Grant Agreement No.: 610764
Instrument: Collaborative Project
Call Identifier: FP7-ICT-2013-10

PANACEA

Proactive Autonomic Management of Cloud Resources

D2.1: Principles of pervasive monitoring

Version: v.1.1

Work package	WP 2
Task	Task 2.1
Due date	01/04/2014
Submission date	11/04/2014
Deliverable lead	Imperial
Version	1.1
Authors	Gokce Gorbil, Eduardo Huedo Cuesta, David Garcia Perez
Reviewers	Olivier Brun, Eliezer Dekel, Dimiter R. Avresky

Abstract	This document presents the principles and requirements of a pervasive cloud monitoring solution needed to support proactive and autonomous management of cloud resources and communications. We first discuss pervasive monitoring methods, and then provide a review of existing monitoring solutions used by the industry and assess their suitability to support pervasive monitoring. We find that the collectd daemon is a good candidate to form the basis of a lightweight monitoring agent that supports high resolution probing, but that it will need to be wrapped around with agent software that provides the higher-level interaction capabilities necessary for pervasive monitoring. Among the higher-level tools that provide additional functionalities, Zabbix, Ganglia and Nagios appear to be the most capable and suitable for the purposes of pervasive monitoring. We also review the current monitoring capabilities of the OpenNebula cloud management software.
Keywords	Cloud monitoring; pervasive monitoring; cloud monitoring tools

Document Revision History

Version	Date	Description of change	List of contributor(s)
V1.0	31.03. 2014	Draft ready	Gokce Gorbil (Imperial) Eduardo Huedo Cuesta (UCM) David Garcia Perez (Atos)
V1.1	09.04.2014	Updated document after reviews	Gokce Gorbil (Imperial) Eduardo Huedo Cuesta (UCM) David Garcia Perez (Atos)

Acknowledgment (if needed) -

*R: report, P: prototype, D: demonstrator, O: other

Project co-funded by the European Commission in the 7th Framework Programme (2007-2013)		
Nature of the deliverable:	R	
Dissemination Level		
PU	Public	**X**
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to bodies determined by the PANACEA project	
CO	Confidential to PANACEA project and Commission Services	

EXECUTIVE SUMMARY

Accurate and fine-grained monitoring of dynamic and heterogeneous cloud resources is essential to the overall operation of the cloud. In this document, we review the principles of pervasive cloud monitoring, and discuss the requirements of a pervasive monitoring solution needed to support proactive and autonomous management of cloud resources.

We provide a review of existing monitoring solutions used by the industry and assess their suitability to support pervasive monitoring. We find that the collectd daemon is a good candidate to form the basis of a lightweight monitoring agent that supports high resolution probing, but that it will need to be wrapped around with agent software that provides the higher-level interaction capabilities necessary for pervasive monitoring. Among the higher-level tools that provide additional functionalities, Zabbix, Ganglia and Nagios appear to be the most capable and suitable for the purposes of pervasive monitoring.

Finally, we present the monitoring capabilities of the popular cloud management software OpenNebula, and briefly discuss how third party monitoring tools such as the ones reviewed in this document can be integrated with OpenNebula in order to provide the foundation for a holistic cloud monitoring and management solution.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS

CPU	Central Processing Unit
CPN	Cognitive Packet Network
DP	Dumb Packet
IaaS	Infrastructure as a Service
I/O	Input/Output
MaaS	Monitoring as a Service
NIST	National Institute of Standards and Technology
OS	Operating System
PaaS	Platform as a Service
QoS	Quality of Service
RNN	Random Neural Network
RTT	Round Trip Time
SaaS	Software as a Service
SLA	Service Level Agreement
SP	Smart Packet
VM	Virtual Machine

1 INTRODUCTION

Cloud providers need to manage increasingly large and complex cloud infrastructures and assure the availability, reliability, elasticity and performance of offered cloud services [1][2]. The increasing scale and complexity of cloud facilities mandate an autonomous and proactive management approach [3][4]. In this management model, human operatives would setup and adjust the general policies that drive the dynamic decisions taken autonomously in the cloud, monitor the general health of the cloud, and intervene in the rare cases that the cloud is unable to handle by itself.

Accurate and fine-grained monitoring of cloud resources is essential to the overall operation of the cloud, especially resource and performance management [5]. Hardware virtualization is employed in the cloud in order to improve utilization of physical resources, such as servers, storage devices and networking equipment. Therefore, there is a need to monitor both physical and virtual resources in the cloud, and the monitoring solution needs to accommodate for volatility in virtual resources and for the migration of VMs [6][7]. In addition to the computing and storage resources, network conditions also need to be monitored [8][9]. This is especially important when the cloud provider operates multiple clouds in physically separate locations, or in hybrid clouds.

In this document, we discuss the principles and requirements of a pervasive monitoring solution needed to support autonomous and proactive management of heterogeneous cloud resources. Such a monitoring solution is required to be scalable in order to be able to handle a large number of resources and probes without causing a significant impact on cloud resources. It needs to be elastic in order to automatically accommodate the high dynamism of the cloud environment due to consumer churn and virtualization. The adopted solution needs to support different monitoring methods and runtime configuration changes. Monitoring data is required to provide an accurate representation of the cloud state and needs to be delivered in a timely fashion in order to enable quick decision making in the face of changes.

We are interested in leveraging the management capabilities of OpenNebula[1] in order to actuate the autonomous decisions taken by the intelligent management technologies developed as part of the PANACEA project, and therefore we present a review of its monitoring capabilities and discuss how third party monitoring tools can be integrated with it.

1.1 An overview of cloud computing

Cloud computing, as defined by the NIST, is “a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction” [10].

Cloud providers have adopted three fundamental service models: Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). The service models are relevant to the monitoring solution as they define what the cloud provider and the consumer can access, and therefore what they wish to and can monitor. The service model also influences the monitoring metrics of interest to the provider and the user.

Based on who can access the cloud infrastructure, the cloud has four deployment models: private, community, public, and hybrid. Similar to the service model adopted by the cloud provider, the deployment model affects monitoring requirements and capabilities. For example, monitoring of metrics related to security is more important in public clouds than in private clouds. In a hybrid cloud, the network connecting the clouds composing the hybrid is an important component and therefore

[1] http://opennebula.org/

needs to be monitored.

1.2 Document structure

The rest of this document is organized as follows. In section 2, we discuss the principles and requirements of pervasive cloud monitoring. Section 3 provides a review of existing cloud monitoring tools and discusses their capabilities for pervasive monitoring. In section 4, we present the monitoring capabilities of the OpenNebula cloud management software, and discuss how third party monitoring tools and the proposed solution can be integrated with it. Section 5 concludes with a summary and presents planned work for the design and development of an integrated monitoring solution.

2 AUTONOMOUS AND PERVASIVE CLOUD MONITORING

The popularity of cloud services is increasing and with this increasing popularity, clouds are growing in size and traffic volume [1]. The cloud provider therefore has to make sure that the performance of its cloud infrastructure and services is keeping up with increasing demand, and to assure the quality-of-service (QoS) demanded by its customers, while also keeping the business profitable. To achieve this, the cloud provider needs to manage increasingly large and complex cloud infrastructures, which mandate an intelligent approach to their management.

Accurate and fine-grained monitoring of cloud resources is essential to the overall operation of the cloud, especially resource and performance management [11]. In this section, we discuss the principles and requirements of the pervasive monitoring solution necessary to support autonomous and proactive management of cloud resources in order to assure the reliability, availability and QoS of cloud services. Although the principles and requirements of monitoring discussed here apply in more general terms to all applications of monitoring, in this document we will focus on monitoring to support resource and performance management.

2.1 Applications of cloud monitoring

Monitoring of the cloud is essential to many activities in the cloud [5]. In order to be able to meet demand and guarantee the performance of cloud services, the cloud provider needs to estimate the expected workload and to quantify the needed capacity and resources to be acquired. Provisioning of measured services is one of the characteristics of cloud computing, and monitoring of service use is crucial to the provider in terms of billing and to the customer for verification of her usage. Troubleshooting in the cloud necessitates comprehensive and continuous monitoring of all cloud components and layers in order to identify the root cause of faults and performance issues.

In this document, we focus on cloud monitoring for the purpose of service and performance management performed by the cloud provider. Both the provider and the consumer employ monitoring in order to assess the performance of the cloud. The consumer is mainly interested in the performance observed by the end users of the services and applications it runs on the cloud [12]-[15]. The provider has a wider interest in the performance of the cloud as a whole, considering aggregate performance with multiple customers concurrently accessing and using the cloud. The provider also needs to monitor the instantaneous and average performance observed by each customer in order to adjust cloud parameters to satisfy SLAs with the customers [16]. Monitoring also enables the provider to formulate more realistic and dynamic SLAs and pricing models by exploiting its knowledge of performance perceived by its customers.

