a mid-size test bed with 128 VMs.

To describe the contribution, this paper has been organized as follows: Section 2 provides an updated state-of-the-art in monitoring solutions for cloud infrastructures. After that, Section 3 shortly describes the basic concepts about the different components involved in the cloud computing infrastructure to have a self-contained contribution. Then, Section 4 provides an overview of a traditional high scalable distributed monitoring solution to enable the reader to realize what the changes are required to fit in cloud computing infrastructures. After that, Section 5 is focused on the main contribution, explaining the design of the monitoring architecture proposed. Then, Section 6 provides implementation details of the proposed architecture. Section 7 describes the different test beds carried out and the intensive testing done in order to validate both the architecture and prototype presented. And finally, Section 8 describes some conclusions about this contribution.

2 Related Works

Open source monitoring solutions for cloud infrastructures are really scarce and only a few proofs of concept are available. Brandt et al. [2] provide OVIS, a distributed monitoring infrastructure for High Performance Computing (HPC). Wuhib and Stadler [18] also provide a distributed monitoring framework for large cloud infrastructures. Kertesz [8] provides an architecture where different cloud providers collect information to decide where to allocate new resources in a federated environment. Dhingra et al. [6] also provide a solution for monitoring resources in cloud infrastructures. Although these works are good attempts, they cannot really fit in public cloud computing infrastructures because they impose the installation of software in the customer s VMs. This is not acceptable in production-level scenarios. Would you enable other people to install software in your machine?

Other approaches such as Moses et al. [14], Romano et at. [17], Massonet et at. [12] and Koenig et al. [10] are designed to monitor specific metrics to provide an added value in the cloud infrastructure like efficient VM migrations, QoS, Data Location and cross-layer monitoring, respectively. Although they are good contributions to the state-of-the-art, they are optimized for specific purposes and do not provide support for: i) monitoring of the virtualization layer; ii) monitoring for large-scale infrastructures; iii) transparent monitoring from the customer s perspective; iv) general purpose and extensible monitoring.

An alternative is to take traditional network monitoring tool for IT infrastructures and to adapt them to the new requirements imposed by the public cloud computing environment. Nagios,[2] Icinga,[3] Zennoss,[4]

[1]What we learnt talking to 60 companies about monitoring, Information available at http://blog.dataloop.io/2014/01/30/what-we-learnt-talking-to-60-companies-about-monitoring/

[2]http://www.nagios.org/

[3]https://www.icinga.org/

[4]http://www.zenoss.com/

Zabbix[5] and OpenNMS[6] are good examples of traditional free tools whereas Nimsoft Monitoring Solution[7] and LiveAction[8] are good examples of commercial ones. From more than 50 tools have been analyzed. Then, those that impose the use of monitoring agents like, for example, Pandora FMS, Ganglia, and XyMon have been discarded due to the fact that it is required a transparent monitoring approach for customer s VMs. It has also been discarded commercial solutions and those that do not provide source code required to enable the integration and/or extension of the monitoring tools to fit in cloud infrastructures. As a result, the five free tools previous indicated were identified as the best candidates. None of these tools are suitable for monitoring cloud infrastructure mainly due to the fact that they do not detect: frequent changes in the topology of the virtual infrastructure, IP address re-assignation to different resources and destruction of virtual resources. Concretely, Zabbix, Zenoss and OpenNMS come with auto-discovery protocols that are not suitable for virtual topologies for the same reasons previous described. The proposed architecture described in this contribution is based on the integration of the control and data planes of the cloud infrastructure providing a new auto-discover protocol. This fact requires the selection of a monitoring architecture that enables the customization of such protocol and thus these three solutions were discarded. Between Icinga and Nagios, Nagios seems to be the better candidate due to its incredible dominance in the market and due to the fact that it has been proven to be suitable for monitoring very large enterprise-class infrastructures. This is the main reason why we have seleceted Nagios.

Aceto et al. [1] have recently provided a complete survey about monitoring architectures for cloud computing infrastructures. This survey describes a number of commercial solutions like CloudWatch,[9] AzureWatch,[10] CloudKick,[11] and CloudStatus,[12] to name a few. However, these commercial vendors have not published how they implement internally their monitoring architecture. In fact, the main intention of this paper is to describe an open-source prototype which possibly provides most of the services available in such commercial products. In this survey, it is also described several open-source and commercial downloadable monitoring architectures like OpenNebula Monitoring Subsystem [13], Nimbus Project,[13] CloudStack ZenPack,[14] Hyperic-HQ,[15] PCMONS [5], Sensu,[16] Nagios and Dargos [16].

Dargos [16] comes with a small set of built-in sensors so that this is a clear disadvantage when compared with the available number of plugins already developed for Nagios. Extensibility is a property that monitoring systems should fulfil to be suitable for cloud infrastructures. There are some approximations in the literature that follows the approach of adapting Nagios to fit in cloud infrastructures. Aparecida de Chaves et al. [5] developed an architecture designed for running in private clouds so that the solution does not fulfill the requirements indicated in Section 1. For instance, they install scripts into the VMs leading to a valid solution only for private clouds where all the VMs belong to the same organization. However, this is not admissible in public clouds where VMs are belonging to customers. G. Katsaros et al. [7] proposed a component for monitoring cloud infrastructure that is notified through the Nagios Event Broker API. This component exposes the data received from Nagios through a REST service. The idea of good, however, authors do not described how Nagios is adapted to reflect the allocation/ de-allocation of VMs and IP addresses which is one of the main adaptions required to fit in cloud infrastructures. Also, there is not any empirical validation of the proposed architecture so that it is impossible to validate and reproduce it. Finally, M. Barbosa et al. [4] proposed a set of Nagios plugins to make Nagios aware of VMs. They proposed two strategies to discover VMs: an active check where a plugin is installed in each physical machine and then Nagios execute them periodically using NRPE;[17] and, a passive checks in which each physical

[5]http://www.zabbix.com/

[6]http://www.opennms.org/

[7]http://www.nimsoft.com/solutions/nimsoft-monitor.html

[8]http://www.actionpacked.com/products/qos-monitor

[9]http://aws.amazon.com/es/cloudwatch/

[10]http://www.paraleap.com/azurewatch

[11]http://www.rackspace.com/cloudkick/

[12]https://status.rackspace.com/

[13]http://www.nimbusproject.org/downloads/

[14]http://wiki.zenoss.org/ZenPack:CloudStack

[15]http://www.hyperic.com/downloads

[16]http://sensuapp.org/

[17]http://exchange.nagios.org/directory/ Addons/Monitoring-Agents/NRPE–2D-Nagios-Remote-Plugin-Executor/details

machine runs a cron task that informs Nagios server through NSCA.[18] Their solution allows also the mapping between physical and virtual machines using the Nagios Event Broker. This solution involves a significant generation of network traffic during VM discovery phase. Also, it makes Nagios aware of new VMs and any topology changes with a significant delay determined by the time interval in active checks or the cron task interval in passive checks which is a clear trade-off with respect to the network traffic generated. In our solution, VMs are registered in Nagios even before they can be reached by ICMP ping and no additional network traffic is generated in the VM discovery phase which is a clear advantage and differentiating point.

