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

PANACEA

Proactive Autonomic Management of Cloud Resources

D4.4: Implementation and Evaluation of the PANACEA integrated System

Version: v.1.0

Work package	WP 4
Task	Task 4.3
Due date	30/03/2016
Submission date	21/04/2016
Deliverable lead	Imperial
Version	0.1
Authors	O. Brun, D. Garcia, E. Gelenbe, L. Wang, A. Rachdi, E. Huedo, D. Avresky
Reviewers	O. Brun, A. Rachdi

Abstract	PANACEA has developed innovative solutions enabling cloud services to manage themselves without direct human intervention, so as to significantly increase their availability and performance, while reducing costs of operations. In this deliverable, we evaluate to what extend PANACEA objectives have been reached using two use cases, related to cloud web hosting and to big data analytics. We present results regarding the individual validation of PANACEA components as well as results for the validation of the global PANACEA solution, evaluating a number of technical metrics to measure the impact of PANACEA.
Keywords	Cloud, proactive autonomic management, Machine Learning, task allocation, routing overlay, autonomic elasticity, cloud web hosting, Hadoop.

Document Revision History

Version	Date	Description of change	List of contributor(s)
V1.0	17.03. 2016	Initial template of D4.4	O. Brun (CNRS), A. Rachdi (QOS)
V1.1	10.04.2016	Description of results obtained for UC1 and UC2	D. Garcia (ATOS), E. Gelenbe (IMPERIAL), L. Wang (IMPERIAL), E. Huedo (UCM)
V2.0	13.04.2016	Description of a live experiment with SMART	O. Brun (CNRS)
V2.1	18.04.2016	Description of the experiments with the MLF	D. Avresky (IRIANC)
V2.2	21.04.2016	Abstract, Executive summary and Conclusion	O. Brun (CNRS)
V2.3	29.04.2016	Experiments with SMART in Use Case 2	D. Garcia (ATOS)
V2.4	30.04.2016	Evaluation of the accuracy of the prediction model for Use Case 2	D. Avresky (IRIANC)

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	

EXECUTIVE SUMMARY

The goal of PANACEA was to significantly increase the availability and performance of services deployed in the cloud by designing and developing innovative solutions for a proactive and autonomic management scheme of cloud services. PANACEA-enabled services are able to manage themselves without direct human intervention, recovering from many inevitable anomalies and autonomously optimizing their performance in changing conditions.

In this deliverable, we evaluate to what extend PANACEA objectives have been reached using two use cases. The first use case is on cloud-hosted web services. It provides a first-stage validation that clearly showcases the PANACEA innovations while easing the integration work due to its simplicity and the familiarity of the involved technologies. The second use case deals with a new cloud service, Data analytics as a service (DAaaS), which provides an end-to-end data analytics platform, from data acquisition to end-user visualization, reporting and interaction. This second use case is used to showcase the PANACEA technology in a more complicated but also more powerful application domain.

We first introduce an application scenario where a company offering mobile fitness services for the sports segment, has to run web services and performs analytics of large volumes of data collected from user devices. We discuss the link with our use cases and explain how this service provider could benefit from our technologies. We then present the results obtained for the two use cases, evaluating a number of technical metrics to measure the impact of PANACEA innovations. For each use case, we present both results for the individual validation of PANACEA components, and results for the integrated validation of the global PANACEA solution.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS

ACM	Autonomic Cloud Manager
AM	Application Manager
API	Application Programming Interface
CAPEX	Capital Expenditure
CDN	Content Delivery Network
CLI	Command Line Interface
CPN	Cognitive Packet Network
CPU	Central Processing Unit
DAaaS	Data Analytics as a Service
DB	Database
DDoS	Distributed Denial of Service
DNS	Domain Name System
FIFO	First-in, First-out
GUI	Graphical User Interface
HDFS	Hadoop Distributed File System
IaaS	Infrastructure as a Service
I/O	Input/Output
IoT	Internet of Things
IP	Internet Protocol
IT	Information Technology
JDK	Java Development Kit
JVM	Java Virtual Machine
MAE	Maximum Absolute Error
ML	Machine Learning
MLF	Machine Learning Framework
NFV	Network Functions Virtualization
NM	Node Manager
O&M	Operations and Maintenance
OCA	OpenNebula Cloud Application Programming Interface
OCCI	Open Cloud Computing Interface
OPEX	Operational Expenditure
OS	Operating System

OVF	Open Virtualization Format
OVA	Open Virtualization Appliance
PaaS	Platform as a Service
QoS	Quality of Service
REST	Representational State Transfer
RM	Resource Manager
SLA	Service Level Agreement
SVM	Support Vector Machine
TPC-W	Transaction Processing Performance Council Web e-Commerce Benchmark
VLAN	Virtual Local Area Network
VM	Virtual Machine
VNF	Virtualized Network Functions
VXLAN	Virtual Extensible Local Area Network
WAN	Wide Area Network
WEKA	Waikato Environment for Knowledge Analysis
WP	Work Package
XML-RPC	Extended Markup Language - Remote Procedure Call
YARN	Apache Hadoop Next Generation MapReduce

1 INTRODUCTION

Even small degradations in the performance and availability of modern computer services can have a considerable business impact in terms of damage to brand reputation, lost revenue and reduced productivity [1][2]. Yet, modern computer services have reached a level of complexity where the human effort required to get them up and running at expected performance levels is becoming prohibitively expensive.

Autonomic computing is emerging as a significant new approach to the design of computer services. Inspired by the autonomic nervous system of the human body, it aims at enabling computer systems to manage themselves with minimal direct human intervention, while at the same time improving their performance and availability.