2.2 Concepts in cloud monitoring

2.2.1 Cloud layers

The cloud has seven layers as specified by the Cloud Security Alliance [10]:

- Facility layer: Consists of the physical infrastructure of the data centre(s) where the cloud is hosted. Monitoring at this layer considers data centre operations, energy consumption, environmental impact, and physical security, such as surveillance and architectural resilience [17].
- Network layer: Consists of the communication links and paths within a cloud, between clouds, and between the cloud and the user.
- Hardware layer: Consists of the physical components of the computing, storage and networking equipment.
- Operating system (OS) layer: Consists of the host and guest operating systems, and the

hypervisors, i.e. the VM managers.

- Middleware layer: Normally only present in the PaaS and SaaS service models, it consists of the software layer between the user application and the OS.
- Application layer: Consists of the user applications running in the cloud.
- User layer: Consists of the end users of the cloud applications and the applications running outside the cloud, e.g. a web browser of the end user.

In the context of cloud monitoring, these layers essentially specify the location of the probes for the monitoring system, i.e. from where measurements are collected [11]. The layer at which a probe is located limits the types of events and resources that the probe can observe and monitor [17][18]. In this document, we consider all layers except the facility layer as relevant to the monitoring solution.

2.2.2 Abstraction levels

In terms of abstraction levels, cloud monitoring can be high-level or low-level. High-level monitoring is concerned with the status of the virtual platform, and gathers information at the middleware, application and user layers. In the IaaS model, the provider is normally not interested in high-level monitoring since it only provides physical resources to the consumer [19]. In the PaaS and SaaS models, the provider, in addition to the consumer and any third parties acting on behalf of either, perform high-level monitoring.

Low-level monitoring is used to refer to monitoring activities that collect information from the lower level resources in the cloud, and it operates at the network, hardware and OS layers. It is mainly used by the provider for the management of the cloud, and low-level monitoring data is not available to the consumer in the PaaS and SaaS models. In the IaaS model, the consumer can perform limited low-level monitoring since it has full control of the guest OS and therefore can collect data at the (guest) OS layer [12]-[14].

A pervasive monitoring solution needs to support both high and low-level monitoring in all service models.

2.2.3 Basic metrics

There are many metrics of interest in the cloud [3][5], and we provide only a sample of the basic metrics here.

- Computing related metrics: Server throughput, defined as the number of requests per second; CPU speed; time per CPU execution cycle; CPU utilization; memory utilization; memory page exchanges per second; memory page exchanges per execution cycle; memory throughput; disk throughput; disk I/O; response time; VM start-up/acquisition/release time; up-time; number of VMs on host; etc.
- Network related metrics: Round trip time (RTT), jitter, throughput; loss rate; available bandwidth; capacity; traffic volume; number of packets/bytes sent/received; number of TCP/UDP sockets open; number of listening sockets open; used ports, etc.
- Other metrics: CPU temperature; CPU voltage; energy consumed by the server; etc.

These metrics will be available at specific layers, and may only be available to the provider or the consumer according to the service model. All metrics can be evaluated in terms of statistical indicators (median, mean, variance, histogram, running average, etc.) and temporal characteristics.

The monitoring solution needs to be able to support many metrics concurrently. We will discuss this in more detail in the following sections.

2.2.4 Reporting methods

Depending on how and when monitoring data is reported, we can classify monitoring solutions as *push* vs. *pull-based* and as *event-based* vs. *periodic*.

- Event-based vs. periodic: Periodic reporting is the simplest form of reporting monitoring data, where data is sent periodically. In event-based monitoring, data is reported when a predefined event happens. Thresholds are often used in event-based monitoring, where a report is triggered when a measured value reaches a given threshold.
- Push vs. pull reporting: In pull reporting, the entity interested in the monitoring data, e.g. the cloud manager, pulls the data from the entity that has the data by polling it. In push reporting, the entity that has the monitoring data pushes it to the entity interested in the data.

Event-based reporting is useful to decrease the traffic load due to monitoring since a report would only be generated when an event of interest occurs. However, determining the events of interest, and in the case of thresholds, the threshold values, could be problematic [3]. The event should be chosen so that the system has time to react and rectify any problem before QoS is violated or an unrecoverable error occurs. This requires timely delivery of reports and estimation of how long it will take for the system to react to the event under the given load [20]. As the system load changes, the reaction time will change, which may require subsequent changes in the registered events. The events and thresholds therefore need to be carefully chosen so that reports are not triggered too early or too late. For certain actions which can be performed in steps, event reporting can be set-up in a staggered manner to trigger each step when necessary.

Similar issues arise in periodic reporting when scalability and monitoring overhead are considered [21]. For example, the monitoring period may need to be decreased when network load increases or when the number of monitored resources becomes large. The period may also need to be adjusted according to the observed metric, e.g. it may need to be increased as a critical value is approached. The monitoring solution should support both event-based and periodic reporting, and should also be aware of the issues discussed here.

The push model is harder to configure than the pull model since many entities need to be informed of configuration changes in contrast to the few polling entities in the pull model. The reporting entities also need to know where to push the data, and maintaining this information is difficult in the case of multiple data receivers that are interested in different metrics. In this situation, a publish-subscribe mechanism can be used to decouple the communication endpoints. The push model is particularly attractive for large scale cloud environments with high resource churn, i.e. where services, VMs and hosts are constantly being added, removed, started and stopped, since maintaining lists of resources to poll for the pull model in such an environment is challenging. The monitoring solution needs to support both models since a hybrid approach provides the best overall solution.

2.3 Pervasive monitoring of cloud resources

The monitoring solution needed to support autonomous and proactive management of cloud resources should be able to collect measurements from a variety of physical and virtual resources at different layers. In addition to monitoring of the computing and storage resources, communications within the cloud and between clouds also need to be monitored. The monitoring solution therefore needs to be *pervasive*. Large scale and complex clouds present challenges in terms of the scalability of the monitoring solution due to the high number and types of resources that need to be monitored. Scalability is also important considering that the monitoring solution needs to be able to handle many metrics concurrently, perhaps as many as a hundred. The pervasive monitoring solution therefore needs to efficiently collect, transfer and analyse data from many probes without impairing the normal operations of the cloud.

2.3.1 Agent based monitoring

The scalability issue in monitoring has been mainly addressed by aggregation and filtering of the monitoring data [7][16][21]-[26]. In addition to the standard aggregation and filtering methods, another approach to improve scalability is *autonomous* monitoring that self-manages and adapts to changes, while simplifying the configuration and control tasks for the cloud manager [4][16]-[21]. The use of configurable software *agents* has been proposed in the literature in order to construct a flexible monitoring architecture [14]. In agent-based monitoring solutions, agents are located on the physical and virtual resources at the appropriate layers in the cloud in order to have access to the metrics of interest. A monitoring agent implements one or more probes that can collect measurements on metrics from the resource either on demand or periodically. Agents can be started automatically at start-up of their respective resource, e.g. physical host or VM, or they can be started and killed on demand by a monitoring manager. Multiple monitoring managers may exist for fault tolerance and load balancing purposes. In the first option where agents start automatically with the resource they monitor, a *push-based* registration with the monitoring manager is most appropriate, and lends *elasticity* to the monitoring solution by enabling runtime discovery of the monitored resources, such as newly added resources and VMs that migrate. In this scheme, the agent informs the monitoring manager of its properties and capabilities, such as the resources and metrics it can monitor, at registration. We believe push-based registration to be a better option for a pervasive monitoring solution due to its advantages.

The role of the monitoring manager is to control the monitoring agents and to activate, reconfigure and deactivate monitoring as required for dynamic resource management. The monitoring manager is also a good candidate to provide the interface between the monitoring solution and any software that uses the monitoring data, e.g. the cloud management software.

The monitoring agents can be configured at runtime by the monitoring managers in order to change the monitoring behaviour of the system. Any time after registration, the agent can be instructed by the monitoring manager to perform event-based or periodic reporting of various metrics. For example, if the cloud manager needs periodic keep-alive messages from hosts and VMs, then these can be set-up. Similarly, event-based alerts such as for high CPU temperature, memory over-utilization, etc. can be configured at this time. Any other core metrics that are used to manage the cloud resources can be requested *on demand* by the monitoring manager. This is an important feature that improves scalability of the monitoring.

We make a distinction between health metrics and core metrics, which enables us to separate different types of monitoring requirements and re-use existing monitoring solutions where applicable. Health metrics are required by software components managing the cloud, and are of interest mainly to obtain an overall view of the health of the cloud infrastructure. Many software for cloud management come equipped with their own monitoring solution in order to observe health metrics. For example, the popular cloud management suite OpenNebula has a basic monitoring solution, which we describe in a later section. Core metrics are used for dynamically allocating resources in order to optimize cloud performance under varying conditions. The final monitoring solution adopted by PANACEA can be a hybrid that combines the existing basic monitoring solution of the cloud management software and the pervasive monitoring solution that drives most of the resource management decisions.