Ceilometer[19] is an OpenStack optional module that provides services to collect measurements of the utilization of the physical and virtual resources deployed in the clouds infrastructure. Although, this module has been designed for billing purposes, it could be considered as a monitoring tool suitable for cloud infrastructures. Ceilometer is integrated in the control plane of the cloud infrastructure. It can retrieve a number of metrics directly form the physical machines. It also can extract metrics from the control plane about the virtual machines by interrogating the hypervisor and thus achieving a transparent monitoring. However, Ceilometer has not been designed as monitoring tool and thus, it has not been integrated with the data plane of the cloud infrastructure. In consequence, it does not have real connectivity with the VMs available in the infrastructure. This fact limits significantly the number of metrics Ceilometer can extract from the VMs. Our proposed architecture goes a step forward in the state of the art achieving a complete integration with both control and data planes and then achieving a wide range of metrics gathered directly from the VMs using agent-less monitoring approach. This is a differentiating aspect of our proposed architecture.

Table 1 shows a complete analysis of the requirements that a monitoring tool should fulfill to be suitable for cloud infrastructures (previously listed in Section 1) compared against all the relevant related works indicated in this section and how the proposed architecture is going a step forward in the state of the art. As the reader can see, IaaSMon is the only one simultaneously providing support for all the requirements identified.

3 Cloud Computing Architecture

Figure 1shows the basic components available in a cloud computing stack for managing cloud infrastructures. Each component in this figure has in parenthesis the name of such service inside OpenStack, this information clarifies the correspondence between design and implementation. In summary, there is a graphical interface where users can ask for new computational resources (see 1 in Fig. 1). These requests are received by the cloud API (see 2 in Fig. 1) which inserts the request into the communication middleware (see 3 in Fig. 1). This middleware interconnects all the modules available in the cloud stack. Depending on the type of request, the messages will travel along different modules in order to achieve its original goal. There are modules for: managing volume storage (see 4 in Fig. 1) deployed in specialized storage computer; for registering new physical machines (see 5 in Fig. 1) that lately are used for placing VMs; for managing the networking of VMs (see 6 in Fig. 1); for managing users and authentication schemes (see 7 in Fig. 1); and finally, there are also services for deciding where to place new VMs (see 8 in Fig. 1) by means of scheduling algorithms. We refer the reader to OpenStack documentation for further details about such modules.

Cloud computing systems are designed to work in highly distributed systems so that the communication middleware may provide approaches for asynchronously decoupling the different modules of the architecture. Generally, this decoupling is achieved using message buses based on queues used to send/receive messages to/from the different modules. These modules are distributed in the physical machines where the infrastructure is deployed. Besides, the physical machines run a virtualization layer that manages VMs. The complete overview of this virtualization stack can be seen in the lower-right side of Fig. 1.

[18] http://exchange.nagios.org/directory/Addons/Passive-Checks/NSCA--2D-Nagios-Service-Check-Acceptor/details

[19] https://wiki.openstack.org/wiki/celiometer

Table 1 Comparative Analysis of current monitoring tools against the proposed architecture

Monitoring Platform	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10
Brandt et al. [2]	×	✓	✓	×	×	×	×	×	×	✓
Wuhib and Stadler [18]	×	✓	✓	×	×	×	✓	×	×	✓
Kertesz et al. [8]	×	✓	✓	×	×	×	✓	×	×	✓
Dhingra et al. [6]	×	✓	✓	×	×	×	✓	×	×	✓
PCMONS [5]	✓	✓	✓	✓	×	×	✓	×	✓	✓
Nagios	✓	✓	✓	✓	×	×	×	×	×	✓
Ganglia	×	✓	✓	×	×	×	×	×	×	✓
DARGOS [16]	×	✓	✓	✓	✓	×	✓	×	✓	✓
Zabbix	✓	✓	✓	✓	×	×	×	×	×	✓
Sensu	×	✓	✓	✓	✓	×	×	×	✓	✓
Hiperic-HQ	×	✓	✓	×	✓	×	×	×	×	✓
Celiometer	✓c	×	✓	✓	✓	✓	✓	✓	×	✓
Azure Watch	✓c	×	✓	×	✓	✓	✓	✓	×	✓
CloudKick	✓c	×	✓	×	✓	✓	✓	✓	×	✓
IaasMon	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

The title of the columns matches with the IDs available in the list of requirements listed in Section 1. ✓* means that only a set of metrics are directly provided by the hypervisor

4 Traditional Monitoring Architecture

The lower-left part of Fig. 1 depicts the conceptual architecture of enterprise-class distributed monitoring. In essence, there is a core service in charge of monitoring capabilities. This core is divided in, at least, 4 different modules. *Notification module* notifies admins when some monitored success occurs (see 9 in Fig. 1); *Event module* processes actions when a monitoring alert occurs (see 10 in Fig. 1); *Performance module* controls the performance of the monitoring software balancing the load between peers (see 11 in Fig. 1); And, *monitoring module* carrying out the monitoring of machines (see 12 in Fig. 1). These modules generate the database that represents the current status of all the monitored machines and services (*status file*) and the log files with all actions done internally in the monitoring architecture (*log files*). A graphical interface shows graphically the current status of all the monitored machines and services using the information available in such databases.

Metrics are gathered by a set of plugins. These plugins gather locally and/or remotely metrics from the different components of the infrastructure. In turn, the remote monitoring of infrastructures and services can be done following an active or passive monitoring, usually associated to push and pull methods, respectively. Also, it can be done using either an agent-based or agent-less approach. Agent-based approach relies on the installation of agent software inside the monitored components in charge of gathering the metrics in the device and sending them back to the monitoring architecture. Agent-less approach relies on different techniques to gather metrics without the need of installing an agent in the monitored component which in turn leads to transparent monitoring approaches. Some examples of agentless monitoring can be: i) the scanning of a particular port in order to determine if the service is responding correctly; ii) the usage of ICMP echo request to get reachability and round trip time metrics; iii) The usage of plugins in the hypervisor of VMs to gather performance information about the VMs.

The monitoring software uses the *config files* to determine what machines and services have to be monitored. These files also specify how to gather these metrics. The *Configuration Management Interface*, NConf[20] for Nagios, enables the dynamic definition of the resources to be monitored by means of an API interface.