The primary goal of PANACEA is to significantly increase the availability and performance of services deployed in the cloud. To this end, we have designed and developed innovative solutions for a proactive and autonomic management scheme of cloud services. PANACEA-enabled services are agile services that are able to autonomously and dynamically

- grow and shrink resources allocated to them as and when needed to meet demand and to draw those resources from the most optimal location,
- dispatch incoming jobs to the best available resources in order to maintain and improve the response time of jobs in response to changing workload conditions,
- detect and quickly recover from path outages or performance degradations of inter-cloud IP routes, always routing flows over the most optimal routes,
- heal themselves by predicting performance degradations or service crash due to the accumulation of errors (e.g., memory leaks) and proactively reconfiguring themselves.

In order to provide these new capabilities to cloud services, the following innovations have been developed in PANACEA [4]:

- **Proactive Service Management Using Machine Learning**: By the time a service failure or QoS violation is detected, it is too late and end-users are already experiencing service downtime or decreased QoS. A proactive approach is therefore required, and the proactive service management solution developed by PANACEA improve service availability and QoS by predicting anomalies such as software failures and response time violations, and then reconfiguring the service before the predicted anomaly occurs (e.g., by rejuvenating a VM or by transferring part of the service to a stand-by VM).
- **Cloud Management Platform for Autonomic Services**: The cloud management solution of PANACEA provides the effector (self-configuration) and sensor (self-awareness) mechanisms necessary for autonomic management of services, which enables unattended services to request or provide information about individual VMs or the service itself and request reconfiguration operations, like migrating to another host or attaching a new device. In particular, it enables services to autonomously grow and shrink resources allocated to them as and when needed.
- **Pervasive Cloud Monitoring**: The proposed monitoring solution *self-optimizes by adaptively prioritizing which nodes to monitor in the cloud*, thereby achieving low overhead while providing timely delivery of fine-grained monitoring data, and

enabling an accurate representation of the state of the cloud, which is critical for good management and allocation decisions.

- **Online QoS-driven Task Allocation**: The online QoS-driven task allocation system uses machine learning techniques and online measurements in order to improve user experience by maintaining or improving QoS, such as response time, in the face of overload conditions and changes in demand and the infrastructure.
- **Routing Overlay**: In order to improve the Internet communication paths between clouds, we have developed *a self-healing and self-optimizing overlay network* that quickly detects path failures and QoS degradations, and adapts the overlay paths in response to failures and congestion

The main expected impact of these new capabilities is improved availability and QoS of cloud services. This will translate into the provisioning of better services by the cloud providers, thus improving revenue, brand name, company visibility, reputation, and competitiveness in the market. In addition, the provider will also see a reduction in its OPEX since autonomic services will require less supervision and manual reconfiguration and intervention. The PANACEA technologies will also have a positive impact on the service providers and their end-users, both of which will benefit from higher availability and improved QoS.

The goal of this deliverable is to evaluate to what extend PANACEA objectives have been reached, using the two use cases defined in deliverable D4.1 [3]. The first use case is on cloud-hosted web services, while the second is about big data in the cloud. The first use case provides a first-stage validation that clearly showcases the PANACEA innovations while easing the integration work due to its simplicity and the familiarity of the involved technologies. The second use case deals with a new cloud service, Data analytics as a service (DAaaS), which provides an end-to-end data analytics platform, from data acquisition to end-user visualization, reporting and interaction. This second use case is used to showcase the PANACEA technology in a more complicated but also more powerful application domain. Table 1 summarizes the technical metrics that will be used to evaluate the impact of each of the PANACEA innovations. Please refer to [3] for a more detailed description of the technical metrics addressed in each use case.

The rest of this document is organized as follows. Section 2 introduces an application scenario where a service provider, offering mobile fitness services for the sports segment, is confronted with various performance and availability issues. The service provider has to run web services and performs analytics of large volumes of data collected from user devices. We discuss the link with our use cases and explain how this service provider could benefit from our technologies. Section 3 is devoted to experiments related to the cloud web hosting use case, whereas Section 4 presents the empirical results obtained in the use case related to Data Analytics. Finally, we conclude in Section 5 with a brief summary.

Innovation	Technical metrics
Proactive Service Management Using Machine Learning	• Improvement in availability offered by the PANACEA innovation compared to an unmanaged service. • Prediction buffer time: time interval between a reliable anomaly prediction is made and when it effectively occurs. • Number of parameters that need to be monitored. • False positive and false negative ratios
Cloud Management Platform for Autonomic Services	• Improvement in job completion time • Amount of resources consumed in order to complete a given job.
Pervasive Cloud Monitoring	• Network overhead due to data exchanged between monitored resources and monitoring managers. • Latency of data acquisition: duration between the time a data request is sent by the monitoring manager until the time it gets back a measurement. • Loss rate of probes and replies. • Local overhead due to the monitoring agent.
Online QoS-driven Task Allocation	• Improvement in response times compared to other typical load balancing algorithms (e.g., round-robin)
Routing Overlay	• Typical network metrics: throughput, packet latency and loss rate. • Path discovery time, i.e., the time it takes to find a new optimal path between two clouds, • Network overhead of the overlay system due to monitoring and control.

Table 1: Technical metrics used to evaluate the impact of PANACEA innovations.

2 APPLICATION SCENARIO

Nowadays, if someone is looking to be more proactive about his health and fitness, "there's an app for that" rings very true. Both iPhone and Android smartphone markets contain a cornucopia of health and fitness applications. Some of these applications are very popular today. For instance, Runtastic, which is offering mobile fitness applications, services and apps for the sports segment since the end of 2009, is claiming to have more than 25 million mobile users and more than 30 million app downloads. The rapidly growing number of users follow their running performance, look up statistics, compare themselves with peer groups and are cheered on via social media elements.