2.3.2 Self-managing overlay networks for cloud monitoring

The monitoring solution needs to be resilient to failures in the monitoring infrastructure and the monitored resources, including the communication network within a cloud and between the clouds. Fault tolerance of the monitoring solution is improved by providing multiple instances of the monitoring manager. A distributed beacon mechanism can be employed among them so that the remaining managers can detect the failure of a manager. Failure of a local probe can be detected by its monitoring agent, which can then attempt to revive the local probe by creating a new probe instance. Failure of a monitoring agent due to failure of its hosting component, e.g. VM or host, needs to be handled gracefully by the monitoring solution and should not affect monitoring of other components

on the cloud.

One of the greatest concerns in cloud monitoring is congestion and failures in the communication networks within the cloud and connecting different clouds in the case of multi-cloud scenarios. Any disruptions in the communication network affect not only the hosted cloud services but also the services used by the cloud provider for the monitoring and management of the cloud. Self-aware adaptive overlay networks [27]-[29] provide an attractive solution to address these issues. In the intra-cloud overlay, a virtual overlay network is built on top of the local physical network that connects the different physical resources, e.g. servers and storage devices, in a cloud. The inter-cloud overlay in turn connects different clouds, e.g. in a hybrid cloud deployment, or clouds controlled by the same provider but hosted at different locations. The overlay approach provides flexibility and the ability to control communication paths over public networks, e.g. the Internet, without having to acquire access to the physical routers and links that compose the network.

The cognitive packet network (CPN) [30]-[35] is a self-aware packet routing protocol that implements many of the principles of pervasive monitoring in a network context in order to measure its performance and the conditions of the network in a distributed manner. Based on its measurements, CPN adaptively find the best routes according to given QoS criteria such as packet delay. In CPN, users of the network specify their QoS criteria, and the routing algorithm is directly related to the QoS desired and specified by the user. CPN supports multiple concurrent flows with different QoS criteria, which is done at the software level and therefore reduces the complexity of routing. With CPN, routing decisions are distributed at each node in the network, and they are taken based on adaptive learning techniques employing random neural networks (RNNs) [36]-[38] and reinforcement learning [39]-[41]. CPN uses three types of packets:

- Smart packets (SPs) for exploring the network,
- Dumb packets (DPs), which are source-routed, to carry the payload, and
- Acknowledgments (ACKs) to collect measurements performed by the SPs and DPs and to train the neural networks in the nodes as explained below.

SPs are generated by the source node on demand, i.e. when the user requests to create a new path with a given QoS goal or when he wants to discover parts of the network state. Note that the QoS goal can be a combination of multiple QoS criteria, such as latency and loss rate. SPs are used to explore the network and discover the best routes satisfying the given QoS goal. At each hop, SPs are routed based on the experiences of previous packets, employing a learning algorithm. One of the most successful learning methods in CPN employs reinforcement learning on RNNs. The RNN is a neural network model inspired by biology, and it is characterized by positive (excitatory) and negative (inhibitory) signals in the form of spikes of unit amplitude which are exchanged between neurons and alter the potential of the neurons. A neuron is connected to one or more neurons, forming a neural network; each neural link has a weight, which can be positive or negative.

In the RNN, the state q_i of the ith neuron represents the probability that it is excited, and satisfies the following system of non-linear equations:

$$q_i = \frac{\lambda^+(i)}{r(i) + \lambda^-(i)}$$

where

$$\lambda^+(i) = \sum_j q_j w_{ji}^+ + \Lambda_i^+$$

$$\lambda^-(i) = \sum_j q_j w_{ji}^- + \Lambda_i^-$$

$$r(i) = \sum_j (w_{ij}^+ + w_{ij}^-)$$

w_{ji}^+ is the rate at which neuron *j* sends excitation spikes to neuron *i* when *j* is excited, w_{ji}^- is the rate at which neuron *j* sends inhibition spikes to neuron *i* when *j* is excited, and $r(i)$ is the total firing rate of neuron *i*. Λ_i^+ and Λ_i^- are the constant rates of the external positive and negative signal arrivals at neuron *i*, respectively. These external signal arrivals follow stationary Poisson distributions. For an *N* neuron RNN, the parameters are the *N*x*N* weight matrices W^+ and W^- which need to be learned from the inputs.

Each node in the CPN stores an RNN for each QoS goal and source-destination pair. Each RNN has a neuron for each communication link at the node. The RNN is used to make adaptive decisions regarding the routing of SPs by routing the SPs probabilistically according to the weights of the neurons, which are updated using reinforcement learning. The QoS goal of the RNN is expressed as a function to be minimized, and the reward used in the reinforcement learning algorithm is simply the inverse of this function. Each received ACK triggers an update of the RNN based on the performance metrics observed by the SP or DP that resulted in the ACK.

As the CPN self-monitors and self-manages the communication paths, it can autonomously adapt to changing conditions in the network, such as congestion and failures. Its self-* properties make the CPN a good solution for the monitoring of communications in the cloud, and for autonomous adaptation of overlay paths. In addition to monitoring the network links and routers, CPN can also be used as the basis of a pervasive monitoring solution employed to collect metrics from the overlay nodes. We will discuss whether and how CPN can be adapted into a pervasive monitoring solution in our future reports on its design.

3 CLOUD MONITORING TOOLS AND SERVICES

In the context of cloud management, monitoring cloud infrastructures is a critical task. Monitoring allows us to obtain insights into the system and the health of the running applications, and perform capacity planning in order to improve scalability and, as a result, take adaptation decisions based on monitoring data.

As discussed in the previous section, a pervasive monitoring solution needs to perform both high-level (application-level) and low-level (infrastructure-level) monitoring. Infrastructure-level monitoring involves collecting metrics related to the CPU, memory, disk, network, etc., from the cloud infrastructure via probes deployed inside VMs and physical nodes. Probes deployed inside the VM can also be leveraged to collect application-level measurements.

In this section, we provide a review of the popular cloud monitoring tools and services currently employed in the industry. This review is not exhaustive and focuses on monitoring tools that are potential candidates for the realization of a pervasive monitoring solution to support proactive and autonomous management of cloud resources. A more comprehensive review of existing monitoring tools can be found in [42].

3.1 Monitoring services

Low-level monitoring, which is performed to monitor the status of the cloud infrastructure, is already employed by the cloud providers as it is an essential component of their management and maintenance tasks. There are many tools available to the provider to set-up a low-level monitoring framework, and we review some of the more popular tools in the next section. As part of the managed cloud services they provide, some cloud providers also offer high-level monitoring services to their cloud consumers, which vary in capability from simple monitoring and alerting to customizable monitoring tailored to the user. This is part of the trend to offer monitoring as a service (MaaS) [43]. There are also third party services such as Monitis and CopperEgg that offer high-level monitoring of popular cloud providers. In this section, we provide a brief overview of some of the existing monitoring services. We note that this review is not exhaustive since our focus is on existing monitoring tools that can support pervasive monitoring for performance and resource management in the cloud, which requires low-level monitoring.

3.1.1 Amazon CloudWatch

Amazon Web Services (AWS)[2] is a collection of public cloud-based services offered by Amazon.com, which include computing, storage, database, analytics, application and deployment services. The most central and well-known of these services are the Amazon Elastic Compute Cloud (EC2)[3], which provides computing facilities in an IaaS model, and the Amazon Simple Storage Service (S3)[4], which provides redundant cloud storage. Among the deployment and management services that Amazon offers, there is a monitoring service called Amazon CloudWatch[5], which enables consumers to collect and track metrics from their applications and instances (VMs) on EC2 and other AWS.

CloudWatch provides certain metrics, such as CPU utilization, latency, and request counts by default, and users can supply their own custom application and system metrics via the CloudWatch API.

[2] http://aws.amazon.com/

[3] http://aws.amazon.com/ec2/

[4] http://aws.amazon.com/s3/

[5] http://aws.amazon.com/cloudwatch/

Metrics and CloudWatch functionality is accessible via the command line, web API, AWS SDK, and the AWS management console. CloudWatch also provides an auto-scaling feature where customers can automatically adapt their computing capacity to demand by dynamically adding and removing VM instances on EC2. The service is available for free with an AWS account, but users can pay to access more advanced features and to monitor more metrics.

3.1.2 Rackspace cloud monitoring

Rackspace[6] is a cloud provider that offers cloud computing, storage, and networking services. They offer a RESTful API-driven high-level cloud monitoring service which can monitor any resource that can be accessed over the Internet, and this service is also available for applications and services hosted by other providers. Their monitoring solution typically polls the monitored resources, e.g. servers, from multiple data centres, but they also provide a monitoring agent that users can install on their VM instances and integrate with their web applications in order to allow local monitoring. Customers with a Rackspace account with managed service level get limited monitoring for free, and can pay for additional monitoring features.