[20]http://nconf.org/dokuwiki/doku.php?id=main

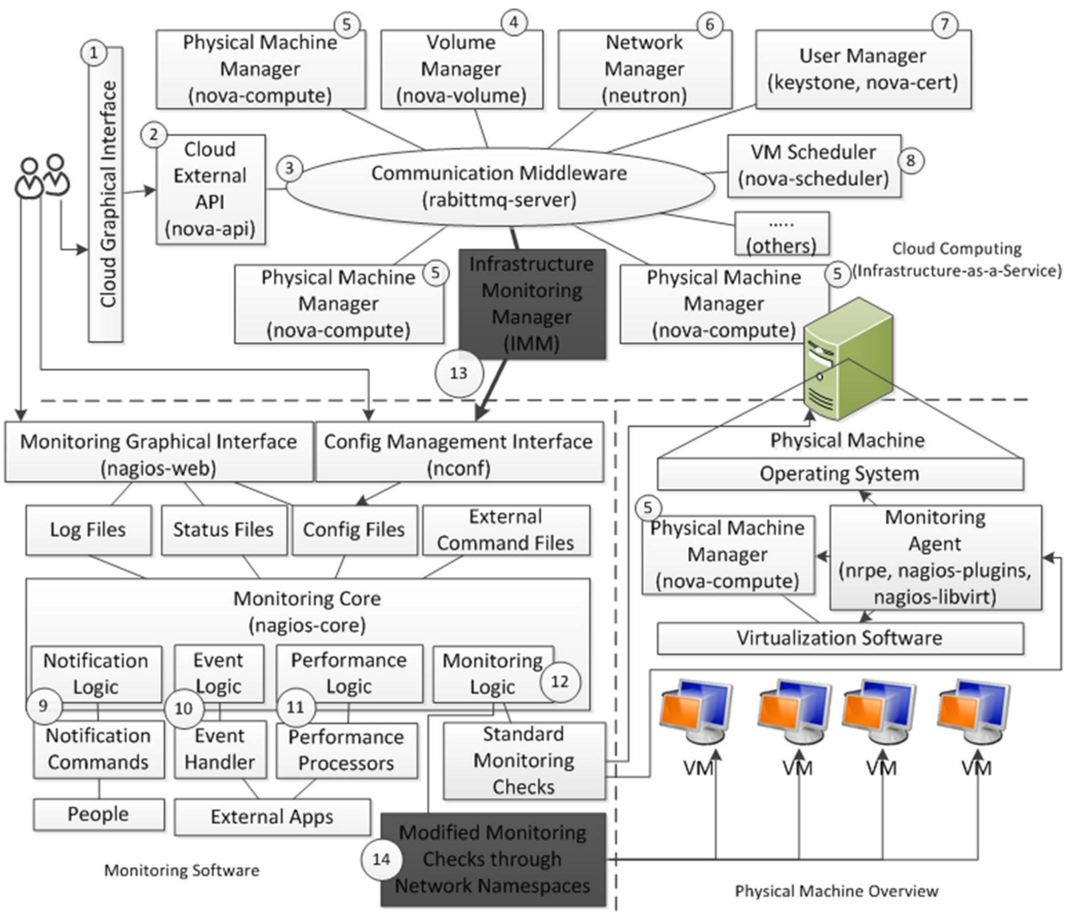

Cloud Graphical Interface
Physical Machine Manager (nova-compute)
Volume Manager (nova-volume)
Network Manager (neutron)
User Manager (keystone, nova-cert)
Cloud External API (nova-api)
Communication Middleware (rabittmq-server)
VM Scheduler (nova-scheduler)
..... (others)
Cloud Computing (Infrastructure-as-a-Service)
Infrastructure Monitoring Manager (IMM)
Monitoring Graphical Interface (nagios-web)
Config Management Interface (nconf)
Log Files
Status Files
Config Files
External Command Files
Monitoring Core (nagios-core)
Notification Logic
Event Logic
Performance Logic
Monitoring Logic
Notification Commands
Event Handler
Performance Processors
Standard Monitoring Checks
People
External Apps
Modified Monitoring Checks through Network Namespaces
Monitoring Software
Physical Machine
Operating System
Monitoring Agent (nrpe, nagios-plugins, nagios-libvirt)
Virtualization Software
VM
Physical Machine Overview

Fig. 1 Proposed Monitoring Architecture for Cloud Computing

5 Proposed Monitoring Architecture for Cloud Computing

The architecture proposed in this paper integrates Nagios with the control and data planes of OpenStack. This integration is achieved by means of the insertion of new architectural components to enable such integration. On the one hand, the *Infrastructure Monitoring Manager* (*IMM*), highlighted in different color in Fig. 1 (see 13 Fig. 1), is the first innovation and clear differentiating point of our architecture with respect to other ones.

As can be seen in Fig. 1, *IMM* is attached to the communication middleware of OpenStack so it receives the messages interchanged in the control plane between the components of the cloud infrastructure. As soon as *IMM* captures the messages from the communication middleware, they are processed producing a set of actions to dynamically adapt the configuration of the monitoring architecture with the up-to-date information. Notice that the messages can be processed by the *IMM* concurrently with the processing done by the associated modules in OpenStack. This simultaneous processing reduces significantly the overhead time of the monitoring architecture and increases substantially its responsiveness. This message interception enables *IMM* to capture changes in the topology, changes in the IP re-assignation, changes in the VM states, as they are happening i.e. close to real time. Table 2 shows a detailed description of the different messages intercepted and the associated actions design to be performed in the new *IMM* architectural component.

IMM receives messages when new VMs are created (#1-#4 in Table 2) or deleted (#5 in Table 2) and also when new physical machines are inserted or removed in the data center (#6 in Table 2). When these messages are received in the *IMM*, it immediately

Table 2 Messages intercepted by the Infrastructure Monitoring Manager (IMM) and the action done accordingly

#	Message Type/ Name in OpenStack	Destination	Description	Action to be performed in IMM
1	Run VM (`run_instance`)	Scheduler	This message ask the scheduler for the best place to allocate a VM	
2	Run VM (`run_instance`)	Compute	This message ask the physical machines to allocate/create a VM	Get user, Tenant, and VM name and store them
3	Allocate VM (`allocate_for_instance`)	Multiple	This message ask for resources for a VM (storage, networking)	Get IP and MAC of the VM and store them.
4	Get New Instance Info (`get_instace_nw_info`)	Multiple	This message provides status information about the VMs	The VM is registered in the monitoring architecture using the information previous stored about it.
5	Deallocate VM (`deallocate_for_instance`)	Multiple	This message ask for destroying a VM	The VM is unregistered in the monitoring architecture and IP address is released
6	Update Physical Machine (`update_service_capabilities`)	Multiple	This message is a heartbeat to update the status of physical machines	The Physical Machines is registered and/or updated in the monitoring architecture

registers the new changes in the monitoring software by means of invocations to the *Configuration Management Interface* of the monitoring software. So, *IMM* acts as a highly scalable auto-discovery process for both physical and virtual topologies which fits perfectly in public cloud computing. It enables to perform the monitoring of all the physical and virtual machines in the cloud infrastructure without any manual configuration of the monitoring architecture, detecting quickly any change in the virtual topology, any re-assignation of IP addresses and any change in the VM states and also it will keep synchronized the status of the complete cloud stack and the monitoring tool. This integration provides an effective coupling of the control plane of the monitoring framework and cloud infrastructure.