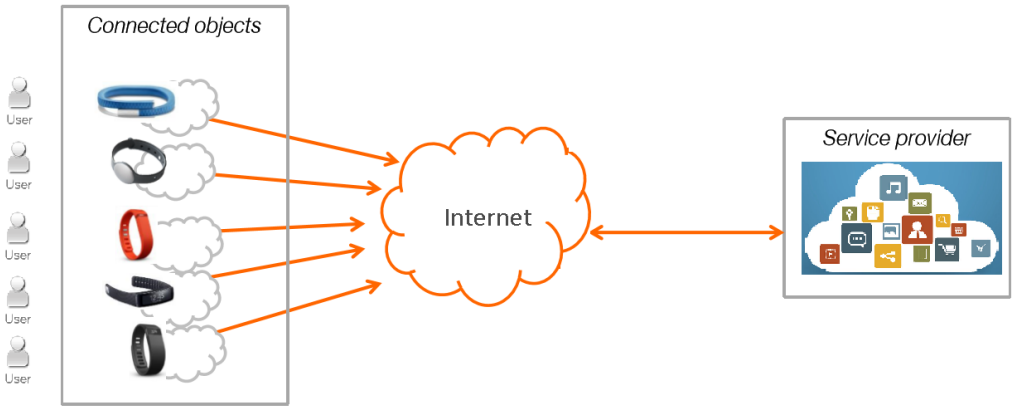


Figure 1: Health and fitness applications

From a service provider perspective, those applications are characterized by three components: (1) the **web services**, (2) the **associated analytics** and (3) the mobile platform. In addition to permanent availability, it's also crucial that service provider can offset peak loads, since customers tend to use their apps simultaneously during their free time – usually early in the morning or in the evening. Increased utilization thereby leads to peak loads in the IT infrastructure - and at different times.

2.1 Web Services

During the peak periods, millions of connections will crave the web service. This basically, will lead to multiple issues that we propose to solve with the PANACEA approach. Among the possible problems, we can find:

- **Provisioning resources**: Dynamic and appropriate resource provisioning is a crucial issue in Cloud Computing. Broadly speaking, the goal is to find a trade-off between the performance offered to users and the cost of the virtual machines used to run the service. Rather than using a fixed amount of allocated resources, resource provisioning can be adaptive and elastic so that allocated resources match closely the real-time workload.

- **Monitoring performance and resource utilization**: the monitoring solution can adaptively prioritize which web servers to monitor in the cloud in order to enable an accurate representation of the state of the system with low overhead.
- **Load balancing**: the task allocation system can improve user experience by forwarding incoming requests to the most appropriate server in order to minimize response time
- **Network performance optimization**: in order to provide permanent availability and an optimal quality of experience to users, we have to address shortcomings of IP routing. The routing overlay will improve the availability and quality of web services by finding an alternate path when an IP route becomes unavailable, even if Internet routing protocols cannot, and routing traffic based on metrics directly related to application performances.

All those new capabilities will be highlighted by Use case 1 experiments.

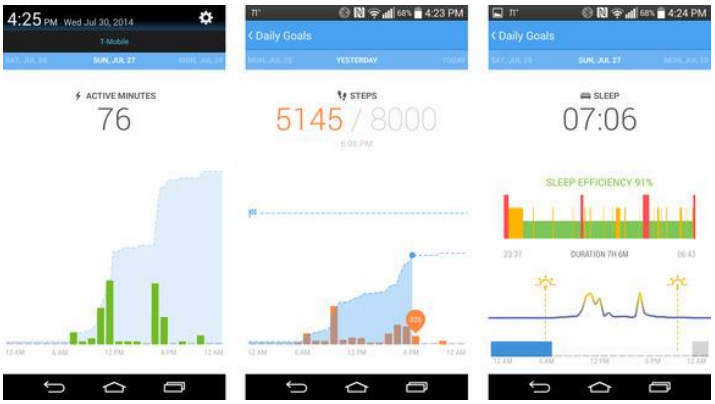


Figure 2: Example of a sport application on smartphone.

2.2 Data Analytics

To customize the user experience, health and fitness applications are providing personal statistics to the user. While personal statistics are handled by the mobile platform, the global statistics (by town, country) or challenges and the trends are handled at the cloud level.

Due to the very large number of users, providing these statistics usually implies the massively parallel processing of large volumes of data, such as with Hadoop Map/Reduce for example. The latter is a computational approach that involves breaking large volumes of data down into smaller batches, and processing them separately and in parallel on multiple computing nodes. Then outputs of these nodes are combined to generate the final result. Experiments from use case 2 will showcase how Panacea components will enhance the performance of such an approach.

