Grant Agreement No.:
Instrument: Collaborative
Call Identifier: FP7-ICT-

610764
Project
2013-10

# PANACEA

## Proactive Autonomic Management of Cloud Resources

# D2.4: Architecture of the Autonomic Cloud with Pervasive Monitoring

Version: 0.3

| | |
|---|---|
| Work package | WP 2 |
| Task | Task 2.3 |
| Due date | 01/07/2015 |
| Submission date | |
| Deliverable lead | IBM |
| Version | 1.0 |
| Authors | All |
| Reviewers | Ana Juan Ferrer, Anouar Rachdi |

| | |
|---|---|
| Abstract | PANACEA proposes innovative solutions for proactive autonomic management of cloud resources. Anomalies are predicted ahead of time and corrective actions are initiated to optimize cloud resources usage. Expected benefits include higher availability, improved security, and better system performance. The leading motto of the system is “being proactive rather than reactive”. With PANACEA cloud services will exhibit the following properties: self-healing, self-configuration, self-optimization, and self-protection. In this document the main components of the system are introduced and the interactions between these components are described. |

| Keywords | System Architecture, Proactive Management, Self* properties |
|---|---|

**Document Revision History**

| Version | Date | Description of change | List of contributor(s) |
|---|---|---|---|
| v0.1 | 01.06.2015 | Organization, Section 3.3 and Global Architecture | Daniel Molina (UCM)<br>Eduardo Huedo (UCM) |
| v0.2 | 10.06.2015 | | Atos's contribution |
| v0.3 | 07.07.2015 | Added the discussion on the pervasive monitoring system (Sec. 3.2), the online QoS-driven task allocation system (Sec. 1.1, 3.5, and 4.1.2), and the PANACEA architecture as realized in use case 1 (Sec. 4.3). | Gokce Gorbil (Imperial)<br>Lan Wang (Imperial) |
| V0.4 | 26.07.2015 | Incorporate comments from Atos | Ana Ferrer (Atos) |
| V0.5 | 27.07.2015 | Incorporate comments from Atos | Anouar Rachdi (QoS Design) |
| V0.6 | 28.07.2015 | Incorporate revised ML section | Dimiter Avresky (IRIANC) |
| V1.0 | 16.07.2015 | Final Integration | Vita Bortnikov, Eliezer Dekel, Benjamín Mandler (IBM) |

## Disclaimer



## Copyright notice



| **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 | |

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

# EXECUTIVE SUMMARY

PANACEA proposes innovative solutions for proactive autonomic management of cloud resources to make the system proactive rather than reactive. In this deliverable we detail the system architecture, including the main components comprising the system and the interactions thereof. The system is comprised of a pervasive monitoring infrastructure providing resources related information to interested parties. In particular a predictive model feeds off the monitoring information and proposed changes to be performed on the cluster (s). Accompanying is a management platform that can carry out the autonomous changes proposed by the prediction component; and support is provided for multi-cloud operations. A self-healing, self-optimizing and highly scalable inter-cloud communications system is provided to assure proper communication among system components. In addition a QoS driven task allocation system is provided to offer load balancing capabilities.

Two use cases were chosen to verify the usefulness of the proposed PANACEA architecture. First, a typical web services workload, and second, data analytics as a cloud service

 

# TABLE OF CONTENTS



 

# LIST OF FIGURES

# ABBREVIATIONS

| | |
|---|---|
| **CU** | Central Unit |
| **DAaaS** | Data Analytics as a Service |
| **DDoS** | Distributed Denial-of-Service |
| **MA** | Monitoring Agent |
| **ML** | Machine Learning |
| **MM** | Monitoring Manager |
| **PMon** | Pervasive Monitoring System |
| **PON** | PANACEA Overlay Network |
| **P-VMS** | Physical Virtual Monitoring System |
| **RA** | Reception Agent |
| **RNN** | Random Neural Network |
| **S-VMS** | Service Virtual Monitoring System |
| **TA** | Transmission Agent |
| **VM** | Virtual Machine |
| **VMS** | Virtual Monitoring System |
| **WP** | Work Package |

# 1 INTRODUCTION

## 1.1 Goals of this document

PANACEA proposes innovative solutions for proactive autonomic management of cloud resources. Advanced Machine Learning (ML) techniques are used to predict anomalies (like time to failure of applications, violation of expected response time of services or DDoS attacks) before they occur. This enables the system to act in advance by proactively reconfiguring itself, rather than just reacting after the fact occurs. Expected benefits include: (a) higher availability, by predicting and reacting to imminent failures before they occur, (b) higher security, by recognizing APT[1] (Advanced Persistent Threat) attacks in their first stages, and (c) higher performance, by predicting workload increases or performance bottlenecks and adapting the capacity in advance. Thus, Panacea is envisioned to tackle a wide range of scenarios, including components failures and varying workloads. The leading motto being "proactive, rather than reactive".

The PANACEA solution is targeted towards critical services having high-availability and high-reliability requirements. With PANACEA, these services will have the autonomic properties:

- *self-healing* against anomalies by recovering from multiple failures, and using proactive rejuvenation of applications and servers for preventing crashes and increasing the availability, predicting the threshold violation of response time of servers,
- *self-configuring* by efficiently mapping user's requirements onto distributed clouds and configuring on-the-fly in the presence of anomalies,
- *self-optimizing* using proactive migration of virtual machines from one cloud resource to another, maintaining the quality of service of end-to-end flows despite path outages and performance failures of the Internet,
- *self-protecting* using proactive reconfiguration of overlay networks to protect against DDoS attacks.

The proposed solution is based on the following components:

- a highly scalable **monitoring system** based-on interacting agents that can read computing and network sensors to collect relevant parameters. The monitoring agents can make autonomous decisions on where to focus the monitoring effort,
- **prediction models** from advanced machine learning techniques allowing to predict in advance software and hardware failures from observations,
- a **cloud management platform for autonomic services** providing self-awareness and self-configuration through sensors and effectors that allow them to make and take proactive reconfiguration decisions (rejuvenation of applications, job migration…),
- online optimization of the **overlay network** between clouds in order to quickly detect failures of Internet paths and to adapt the routes used to the congestion in the Internet,
- an **online QoS-driven task allocation system** that uses machine learning techniques based on random neural networks and online measurements in order to learn the current condition of the available resources and which decisions lead to good performance, and dynamically performs task allocations based on this learned information in order to maintain and improve the quality-of-service observed by the users of the services hosted in the cloud.

[1] **https://en.wikipedia.org/wiki/Advanced_persistent_threat**

[2] https://github.com/dsa-research/onegate-panacea

The main objective of this document is to describe how the above components interact in order to achieve PANACEA goals.

## 1.2 Organization of this document

This document is organized as follows:

- Chapter 2 provides background information on proactive and autonomic computing.
- Chapter 3 presents the main components of the PANACEA architecture.
- Chapter 4 discusses how the different components interact so as to achieve PANACEAs goals.
- Chapter 5 describes the use cases deployment with PANACEA.

# 2 AUTONOMIC AND PROACTIVE COMPUTING

This chapter provides background information on autonomic and proactive computing. Section 2.1 presents the main properties of proactive autonomic systems. Section 2.2 describes the architecture for autonomic computing defined by IBM.

Proactivity or proactive behaviour refers to anticipatory, change-oriented and self-initiated behaviour, which involves acting in advance of a future situation, rather than just reacting. In particular, proactive systems predict anomalies (like time to failure of cloud applications or DDoS attacks) before they occur.

## 2.1 Properties of Proactive Autonomic Systems

Autonomic systems have the following properties [1], including the list of features that should be exposed by any Cloud manager wishing to deploy autonomic self-managing systems:

- **Self-configuring**: The ability to define themselves "on-the-fly".
  - Change configuration parameters (capacity, placement, connectivity...) at runtime
  - Connect new devices at runtime (hot plugging)
- **Self-healing**: Discover, diagnose, and react to disruptions.
  - Detect faults and recover from them
  - Perform software rejuvenation
  - Manage spares for live migration of applications
- **Self-optimizing** (**self-adapting**): Monitor and tune resources automatically.
  - Obtain information of the execution environment at runtime (self-awareness) and adapt to it (self-configuration)
  - Manage elasticity
- **Self-protecting**: Anticipate, detect, identify and protect themselves against threats from anywhere.
  - Define and manage user access
  - Detect attacks and recover from them
  - Perform backup and recovery

In principle, no special functionality has to be included in the Cloud manager to support proactive systems, provided that autonomic systems are supported, since the proactive behavior would rely on the systems themselves. Proactive systems provide a number of benefits, including:

- Higher availability, by predicting and reacting to failures before they occur.
- Higher security, by recognizing APT (Advanced Persistent Threat) attacks in their first stages.
- Higher performance, by predicting workload increases or performance bottlenecks and adapting the capacity on advance.

## 2.2 Architecture for Autonomic Computing

IBM defined an architecture for autonomic computing [1] [2], which is depicted in Figure 1.

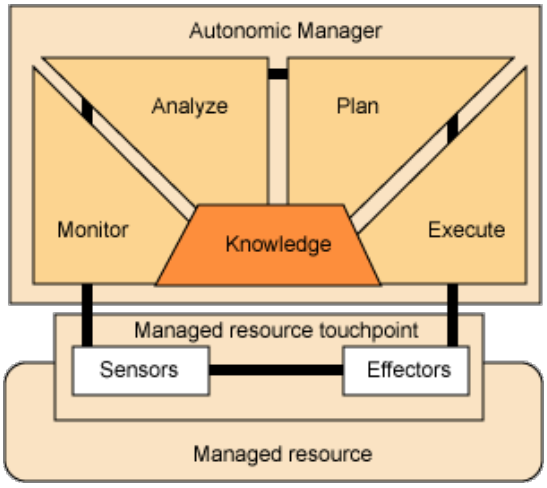


*Figure 1: IBM's architecture for autonomic computing [1].*

A **managed resource** is a hardware or software component that can be managed. A managed resource could be a server, storage unit, database, application server, service, application or other entity. A **touchpoint** implements **sensors** (to gather details) and **effectors** (to change the behaviour) for a managed resource's manageability mechanisms. It employs mechanisms such as log files, events, commands, application programming interfaces (APIs) and configuration files.

For a system component to be self-managing, it must have an automated method to collect the details it needs from the system; to analyse those details to determine if changes are required; to create a plan, or sequence of actions, that specifies the necessary changes; and to perform those actions. **Autonomic managers** implement intelligent control loops, automating the "monitor, analyse, plan and execute" parts.

The sketched architecture served as a starting point and inspiration for the Panacea architecture described therein. The architecture has been modified to fit the project's intended goal which is an autonomous cloud federation system. The managed resources presented in Figure 1 correspond to cloud related resources such as VMs, memory, etc. The relevant resources are being monitored by the Panacea monitoring system described in section 3.2, and are being fed into the knowledge, analysis, and planning components, represented in Panacea within the ML framework (see section 3.1). Finally the execution is governed by the cloud management platform (section 3.3), which can carry out the instructions provided by the analysis and planning components, to proactively take care of failures before they occur.