However, the integration needs to be done in the data plane as well. Concretely, the monitoring tool needs to have also connectively in the data plane to enable the communication with all the VMs and physical machines of the cloud infrastructure. The reader may be aware that communications between VMs are isolated between them, using different VLANs, different IP ranges, etc. Then, only VMs belong to the same cloud user can communicated between them. This is a challenge due to the fact that the monitoring tool needs to interact in a secure way with all the VMs of the cloud infrastructure (with different IP ranges) in a transparent way. To achieve this integration in the data plane, it has been provided another novel architectural component, the *Cloud monitoring Checks* (*CMC*) (see 14 Fig. 1). *CMC* enables the monitoring tool to communicate with any VMs of the cloud infrastructure regardless its location and/or owner. OpenStack networking (Neutron) makes use of Linux namespaces to create different communication networks and secure boundaries between all the VMs associated to a particular cloud user and to isolate these networks between them. So, the *CMC* enables the monitoring tool to communicate with any Linux namespace achieving communication with all the VMS to be monitored. The novel *CMC* module is another clear differentiating point of our contribution and enables the monitoring tool to have networking admin privileges to monitor efficiently all the machines of the architecture. We have defined two types of Nagios checks: those that collect metrics from physical machines (performing the checking through the control plane), and those that collect metrics from virtual machines (performing the checking through the data plane - using the namespace of the network where the VM are located).

In scenarios where there are different types of resources (physical and virtual machines) to be monitored, it is important to be able to define hierarchical groups of resources to display and manage them as a whole, for example, assigning a monitoring metric to a whole group in order to make easier the management of the monitoring architecture. Figure 2 represents the hierarchical structure proposed for the monitoring of public cloud infrastructures and services. Figure 2 represents *all the machines* in the top of the tree. These machines are subdivided in three logical groups, representing physical machines of the infrastructure (see 3 in Fig. 2), virtual machines used internally for the own cloud infrastructure for specific purposes, usually referred as Management VMs (see 4 in Fig. 2), and also customer s VMs (see 2 in Fig. 2) belonging to the different customers (indistinctly referred as tenants) (see 1 in Fig. 2). White nodes in Fig. 2 represent those automatically generated and managed by *IMM* as part of the dynamic auto-discovery process and grey node are statically created as part of the bootstrapping process of the proposed architecture. This division enables us to assign different metrics to each logical group. For example, the `ping` service shown in Fig. 2 and attached to *all machines* group means that all the machines will be pinged to know their status whereas the `free disk` service shown in Fig. 2 is also attached to Management VMs. Thus, the hierarchical groups of resources that are under the complete control of the public cloud provider can be exhaustively monitored using an agent-based approach since installing software inside of such resources is not a problem. Also, customer s VMs can be monitored using only and exclusively transparent and non-intrusive techniques like ICMP connections, TCP/UDP connections, metrics extracted directly from the hypervisor, etc.

When the *IMM* captures a message from the message bus, it recognizes the type of resource to be monitored from the type of message being intercepted being able to differentiate between physical and virtual machines. This enables to differentiate types of resources. Also, management VMs are identifies due to the usage of special and reserved names. Thus, the name of VMs is used to discriminate between customer s VMs and management VMs. With this information it is possible to insert the identified machine in the associated logical group of the hierarchy. For the sake of simplicity, Fig. 2 only shows a couple of monitored metrics per each group, however, the monitoring architecture manages hundreds of them enabling a complete customization and management of the logical groups. It is worthy to mention that the way in which *IMM* dynamically assign monitored resourced with in the proposed tree available in Fig. 2 is also another innovation of the proposed architecture.