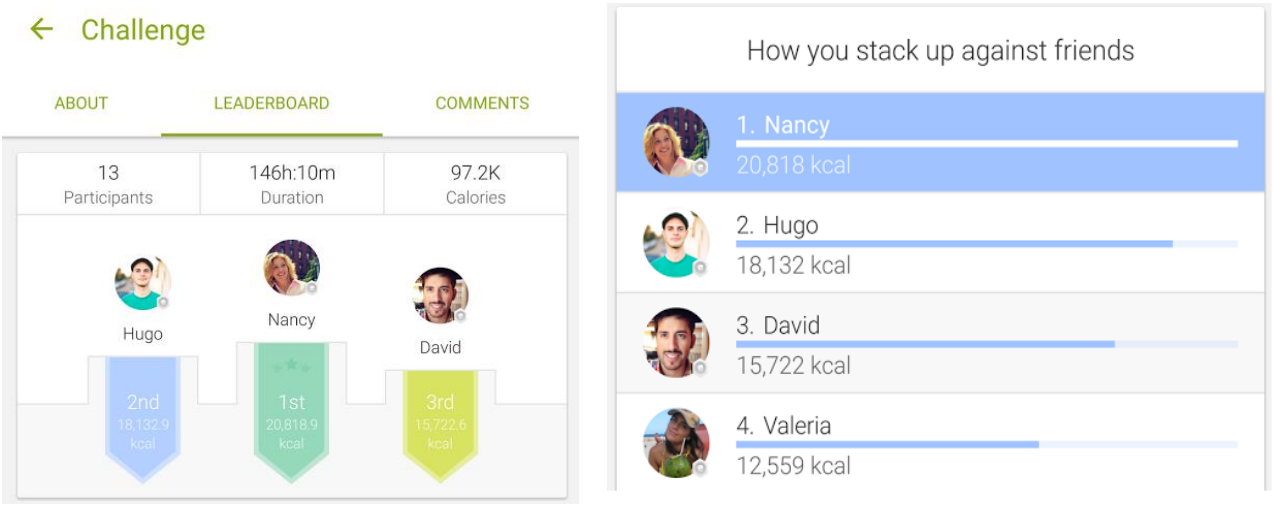


Figure 3: Comparison of user' s performance with peer groups in sport applications.

PANACEA can help the service provider to handle some of the data analytics issues while minimizing the overall effort. For example:

- **Poor monitoring strategy:** The monitoring solution selected to monitor the health of the application is difficult to tune. The application has many peak hours (for example: European users that go around the same hour to the gym) and many times the monitoring infrastructure is collecting fine-grained data that is unnecessary over other periods of the day.
  ⇒ The **Pervasive Monitoring** solution proposed by PANACEA will help to fine tune the collection of data depending on the amount of information received, collecting only few samples of a metric when no variation is detected and increasing the sampling rate when a lot of variation happens. In this way, the application is not storing hundreds of GB of unnecessary monitoring data and the service provider will not miss important information because it underestimated the monitoring resources.
- **Self-configuring:** As mentioned above, the workload of sport applications is highly volatile and the volumes of data to process for producing statistics vary a lot over time (the application servers tend to get a lot more of data the weekends). Online policies allowing to automatically adjust resources allocated with the real-time workload are therefore required.

  ⇒ The **Self-configuring technologies** of PANACEA are ideal for this. The application can use monitoring data to infer its state and grow or shrink the amount of allocated resources as, when and where needed (in different countries if the load comes from a specific place). In this way, the application will use only the necessary resources, without overprovisioning. Since this process is automatic, the OPEX costs are also decreased.

- **Self-Healing:** Even as we increasingly rely on the correct operation of distributed applications, it is becoming ever harder to make them entirely bug-free before deployment in the cloud. There is therefore a need to design autonomic applications that can monitor themselves online and quickly detect and recover from failures arising either from program faults or from changes in the environment. The massively parallel processing of big data would clearly benefits from such an approach.
  - ⇒ the **proactive service management** solution developed by PANACEA improve service availability and QoS by predicting anomalies such as software failures and response time violations, and then reconfiguring the service before the predicted anomaly occurs (e.g., by rejuvenating a VM or by transferring part of the service to a stand-by VM).
  - ⇒ The **PANACEA overlay network** can protect the application from network failures by finding alternate path, even when Internet routing protocols cannot.
  - ⇒ The **Frappe Framework** can also help increase availability since it manages critical configuration and operations information that is stored and shared between cluster members and services.
  - ⇒ The **overlay simulation platform** can help the service provider to anticipate the behaviour of its application under adverse events in a nearly identical production environment.
- **Self-Optimizing:** end-users experience can be negatively impacted by bad performance due to servers overload or network delays. PANACEA provides a twofold answer to this problem.
  - ⇒ **Autonomic job allocation** uses online measurements to make fast task allocation decisions in order to minimize the response time of jobs.
  - ⇒ **PANACEA overlay network** discovers the optimal routes within the overlay network for service-specific routing metrics with a quite limited measurement overhead.

All those capabilities will be highlighted by Use case 2 experiments.

3 EXPERIMENTS IN USE CASE 1

3.1 Experiment setup

In this use case, we consider a typical tiered web application consisting of the web servers that host the application logic and also provide a user interface to the clients, and of the database that stores all the information (users, transactions, etc). We use the specific technologies of the Apache Tomcat web server[1], the MySQL database server[2] implemented in Java. In order to emulate web clients and requests, we have used either the TPC-W benchmarks[3] or the HTTPerf tool.

The TPC-W benchmark is a web server and performance benchmark proposed by the Transaction Processing Performance Council. It defines a complete multi-tier e-commerce web application simulating an online bookstore, the architecture of which is shown in Figure 4. The client tier consists of the web clients accessing the online store, which are emulated and generate a configurable rate of requests. The client requests can be of different types, e.g., searching, browsing, ordering, and it is possible to configure different types of clients, e.g., heavy browsers. The client requests pass through a network, which may be emulated or a real network, and arrive at the web application. There are different deployment options for the web application; all components of the application can be hosted on a single server, or the different components can be spread across multiple servers, e.g., one server hosting the HTTP server, the other hosting the application server, and another server hosting the database.

On the other hand, Httperf is a simple tool for measuring web server performance. It provides a flexible facility for generating various HTTP workloads and for measuring server performance. On the contrary of TPC-W, the focus of httperf is not on implementing one particular benchmark but on providing a robust, high-performance tool that facilitates the construction of both micro- and macro-level benchmarks.

In addition, inside certain VMs hosting the application servers, we inject anomalies in the form of memory leaks of random sizes and unterminated threads in order to emulate a misbehaving or poorly configured web application.

Use case 1 was used in a number of different scenarios for progressively validating the individual PANACEA innovations, and for their integration. We first present empirical results validating the individual components of the PANACEA solution. Thus, empirical results for the validation of the Machine Learning Framework are presented in Section 3.2, whereas results for the validation of the autonomic the task allocation system, the autonomic communication system and the simulation/emulation environment are presented in sections 3.3, 3.4 and 3.5, respectively. Section 3.6 is devoted to the integrated validation of these components.