# 3 MAIN COMPONENTS OF PANACEA

Enclosed is the description of the main components of the PANACEA architecture. Section 3.1 is devoted to the Prediction Models, derived from the Machine Learning (ML) framework, for predicting the time to crash of cloud applications and the response time of servers. In Section 0, we describe the multi-agent pervasive monitoring system of PANACEA. Section 3.3 presents the Cloud management platform, based on OpenNebula, and how it has been modified in PANACEA to support autonomic services. Section 3.4 is devoted to the overlay routing system of PANACEA. In Section 3.5, we present the online QoS-driven task allocation system and its relation to the other PANACEA components in the global architecture.

## 3.1 Prediction Models from ML

A global architecture, based on ML framework, for a proactive management of cloud resources is realized. The developed architecture can be used for forming Hybrid Clouds and controlled by the Intra-Autonomic Cloud Managers (Intra-ACM). They are organised by the Inter-Autonomic Cloud Manager (Inter-ACM). It is in charge of orchestrating the Hybrid Clouds.

The ML Framewrok is created to generate Machine Learning (ML) prediction models in private, public and hybrid clouds. The Framework generates predictions models with all features (monitored parameters of the physical resources) or with a reduced set of parameters, based on Lasso Regularization technique. The users can make a choice, based on the operational requirements of their cloud applications. The goal of this research is to show the effectiveness of a Global Architecture for a Proactive Management in a (geographically) distributed environment.

The initial step to build ML prediction models is to collect runtime data from the system under monitoring. The abstract architecture of the system when collecting this data is shown in *Figure 2*. To build the initial database file (Raw Knowledge Base), namely the input file to the ML Framework, different types of anomalies can be injected in monitored system to simulate a software environment which eventually becomes faulty ( memory leaks, unterminated threads, number of threads etc.. ).

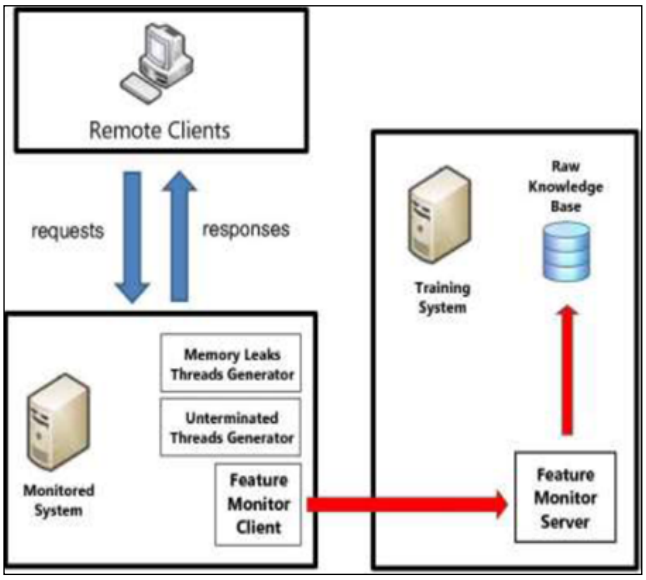


*Figure 2: Architecture for Data Collection*

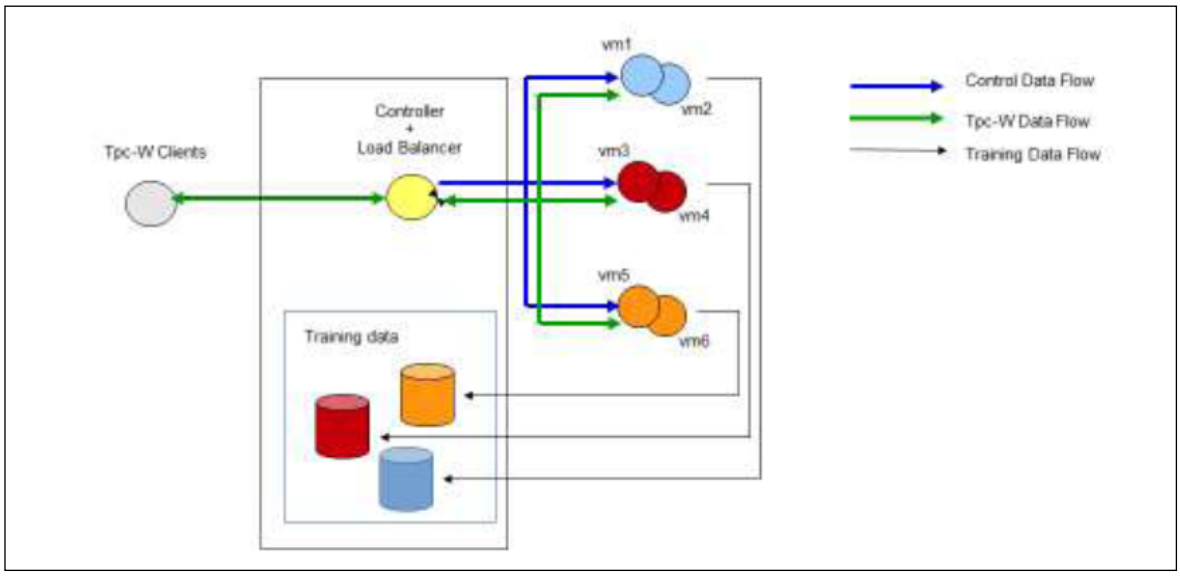


*Figure 3: Controller and Load Balancer for VMs*

The basic modules of the Global Architecture for Proactive Management in a (geographically) distributed environment are shown in *Figure 3*. The number of the VMs can scale (up/down) to any size in a given cloud region.

Each web server hosts the TPC-W Web Application standard benchmark , which requires several pieces of software to run correctly: Apache Tomcat 6.0.41 , A Java implementation of TPC-W , MySql Server version 5.1.41-3. A Feature Monitor Client, which is a proprietary software package aimed at collecting resource usage statistics of the VM Client machine and send it to the Feature Monitor Server (*Figure 2*). The Java implementation of TPC-W relies on Java HTTP Servlet, a standard for implementing java classes for handling HTTP requests, allowing to manage dynamic contents in a web server through the Java platform. Multiple TPC-W users, implemented as emulated browsers for a complete description of the behaviour of the clients are used to simulate the traffic generated by end users. Given the fact that TPC-W Clients can be (geographically) distributed, and provided that their location is of no constraint to the generality of our approach, they are depicted as a single box in system.

Based on *Figure 2* and *Figure 3*, the Intelligent collection of Training Data and usage of Machine Learning algorithms for creating prediction models designed for a proactive management of Intra cloud resources can be achieved. Intra ACM is communicating with other Intra ACMs and Inter ACM.

The implemented ML framework predicts the time to crash of applications. This ML prediction model is used by cloud region controllers (which implement Intra ACMs) to determine when a VMs is approaching its failure point. In this way, the controller is able to autonomously decide when to send a control message to this about-to-crash VM, in order to proactively rejuvenate it. At the same time, the Intra ACM controller sends a control message to a spare (standby) VM, which is then activated (upon the receipt of this control message) and takes the place of the rejuvenating one.

In particular, the Intra ACM/InterACM infrastructure allows to build public/hybrid distributed cloud architectures, which are able to self-tune themselves so as to cooperate for keeping under a selected threshold the response time and increase the availability. By relying on the same ML prediction models, the various Intra ACM controllers can redistribute the incoming load among the various cloud regions, so as to reduce at the same time the rejuvenationfrequency.

The working prototype of a system allows implementing self* properties (healing, rejuvenation, optimizing, reconfiguring) and seamless application execution. We realized a highly available web server by using a set of virtual machines in several cloud regions. This framework can be used for any

cloud applications (not only web servers) that requires a large number of resources and has a high probability to fail, and runtime constraints. It allows creating mission-oriented critical applications in hybrid clouds.

The implemented ML framework predicts the time to crash of applications, so that a proactive rejuvenation of the application can be periodically performed before the system fails and the response time is lower than a given threshold.

Our solution allows scalability at two levels: the Intra ACM controllers can proactively decide whether to activate new VMs, in case the workload is increasing too much. Moreover, multiple cloud regions can be connected to each other, even at runtime, so that the system can rely on a larger number of VMs if current dynamics require so. The distributed architecture performs automatic activation of VMs, which requires seconds.

The obtained results show that Inter ACM is able to properly coordinate the behaviour of multiple regions. In particular, Inter ACM is able to promptly detect the state of entire regions, and its internal metrics quickly converge to the actual values, even in case of strong variations of the system (an entire region is removed).

The available cloud resources can be distributed over the services offered by different cloud providers, and/or over services offered by the same cloud provider in different geographical locations, and/or over private cloud infrastructures.

The communications between Intra/Inter ACM are provided by the Overlay Routing System in Section 3.4. It protects inter-cloud communications from path outages and performance degradations of the Internet. The overlay network is formed of software routers called Proxies and deployed at the cloud sites. Overlay Network uses two other types of software agents Transmission (TA) and Reception (RA) agents installed on Intra/Inter ACM in cloud regions.

Replication Service for high availability, Partition Tolerant leader election and locking mechanisms, which are presented in Section 3.6 Configuration and Configuration Service, will be used by Intra/Inter ACMs for coordinating their activities. This coordination is very vital when the critical information is maintained and used for ML framework in hybrid clouds, such as large sizes of: Training Data Base, software nodules for different ML techniques and generated data base for ML prediction models.

A more detailed description can be found in Deliverable 3.3 [16].

## 3.2 Pervasive Monitoring System

The role of the *pervasive monitoring system (PMon)* within the PANACEA cloud management architecture is to collect measurements from physical hosts, virtual machines, services and applications, and provide the monitoring data to the entities that make management and allocation decisions based on the observed conditions of the cloud, such as the autonomous service managers and the cloud managers.

The pervasive monitoring system adaptively prioritizes which nodes to monitor, thereby decreasing overhead while providing timely delivery of fine-grained monitoring data. The main purpose of PMon within the PANACEA architecture is to enable better, i.e., more responsive and closer to optimal, management and allocation decisions to be made dynamically at run-time.

PMon consists of *monitoring agents (MA)* and *monitoring managers (MM)*. The MAs are located on the physical hosts and the VMs, and therefore have access to measurements from the physical infrastructure and also from within the virtual machines, i.e., from the services and applications running within the VMs. Each MA implements one or more probes that periodically collect measurements from its node, which are stored locally and made available to other PANACEA components via the MMs.

The MAs employ *machine learning techniques based on random neural networks (RNN)* [11-15] that continually learn which parts of the cloud are changing the most, and dynamically focus the monitoring on those areas as needed in a distributed manner. The current implementation of the MA leverages existing tools for collecting and storing measurements, such as *collectd* and *rrdtool*. However, the core functions of PMon are independent of these tools, and they can be replaced by equivalent or better software packages in future implementations.