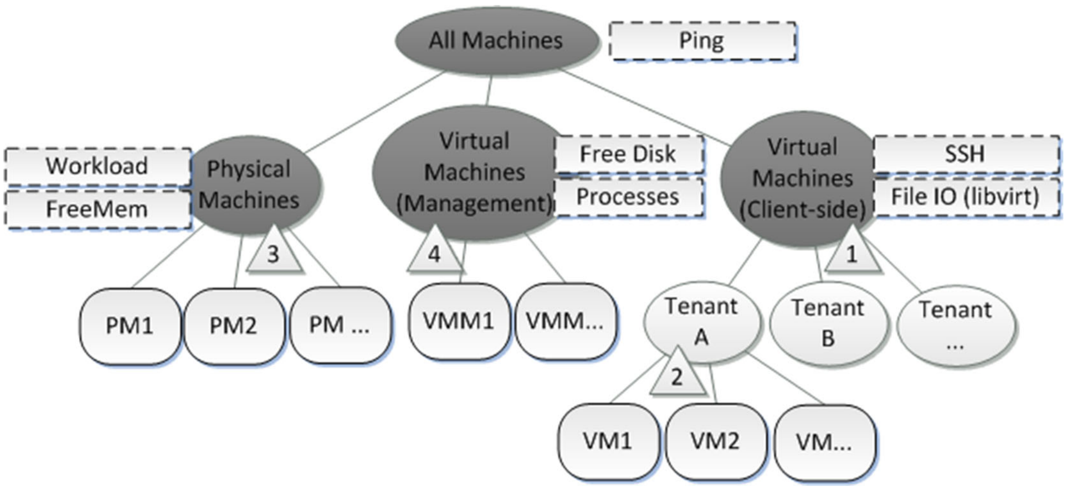


Fig. 2 Logical Grouping for Monitoring Cloud Computing Machines

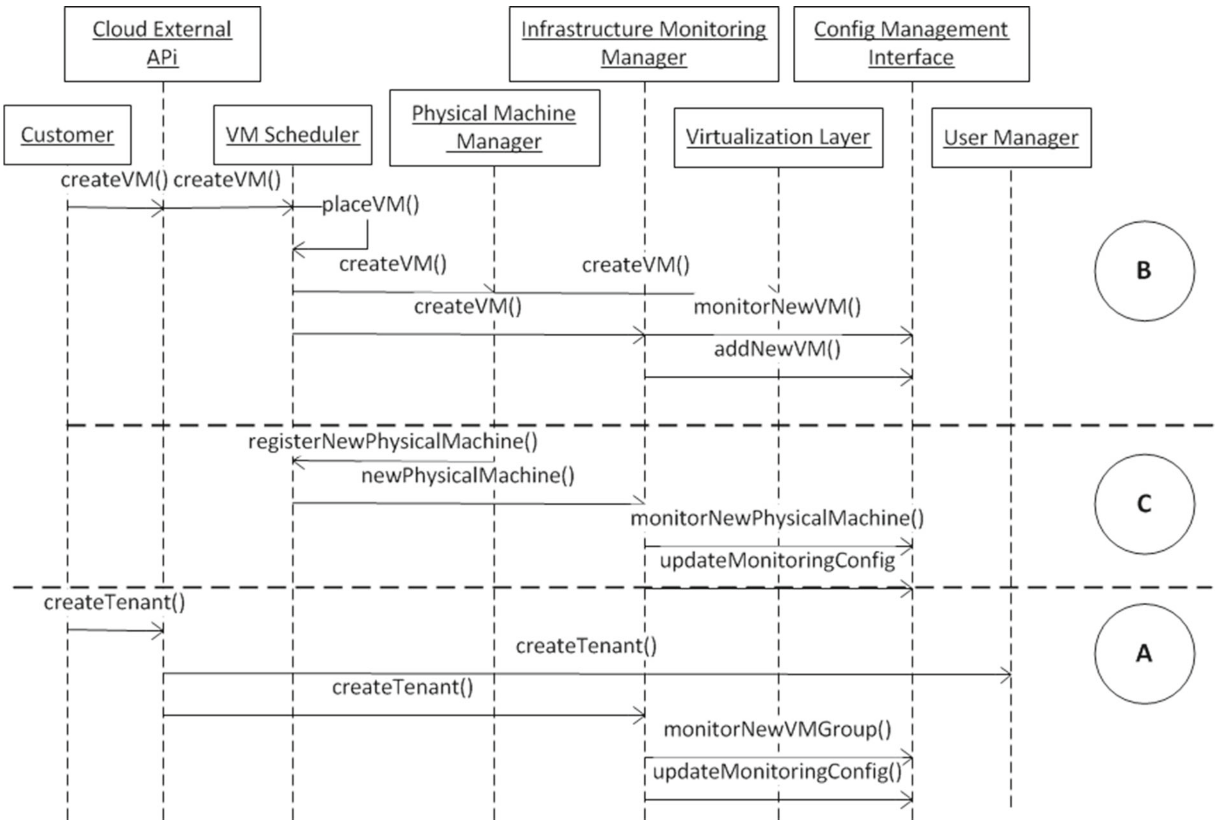


Fig. 3 Sequence Diagram for the use case in which a customer is creating the first VM in the Cloud Provider

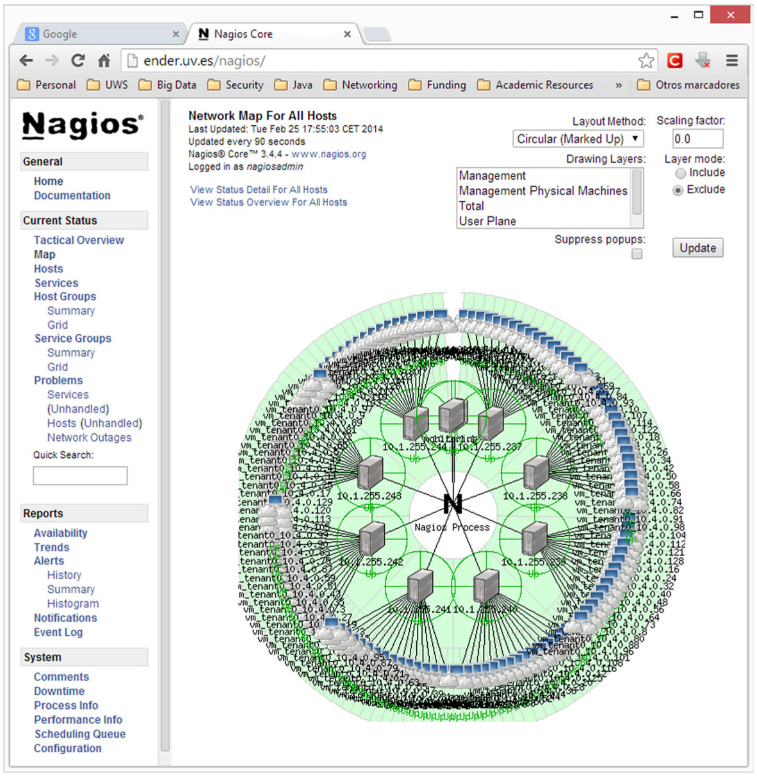
6 Implementation

The architecture presented in the previous sections has been implemented and released to the community as an open source project under GPL license. IaaSMon can be downloaded together with the documentation and the source code[21] to validate, reproduce and use this research work. *IaaSMon* has been developed completely in Java. This module is independent of the rest of the services in the cloud stack and only requires to be attached to the RabbitMQ[22] message bus used to interconnect the different OpenStack modules. The prototype subscribes to the relevant exchanges in RabbitMQ in order to receive messages about creation and elimination of VMs and also registration of physical machines. The module is defined in approximately 3000 lines of Java code which are complemented with another 3000 lines of scripting for doing the automatic installation and configuration of the complete monitoring architecture. IaaSMon has the innovative capability to update Nagios in two different ways, either as soon as the infrastructure knows the future IP address of the VM or when the VM has already been successfully created. The former approach is optimized for responsiveness whereas the latter is optimized for consistency, using respectively the message #3 and message #4 in Table 2 to update Nagios. To use IaaSMon, it is assumed that there is an OpenStack infrastructure and a Nagios monitoring tool already installed in the cloud infrastructure. On top of that, *IaaSMon* provides an installation script that performs the initial configuration and set up of the new architectural components proposed, i.e. *IMM* and *CMC*.