[1] http://tomcat.apache.org/

[2] http://www.mysql.com/

[3] http://www.tpc.org/tpcw/

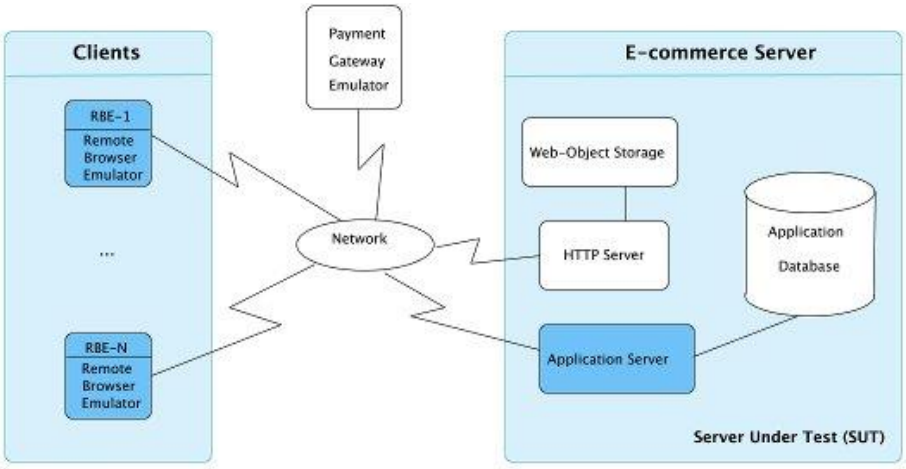


Figure 4 TPC-W multi-tier web application architecture

3.2 Machine Learning Framework for Proactive Self-* (Healing, Optimizing and Configuring)

The initial application of this use case is the validation of the MLF, which has been done and its results have been presented in D3.1 [25] and D3.2 [26]. The main aim of these experiments are to show that given a fixed workload and anomaly pattern, the MLF can predict the time-to-crash of the application server and the time-to-violation of a given response time threshold with small maximum absolute error. A complementary goal was to show that the MLF can reduce the number of metrics that need to be monitored during the operational phase by the application of Lasso regularization. Our experimental results show that the MLF can perform all these functions, as discussed in detail later in this section, and that this innovation can demonstrate an increase of at least 20% in the availability of web applications, which was one of the major goals of the project.

3.2.1 Introduction

Performance and availability of computer systems is affected by the accumulation of anomalies of different nature, such as memory leaks, unterminated threads, unreleased locks, and/or file fragmentation. Most of the times, occurrences of anomalies are complex to predict. Further, discovering the related causes and eliminating errors via software debugging may be hard. In [27], it has been shown that in web applications an average of 40% of anomalies is due to software errors. As a consequence, these kinds of problems need to be addressed by run-time management of systems. This would require continuous monitoring of a system to detect and manage unexpected behaviours, such as excessive performance degradation or service outage.

In many cases (such as 24/7 web applications), the ability to timely recovery the system from failures, and/or to restore adequate performance level, is of paramount importance. An approach to cope with the problem of accumulation of anomalies consists of periodically performing recovery actions forcing the system to a "clean" state (namely, a state where the system works without—or with a reduced number of— anomalies). This prevents the occurrence of further anomalies from (quickly) leading the system to unexpected behaviours.

We developed a recovery action " proactive software rejuvenation" [28], which is typically performed by restarting the application responsible of generating anomalies, or even the hosting machine. A recovery action can be performed at pre-established time instants (time-based approach), or proactively, i.e. when the system is predicted to approach an undesired state. In the first case, actions are performed independently of the actual working state of the system. Conversely, in the second case, they are performed based on run-time system monitoring. A proactive approach requires more complex techniques than the time-based counterpart. In fact, estimating the best-suited time for recovering the system from failures, or to restore adequate performance level, can be non-trivial, given that predicting the occurrence of anomalies and their effects on the system is typically hard. On the other hand, benefits of proactive management can be significant, in terms of both system service downtime and system performance.

Recently, in [30], it has been shown that the proactive management of software anomalies can efficiently exploit Machine Learning (ML) algorithms to predict the time to crash of applications. In the proposed approach, the system has been trained until crashing in the presence of anomalies, and values of some system feature have been collected (CPU utilization, memory usage, etc.). After having been collected, values of system features are fed to the ML algorithm for building a model to predict the Remaining Time to Crash (RTTC) of the system. The benefits of this approach are related to the fact that a recovery action can be executed before the (predicted) failure time, or even before that the system performance goes below a given level. As well, actions could take place before some Service-Level Agreement (SLA) levels are violated, e.g. the response time increases over a given threshold, or availability is below a certain percentage. In[30], we also showed that ML-based approaches can be used to predict the occurrence of both system crashes and other events, such as violations of performance thresholds, also in the presence of different kinds of software anomalies.

In the sequel, we present a ML-based framework for Proactive Client-server Application Management (PCAM) in the cloud. PCAM exploits failure prediction models produced by a ML framework that we presented in [30], called F2PM. PCAM targets a client-server application model, with replicated server instances. We assume that server instances are deployed on virtual machines (VMs) provided by a cloud IaaS (Infrastructure as a Service) [31]. Server instances can be added/removed at run-time (by adding/removing VMs), in order to dynamically scale the server pool according to the system workload. Also, server instances are subject to software anomalies, which can lead VMs to fail, as well as to cause degraded performance level over time. PCAM [30] is able to trigger recovery actions for any VM on the basis of run-time predictions of the its Remaining Time to Failure (RTTF) and on VM performance measurements. The RTTF of a VM is predicted based on the time when a given failure condition is expected to be true. The failure condition corresponds to a system crash or to the violation of some user-defined thresholds related to system performance requirements (e.g. when the average response time exceeds a user-defined value). PCAM uses RTTF prediction models generated according to the scheme used in F2PM.