3.1.3 Monitis

Monitis[7] offers network and IT systems monitoring services, including cloud monitoring services for AWS, Rackspace and GoGrid[8]. Their agent-based cloud monitoring service allows users to track their VM instances, set-up events and receive notifications in cases of VM failures, and configure monitoring services that should be started with a new VM instance. Monitis also provides an SDK and an API to their monitoring services, enabling users to write their custom scripts.

3.1.4 CopperEgg

CopperEgg[9] provides SaaS-based cloud management and monitoring services for servers and applications. Their primary product is RevealCloud, which provides real-time monitoring for many operating systems, including Linux, Windows and Mac OS X. It supports a notably low update frequency of 5 seconds, a very easy-to-install monitoring agent and a web interface. CopperEgg also allows the customers to store their monitoring data for up to one year.

3.2 Monitoring tools

3.2.1 Collectl

Collectl[10] collects system information about the CPU, memory, disk, and the network. It is a lightweight monitoring tool but it provides only basic monitoring. The monitoring abilities of Collectl are provided by plug-ins, and therefore it does not have any external dependencies. However, most of the plug-ins is for Linux only, and this limits Collectl to Linux-based systems. Collectl does not generate graphs, but there are plug-ins available for Ganglia and Graphite that can be used to create graphs based on monitoring data collected by Collectl.

[6] http://www.rackspace.com/

[7] http://www.monitis.com/

[8] http://www.gogrid.com/

[9] http://copperegg.com/

[10] http://collectl.sourceforge.net/index.html

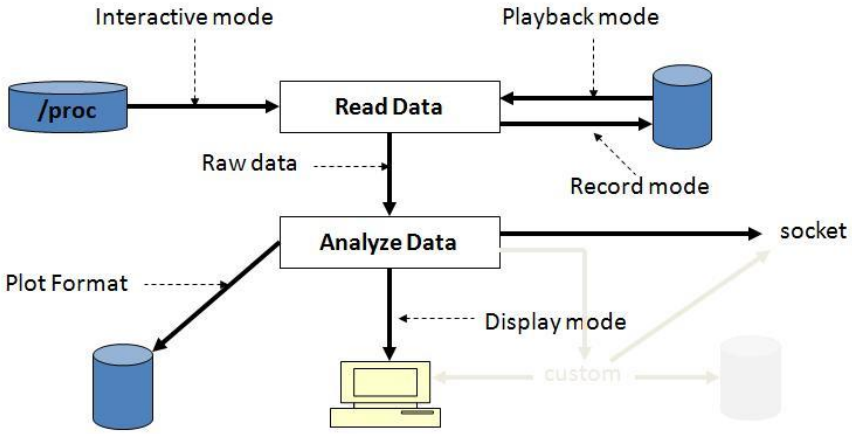

Figure 1: Basic architecture of Collectl

3.2.2 System Activity Report (SAR)

SAR[11] is a tool developed for the Solaris operating system that was later ported to other UNIX systems such as Linux. It offers very basic monitoring of the CPU, disk, and I/O, and overall system performance. It offers a basic graphic representation tool called SAG (System Activity Grapher).

3.2.3 System Information Gatherer and Reporter (SIGAR)

The SIGAR API[12] offers a portable interface for gathering system information such as CPU, processes, disk, network, etc. for different operating systems such as Linux, Solaris, Windows, Mac OS X, etc. The API offers bindings for different programming languages such as Java, Python, Ruby, Erlang, C#, etc. SIGAR has been incorporated into third party projects, but its development appears to have stopped in the last years.

3.2.4 Ganglia

Ganglia[13] is a scalable distributed monitoring system for high-performance computing systems, such as computing clusters and grids. It is able to federate the monitoring information coming from several clusters by creating a tree of point-to-point links among representative nodes, as shown in Figure 2. The communication between the probes and the different monitoring aggregators relies on a multicast-based publish-subscribe protocol. It employs popular and proven technologies such as XML to represent data, XDR to transfer monitoring data in a compact form, and the RRDtool[14] for data storage and visualization.

[11] http://sebastien.godard.pagesperso-orange.fr/

[12] https://support.hyperic.com/display/SIGAR/Home

[13] http://ganglia.sourceforge.net/

[14] http://oss.oetiker.ch/rrdtool/

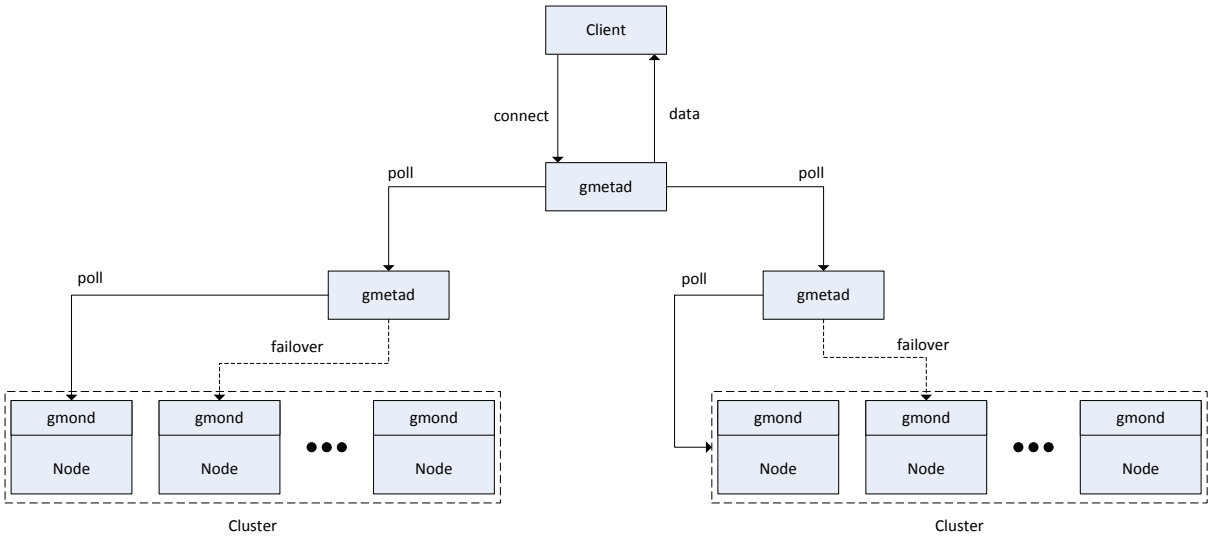

Figure 2: Deployment diagram of Ganglia components

Ganglia supports diverse operating systems such as UNIX-based ones (Linux, Solaris, HP-UX, Mac OS X, etc.) and Windows. Implementations for several metrics are supplied out-of-the-box, but custom clients can also be created using the offered specification. Ganglia also provides a graphical interface for visualization of the metrics, in addition to an advance alert system.

3.2.5 Nagios

Nagios[15] is designed to monitor applications, services, operating systems, network protocols, system metrics, and infrastructure components with a single tool. It has a complete alert system that allows the configuration of actions to take when given conditions are satisfied, and thus helps to react to problems in the infrastructure and decrease downtime.

Nagios supports Windows systems via add-ons and Linux systems, although other UNIX-based systems can still use Nagios. It offers a complete graphical tool that allows visualization of different monitoring metrics and events. An example of integrating Nagios into a more comprehensive cloud monitoring solution is provided in [44].

3.2.6 Monit

Monit[16] is a utility for managing and monitoring processes, programs, files, directories, and devices on a Unix system. It is designed to conduct automatic maintenance and repair and can execute meaningful causal actions in error situations.

It is able to monitor files, directories, daemons, and devices such as memory or CPU. It logs all its actions in log files or uses syslog for this purpose. It is only available in UNIX-based systems, including Linux.

By default, Monit does not offer historical storage of monitoring data. For that purpose, the developers offer the complementary commercial tool M/Monit.

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

[16] http://mmonit.com/

3.2.7 Zabbix

Zabbix[17] is an open source tool designed for monitoring networks and applications. It is designed to monitor and track the status of various network services, servers and network hardware. It uses SQL databases (for example MySQL, PostgreSQL, etc.) to store historical data, and presents a web front-end and an API to access the monitoring information.

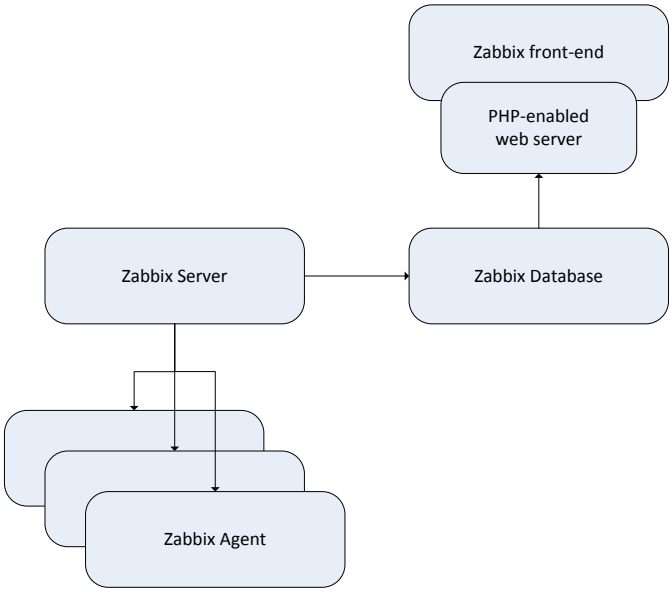

Figure 3: Block diagram of Zabbix

It can monitor services like http and smtp out of the box and it provides a Zabbix agent that can be installed in the hosts in order to monitor things like CPU, disk, network, etc. The agents can use a variety of protocols, including IPMI, ssh, TCP, etc., that improves their integration into existing infrastructures. Zabbix can be used in both Linux and Windows systems.