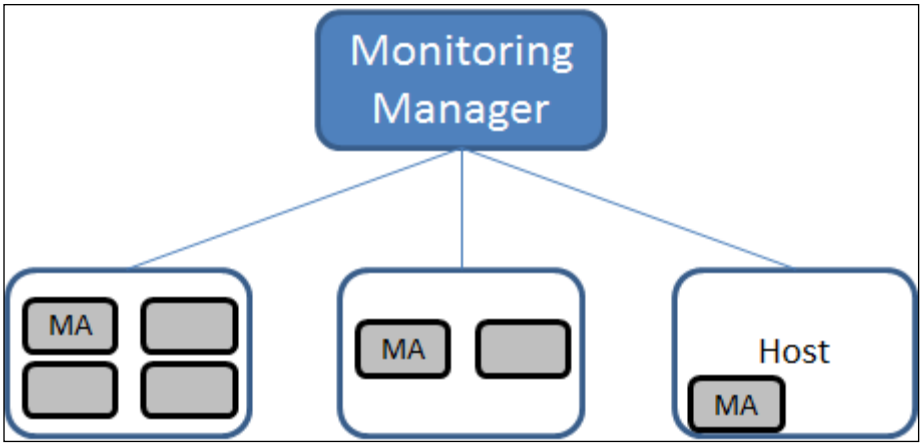


*Figure 4: Pervasive Monitoring System Srchitecture*

The monitoring manager (MM) implements the control functions of the monitoring system, and provides an interface to PMon. The MMs collect the monitoring data from the MAs via probes, and provide it to the interested entities, e.g., the autonomic cloud managers and the service managers, via standardized interfaces, which are described later in this section. Each MM controls a set of MAs, activating, reconfiguring, and deactivating measurements at MAs as necessary.

The MM also provides the interface between PMon and the components that use the monitoring data, i.e., the cloud management software. The requirements and the design of the pervasive monitoring system have been presented in D2.1 [4] and D2.2 [5]. In this document, we describe the relationship and the interactions between PMon and other components in the PANACEA system (Sections 4.1.3 and 4.2.4), and present how PMon will be deployed in the PANACEA use cases 1 and 2 (Sections 5.1 and 5.2).

### 3.2.1 The control interface between the monitoring manager and PANACEA components

The monitoring manager provides the control interface to the consumers of the monitoring data in the PANACEA architecture. The control interface is very simple, and allows data consumers to register to the MM, to start monitoring for a specific metric, and to stop monitoring for a specific metric. When there are no registered data consumers for a given metric, the MM stops all monitoring activity for that metric. If there are no metrics left to monitor, then all monitoring activity is ceased until a new consumer or metric is registered to the MM.

The control interface is implemented via JSON-RPC, with the data consumers sending JSON-encoded RPC requests to the MM in order to register to, start, and stop monitoring activities. This allows the data consumers and the MM to reside on different machines, either on the same cloud, or on different clouds.

### 3.2.2 The data interface between the monitoring manager and PANACEA components

The monitoring manager indirectly provides the data interface to the consumers of the monitoring data. The data interface allows the consumers to read the monitoring data, either locally when the consumer and the MM reside on the same machine, or remotely when they are on different machines.

In the co-residence case, the MM updates a special memory-mapped file that records the last update time of the monitoring data per metric. The consumers read this file at will in order to learn whether new data is available, and they then read the data directly from local storage, e.g., via the interface provided by rrdtool.

A similar method is used in the remote case: the MM updates a special collection in the database that stores the monitoring data with the last update time per metric, and the consumers first access this collection using the interface provided by the database system, e.g., MongoDB, and then access the monitoring data, again via the interface of the database.

In this implementation, we have adopted the approach of the consumers checking whether new data is available. Future implementations may add the option of the MM informing the registered entities when new data is available, although this feature has to be implemented and configured carefully so as not to cause excessive network overhead when the MM and the consumers are not co-located. This implementation also delegates the interface between the consumers and the data to the database storing the monitoring data in order to reduce development effort. Future implementations may provide this functionality through the MM.

## 3.3 Cloud Management Platform for Autonomic Services

We have defined an architecture to provide support for autonomic systems in typical clouds, by means of self-awareness (sensor mechanism) and self-configuration (effector mechanism). These mechanisms will be used by the PANACEA Intra-ACM, running in a VM, to perform proactive autonomic management of cloud services. Usually, the Cloud Manager manages physical and virtual resources (servers, networks, VMs, images…), while the Service Manager manages services automatically, including elasticity. Services are composed of interconnected VMs playing different roles and each role has a cardinality (number of VMs) and can have deployment dependencies with other roles.

Based on the analysis performed in the OpenNebula use case of D4.1, the described architecture has been concreted to use OpenNebula as Cloud Manager and OneFlow as Service Manager. In this case, self-awareness and self-configuration are provided by extending the OneGate component provided by OpenNebula. OneGate is very similar to Amazon CloudWatch, so that a solution based on this mechanism could be easily ported to other Cloud managers.

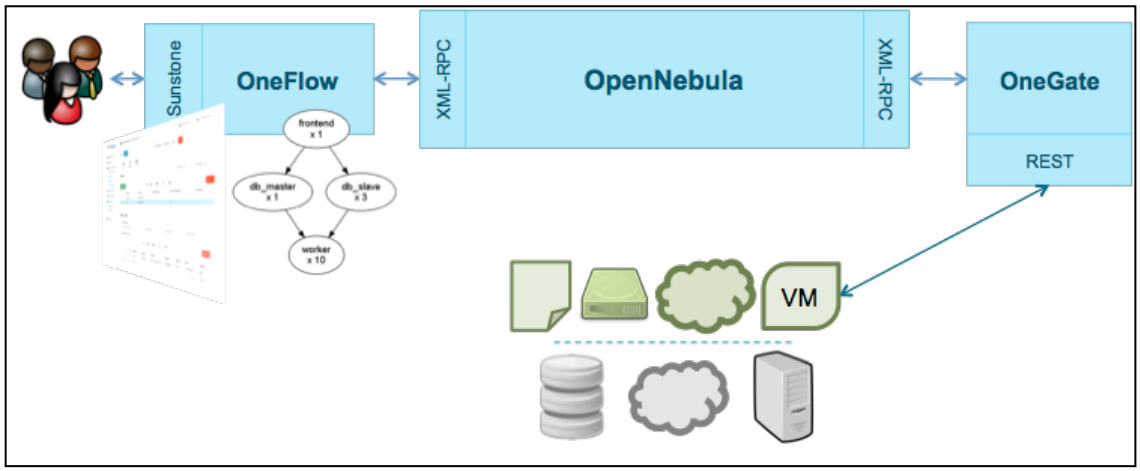


*Figure 5: OpenNebula-based autonomic Cloud architecture.*

Since version 4.4, OneGate allows VM guests to *push* monitoring information to OpenNebula. Users and administrators use it to gather metrics, to detect problems in their applications, and to trigger OneFlow elasticity rules. It is basically a server that listens to HTTP connections from the VMs using

a REST interface. OpenNebula assigns an individual token to each VM instance, and applications running inside the VM use this token to contact OneGate in order to request operations to OpenNebula (initially, just to send monitoring metrics).

OneGate checks the VM ID and the token sent and, if valid, the operation is forwarded to OpenNebula through its XML-RPC interface. For example, in an update operation (PUT action in REST), new information is placed inside the VM's user template section, so that the application metrics are visible from the command line, Sunstone, or APIs.

To provide self-awareness, OneGate now also allows VM guests to *pull* VM information from OpenNebula (GET actions in REST). For VMs that are part of a Service, they can also retrieve information from other VMs or retrieve the Service information. Moreover, by pushing data from one VM and pulling data from another VM, nodes that are part of a Service can pass values from one to another.

To provide self-configuration, more operations on VMs (besides update) have been implemented, like shutdown, resched, stop, suspend, resume, hold, release… (POST actions in REST). Also, the scale operation on the service has been implemented to allow a VM to modify the cardinality (number of VMs) of a given role, providing service auto-scaling. More complex operations (e.g. requiring more parameters) are being added, like resize or migrate.

An easy to use command line interface (the onegate command) has been developed to hide all the implementation details to the user. A running prototype[2] is currently available, together with installation[3], configuration[4] and usage[5] guides.

## 3.4 Overlay Routing System

We have designed a communication system shielding inter-cloud communications from path outages and performance degradations of the Internet. This communication system consists of a *self-healing*, *self-optimizing* and *highly scalable* routing overlay that accepts customized routing policies formulated by distributed applications according to their own needs.

As shown in *Figure 6*, the overlay network is formed of software routers called Proxies and deployed at the cloud sites and possibly in other locations in the Internet.

[2] https://github.com/dsa-research/onegate-panacea

[3] https://github.com/dsa-research/onegate-panacea/wiki/Installation

[4] https://github.com/dsa-research/onegate-panacea/wiki/Preparing-VMs-and-Services

[5] https://github.com/dsa-research/onegate-panacea/wiki/Usage

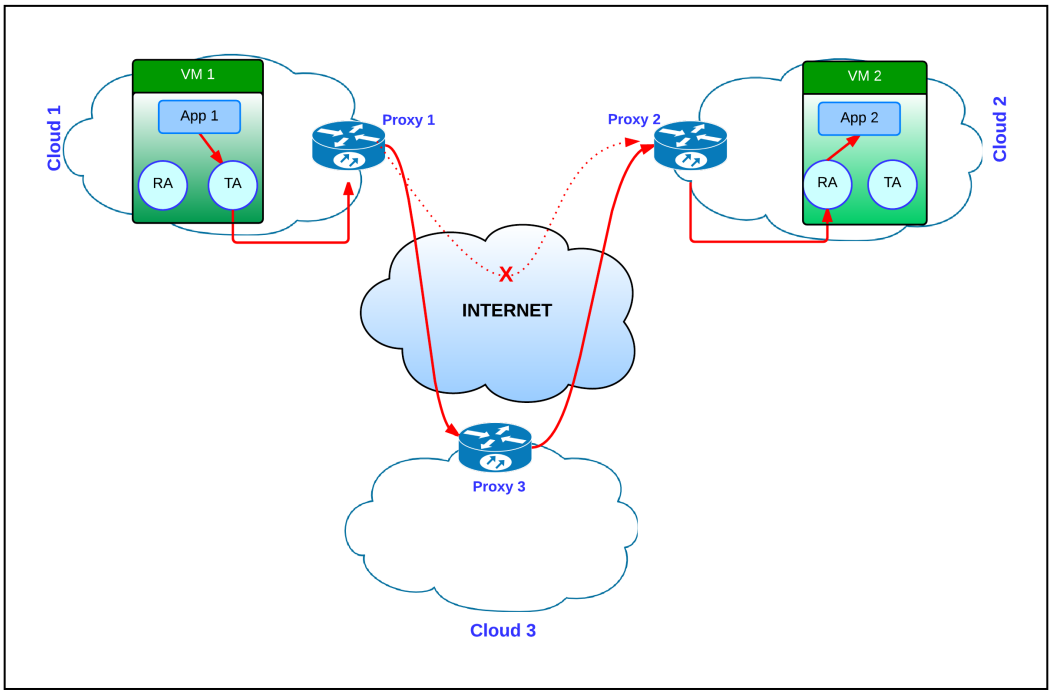


*Figure 6: Architecture of the PANACEA Overlay Network (PON)*

In addition to Proxies, the PANACEA Overlay Network (PON) uses two other types of software agents. Transmission (TA) and Reception (RA) agents, executed on each VM running a task of the distributed application, and are activated automatically at start-up of their respective VMs. They allow controlling the path of data through the network (without the application even being aware that its data flows are routed over the overlay):

- **Transmission agent**: the role of the Transmission Agent (TA) is to intercept the packets sent by the application running in the same VM and to forward them to the local Proxy using IP-in-IP encapsulation. On the data-forwarding path, the TA is therefore the entry node in the overlay network.
- **Reception agent**: the role of the Reception Agent (RA) is to receive the packets sent by the local Proxy and to deliver the original packets to the local application running in the same VM. On the data-forwarding path, the RA is therefore the exit node of the overlay network.