Particularly, with respect to previous literature studies, our approach advances in several directions:

1. We target more complex and largely common application deployments, where multiple server instances are replicated on different hosting machines, and where different kinds of anomalies can occur, also with different occurrence patterns;

2. We use both ML-based predictions and run-time system performance measurements to decide when a recovery action has to be executed;
3. We evaluate our framework in the case of a test-bed application reproducing a real-world scenario, where we run the system with different configurations and by injecting different anomaly patterns.

The PCAM approach, on the one hand, aims at improving both the system availability (by reducing the system downtime due to failures) and the system performance (by keeping it up to user-established levels). On the other hand, it allows to reduce significantly the management effort for keeping the system operational, supporting self-* (healing, optimizing, configuring) system properties. PCAM is not bound to specific applications, because it only requires monitoring parameters at hosting machine and operating system level. Therefore, besides our target experimental web applications, PCAM can be used, as well, in other kinds of client-server applications, acting in a completely application-agnostic way.

3.2.2 Experimental evaluation

The results presented below have been obtained on the System Architecture shown in Figure 5.

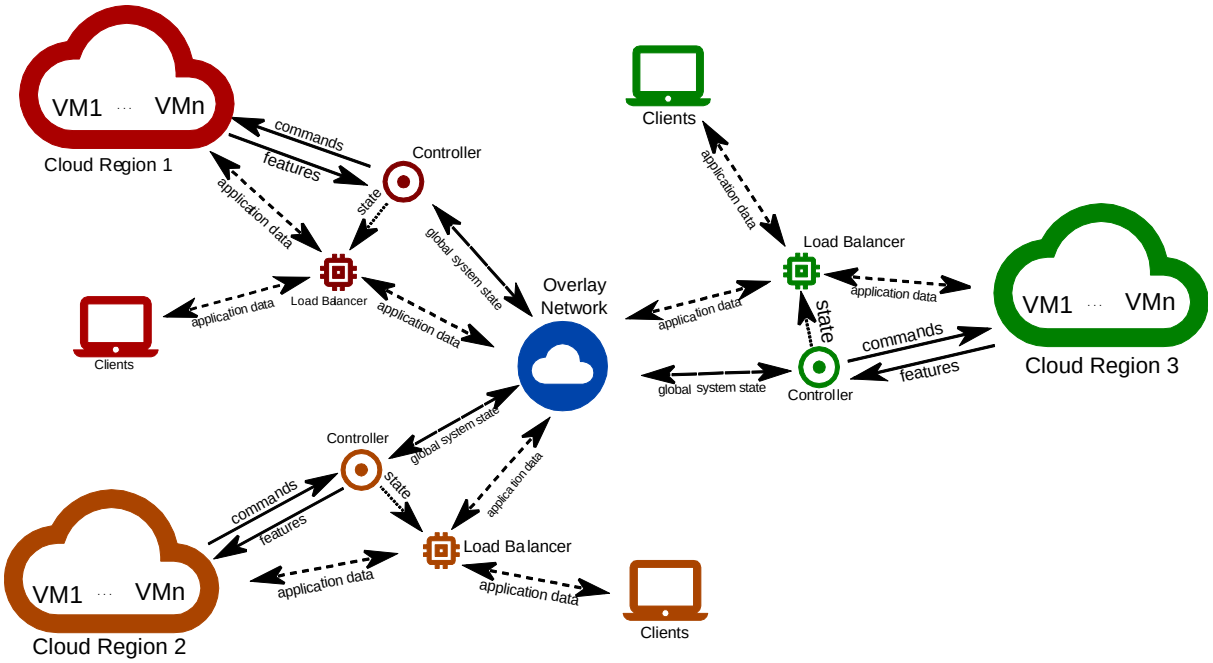


Figure 5 System Architecture for a proactive management of cloud resources based on Machine Learning.

We present experimental results [28] for the cases of two scenarios. In the first one, we used only one couple of slave VMs, say c1. In this scenario, client requests are processed by one active VM. Both memory leaks and unterminated threads are injected in slave VMs. When the predicted RTTF of the active VM is smaller than the threshold T, the MU executes the procedure for switching the active machine from VM11 to VM21 or vice versa. In the second scenario, we used three couple of VMs, say c1, c2 and c3. Thus, in this case, client requests

are processed by three active VMs (one per couple of VMs). The active VM of a couple is switched independently of the other couples of VMs. In this scenario, we injected both memory leaks and unterminated threads in VM11 and VM21. Conversely, we injected only unterminated threads (memory leaks) in VM12 and VM22 (VM13 and VM23). In both experiments, we used the prediction model generated by M5P algorithm.

Initially, in both scenarios, we set the threshold T equal to 300 seconds. Upon restarting a VM, the probability of generating anomalies for the VM is randomly selected in the interval (0, 1], and the size of objects is randomly selected between 10 Kb and 1 Mb. As for the number of clients (emulated web browsers), we used 32 concurrent clients in the first scenario, and 64 in the second one. For both scenarios, we collected data related to all system features of the VMs, the response time measured by the clients and the predicted time to crash provided by the MLP.

We run the first experiment with 2 VMs for one week. In Figure 6, we show some results related to a time window extracted from the whole experiment for this first scenario. We report various measured features, namely number of active threads, free memory, used swap memory, and (total) CPU usage. Additionally, we report the response time measured by placing software probes in the Emulated Browsers, and the predicted RTTF for the VM that was activated upon each switching. By the plot, we can see that the accumulation of anomalies leads to a continuous decrease in free memory, with a subsequent increase in the usage of swap. Similarly, the number of active threads grows. The effects of these anomalies on the end users is shown by the response time, which grows as long as the effects of accumulated anomalies produce a performance degradation of the active VM. Further, the predicted RTTF for the active VM shows a decreasing trend while increasing the amount of accumulated anomalies. Vertical red lines, in Figure 6, represent rejuvenation points, namely time instants where the active VM is switched. Indeed, after the occurrence of each vertical red line, we can see that the amount of available resources (e.g., memory free, threads, swap used) of the new active VM shows an anomaly-free state, and the predicted RTTF immediately increases. Yet, the response time measured by end users (emulated browsers) drops down. Particularly, we note that PCAM ensured an upper bound value of the average response time equal to 4.5 seconds (as we set in the VM failure condition).