3.2.8 collectd

collectd[18] is a system daemon that collects system performance statistics periodically, and provides several mechanisms to store the monitoring data, for example using the commonly employed RRDtool. It is able to gather many different types of statistics, from CPU utilization to custom probes. collectd was designed to be extensible, and by default it only provides a limited threshold checking facility. However, its functionalities can easily be extended via plug-ins, and there are plug-ins to interface collectd with higher-level monitoring tools such as Nagios and Graphite[19] in order to leverage additional capabilities provided by these tools. For example, Nagios can be used to provide more complex alert and reaction mechanisms based on data provided by collectd, and Graphite can be used to provide graphical visualization of monitoring data collected by collectd.

collectd natively supports Linux and UNIX-based operating systems, and a third-party client (SSC Serv[20]) can be installed on Windows systems to interface with collectd. collectd also supports the

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

[18] http://collectd.org/

[19] http://graphite.wikidot.com/

[20] http://ssc-serv.com/

collection of metrics from network devices via SNMP, which is provided by most networking equipment.

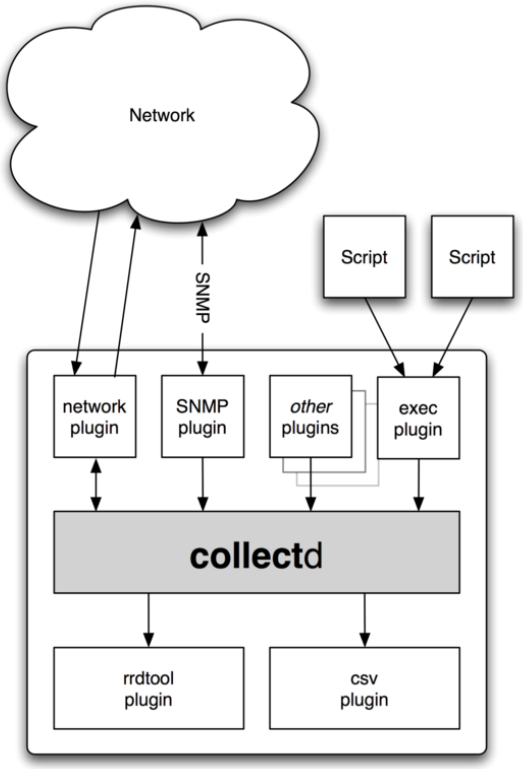

Figure 4: collectd architecture

3.2.9 sFlow

sFlow[21] is a sampling technology to monitor network traffic. It is widely supported in the networking industry and devices such as routers, switches, etc. provide native interfaces to send information to sFlow collectors. Although by itself sFlow would not be able to provide pervasive monitoring, it can be integrated with some of the other presented solutions, such as Ganglia and Nagios, which offer native support to collect information from an sFlow server.

3.2.10 Munin

Munin[22] is a client-server based monitoring tool capable of monitoring computers, networks and other resources that can be represented as numerical values. The main objective of the Munin project is to be able to easily represent those results as graphs from stored RRD files.

Munin works mainly in Linux systems, although it can receive inputs from Windows systems thanks to third-party plug-ins. Using the plug-in architecture provided, it can be extended to perform actions based on the gathered monitoring information.

[21] http://www.sflow.org/

[22] http://munin-monitoring.org/

3.3 License comparison

The following table presents a license comparison between the presented solutions:

Monitoring Solution	License	Notes
Collectl	Artistic Licence, GPLv2	
System Activity Report	GPLv2	
SIGAR	Apache License v2	
Ganglia	BSD	
Nagios	GPLv2	Also offered is a set of commercial tools that build upon the open source solution provided by Nagios.
Monit	AGPL	Also offered is a commercial tool called M/Monit that can monitor more than one host.
Zabbix	GPL	
collectd	GPLv2	
sFlow	sFlow License[23]	The product can be used free of charge, but the license belongs to InMon Corporation.
Munin	GPLv2	

Table 1: License comparison of the reviewed monitoring tools

3.4 Integration with the pervasive monitoring solution

Out of the monitoring tools that we have reviewed, Zabbix, Ganglia and Nagios are the ones that are the most suited for the realization of a pervasive monitoring solution as required in PANACEA. They offer a single tool for the monitoring of different server and application metrics, enable the user to set-up and receive alerts and notifications, provide aggregation mechanisms for the monitored metrics, and keep a historical record of the monitoring data. All three projects are widely used and have strong community support.

collectd offers a good solution to monitor one host, but it will need to rely on a third party tool to aggregate the metrics coming from different hosts and applications, to set-up complex alerts and for the visualization and analysis of the data.

We believe that out of the monitoring tools we have reviewed, collectd is the best candidate to form the basis of the monitoring agent. A monitoring agent will be installed at every physical and virtual resource, e.g. physical hosts and VMs that we want to monitor. At the core of the monitoring agent, we can use collectd to provide the local probing mechanism. collectd is attractive for this purpose

[23] http://www.sflow.org/developers/licensing.php

since it is written in C as an operating system daemon, and therefore it is lightweight and offers high-resolution probing, supporting a probing frequency of up to 10 seconds for many concurrent probes. The monitoring performance of collectd is good, supporting a large number of probes, metrics and very large datasets. It can probe at different layers, e.g. at the application, middleware and OS layers and thus can support both high-level and low-level monitoring. It is also customizable via plug-ins, and there is an active community behind it.

Another advantage of collectd is that it is distributed, i.e. there is no central controlling or reporting entity. Measured metrics can be written to local files or a local database, e.g. via RRDtool, or they can be reported over the network to one or more collectd servers. In the case where data is collected at one host and stored at another, the collectd instance doing the measurements and sending them is called the client, and the one receiving and storing them the server. Reporting of monitoring data is highly configurable via networking, and collectd supports multicast and unicast communications, in addition to a proxy mode. Cryptographic extensions are supported by the network plug-in, and therefore the network traffic can be signed and encrypted to increase security of monitoring over public networks. The multicast communication mode is especially useful to decouple the communication end-points, i.e. the server does not need to know the addresses of the clients in order to be able to communicate with them, and vice versa.

collectd's customizable plug-in mechanism and its support for various networking modes lends it flexibility and thus allows us to adapt collectd to our final design of the pervasive monitoring solution, which will be specified in a later document. In the pervasive monitoring solution, the basic probing and communication capabilities provided by collectd can be wrapped by a software agent in order to provide an interface between collectd and the monitoring solution. We think that this may be the most appropriate approach for monitoring data that is consumed by another autonomous management system such as the one being developed within PANACEA. In this case, the monitoring agent will receive requests and commands via an API that will be developed, and in turn configure its collectd instance according to the received requests.

For human consumption of data, collectd needs to be interfaced with a higher-level monitoring tool in order to add alerting mechanisms, and visualization and analysis capabilities. Nagios appears to be the best choice for this purpose due to its extensibility and flexibility. Measurements from collectd can be provided to Nagios via the collectd-nagios plug-in, and alerts for the user can be configured and transmitted by Nagios. Nagios would also present a front-end to the human user for visualization of metrics and for the configuration of the collectd instances via the monitoring agents. In order to support configuration of the monitoring agent via Nagios, Nagios will need to be extended [44]. In the design of our pervasive monitoring solution, we will prioritize capabilities needed by the PANACEA management system. However, providing an interface for human users is desirable, and we will discuss the details of how Nagios can be leveraged for this purpose in a later document.

4 MONITORING CAPABILITIES OF OPENNEBULA

In order to build and manage a cloud, it is essential to have good cloud management software. The cloud community currently favours two open-source industry standard cloud management toolkits: OpenStack[24] and OpenNebula[25]. Both projects provide tools and services in order to set-up, configure and manage a cloud platform, supporting public, private and hybrid clouds. In the design and development of the proactive and autonomous cloud management system proposed in the PANACEA project, the cloud management toolkit plays an important role as it provides the "dumb" interface that will be employed by the "smart" management system in order to control the cloud and actuate its decisions. We have selected to employ OpenNebula as the basis of the cloud management system since it is easier to install and configure than OpenStack, provides easier integration of existing components due to its central project governance model, and provides a more mature software stack.