Each Proxy is in charge of monitoring the quality of the overlay paths towards certain destinations, selecting the best paths and forwarding the packets emitted towards them. As shown in Figure 6, the traffic flows are routed over the overlay network, enabling us to avoid congested or failed parts of the Internet. As described in Figure 7, the Proxy is constituted of three different software entities:

- **Monitoring agent**: it monitors the quality of the Internet paths between the local cloud and the other clouds in terms of latency, bandwidth, and loss rate. The monitoring agent can be queried by the routing agent in order to discover the quality of a given path according to a certain metric. It can also be configured to monitor the availability of a path at regular time intervals.
- **Routing agent**: This agent is configured to optimize a service-specific routing metric towards certain destinations. To this end, it drives the monitoring agent so as to discover an optimal path (e.g., low-latency, high-throughput, etc.) with a minimum monitoring effort. For each destination, the optimal path towards that destination discovered by the routing agent is

written in the routing table of the forwarding agent.

- **Forwarding agent**: this agent[6] is in charge of forwarding each incoming packet to its destination on the path it was instructed to use by the routing agent. We use source routing, that is, the routing table of the source Proxy describes the complete path to be followed by a packet to reach its destination. Each subsequent Proxy simply determines the next forwarding hop from the information contained in the PANACEA header. The final Proxy forwards the packet to the appropriate RA that then delivers the original packet to the destination application.

We refer to [6] for a more detailed presentation of the PON architecture.

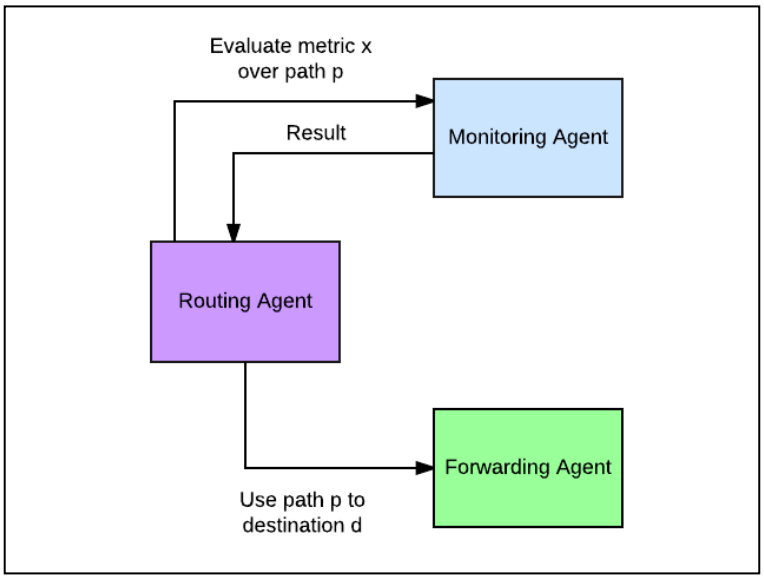


*Figure 7: Interactions between the entities constituting the proxy.*

## 3.5 Online QoS-Driven Task Allocation System

The online QoS-driven task allocation system *dispatches incoming jobs to the best available resources in the cloud* in order to maintain and improve the end-user experience, thereby providing self-optimization of unattended managed services at the user experience level. An application of the task allocation system is as a QoS-based load balancer for cloud-hosted web services (use case 1), where the load balancer dispatches incoming service requests to the application server that provides the best QoS, e.g., the lowest response time, as observed via online measurements.

The task allocation system uses machine learning techniques based on random neural networks in order to learn the current conditions of the resources and which allocation decisions may lead to good performance, and dynamically performs allocation decisions based on this information.

The task allocation system consists of the task controller, e.g., the load balancer, and the worker nodes. The task controller receives the incoming jobs arriving at the service, and dispatches them to the worker nodes according to the implemented task scheduling algorithm. In addition to being the entry point for the service hosted in the cloud, the task controller maintains a local view of the conditions of the cloud and the worker nodes, which is updated using online measurements relating to the performance observed for each executed job.

Although the task allocation system implements basic service monitoring functions that it can fall back

[6] It is possible to use multiple forwarding agents for load-balancing purposes.

upon in case of any failure of external systems, it normally interacts with the pervasive monitoring system in order to collect measurements related to its decisions and the current conditions in the cloud.

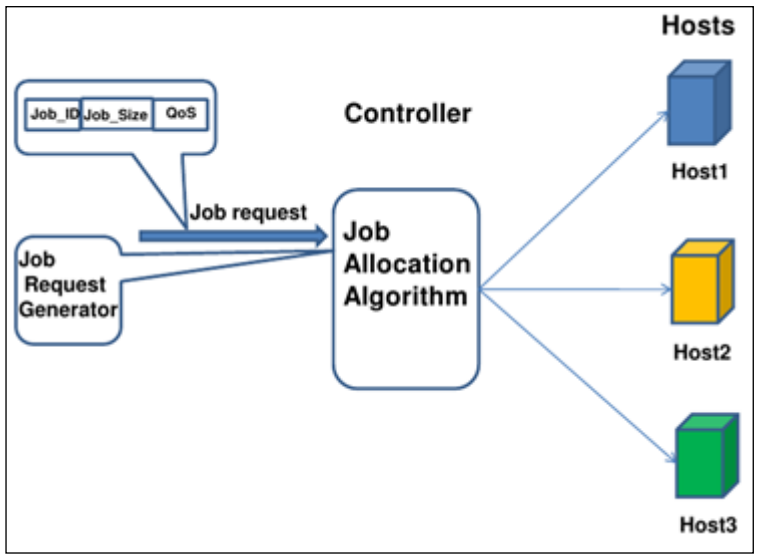


*Figure 8: Autonomic job allocation*

The task allocation system is a service add-on component that is provided by the PANACEA system for unattended managed services that fit the job-based or request-based service model with multiple workers sharing a queue of jobs, and that can therefore take advantage of the self-optimization function provided by the task allocation system.

This system is therefore different than the other management components in the PANACEA architecture, e.g., the cloud manager and the service manager, in that it is more a part of the managed service rather than of the PANACEA management system. The task allocation system deployment, and its interactions with the other PANACEA components, mainly with the pervasive monitoring system, is described in Sections 4.1.4 and 4.2.5.

## 3.6 Configuration and Coordination Service

This service is a core component used by other system components to provide high availability, fault tolerance and scalable management of the system. It will provide replication services, leader election and distributed locking and should be used wherever critical information is maintained. The distributed locking mechanism is crucial to the PANACEA operation as it minimizes the conflicts in executing control operations in a distributed environment. In particular, this includes: Replication Service for high availability, and Partition Tolerant leader election and locking mechanism.

For this purposes we are leveraging highly available and partition tolerant cohort of services that uses quorum based replication. The Frappe replication technology was built especially for coordinating data-centric distributed systems. It is that is from the same family as Zookeeper and Chubby. Frappe is built to be hosted in the cloud and be ready for its elastic nature.

This service will be used by all core components that maintain state information that is required for the continuous operation of the entire system. More specifically, the replication system will guard the large knowledge base that is maintained by the ML component. In addition, required monitoring information will be maintained highly available by this service. The management system of the cloud, including critical agents of the system will use this service to maintain configuration information and parameters that are required for the continuous operation of the system. It will provide a membership service which will feed the system with correct up to date information into the entities which are currently alive within the system, including critical information as to which services are hosted in which parts of the cloud. Providing a consistent and assured leader election process will enable other

system components to operate under the assumption that no split-brain or other such scenarios may cause havoc when more than a single master is attempting to orchestrate the cloud operations simultaneously.

# 4 GLOBAL PANACEA ARCHITECTURE

## 4.1 Architecture for a single cloud

We first consider the global architecture for a single cloud. The proposed architecture is described in *Figure 9*.

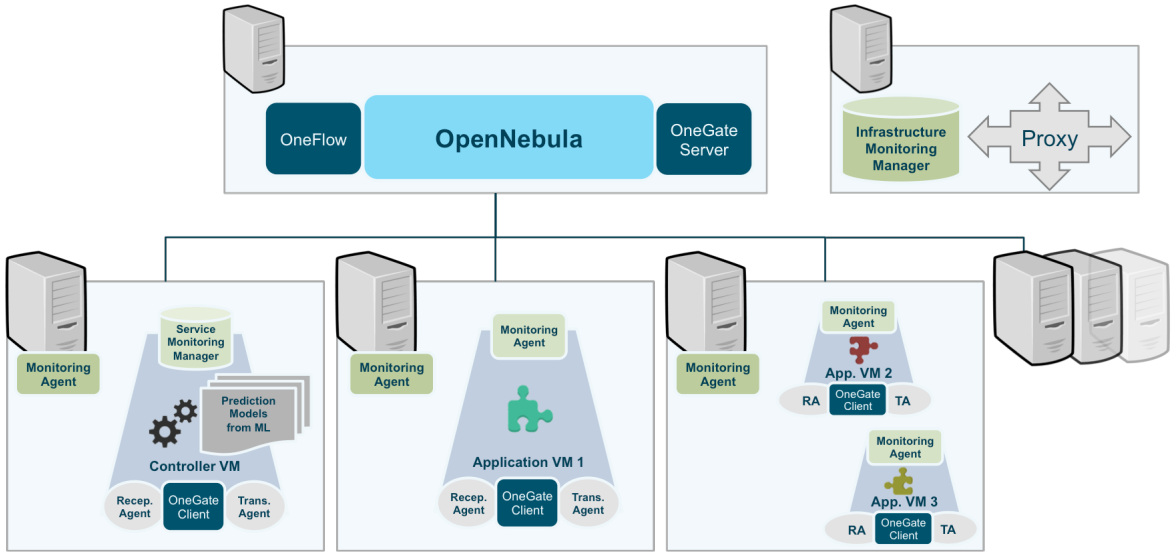


*Figure 9: PANACEA architecture for a single cloud.*

### 4.1.1 Cloud Infrastructure

As depicted in *Figure 9*, OpenNebula acts as the Cloud Manager, managing both physical and virtual resources. On top of it, OneFlow acts as the Service Manager, allowing services to be deployed and managed as a whole. The OneGate Server is used to support self-awareness and self-configuration to autonomic services, providing a REST interface for VMs to interact with OpenNebula.

There is one Pervasive Monitoring System for the infrastructure and one for each service. The infrastructure Pervasive Monitoring System consists of a Monitoring Manager, running in a physical host (which could be the front-end host running OpenNebula) or even in a VM, and one Monitoring Agent running on each physical host acting as worker node for OpenNebula.

For the Routing Overlay, the Overlay Proxy is deployed in a physical host or even in a VM.

### 4.1.2 Autonomic Services

Several Autonomic Services can be deployed on the Cloud. Each Autonomic Service is composed of one Controller VM (a.k.a. the Intra-ACM) and several Application[7] VMs.

The Controller VM runs an agent that uses Prediction Models, obtained from Machine Learning, to make decisions based on different monitoring metrics. To perform these decisions, it contacts the OneGate Service using the OneGate Client available in the VM.

The Application VM runs a component of the application, which can be in active or stand-by status.