In order to better explain the behavior of IaaSMon, Fig. 3 shows a sequence diagram for three different case studies where: A) a customer is registered in the cloud provider; B) a customer creates and launch a new VM, we assume that this VM is the first one

[21] http://sourceforge.net/projects/iaasmon/

[22] http://www.rabbitmq.com



Network Map For All Hosts
Last Updated: Tue Feb 25 17:55:03 CET 2014
Updated every 90 seconds
Nagios® Core™ 3.4.4 - www.nagios.org
Logged in as *nagiosadmin*

Layout Method: Circular (Marked Up)
Scaling factor: 0.0
Layer mode: Exclude

Fig. 4 Nagios Snapshots showing the mapping between physical and virtual machines dinamically discovered by IaaSMon

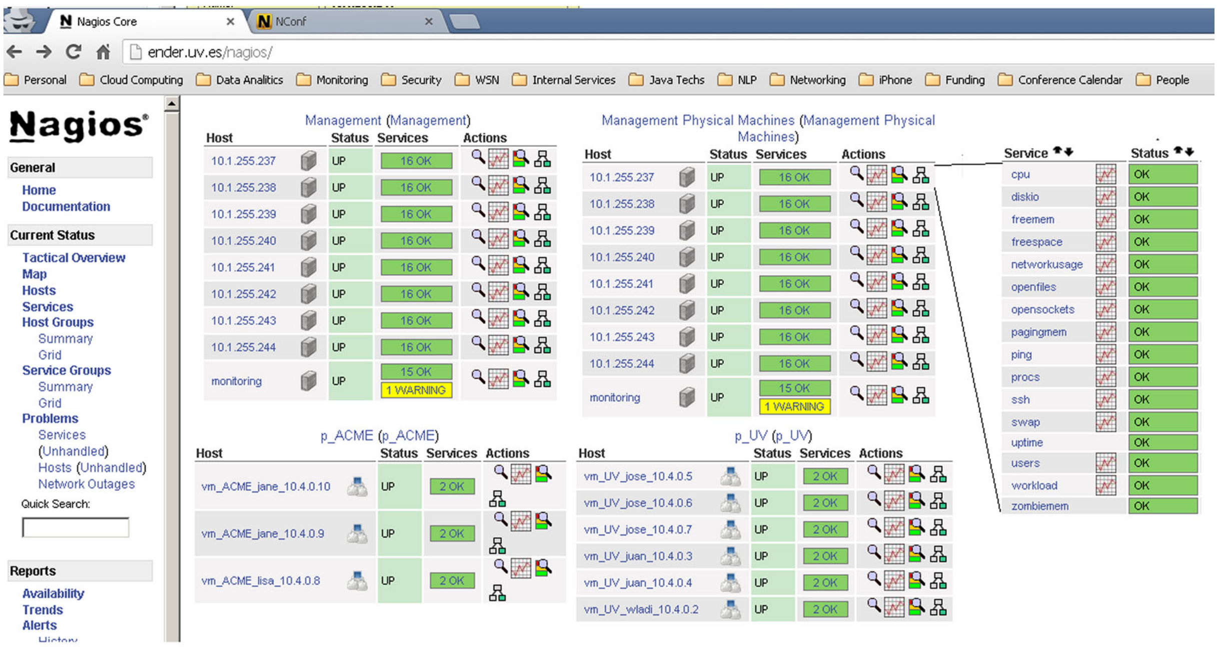
created by the user in the public cloud provider so that IaaSMon does not have any previous information available for this customer; C) the physical machines update periodically their status. The name of the methods available in Fig. 3 does not match exactly with the implemented ones but we have decided to provide more descriptive names for readability purposes. Figure 3 assumes that IaaSMon is setup in the mode optimized for responsiveness.

For case study A), it is worthy to see in Fig. 3 how the creation of a new tenant is intercepted by our IMM producing the creation of a new logical group associated to such tenant in the tree structure previously explained. For the case study B) where the user is creating a new VM, notice that *IMM* intercepts the `createVM` message and then performs HTTP calls to NConf with the new information to be configured in Nagios. Then, NConf generates the new *config files* and updates Nagios accordingly. *IaaSMon* support the usage of a predefined VM name prefix and/or a predefined tenant name to discriminate different types of VMs (customer VMs and management VMs). Finally, see how *IMM* is intercepting periodically the status of physical machines available within the cloud infrastructure in case study C) by means of the periodic heart beat sent to update their status into the cloud infrastructure.

By default, each logical group of machines (shown in Fig. 2) comes with a wider set of predefined monitoring metrics. Moreover, simple OpenStack deployments do not use any VM for management purpose so that this group could be always empty in some deployments. We decided to include it to provide further support to other open cloud stacks like CloudStack[23] which make use of management VMs. IaaSMon has been designed to allow cloud providers to combine automatic configuration with fine-grain manual configuration of the different monitored resources. So, the information enforced automatically by IaaSMon can be extended and fully customized manually using the NConf web page. This is very powerful and another differentiating point because it enables the manual customization of the monitoring system while the whole automated discovery technique is in place.

For the agent-based monitoring approach, the monitoring architecture relies in *NRPE*, Nagios Remote

[23] http://incubator.apache.org/cloudstack


Nagios Core | NConf

ender.uv.es/nagios/

Personal | Cloud Computing | Data Analitics | Monitoring | Security | WSN | Internal Services | Java Techs | NLP | Networking | iPhone | Funding | Conference Calendar | People

Nagios®

General
- Home
- Documentation

Current Status
- Tactical Overview
- Map
- Hosts
- Services
- Host Groups
  - Summary
  - Grid
- Service Groups
  - Summary
  - Grid
- Problems
  - Services (Unhandled)
  - Hosts (Unhandled)
  - Network Outages

Quick Search:

Reports
- Availability
- Trends
- Alerts

Management (Management)

Host	Status	Services	Actions
10.1.255.237	UP	16 OK	
10.1.255.238	UP	16 OK	
10.1.255.239	UP	16 OK	
10.1.255.240	UP	16 OK	
10.1.255.241	UP	16 OK	
10.1.255.242	UP	16 OK	
10.1.255.243	UP	16 OK	
10.1.255.244	UP	16 OK	
monitoring	UP	15 OK 1 WARNING	

Management Physical Machines (Management Physical Machines)

Host	Status	Services	Actions
10.1.255.237	UP	16 OK	
10.1.255.238	UP	16 OK	
10.1.255.239	UP	16 OK	
10.1.255.240	UP	16 OK	
10.1.255.241	UP	16 OK	
10.1.255.242	UP	16 OK	
10.1.255.243	UP	16 OK	
10.1.255.244	UP	16 OK	
monitoring	UP	15 OK 1 WARNING	

Service	Status
cpu	OK
diskio	OK
freemem	OK
freespace	OK
networkusage	OK
openfiles	OK
opensockets	OK
pagingmem	OK
ping	OK
procs	OK
ssh	OK
swap	OK
uptime	OK
users	OK
workload	OK
zombiemem	OK

p_ACME (p_ACME)

Host	Status	Services	Actions
vm_ACME_jane_10.4.0.10	UP	2 OK	
vm_ACME_jane_10.4.0.9	UP	2 OK	
vm_ACME_lisa_10.4.0.8	UP	2 OK	

p_UV (p_UV)

Host	Status	Services	Actions
vm_UV_jose_10.4.0.5	UP	2 OK	
vm_UV_jose_10.4.0.6	UP	2 OK	
vm_UV_jose_10.4.0.7	UP	2 OK	
vm_UV_juan_10.4.0.3	UP	2 OK	
vm_UV_juan_10.4.0.4	UP	2 OK	
vm_UV_wladi_10.4.0.2	UP	2 OK	

Fig. 5 Nagios Snapshots: A) Mapping physical /virtual machines dinamically discovered. B) Monitoring logic groups and Monitoring metrics associated to each group

Plugin Executor and *NSCA*, Nagios Service Check Acceptor, to act respectively as active and passive agents to gather metrics remotely. We also rely in the wide set of Nagios *plugins* available to gather a vast number of metrics using both agent-based and agent-less approaches.