In Figure 7, we plot measurements related to a time window of the experiment for the second scenario, where three couples of VMs are managed by VMC. As mentioned, the three couples of VMs are subject to different combinations of anomalies. Also in Figure 7, vertical dashed red lines represent rejuvenation points. By the results, we can see that the injection of different kinds of anomalies lead the different VMs to reach the failure point at different wall-clock time instants. Nevertheless, VMC is able to manage independently these different couples without any loss of timeliness in the rejuvenation action.

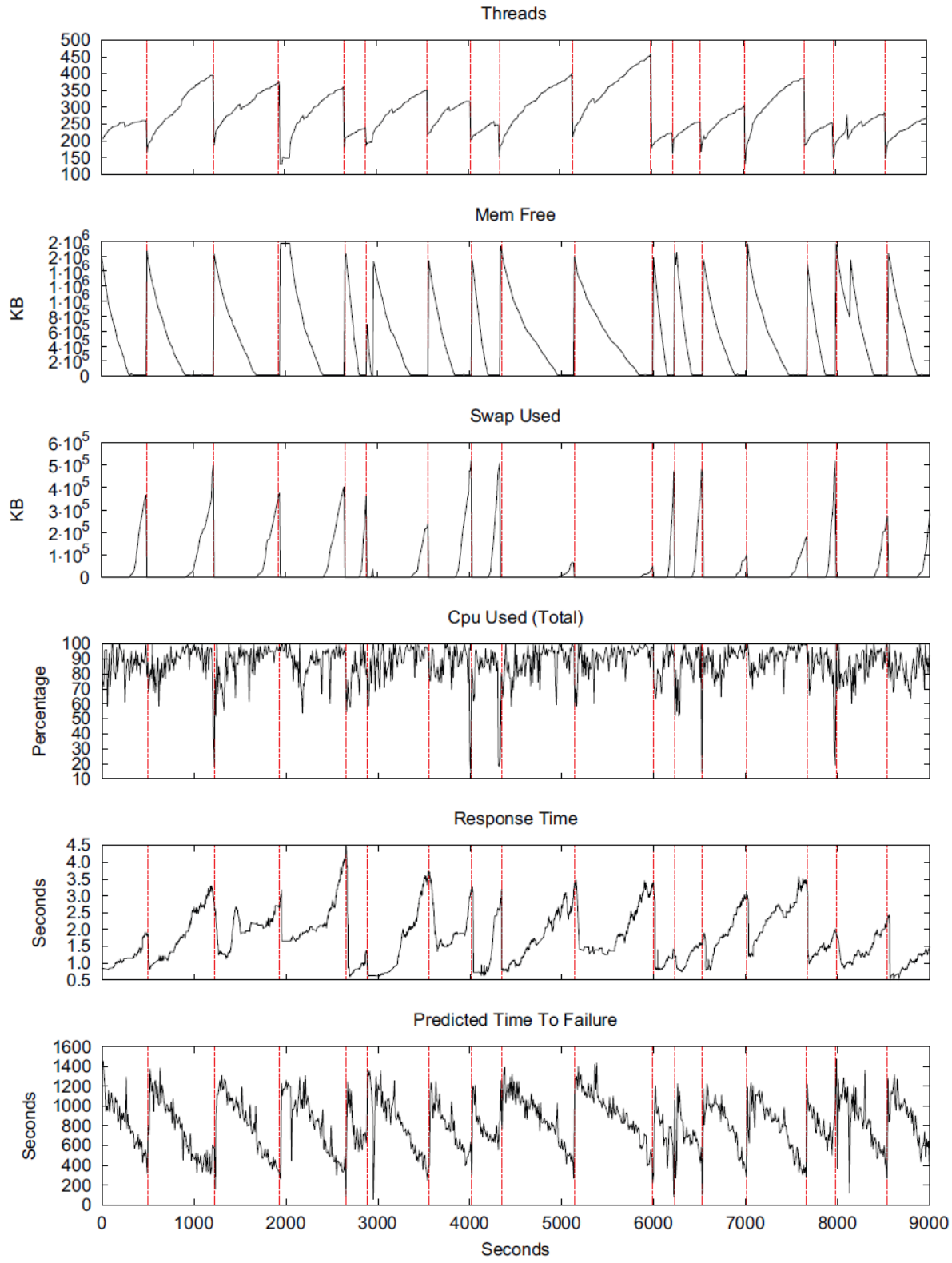


Figure 6 System features, response time and predicted RTTF for the scenario with 2 VMs and Lasso (with reduced parameters) as a predictor.