[7] **The Application is used here as a generic term for the application that is being managed by PANACEA. This is the Web Service in use case 1 and the relevant parts of the Hadoop deployment in use case 2.**

There is a service Pervasive Monitoring System for each service. It consists of a Monitoring Manager, running in the Controller VM, and one Monitoring Agent running on each Application VM. Therefore, Prediction Models can consume monitoring information both from the cloud infrastructure and from the service.

Each VM also runs one Transmission Agent and one Reception Agent to intercept outgoing and process incoming network packets.

The orchestration of all components, their location, role and state are maintained in synchronization by the coordination services.

### 4.1.3 Pervasive Monitoring System in a Single Cloud Deployment

As described in Section 0, the pervasive monitoring system consists of the monitoring agents (MA) and the monitoring managers (MM). The MAs are installed at each physical host and VM. For logical management of monitoring functions and improved scalability, PMon constructs intra-cloud communication substrata in the form of trees in a single cloud deployment. Each monitoring communication substrata comprises a *virtual monitoring system (VMS)*. There is a single VMS per cloud for monitoring the physical infrastructure, which is called the physical VMS (P-VMS, see Figure 15).

In addition to the P-VMS, there is one service VMS (S-VMS) per managed service deployed in the cloud (see Figure 14). The MAs form the nodes in the VMS, and the MM is attached to the root node of the VMS and controls the monitoring activities and pulls the monitoring data from the MAs. The communication architecture employed in the multi-cloud case is discussed in Section 4.2.4.

The placement of the P-VMS MM and the S-VMS MM is flexible, but in order to improve the performance of the communication between the managers that use the monitoring data and the respective MM, it is advantageous to place the autonomic cloud manager and the P-VMS MM on the same physical host or on the same VM, and likewise for the autonomic service manager and the S-VMS MM.

The pervasive monitoring system has been purposefully kept separate from the monitoring activities of the OpenNebula system, which needs to monitor the hosts and the VMs for managing the cloud. In this proof-of-concept implementation of the PANACEA system, OpenNebula relies on its own monitoring system for its management activities, and PMon is used for proactive and autonomic management of the cloud and services by the autonomic service managers and the cloud managers. In future implementations, PMon can be integrated into OpenNebula's native monitoring system in order to improve its performance.

### 4.1.4 Online QoS-Driven Task Allocation System in a Single Cloud Deployment

The online QoS-driven task allocation system manages jobs and requests arriving at a self-managed service hosted in the cloud, and dispatch them to the best resource available within the service in order to improve QoS for the user. As the task allocation system is an optional component that a self-managed service can exploit; only some services may include this component.

The entry point for all jobs and requests arriving at the service when the task allocation system is used will be the task controller, which is a piece of software that will most likely run within its own VM. The service needs to be configured so that the service users know the address of the entry point. The worker nodes also need to be aware of the task controller as they have to register to it when they start up.

This can be achieved via the self-awareness functions provided by OpenNebula. Alternatively, the address of the task controller and the addresses of the worker nodes can be given as part of the service description, which can again be accessed via the self-awareness functions.

Each task allocation system is specific to its constituent service, and interacts with the S-VMS MM in order to get the monitoring data it needs to make its allocation decisions. Therefore, it is advantageous to place the S-VMS MM and the task controller on the same physical host, or on the same VM.

## 4.2 Architecture for multiple clouds

The architecture for multiple clouds can be implemented using OpenNebula's features. More precisely, three possible scenarios can be envisioned:

- Cloud bursting (or Hybrid cloud)[8]
- Cloud federation[9]
- Cloud brokering

OpenNebula cloud bursting or federation capabilities can be used for the first two scenarios, as described below.

### 4.2.1 Cloud Bursting

Cloud bursting is a model in which the local resources of a Private Cloud are combined with resources from remote Cloud providers. The remote provider can be a commercial Cloud service (e.g., Amazon EC2) or a partner infrastructure running a different OpenNebula instance. Such support for cloud bursting enables highly scalable hosting environments.

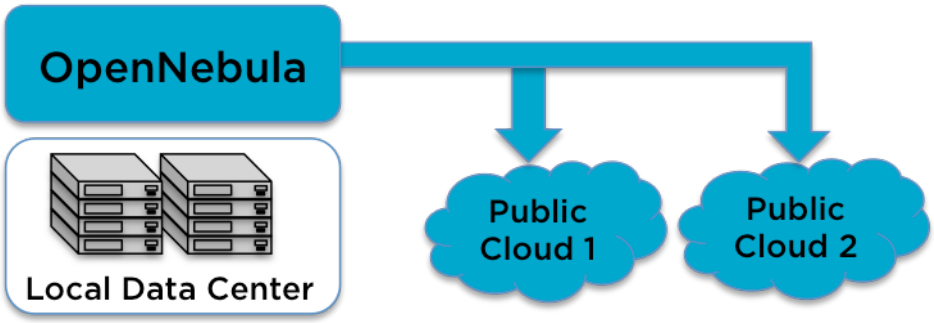


*Figure 10: Cloud Bursting with OpenNebula.*

OpenNebula's approach to cloud bursting is quite unique in that using and maintaining the cloud bursting functionality is transparent to both end users and cloud administrators:

- The transparency to cloud administrators comes from the fact that a remote data centre is modelled as any other host (albeit of potentially a much bigger capacity), so the scheduler can place VMs in the remote data centre as it will do in any other local host.
- The transparency to end users comes from the fact that the same VM template in OpenNebula can describe the VM if it is deployed locally and also if it gets deployed in a remote cloud infrastructure. So users just have to instantiate the template, and OpenNebula will transparently choose if that is executed locally or remotely.

[8] http://docs.opennebula.org/4.6/advanced_administration/cloud_bursting/introh.html

[9] http://docs.opennebula.org/4.6/advanced_administration/data_center_federation/introf.html

### 4.2.2 Data Centre Federation

Several OpenNebula instances can be configured as a *Federation*. Each instance of the Federation is called a *Zone*, and they are configured as one master and several slaves. An OpenNebula Federation is a tightly coupled integration. All the instances will share the same user accounts, groups, and permissions configuration (see *Figure 11*). The typical scenario for an OpenNebula Federation is a company with several Data Centres, distributed in different geographic locations.

For the end users, a Federation enables using the resources allocated by the Federation Administrators no matter where they are. The integration is seamless, meaning that a user logged into the Sunstone web interface of a Zone will not have to log out and enter the address of the other Zone. Sunstone allows users to change the active Zone at any time, and it will automatically redirect the requests to the right OpenNebula at the target Zone.

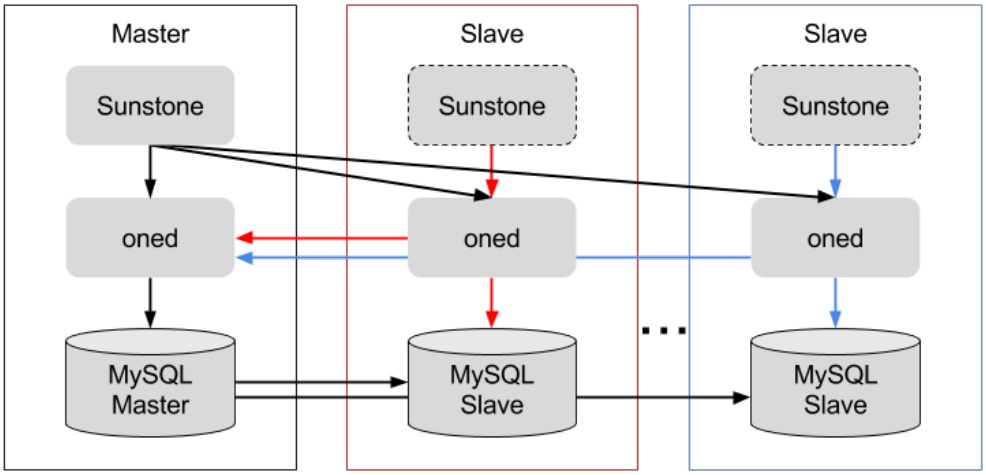


*Figure 11: Architecture of an OpenNebula federation.*

### 4.2.3 Deployment of the Routing Overlay

The PANACEA Overlay Network (PON) is composed of several software agents. First, it uses software routers called Proxies that are in charge of monitoring the quality of overlay paths towards certain destination, discovering the optimal paths towards these destinations and forwarding applications packets emitted towards them. As already mentioned, each Proxy is constituted of three different software agents:

- **Monitoring agent (Monitor)**: it monitors the quality of the Internet paths (in terms of latency, bandwidth, or loss rate) between the local cloud and a destination cloud upon request of the routing agent.
- **Routing agent (Router)**: This agent drives the monitoring agent so as to learn as fast as possible optimal path (e.g., low-latency, high-throughput, etc.) towards certain destinations with a constant monitoring effort.
- **Forwarding agent (Forwarder)**: this agent holds a routing table updated by the Router, and used to forward packets on alternative routes.

We note that the three components of a Proxy can run on the same (physical or virtual) host or on different hosts.

In order to deploy a PANACEA-enabled multi-cloud service, two other components have to be used:

- **Transmission Agent (TA)**: this agent intercepts packets emitted by applications running on the same VM and forwards them to the local Forwarder using IP-in-IP encapsulation. A direct

communication channel is established between each TA and its local Router using connection-oriented sockets, thereby allowing the local Router to activate or deactivate the interception of IP packets sent towards specific destinations.

- **Reception Agent (RA)**: this agent decapsulates the PON packet and forwards the original data packet to the destination task running on the same VM using a raw socket.

Each cloud needs to run at least one Proxy (Router, Monitor and Forwarder). Moreover, each application VM needs to run one TA and one RA. These software agents need some configuration information in order to work together with other PON agents running in distant clouds. Some of the parameters are directly related to the VM (e.g., its IP address) and can be discovered independently, whereas others (IP addresses of distant forwarders or routers…) have to be communicated by a **Central Unit (CU)**.

The Central Unit has an overall view of running agents in different clouds and is configured for a specific application. In each cloud, individual agents register at the CU by transmitting their own information (cloud identifier, role and IP address) to the CU and receive in return the required information about all other agents from the CU. This process is illustrated in **Erreur ! Source du renvoi introuvable.** Figure 12.