Figure 4 shows a screenshot of Nagios monitoring 128 VMs allocated in 8 physical nodes using Open-Stack and *IaaSMon*. This figure shows the correlation between physical and virtual machines graphically by means of a two-layer graph visualizing graphically workload distributions, system failures, etc. This is achieved by means of the correlation of the information intercepted in messages of the OpenStack control plane providing useful and valuable information to the administrator of the cloud provider. This correlation is detected and enforced by the IMM.

Figure 5 shows a screenshot where the cloud administrator can see: i) the logical groups previously described in Fig. 2; ii) all the VMs grouped by tenant (*ACME* and *UV* are two example tenants which contain some VMs each) which has been automatically discovered by IaaSMon; and, users and VMs among other monitoring information. For each, VM there is a number of services and metrics being monitored and for each metric, Nagios provides different graphical and numerical information shown to the user.

In term of scalability, both OpenStack and Nagios have intensively proved their scalability due to the number of large-scale deployments being done so far for each of these tools. OpenStack addressed scalability in its own design where it is deployed in a complex distributed environment. Nagios manages scalability by means of the usage of DNX[24] (Distributed Nagios Executor). *DNX* provides Nagios with a distributed master-slave infrastructure where Nagios can scale the monitoring of thousands of services and resources efficiently.

7 Evaluation

IaaSMon prototype has been validated empirically in a mid-size production-ready cloud infrastructure composed by 8 high-dense Bull R424-E3 blades with 2 Xeon ES-2650 2Ghz, 1TB SATA III, 32 GB @

[24]http://dnx.sourceforge.net

1600Mhz wired with two gigabit networks over which we installed an OpenStack Folsom single-host, Nagios 3.4.4, NConf 1.3.0 and our prototype.

We have created 128 VMs, one by one, at a constant rate. This rate has been fixed between 16 and 18 seconds. Smaller numbers make the infrastructure to fail creating at least one VM due to the high stressing of the system even without the monitoring architecture running. Bigger numbers makes monitoring overhead negligible and the main purpose of this experiment is to analyze the behavior of the monitoring architecture is the worse conditions in order to enable the reader to see the worse-case scenario. Times have been measured with and without our prototype running in order to analyze the overhead imposed by IaaSMon into the system. It is worthy to mention that 128 VMs is exactly the number of cores available in the blades used. It has been decided to do not increase this number in order to do not bias the results provided. A higher number of VMs will cause much slower VMs leading to low overhead for the monitoring architecture and we are focused on providing information for the worst case scenario. Figure 6 shows the average of 5 executions measuring the time where the scripts initiates the creation of the first VM to the arrival of at first ICMP echo reply received for all the launched VMs. 20 different metrics are monitored from each VM and 40 different metrics are monitoring for each physical machine. As seen in Fig. 6, in the worst scenario (T=16), where the infrastructure is really stressed, our monitoring module only imposes around 2 % of overhead time from the customer s perspective whereas this percentage become negligible (near to 0 %) for less stressed situations. These numbers clearly validate the scalability, feasibility and good performance exposed of our architecture. Notice that more stress levels will automatically lead to failures in the creation of VMs even without any monitoring tool running. It means that the worst case scenario imposes a 2 % of overhead.

Figure 7 shows the most and less stressed scenarios with respectively requests intervals at 16 and 18 seconds at top and bottom subfigures, respectively. These figures show how our system behaves along the time when the number of VMs are being increased at a constant rate. The following times have been gathered: *API Time* represents the time between client creates a request and thus requests is inserted in the communication bus; *OpenStack Time* is the time required to process the message in OpenStack; *Monitoring Time* is the time required by IaaSMon for starting the monitoring of all the services of the new VM in Nagios; And finally, *VM Ping Time* is the time when the machine answers to the first ICMP echo request. Both subfigures available in Fig. 7 show a linear increase in the total time associated to process VMS when increasing in the number of VMs mainly due to the increase in the workload of the whole system. For this

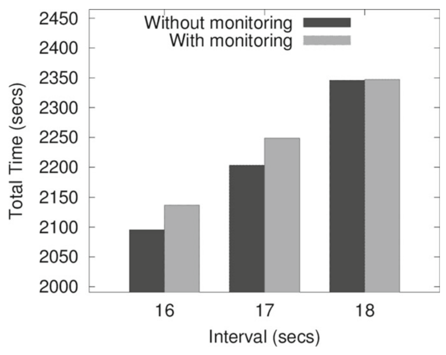


Fig. 6 OpenStack performance comparison with and without our architecture when creating 128 VMs at different arrival rates of creation requests

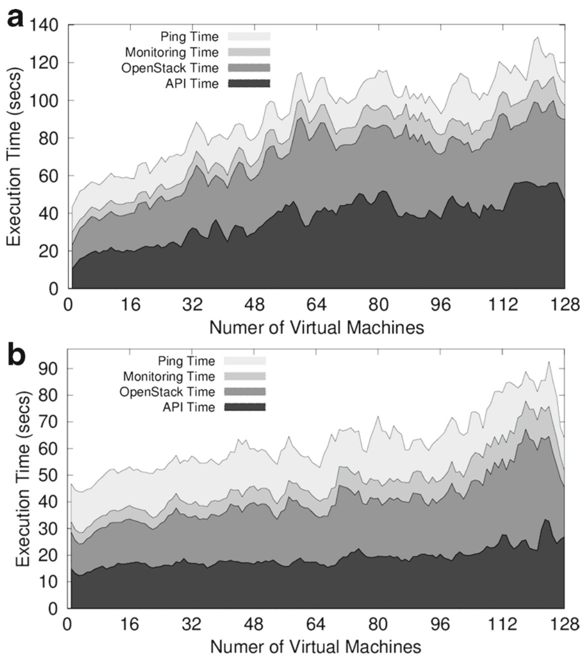


Fig. 7 Scalability of the Monitoring architecture when ranging the number of VMs between 1 and 128 for a constant arrival interval time of create VM requests: 16 seconds (top) and 18 seconds (bottom)

experiment, IasSMon has been optimized for responsiveness and thus it reacts as soon as the infrastructure knows the future IP address. The mean time required by *IMM* to update Nagios is 8,7 and 6,9 seconds for 16 and 18 interval requests, respectively. It represents in average around 10 % more of the overall time required by OpenStack in both cases. Note that even having 10 % of overhead time from the OpenStack perspective, there is not any overhead at all from the customer perspective since the VMs are being booted simultaneously to the configuration of the monitoring system and, in consequence, VM are being reached (*pinged*) always after the finalization of the configuration of the monitoring system. This parallelization is the reason of the low overhead (2 % in the worst case scenario tested) shown in Fig. 6 and are a clear sign of the scalability of the proposed architecture.