As all cloud management software, OpenNebula relies on monitoring the cloud infrastructure in order to assess the state of resources and provide related information to the cloud manager. In this section, we provide an overview of the current monitoring capabilities provided by OpenNebula, and consider how a separate pervasive monitoring solution can be integrated with it.

4.1 Overview of the OpenNebula monitoring system

The monitoring subsystem[26] gathers information relative to the hosts and the virtual machines, such as the host status, basic performance indicators, as well as VM status and capacity consumption. This information is collected by executing a set of static probes provided by OpenNebula. The output of these probes is sent to OpenNebula in two different ways: using a push or a pull paradigm.

4.1.1 The pull model

When using this mode, OpenNebula periodically and actively queries each host and executes the probes via ssh. In KVM and Xen this means establishing an ssh connection to each host and executing several scripts to retrieve this information. Note that VMware uses the VI API for this, instead of an ssh connection.

This mode is limited by the number of active connections that can be made concurrently, as hosts are queried sequentially. This monitoring model is adequate when the infrastructure has a low number of hosts (e.g. 50 or less) and if low update frequencies are acceptable (e.g. for 50 hosts, the monitoring period would typically be around 5 minutes). It is also appropriate if hosts communicate through an insecure network.

4.1.2 The push model

In the UDP-based push model, each host periodically sends monitoring data via UDP to the frontend, which collects it and processes it in a dedicated module. This distributed monitoring system resembles the architecture of dedicated monitoring systems, using a lightweight communication protocol, and a push model.

This model is highly scalable and its limit (in terms of number of VMs monitored per second) is bounded by the performance of the server running the OpenNebula daemon (*oned*) and the database server. The push mode is limited to Xen and KVM, and it is the default monitoring mode in

[24] https://www.openstack.org/

[25] http://opennebula.org/

[26] http://docs.opennebula.org/4.4/administration/monitoring/mon.html

OpenNebula versions 4.4 and up. It is better suited for infrastructures with medium-to-high number of hosts (e.g. more than 50), providing a responsive system that supports high update frequencies. When the push model is used, hosts must communicate through a secure network since UDP datagrams are not encrypted and their origin is not verified.

In order to enable UDP-based push monitoring, OpenNebula starts an OpenNebula-specific *collectd* daemon, which bears no relation to the collectd daemon we reviewed in the previous section. This daemon runs in the frontend host that listens for UDP connections on port 4124. In the first monitoring cycle, OpenNebula connects to the host using ssh and starts a daemon that will execute the probe scripts as in the ssh-based pull model, and sends the collected data to the collectd daemon in the frontend with a user configurable period. This way the monitoring subsystem doesn't need to make new ssh connections to the hosts when it needs data.

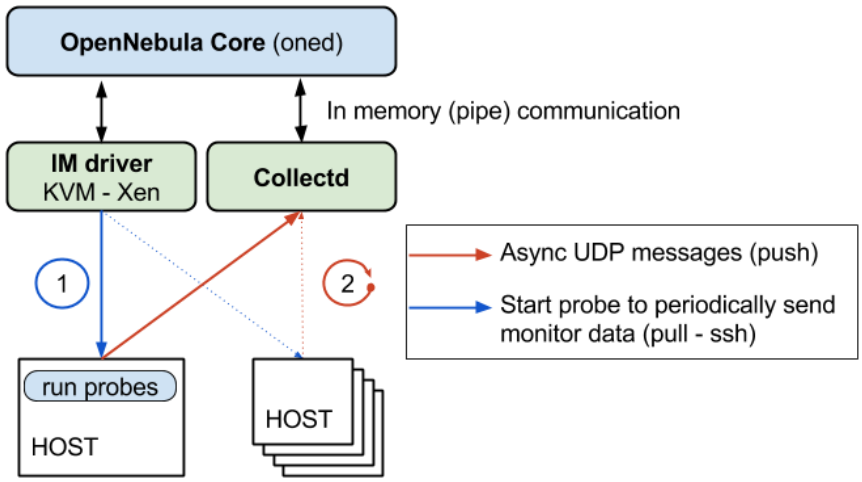

Figure 5: OpenNebula monitoring architecture in the push model

If the agent stops in a specific host, OpenNebula will detect that no monitoring data is received from that host and will automatically fall back to the ssh-pull model, thus starting the agent again in the host.

4.2 Customizing monitoring drivers

Monitoring drivers (or IM drivers) collect host and VM monitoring data by executing a set of probes in the hosts. These data are either actively queried by OpenNebula (pull model) or sent periodically by an agent running in the hosts to the frontend (push model).

The default probes are installed in the frontend in the following path:

- KVM and Xen:`/var/lib/one/remotes/im/<hypervisor>-probes.d`
- VMware and EC2: `/var/lib/one/remotes/im/<hypervisor>.d`

In the case of KVM and Xen, the probes are distributed to the hosts, therefore if the probes are changed, they must be distributed to the hosts by running `onehostsync`.

An IM diver is composed of one or more scripts that write to `stdout` information in a (key,value) format:

```
KEY1="value"
KEY2="another value with spaces"
```

The drivers receive the following parameters:

Argument	Description
$1	The name of the hypervisor (kvm, xen, etc.)
$2	Path of the *datastores* directory in the host
$3	The port on which *collectd* is listening
$4	The period for the UDP-push model, in seconds
$5	Id of the host
$6	Name of the host

Table 2: Parameters accepted by the OpenNebula monitoring drivers

It is possible to add any key and value for later use in the *rank* and *requirements* fields used for scheduling, but there are some basic values that should be output:

Key	Description
HYPERVISOR	Name of the hypervisor of the host, useful for selecting the hosts with a specific technology
TOTALCPU	Number of CPUs multiplied by 100, e.g. a 16-core machine will have a value of 1600
CPUSPEED	Speed of the CPUs, in MHz
TOTALMEMORY	Maximum memory that can be used for VMs. It is advised to subtract the memory used by the hypervisor.
USEDMEMORY	Memory used, in KB
FREEMEMORY	Available memory for VMs at the moment, in KB
FREECPU	Percentage of idling CPU multiplied by the number of cores (if 50% of the CPU is idling in a 4 core machine the value will be 200)
USEDCPU	Percentage of used CPU multiplied by the number of cores
NETRX	Number of bytes received from the network
NETTX	Number of bytes sent to the network

Table 3: Values output by the monitoring drivers

There are some examples of probes under the `/var/lib/one/remotes/im` directory. The scripts should also provide information about the VMs running in the host. This is useful as it will only need one call to gather all that information about the VMs in each host. The output should be in this form:

```
VM_POLL=YES
VM=[
  ID=86,
  DEPLOY_ID=one-86,
  POLL="USEDMEMORY=918723 USEDCPU=23 NETTX=19283 NETRX=914 STATE=a"]
```

```
VM=[
  ID=645,
  DEPLOY_ID=one-645,
  POLL="USEDMEMORY=563865 USEDCPU=74 NETTX=2039847 NETRX=2349923 STATE=a" ]
```

The first line (`VM_POLL=YES`) is used to indicate to OpenNebula that VM information will follow. Then the information about the VMs is output in the following format:

Key	Description
ID	ID of the VM. It can be -1 in case this VM was not created by OpenNebula.
DEPLOY_ID	Name of the hypervisor or VM identifier
POLL	VM monitoring information, in the format described below

Table 4: VM information provided by monitoring

The monitoring information in the `POLL` variable includes the following variables:

Variable	Description
STATE	State of the VM, indicated by a single character and described in the next table
USEDCPU	Percentage of CPU consumed. Two fully-consumed CPUs would have a value of 200.
USEDMEMORY	Used memory, in KB
NETRX	Number of bytes received from the network
NETTX	Number of bytes sent to the network

Table 5: Variables returned in the poll value in the VM information

Possible values for `STATE` are:

State	Description
N/A	State error detected, i.e. the monitoring could be done but it returned an unexpected output
a	Active. The VM is alive, but not necessarily running since it could be blocked, booting, etc.
p	Paused
e	Error. The VM crashed or its deployment failed.
d	Disappeared. The hypervisor does not know the VM anymore.

Table 6: Possible VM states as returned by the monitoring driver

Information manager (IM) drivers are also responsible to collect datastore sizes and their available space. The following values are provided:

Variable	Description
DS_LOCATION_USED_MB	Disk space used in the datastore location, in MB
DS_LOCATION_TOTAL_MB	Total space reserved for the datastore location, in MB
DS_LOCATION_FREE_MB	Free space in the datastore location, in MB
ID	Id of the datastore. This is the same as the name of the directory.
USED_MB	Disk space used for the datastore, in MB
TOTAL_MB	Total space for the datastore, in MB
FREE_MB	Free space for the datastore, in MB

Table 7: Information provided on datastores

The datastore location is the path where the datastores are mounted. By default it is `/var/lib/one/datastores`, but a different location can be specified in the second parameter of the script call.