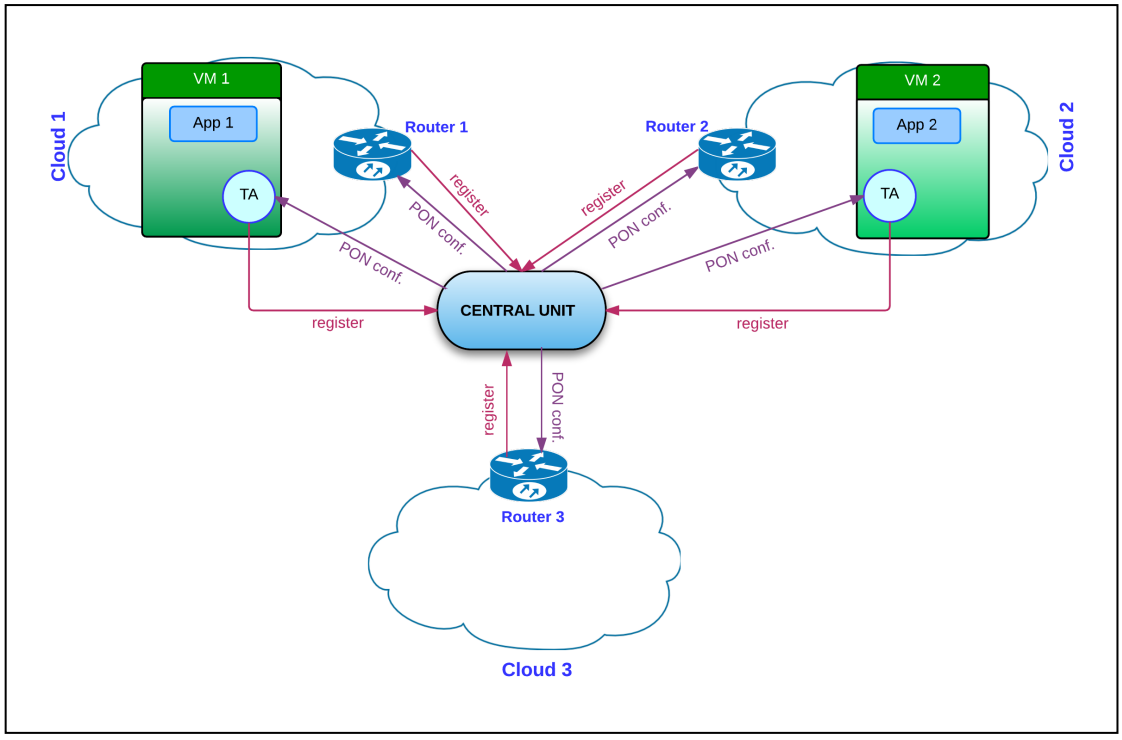


Figure 12: Registries at the Central Unit of PANACEA Overlay Network

In the following we present a deployment scenario in a multi-cloud context. In order to simplify the presentation, we shall assume that the three software agents (Router, Monitor and Forwarder) constituting a Proxy run on the same virtual machine. The role of this VM is tagged as PROXY. On the other hand, all application VMs run one TA and one RA and they get the APP tag.

We emphasize that the role of a VM may contain more information than just APP or PROXY. Consider for instance the scenario described in *Figure 7*. In that scenario, the roles assigned to the VM running Proxies 1, 2 and 3 are PROXY.SRC, PROXY.DST, and PROXY, respectively. The CU is configured to interpret this information and, therefore announces to Router 1 that it has to learn an optimal route towards Proxy 2, possibly passing through the intermediate point Proxy 3. On the other hand, the CU informs Proxies 2 and 3 of the existence of Proxy 1, but do not instruct them to monitor the routes to that Proxy. Similarly, in a multi-cloud Hadoop M/R scenario, the Proxy in the cloud

where the Master node is running is assigned the role PROXY.MASTER, whereas Proxies of clouds where only slave nodes are running are assigned the role PROXY.SLAVE. The CU uses this information to instruct the Master Proxy (resp. Slave Proxies) to monitor routes to Slave Proxies (resp. Master Proxy).

The different steps to follow for the deployment of the PON are as follows.

**VM startup**

All VMs include the PON software and the kind of VM (Proxy or APP) is determined upon VM creation. Based on the VM type the correct executable is launched upon start up.

**Operations on PROXY VM**

The Router starts by establishing a connection with the CU. As long as this connection is not ready, the routing agent waits. Once this connection is established, the Router registers with the CU by sending it its cloud identifier and IP address[10]. In return, the Router receives from the CU a unique identifier as well as the information about all other Routers in distant clouds.

Once this initial communication phase with CU is performed, the Router launches the monitoring and forwarding agents and the cloud PROXY can receive its application packets from the local TAs. We emphasize that the list of Routers registered at the CU is dynamic, meaning that new Routers can join (or even leave) the PON at any time. The connection between the Router and the CU is therefore maintained as long as needed, so that the Router can receive update messages from the CU when the PON topology changes.

**Operations on APP VM**

Both the TA and the RA have to run on an APP VM. However, only the TA needs configuration information regarding other agents in distant clouds and in its own cloud. The operations performed by a TA are pretty similar to those performed by a Router. A connection with the CU is first established upon TA startup. The TA then registers with the CU and gets from it a unique identifier and the information about PON configuration.

Once the initial configuration is obtained, the TA starts the RA and the APP VM is ready to intercept, encapsulate and transmit packets to the PROXY VM. The communication channel between the TA and the CU is maintained as long as needed in order to receive update messages from the CU regarding new participants in the PON.

**Operations at the Central Unit (CU)**

The CU is the PON manager. It has an address such as "cu.cloud1.com" that is known by all VMs running PON agents. The CU manages PON agents located in different clouds and provides them with information on the PON configuration. Routers need to know about other PON routers running in distant clouds, whereas TAs need to know about distant Forwarders and APP VMs behind them. The CU maintains a register describing the PON configuration (identifier, role and IP address of each VM in each cloud).

When a new VM joins the PON network, it first registers with the CU by sending its own identification information (cloud identifier, role and IP address). The CU records this information in the appropriate table, assigns a unique ID to the new VM and then sends in return a description of the PON configuration to the new VM. It also sends update messages to all other already registered VMs in order to inform them about the new VM.

*Figure 13* shows a possible sequence diagram of message exchanges between the CU and other agents in the case of the PON topology of Figure 12.

[10] Under our assumption, the local Monitor and Forwarder have the same IP address as they run on the same VM. However in case that Monitor or Forwarder run on a different VM, they need to communicate with the CU directly using a similar mechanism as the one used for the Router.

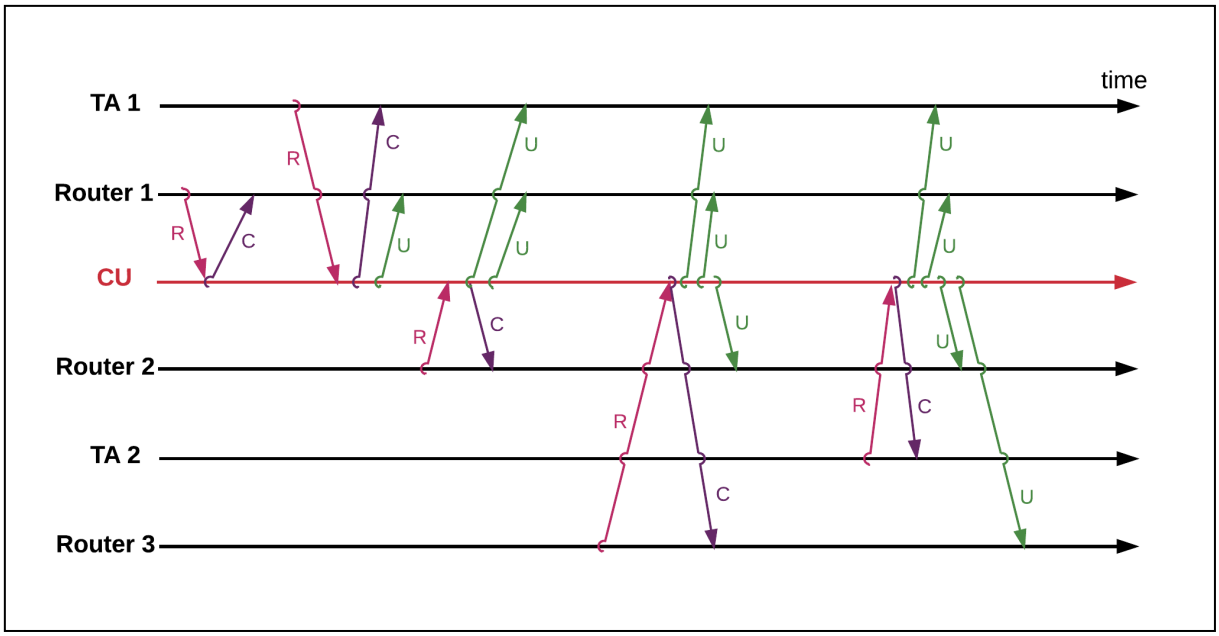


*Figure 13 Register (R), Configuration (C), and Topology Update (U) messages.*

### 4.2.4 Pervasive Monitoring System in a Multi-Cloud Deployment

The multi-cloud deployment of PMon follows the same architecture as the single cloud case, using overlay trees[11] to manage the monitoring actions and to connect the different clouds together. For a managed service deployed across multiple clouds, there are no changes to the S-VMS for monitoring the components of the service as long as the VMs in the different clouds can communicate over the network, e.g., the Internet.

The difference between the single cloud versus the multi-cloud scenario is the structure of the S-VMS as the MAs will re-organize themselves differently in the overlay tree in the multi-cloud case by taking into account the communication delays between MAs residing on different clouds (see Figure 14). However, the construction and maintenance of the overlay tree is automatically handled by the MAs, and is transparent to the other PANACEA components.

[11] **These overlay trees are communication substrata which are internal to the Pervasive Monitoring System, and have no relation to the PANACEA Overlay Network described in Section 3.4.**

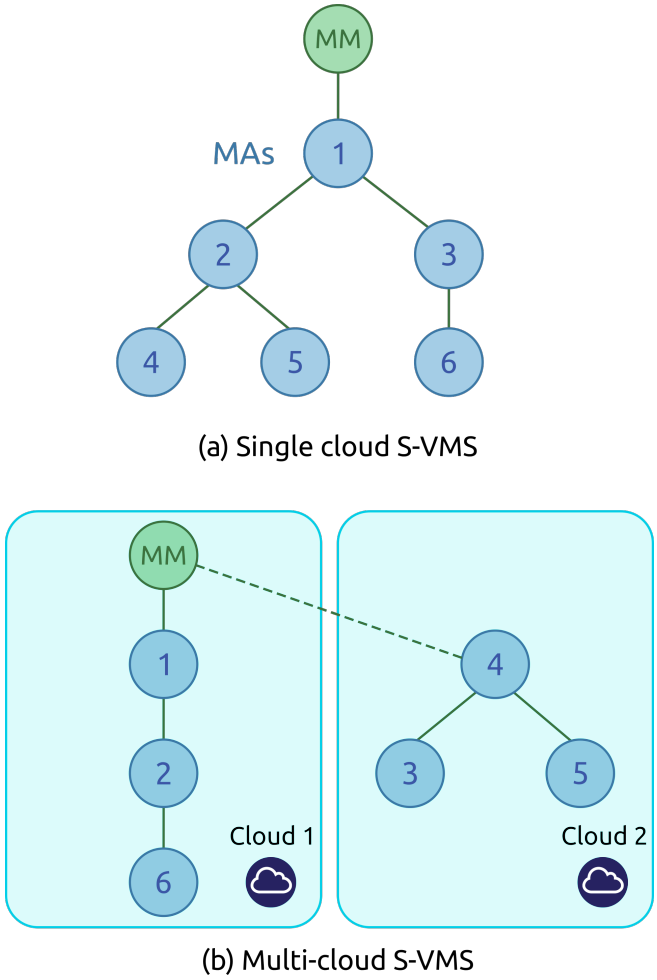


Figure 14: S-VMS deployment in single and multi-cloud scenarios

For monitoring the physical infrastructure in the multi-cloud case, in addition to the cloud specific P-VMSs, of which there is one per cloud, another inter-cloud P-VMS is constructed that connects to the MMs of the intra-cloud P-VMSs (see Figure 15). In this inter-cloud P-VMS, the intra-cloud MMs become the leaf nodes, and a new inter-cloud MM handles the monitoring across the multiple clouds.