In the two scenarios shown in Fig. 7, the monitoring system is configured in average 15 seconds before the VMs are reachable. This fact demonstrates the good responsiveness of our proposal even in the worst case scenario.

8 Conclusion

In this contribution, it has been proven the advantages in terms of overhead and scalability associated to the early reaction in the configuration of Nagios due to the integration of the monitoring solution with the control and data planes of the public cloud stack which leads to a rapid and adaptive monitoring solution suitable for monitoring public clouds. It has been also described a complete distributed and high scalable monitoring solution suitable for monitoring public cloud infrastructures. The solutions fulfill all the requirements identified for public cloud infrastructures. A novel auto-discovery process for both virtual and physical infrastructure has also been provided that allows the correlation between physical and virtual machine lately used for correlating monitored metrics. It has been demonstrated how a Nagios can be adapted to fit in cloud computing scenarios achieving all the benefits of years of already available development in the monitoring field. It has been provided a new component for OpenStack which enable other researchers and practitioners to monitor their cloud infrastructure even in public cloud scenarios.

As future work, it is planned to extend the integration of the control plane of the cloud stack and the monitoring solution in order to cover advanced scenarios as migration of VMs, correlation of VM sharing the same golden image and etc.

Acknowledgments This work is supported by the European Commission Horizon 2020 Programme under grant agreement number H2020-ICT-2014-2/671672 - SELFNET (Framework for Self-Organized Network Management in Virtualized and Software Defined Networks) and by the Universitat de Valencia Research Infrastructure Grant 2014.

References

1. Aceto, G., Botta, A., de Donato, W., Pescap, A.: Cloud monitoring A survey. Comput. Netw.: Int. J. Comput. Telecommun. Netw. **57**(9), 2093–2115 (2011)
2. Brandt, J., Gentile, A., Mayo, J., Pebay, P., Roe, D., Thompson, D., Wong, M.: Resource Monitoring and Management with OVIS to Enable HPC. In: Cloud Computing Environments IEEE International Symposium on Parallel & Distributed Processing, Rome (2009)
3. Cuomo, A., Modica, G.D., Distefano, S., Puliafito, A., Rak, M., Tomarchio, O., Venticinque, S., Villano, U.: An SLA-based broker for cloud infrastructures. J. Grid Comput. **11**, 1–25 (2013)
4. Barbosa de Carvalho, M., Zambenedetti Granville, L.: Incorporating Virtualization Awareness in Service Monitoring Systems. In: IFIP/IEEE International Symposium on Integrated Network Management (2011)
5. Aparecida de Chaves, S., Brundo Uriarte, R., Becker Westphall, C.: Toward an architecture for monitoring private clouds. IEEE Commun. Mag. **49**, 130–137 (2009)
6. Dhingra, M., Lakshmi, J., Nandy, S.K.: Resource Usage Monitoring in Clouds. In: De ACM/IEEE 13th International Conference on Grid Computing (2012)
7. Katsaros, G., Kubert, R., Gallizo, G.: Building a Service-Oriented Monitoring framework with REST and Nagios. In: IEEE International Conference on Services Computing (2011)
8. Kertesz, A., Kecskemeti, G., Marosi, A., Oriol, M., Franch, X., Marco, J.: Integrated Monitoring Approach for Seamless Service Provisioning in Federated Clouds. In: 20th Euromicro International Conference on Parallel, Distributed and Network-based Processing, Munich, Germany (2012)
9. Ketesz, A., Kecskemeti, G., Oriol, M., Kotcauer, P., Acs, S., Rodrguez, M., Merc, O., Marosi, A.C., Marco, J.,

Franch, X.: Enhancing federated cloud management with an integrated service monitoring approach. J. Grid Comput. **11**, 699–720 (2013)

10. Koenig, B., Alcaraz Calero, J.M., Kirchnick, J.: Elastic monitoring framework for cloud infrastructures. IET Commun. **6**, 1306–1315 (2011)
11. Lorido-Botran, T., Miguel-Alonso, J., Lozano, J.A.: A review of auto-scaling techniques for elastic applications in cloud environments. J. Grid Comput. **12**, 559–592 (2015)
12. Massonet, P., Naqvi, S., Ponsard, C., Latanicki, J., Rochwerger, B., Villari, M.: A Monitoring and Audit Logging Architecture for Data Location Compliance in Federated Cloud Infrastructures. In: IEEE International Parallel & Distributed Processing Symposium, Anchorage, Alaska (2011)
13. Milojicic, D., Llorente, I.M., Opennebula, R.S.M.: A cloud management tool. IEEE Internet Comput. **15**(2), 11–14 (2011)
14. Moses, J., Iyer, R., Illikkal, R., Srinivasan, S., Aisopos, K.: Shared Resource Monitoring and Throughput Optimization in Cloud-Computing Datacenters. In: IEEE International Parallel & Distributed Processing Symposium, Anchorage, Alaska (2011)
15. Petcu, D.: Consuming resources and services from multiple clouds: From terminology to cloudware support. J. Grid Comput. **12**, 321–345 (2014)
16. Povedano-Molina, J., Lopez-Vega, J.M., Lopez-Soler, J.M., Corradi, A., Dargos, L.F.: A highly adaptable and scalable monitoring architecture for multi-tenant clouds. Futur. Gener. Comput. Syst. **29**(8), 2041–2056 (2013)
17. Romano, L., De Mari, D., Jerzak, Z., Fetzer, C.: A Novel Approach To QoS Monitoring In The Cloud. In: First International Conference on Data Compression, Communications and Processing, Palinuro, Italy (2011)
18. Wuhib, F., Stadler, R.: Distributed Monitoring and Resource Management for Large Cloud Environments. In: IFIP/IEEE International Symposium on Integrated Network Management, Dublin, Ireland (2011)