4.3 Creating a new monitoring driver

OpenNebula provides two IM probe execution engines:

- `one_im_sh` is used to execute probes in the frontend. VMware uses this engine as it collects data via an API call executed in the frontend.
- `one_im_ssh` is used when probes need to be run remotely in the hosts. This is the case for Xen and KVM.

Both `one_im_sh` and `one_im_ssh` require an argument, which indicates the directory that contains the probes. This argument is appended with ".d". For example, for VMware the execution engine is `one_im_sh` (local execution) and the argument is `vmware`, and therefore the probes that will be executed in the hosts are located in `/var/lib/one/remotes/im/vmware.d`.

If the new IM driver wishes to use the OpenNebula *collectd* component, it needs to:

- Use `one_im_ssh`.
- The `/var/lib/one/remotes/im/<im_name>.d` should only contain two files, the same that are provided by default inside `kvm.d` and `xen.d`, which are: `collectd-client_control.sh` and `collectd-client.rb`.
- The probes should be placed in the `/var/lib/one/remotes/im/<im_name>-probes.d` folder.

4.4 Integrating Ganglia with OpenNebula

OpenNebula provides drivers to get information about hosts and VMs from Ganglia. These drivers can potentially improve the scalability of the monitoring for OpenNebula since they don't use the ssh-pull model in order to get the information. However, they require more work from the system administrator as Ganglia should be properly configured and `cron` jobs must be installed on the nodes to provide VM information to the Ganglia system.

VM information is not gathered by Ganglia so we need to provide a script to get that information. The same probe that gets information for Xen and KVM (`/var/lib/one/remotes/vmm/xen/poll` or `/var/lib/one/remotes/vmm/kvm/poll`) can be used to get this data and push it to Ganglia. This probe has to be copied to each of the nodes or put in a path visible by all the nodes. It will be executed by the cloud administrator, so a good place is the home of the `oneadmin` user (~). VM information then needs to be pushed to a metric called OPENNEBULA_VMS_INFORMATION, which is in turn pushed to Ganglia with the following command:

```
gmetric -n OPENNEBULA_VMS_INFORMATION -t string -v '$HOME/tmp/poll -kvm'
```

To make it refresh automatically (e.g. every minute), this command should be added to the `cron` system, so it is executed as a job periodically. The maximum size of Ganglia values is 1400 bytes by default. This currently limits the total amount of VMs monitored per host.

4.5 Application monitoring using OneGate

OneGate allows VM guests to push monitoring information to OpenNebula. Users and administrators can use it to gather metrics, detect problems in their applications, and trigger OneFlow elasticity rules. OneGate is a server that listens to http connections from the VMs. OpenNebula assigns an individual authorization token to each VM instance, and applications running inside the VM use this token to send monitoring metrics to OneGate. When OneGate checks the VM id and the token sent, the new information is placed inside the VM's user template section. This means that the application metrics are visible from the command line, Sunstone, or the APIs.

In order to use OneGate, the cloud administrator must first configure and start the OneGate server and the VM template must set the `context/token` attribute to `yes`:

```
CPU     = "0.5"
MEMORY  = "128"
DISK = [
  IMAGE_ID = "0" ]
NIC = [
  NETWORK_ID = "0" ]
CONTEXT = [
  TOKEN = "YES" ]
```

When this template is instantiated, OpenNebula will automatically add the ONEGATE_URL context variable, and a *token.txt* will be placed in the context CD-ROM. This *token.txt* file is only accessible from inside the VM.

```
[.../...]
CONTEXT=[
  DISK_ID="1",
  ONEGATE_URL="http://192.168.0.1:5030/vm/0",
  TARGET="hdb",
  TOKEN="YES" ]
```

The contextualization CD-ROM should contain the *context.sh* and *token.txt* files:

```
# mkdir /mnt/context
# mount /dev/hdb /mnt/context
# cd /mnt/context
# ls context.sh token.txt
# cat context.sh
# Context variables generated by OpenNebula
DISK_ID='1'
ONEGATE_URL='http://192.168.0.1:5030/vm/0'
TARGET='hdb'
TOKEN='yes'
```

```
# cat token.txt
yCxieDUS7kra7Vn9ILA0+g==
```

With that data, the VM can perform this http request message:

```
Request: PUT ONEGATE\_URL.
Headers: X-ONEGATE-TOKEN: token.txt contents.
Body: Monitoring values, in the usual ATTRIBUTE = VALUE OpenNebula
syntax.
```

For example, using the `curl` command:

```
$ curl -X "PUT" http://192.168.0.1:5030/vm/0 --header "X-ONEGATE-
TOKEN: yCxieDUS7kra7Vn9ILA0+g==" -d "APP_LOAD = 9.7"
```

The new metric is stored in the user template section of the VM:

```
$ onevm show 0
...
USER TEMPLATE
APP_LOAD="9.7"
```

4.6 Accessing monitoring information

The XML-RPC interface provides operations to obtain information from every resource it manages: `one.host.info`, `one.hostpool.info`, `one.vm.info`, `one.vmpool.info`, etc. It also provides operations to obtain monitoring information from hosts and VMs: `one.host.monitoring`, `one.hostpool.monitoring`, `one.vm.monitoring` and `one.vmpool.monitoring`. For VMs, it even provides operations to obtain history records: `one.vmpool.accounting`.

The OCCI interface provides list and show commands (both of them are GET methods) to respectively list the contents of each resource collection and to obtain information from each kind of resource it manages: STORAGE, NETWORK, COMPUTE, INSTANCE_TYPE, and USER. However, it does not provide monitoring information, besides the state in a COMPUTE resource (i.e. a VM).

The EC2 interface provides operations to describe images, instances, volumes, addresses, key pairs and tags. However, like OCCI, it provides the state of an instance (i.e. a VM) as the only monitoring information.

Finally, OpenNebula saves its state and lots of monitoring and accounting information in a persistent database, using MySQL or SQLite. Therefore, the database could be easily interfaced with any DB tool to extract monitoring information.

4.7 Integration of the pervasive monitoring solution and OpenNebula

The pervasive monitoring solution that will provide the measurements and alerts that will be used by the smart cloud management system can be provided directly to the management system, by-passing the OpenNebula system. However, OpenNebula itself requires basic monitoring for its management operations, and this type of monitoring can be done using the monitoring capabilities available in OpenNebula. The UDP-based push model is especially useful for this purpose. If during the design of the smart management system a need arises for the integration of the pervasive monitoring solution and OpenNebula, either in order to use OpenNebula as an interface to the data collected by the pervasive solution, or to satisfy the monitoring requirements of OpenNebula itself, this integration can be accomplished by writing a new driver for OpenNebula in order to interface the monitoring agents directly with OpenNebula, or by integrating OpenNebula with a third-party tool such as Nagios or Ganglia that acts as an intermediary between the monitoring agents and OpenNebula. In the case of Ganglia, this integration would be simplified by the fact that an OpenNebula monitoring driver is

already available for it. We leave whether we need to integrate OpenNebula and the pervasive monitoring solution that will be developed in PANACEA, and if yes, how the integration will be done, as future work since these design decisions will heavily depend on the overall architecture of the smart management system.

5 CONCLUSIONS

We presented the principles and requirements of pervasive cloud monitoring, focusing on monitoring activities necessary to support the autonomous and proactive management of cloud resources in order to improve the availability, reliability and performance of cloud services. We reviewed an agent-based design for a pervasive monitoring solution and self-aware adaptive overlay networks in order to monitor the network and construct a resilient monitoring architecture.

We reviewed some of the popular cloud monitoring services and tools in terms of their suitability to support pervasive monitoring. The collectd daemon is a lightweight solution that supports basic monitoring activities and provides high resolution measurements without causing significant load on the computing resources of its host. We will therefore use it in our monitoring solution as the basis of our local probes for the monitoring agent. We will also review more closely whether we can integrate any of the existing monitoring tools we have identified as suitable, i.e. Zabbix, Nagios and Ganglia, in the development of our pervasive monitoring solution.

REFERENCES

[1] M. Armbrust, A. Fox, R. Griffth, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, M. Zaharia. "A view of cloud computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50-58, 2010.

[2] M. Ahmed, A. Sina, Md. R. Chowdhury, M. Ahmed, Md. M. H. Rafee. "An advanced survey on cloud computing and state-of-the-art research issues," *International Journal of Computer Science Issues*, vol. 9, no. 1, 2012.

[3] S. Meng, L. Liu, T. Wang. "State monitoring in cloud datacenters," *IEEE Transactions on Knowledge and Data Engineering*, vol. 23, no. 9, pp. 1328-1344, Sep. 2011.

[4] T. Lorimer, R. Sterritt. "Autonomic management of cloud neighborhoods through pulse monitoring," in *Proc. 5th IEEE International Conference on Utility and Cloud Computing (UCC'12)*, pp. 295-302, Nov. 2012.