This approach prevents the duplication of cloud specific monitoring data at multiple locations, i.e., at each intra-cloud P-VMS MM and also at the inter-cloud P-VMS MM. With this approach, the monitoring of the physical infrastructure across multiple clouds is handled solely by the inter-cloud P-VMS, while per cloud monitoring is handled by their respective intra-cloud P-VMS. This also improves access to the monitoring data as management components that only need data from a single cloud can directly access it from within the same cloud via the intra-cloud P-VMS MM.

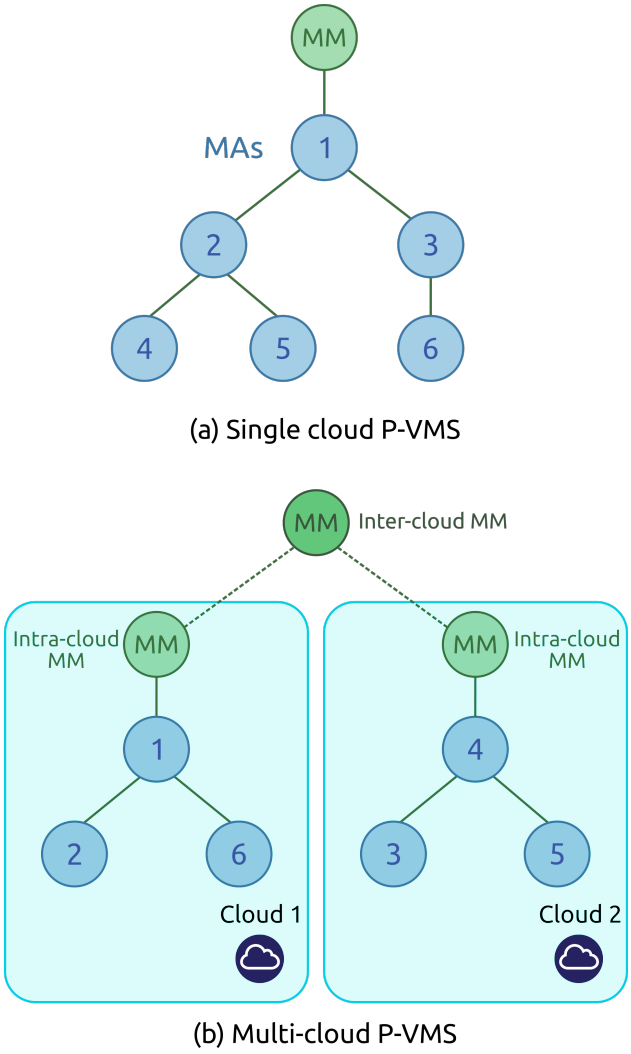


Figure 15: P-VMS deployment in single and multi-cloud scenarios

It is worthwhile to mention that the pervasive monitoring system handles monitoring of the physical and virtual computing and storage resources, i.e., the physical hosts and the VMs, while the overlay network agents discussed in Section 4.2.3 handle the monitoring of the network between the clouds.

In order to not duplicate monitoring effort, we have not implemented the network monitoring functionality within PMon, and such information can be obtained from the overlay network agents if necessary. Future implementations of PMon may include monitoring of the inter-cloud communication network.

### 4.2.5 Online QoS-Driven Task Allocation System in a Multi-Cloud Deployment

The deployment of the task allocation system does not change significantly in the multi-cloud scenario since the task allocation system is service-specific, and works with the already instantiated worker nodes that are part of the service.

In a multi-cloud scenario, the communication delays and losses between the task controller and a worker node located in the same cloud would be very different than those between the task controller and a worker node located in a different cloud. Therefore, these delays and losses should be known and used by the task allocation algorithm.

Another consideration is the placement of the task controller in a multi-cloud scenario. The task controller will be placed in one of the clouds that are hosting the multi-cloud service, but the decision

of which cloud it should be initialized in, and whether it should be moved to a different cloud during the operation of the service in order to improve reliability or performance is left to the autonomic service manager.

# 5 USE CASES WITHIN THE GLOBAL PANACEA ARCHITECTURE

## 5.1 Use case 1: Web Services in the Cloud

In the use case of web services hosted in the cloud, which was described in D4.1 [10], we employ the TPC-W benchmark in order to realize an online bookstore web application using the Apache Tomcat application server and the MySQL database. In the next section, we provide a brief overview of the TPC-W benchmark, and later we present the PANACEA architecture in use case 1.

### 5.1.1 TPC-W benchmark

The TPC-W benchmark[12] is a transactional web e-commerce benchmark, which simulates the activities of a business oriented transactional web server. TPC-W models an online bookstore and a set of operations in order to exercise the web application in a representative manner. The web application follows a typical two-tier web architecture, with one or more application servers implementing the application tier, and one or more database servers implementing the data tier (see Figure 16).

In the implementation used in our project, Apache Tomcat is used as the application server and the web container of the Java Servlets. MySQL is used as the database server. The application server handles multiple concurrent browser sessions, serves static and dynamic content, and performs online transaction processing by communicating with the database server. TPC-W's online bookstore implementation models the searching, browsing, shopping carts and secure purchasing, best sellers and new products, customer registration, and administrative updates functions.

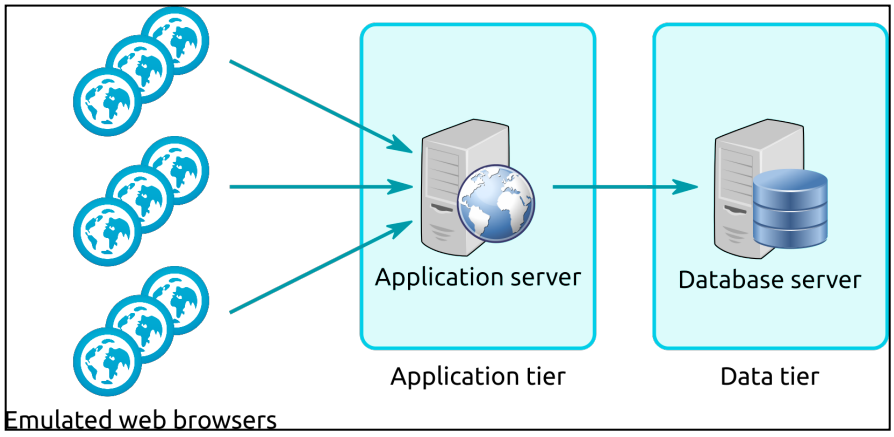


Figure 16: Two-tier architecture of the TPC-W based web services use case

In addition to modelling the web application, TPC-W also models the user browser sessions, i.e., the workload on the web application (see Figure 16). These emulated web browsers simulate the activities of multiple concurrent web users, with each one requesting web pages and images from the application server.

Depending on whether the requested content is static or dynamic, the application server may need to communicate with the database server in order to dynamically generate a response. The database server stores all of the data necessary to implement the online bookstore, including the inventory, customer records, orders, etc.

Through the emulated browser sessions, TPC-W simulates three types of workload by varying the ratio of browse to buy actions: primarily shopping (WIPS), browsing (WIPSb), and web-based

[12] **http://www.tpc.org/tpcw/**

ordering (WIPSo). The main performance metric tested by TPC-W is throughput, measured as the number of web interactions per second (WIPS).

### 5.1.2 PANACEA architecture in use case 1

In addition to the service components, i.e., the application server and the database server, defined in the TPC-W benchmark, the PANACEA architecture defines the following components for proactive and autonomic management of the cloud: autonomic cloud manager, autonomic service manager, online QoS-driven task allocation system, pervasive monitoring system, the machine learning framework, the overlay routing system, and self-awareness and self-management tools which are used by the service manager and the cloud manager to affect management actions.

Figure 17 shows a possible deployment of use case 1 in the cloud; in this figure, we only show the PANACEA architecture and the web service as deployed on a single cloud. The deployment on other clouds would be similar to the one shown here. We also omit the physical infrastructure related components from the figure, e.g., the cloud controller running the servers for OpenNebula, OneGate and OneFlow, the overlay proxy, and the monitoring system for the physical infrastructure (P-VMS).

Figure 17 shows the online bookstore web application realized via multiple application servers and a single database server. The self-managing web service has an associated service manager and a monitoring system (S-VMS). In this deployment, the service management related components have been instantiated in a single VM called the service controller VM. However, other deployments are possible, where the different management components reside on different VMs.

The web application also includes the online QoS-driven task allocation system as a load balancer. The task controller is implemented as a new load balancing algorithm of the *mod_proxy* module of the Apache server, and communicates with the S-VMS MM in order to get the monitoring data.

The machine learning agent running in each active VM locally accesses the monitoring data, and informs the service manager running in the controller VM when it predicts a failure or QoS violation. The service manager then takes a decision on which standby VM to activate and affects this action via the OneGate client, and also rejuvenates the VM that is predicted to fail. The rejuvenation process is initiated by the ML component which reaches the conclusion that the VM in question is about to fail. The operation itself is passed to the cloud management system which initiates the rejuvenation process itself by allocating a new standby VM to take over the failing one, and re-starting the soon to fail VM and assigning it as a standby.

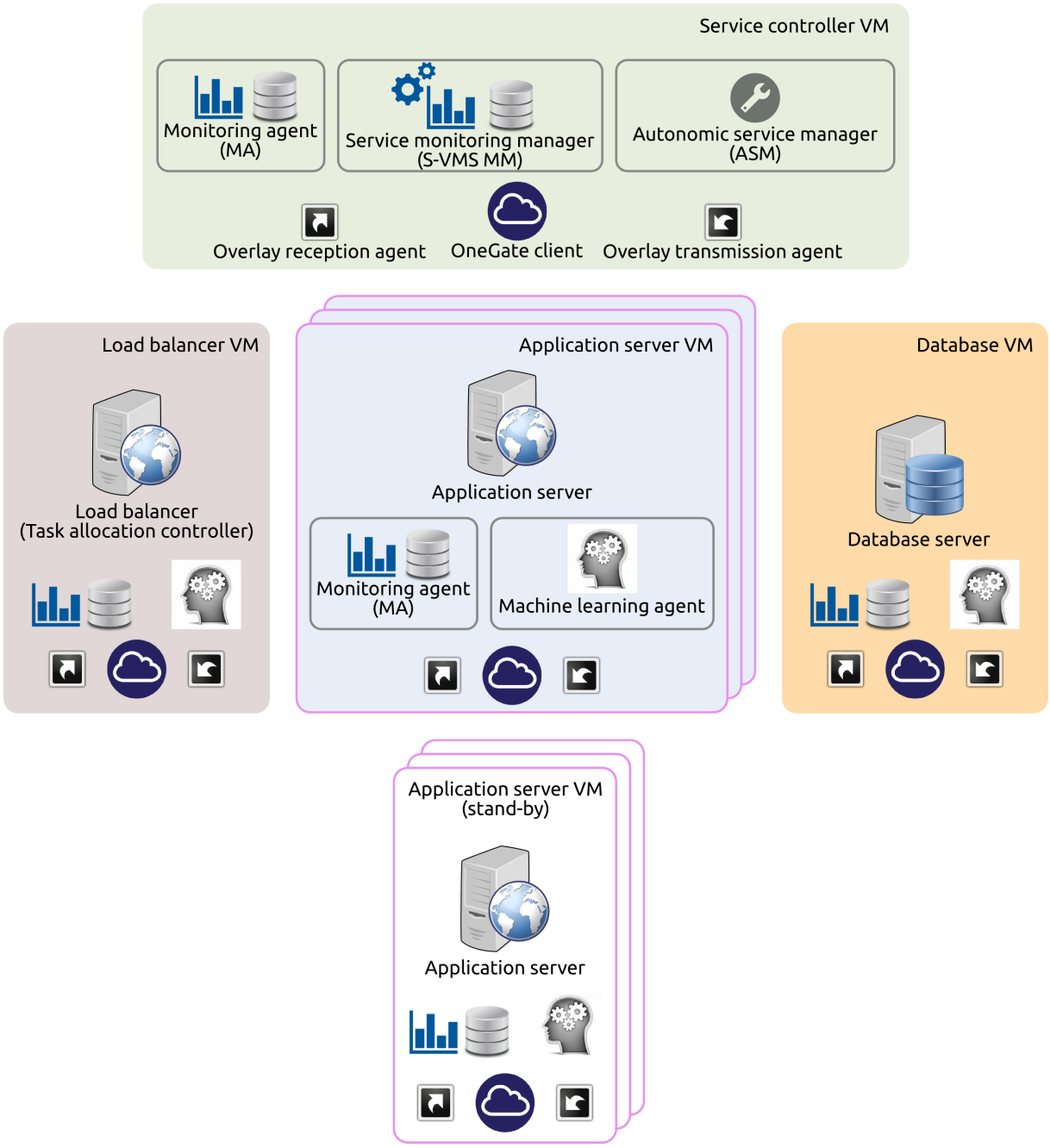


Figure 17: The PANACEA architecture in use case 1 (web services in the cloud)

## 5.2 Use case 2: Data Analytics as a Service

In this section we present the adaptation of the second use case, Data Analytics as a Service (DAaaS), to the PANACEA architecture presented in the previous sections. This use case was previously described at length in the deliverable D4.1 [10].

As it was discussed in D4.1, the main validation of the PANACEA solution for the DAaaS is focused on the adaptation of Apache Hadoop[13] to our proposed solution. In the following subsections we will

[13] **https://hadoop.apache.org/**

first introduce a short description of Apache Hadoop components, followed by a general block diagram description of the PANACEA architecture for DAaaS.

### 5.2.1 Apache Hadoop architecture

For this proposed use case we are using Apache Hadoop 2.6.x[14] that at the time of writing is the most stable version of Apache Hadoop (Apache released the version 2.7.0 at the end of April of the present year, but it is still not recommended for production environments[15]). A typical installation of Hadoop is composed of two central services and a lot of machines (physical or virtual) that are used to store data and perform calculations; those later machines are known in Hadoop nomenclature as resources or nodes.

In a very high level architecture, it is possible to say that Apache Hadoop it is composed of four main components:

- Resource Manager: Known as YARN[16], is responsible for assigning the different tasks that the Hadoop cluster needs to perform to the different Hadoop nodes. Hadoop follows a Master-Slave architecture, where the Resource Manager takes full control of the work needed to be performed by the Hadoop slaves/workers.
- Node Manager: Is responsible for obtaining the requests from the Resource Manager in the slave/worker node. It assigns resources to the different tasks that this node needs to run and is responsible to notify back to the Resource Manager on the progress and state of those tasks.
- NameNode: Is responsible for maintaining the Hadoop File System (HDFS). HDFS it is a distributed file system that keeps data replicated between the different nodes assigned to the NameNode. The data it is replicated in a way that if a node fails, no data is lost.
- DataNode: Is the component responsible for controlling the data stored inside a specific worker or slave. It is also responsible for accepting the requests to add/update/delete more data to the node, coming from the NameNode or other Hadoop workers.

Since Hadoop 2.0 the internal architecture of it changed greatly. Now each Hadoop application submitted to perform some kind of calculation runs in its own independent environment from the rest of the applications. YARN assigns partitions of the workers to it. A Hadoop application can be divided into the following components:

- Application Master: Is responsible for coordinating the workflow of the different calculations a Hadoop application needs to perform.
- Container: Is a set of resources from a Hadoop worker assigned to a specific Hadoop application to perform calculations.

### 5.2.2 Adaptation of DAaaS to the proposed PANACEA architecture

In this section we detail how a typical installation of Hadoop was modified to add to it the four autonomic properties: self-healing, self-configuring, self-optimizing, and self-protecting. A typical deployment diagram for the deployment of DAaaS can be seen in *Figure 18*:

[14] **https://hadoop.apache.org/releases.html#18+November%2C+2014%3A+Release+2.6.0+available**

[15] **https://hadoop.apache.org/releases.html#21+April+2015%3A+Release+2.7.0+available**

[16] **http://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html**

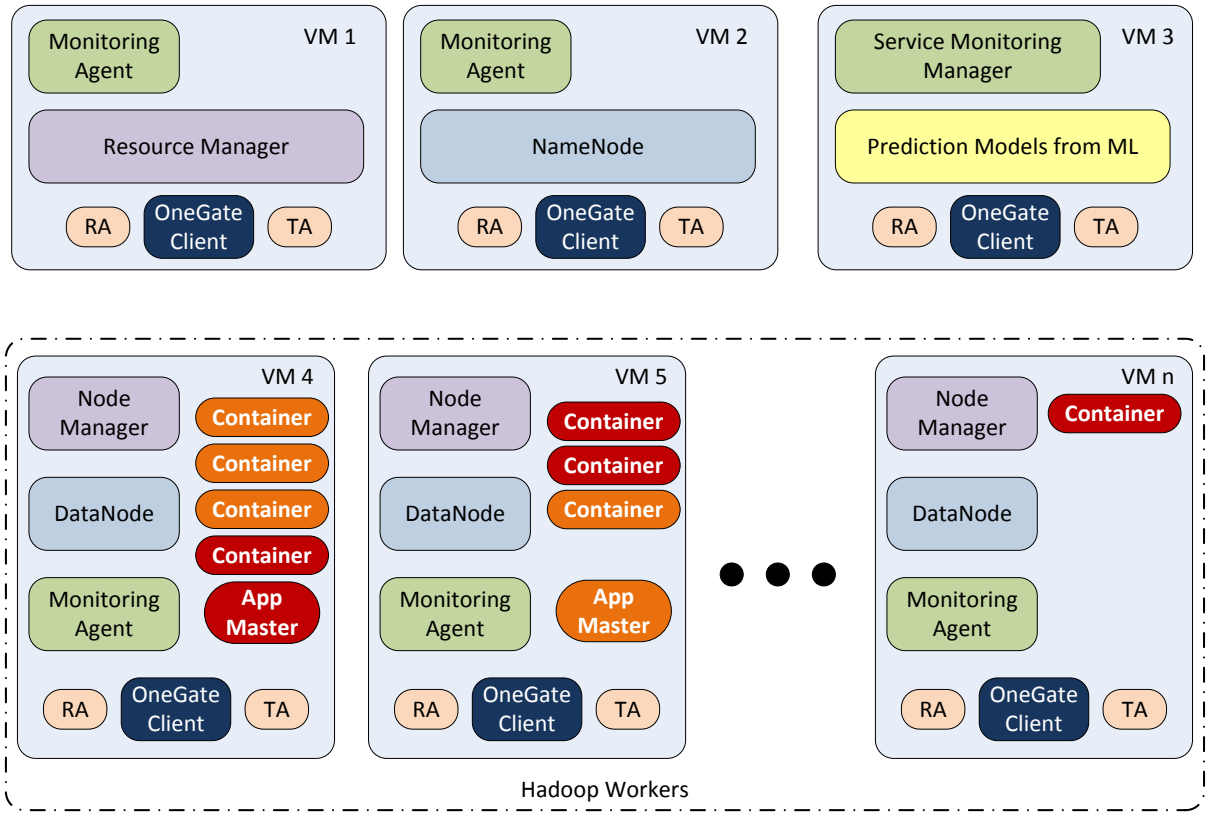


*Figure 18 Block diagram of the deployment of DAaaS Hadoop in PANACEA.*

Using this figure as base, let's go one by one around the autonomic features of PANACEA.

- *Self-healing*: In deliverable D4.1 [10] we introduced typical failure scenarios for a Hadoop cluster. Monitoring data collected while those typical failures occur will be used as a base for training the Machine Learning framework. From there Prediction Models will be created and put into a module that will continuously read the Hadoop monitoring information to anticipate possible problems in the cluster and proceed to initiate different rejuvenation actions.
- *Self-configuring*: Hadoop VM images were adapted to take full advantage of the new tools provided by OpenNebula: OneFlow and OneGate. Apache Hadoop has been configured as a deployable service by OneFlow and a set of logic and scripts have been created to take advantage of OneGate to scale-up or down the Hadoop deployment.
- *Self-optimizing*: Thanks to the usage of the PANACEA Pervasive Monitoring system, Hadoop can monitor itself and know when the amount of resources is too high or too low depending on the incoming load. By monitoring this information and by the usage of OneGate, Hadoop can request the creation or deletion of VMs to act as Hadoop workers. In this way, we optimize the amount of resources used during the lifetime of the cluster.
- *Self-protecting*: All VMs in the Hadoop cluster are equipped with the overlay network modifications, in this way, when deployed in an inter-cloud scenario; they could be protected against DDOS attacks.

# 6 SUMMARY AND FUTURE WORK

In this document we presented the architecture of PANACEA. We provided details on the various components, their technologies, and the way they interact with each other, in order to provide proactive autonomic management of cloud resources. The use cases are utilized in order to demonstrate the PANACEA capabilities in common cloud situations. After investing initially in building the separate components we are now moving to an integrated practical cloud system. Using the architecture blueprint provide in this document, we started integrating the various technologies that enable the proactive autonomic management proposed by PANACEA. We have already integrated most of our technologies and reported some very promising results. We continue to work on this integration both in the single cloud instance and in the federated case.

# REFERENCES


**[1]** J. O. Kephart, D. M. Chess: "The vision of autonomic computing", Computer 36 (1): 41-50, 2003. http://dx.doi.org/10.1109/MC.2003.1160055

**[2]** A. G. Ganek, T. A. Corbi: "The dawning of the autonomic computing era", IBM Systems Journal 42 (1): 5-18, 2003. http://dx.doi.org/10.1147/sj.421.0005

**[3]** PANACEA D5.1: Market analysis, business models and value chain. Feb. 2015

**[4]** PANACEA D2.1: Principles of pervasive monitoring. Apr. 2014

**[5]** PANACEA D2.2: Design of a pervasive monitor. Dec. 2014

**[6]** PANACEA D2.3: Autonomic Communication Overlay. March 2015

**[7]** ETSI: "Network Functions Virtualization – Introductory White Paper," Oct. 2012

**[8]** PANACEA D3.1: Implementation of a virtualization framework at a node level of the overlay, based on open source software and developed in the project tools, for realizing the machine learning framework. Oct. 2014

**[9]** PANACEA D3.2: Machine learning framework and global architecture for proactive management. Dec. 2015.

**[10]** PANACEA D4.1: Description of feasible use case. March 2014.

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

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

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

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

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

**[16]** PANACEA D3.3: Machine Learning solution for Intra and Inter Autonomic Cloud Managers (ACMs) for a proactive management of cloud resources. July 2015


 