[5] G. Aceto, A. Botta, W. de Donato, A. Pescape. "Cloud monitoring: a survey," *Computer Networks*, vol. 57, no. 9, pp. 2093-2115, Jun. 2013.

[6] F.-F. Han, J.-J. Peng, W. Zhang, Q. Li, J.-D. Li, Q.-L. Jiang, Q. Yuan. "Virtual resource monitoring in cloud computing," *Journal of Shanghai University (English Edition)*, vol. 15, no. 5, pp. 381-385, 2011.

[7] B. Konig, C. J. M. Alcaraz, J. Kirschnick. "Elastic monitoring framework for cloud infrastructures," *IET Communications*, vol. 6, no. 10, pp. 1306-1315, Jul. 2012.

[8] L. Atzori, F. Granelli, A. Pescape. "A network-oriented survey and open issues in cloud computing," in *Cloud Computing: Methodology, Systems, and Applications*, pp. 91-108, CRC Press, 2011.

[9] N. M. Chowdhury, R. Boutaba. "A survey of network virtualization," *Computer Networks*, vol. 54, no. 5, pp. 862-876, 2010.

[10] P. Mell, T. Grance. "The NIST definition of cloud computing," *NIST Special Publication 800-145*, Sep. 2011. [Online]. Available: http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf

[11] J. Montes, A. Sanchez, B. Memishi, M. S. Pereze, G. Antoniu. "GMonE: a complete approach to cloud monitoring," *Future Generation Computer Systems*, vol. 29, no. 8, pp. 2026-2040, Oct. 2013.

[12] P. Sharma, S. Chatterjee, D. Sharma. "CloudView: Enabling tenants to monitor and control their cloud instantiations," in *Proc. 2013 IFI/IEEE International Symposium on Integrated Network Management (IM'13)*, pp. 443-449, May 2013.

[13] Z. ur Rehman, O. K. Hussain, S. Parvin, F. K. Hussain. "A framework for user feedback based cloud service monitoring," in *Proc. 6th International Conference on Complex, Intelligent and Software Intensive Systems (CISIS'12)*, pp. 257-262, Jul. 2012.

[14] R. Aversa, L. Tasquier, S. Venticinque. "Management of cloud infrastructures through agents," in *Proc. 3rd International Conference on Emerging Intelligent Data and Web Technologies (EIDWT'12)*, pp. 46-52, Sep. 2012.

[15] K. Alhamazani, R. Ranjan, F. Rabhi, L. Wang, K. Mitra. "Cloud monitoring for optimizing the QoS of hosted applications," in *Proc. 4th IEEE International Conference on Cloud Computing Technology and Science (CloudCom'12)*, pp. 765-770, Dec. 2012.

[16] J. Povedano-Molina, J. M. Lopez-Vega, J. M. Lopez-Soler, A. Corradi, L. Foschini. "DARGOS: A highly adaptable and scalable monitoring architecture for multi-tenant clouds," *Future Generation Computer Systems*, vol. 29, no. 8, pp. 2041-2056, Oct. 2013.

[17] J. Spring. “Monitoring cloud computing by layer, part 1,” *IEEE Security & Privacy*, vol. 9, no. 2, pp. 66-68, Mar. 2011.

[18] J. Spring. “Monitoring cloud computing by layer, part 2,” *IEEE Security & Privacy*, vol. 9, no. 3, pp. 52-55, May 2011.

[19] T. S. Somasundaram, K. Govindarajan. “Cloud monitoring and discovery service (CMDS) for IaaS resources,” in *Proc. 3rd International Conference on Advanced Computing (ICoAC'11)*, pp. 340-345, Dec. 2011.

[20] Y. Meng, Z. Luan, Z. Cheng, D. Qian. “Differentiating data collection for cloud environment monitoring,” in *Proc. 2013 IFIP/IEEE International Symposium on Integrated Network Management (IM'13)*, pp. 868-871, May 2013.

[21] G. Katsaros, G. Kousiouris, S. V. Gogouvitis, D. Kyriazis, A. Menychtas, T. Varvarigou. “A self-adaptive hierarchical monitoring mechanism for clouds,” *Journal of Systems and Software*, vol. 85, no. 5, pp. 1029-1041, May 2012.

[22] J. S. Ward, A. Baker. “Monitoring large-scale cloud systems with layered gossip protocols,” *arXiv Computing Research Repository*, vol. abs/1305.7403, May 2013.

[23] H. T. Kung, C.-K. Lin, D. Vlah. “CloudSense: Continuous fine-grain cloud monitoring with compressive sensing,” in *Proc. 3rd USENIX Workshop on Hot Topics in Cloud Computing (HotCloud'11)*, Jun. 2011.

[24] X. Lu, J. Yin, Y. Li, S. Deng. M. Zhu. “An efficient data dissemination approach for cloud monitoring,” *Service-Oriented Computing*, series LNCS, vol. 7636, pp. 733-747, 2012.

[25] C. Canali, R. Lancellotti. “Improving scalability of cloud monitoring through PCA-based clustering of virtual machines,” *Journal of Computer Science and Technology*, vol. 29, no. 1, pp. 38-52, Jan. 2014.

[26] C. Canali, R. Lancellotti. “Automated clustering of VMs for scalable cloud monitoring and management,” in *Proc. 20th International Conference on Software, Telecommunications and Computer Networks (SoftCOM'12)*, pp. 1-5, Sep. 2012.

[27] E. Gelenbe, “Steps toward self-aware networks,” *Communications of the ACM*, vol. 52, no. 7, pp. 66-75, Jul. 2009.

[28] E. Gelenbe, R. Lent, A. Nunez. “Self-aware networks and QoS,” *Proceedings of the IEEE*, vol. 92, no. 9, pp. 1478-1489, Sep. 2004.

[29] R. Lent, O. H. Abdelrahman, G. Gorbil. “A low-latency and self-adapting application layer multicast,” *Computer and Information Sciences*, vol. 62, series LNEE, pp. 169-172. Sep. 2010.

[30] G. Sakellari. “The cognitive packet network: a survey,” *The Computer Journal*, vol. 53, no. 3, pp. 268-279, Jun. 2009.

[31] E. Gelenbe, Z. Xu, E. Seref. “Cognitive packet networks,” in *Proc. 11th International Conference on Tools with Artificial Intelligence*, pp. 47-54, Nov. 1999.

[32] E. Gelenbe, R. Lent, Z. Xu. “Measurement and performance of a cognitive packet network,” *Journal of Computer Networks*, vol. 37, no. 6, pp. 691-701, Dec. 2001.

[33] E. Gelenbe, M. Gellman, R. Lent, P. Liu, P. Su. “Autonomous smart routing for network QoS,” in *Proc. 1st International Conference on Autonomic Computing*, pp. 232-239, May 2004.

[34] E. Gelenbe. “Sensible decisions based on QoS,” *Computational Management Science*, vol. 1, no. 1, pp. 1-14, Dec. 2003.

[35] E. Gelenbe, M. Gellman, P. Su. “Self-awareness and adaptivity for QoS,” in *Proc. 2003 IEEE International Symposium on Computers and Communications*, pp. 3-9, Jun. 2003.

[36] E. Gelenbe. “Random neural networks with negative and positive signals and product form solution,” *Neural Computation*, vol. 1, no. 4, pp. 502-510, 1989.

[37] E. Gelenbe, J. M. Fourneau. “Random neural networks with multiple classes of signals,” *Neural Computation*, vol. 11, no. 4, pp. 953-963, May 1999.

[38] E. Gelenbe, S. Timotheou. “Random neural networks with synchronised interactions,” *Neural Computation*, vol. 20, no. 9, pp. 2308-2324, Sep. 2008.

[39] E. Gelenbe. “Learning in the recurrent random neural network,” *Neural Computation*, vol. 5, no. 1, pp. 154-164, Jan. 1993.

[40] E. Gelenbe, K. Hussain. “Learning in the multiple class random neural network,” *IEEE Transactions on Neural Networks*, vol. 13, no. 6, pp. 1257-1267, Nov. 2002.

[41] U. Halici. “Reinforcement learning with internal expectation for the random neural network,” *European Journal of Operational Research*, vol. 126, no. 2, pp. 288-307, Oct. 2000.

[42] K. Alhamazani, R. Ranjan, K. Mitra, F. A. Rabhi, S. U. Khan, A. Guabtni, V. Bhatnagar. “An overview of the commercial cloud monitoring tools: research dimensions, design issues, and state-of-the-art,” *arXiv Computing Research Repository*, vol. abs/1312.6170, Dec. 2013.

[43] S. Meng, L. Liu. “Enhanced monitoring-as-a-service for effective cloud management,” *IEEE Transactions on Computers*, vol. 62, no. 9, pp. 1705-1720, Sep. 2013.

[44] S. de Chaves, R. Uriarte, C. Westphall. “Toward an architecture for monitoring private clouds,” *IEEE Communications Magazine*, vol. 49, no. 12, pp. 130-137, Dec. 2011.