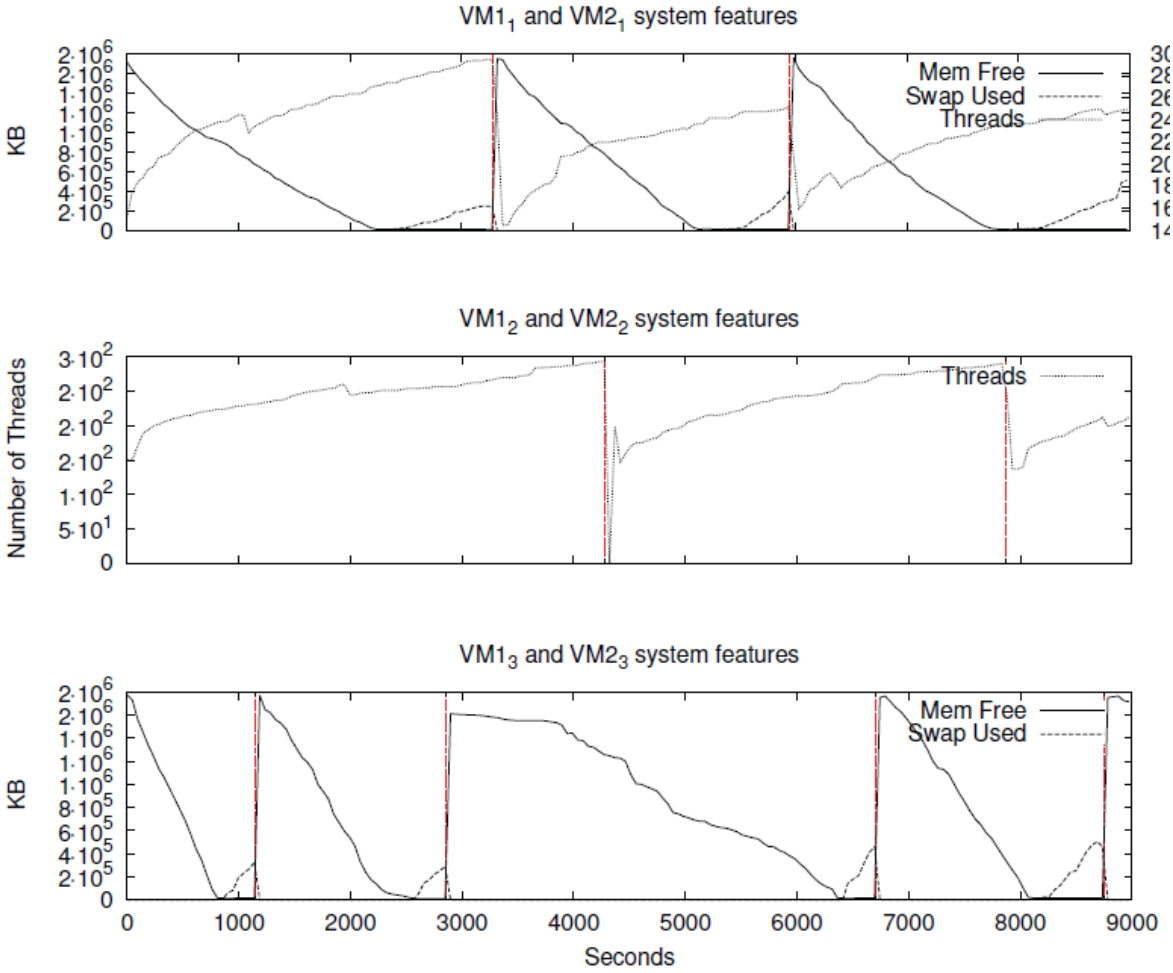


Figure 7. System Features for the scenario with 6 VMs and Lasso (with reduced parameters) as a predictor.

To complete the study, and to show the accuracy of PCAM, we measured the number of false negatives in [28]. As mentioned previously, F2PM allows the user to define criteria to determine whether the system under monitoring should be considered as failed or not. These criteria are used, during the datapoints collection of the training phase, to mark specific datapoints as system failure points. At run-time, when VMC detects that the predicted RTTF is lower than the threshold T, a rejuvenation action of the active VM takes place. Nevertheless, due to prediction errors, it could be possible that the system fails although the predicted RTTF is greater then T. This is exactly what we consider as a false negative. Namely, VMC treats the system as still working at an acceptable level, while it is actually not (i.e. it has already failed). In order to count the number of false negatives, we modified VMC in order to check if, based on values of features received by the active VM, the failure condition has been met.

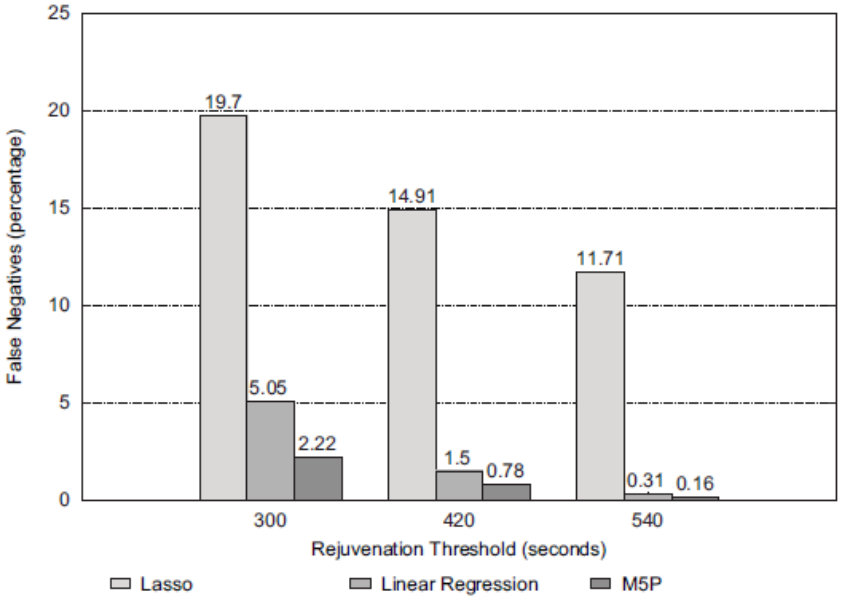


Figure 8 Percentage of false negatives using different thresholds in the case of memory leaks and unterminated threads.

In Figure 8, we report the percentage of false negatives we measured, for T equal to 300, 420 and 540 seconds, respectively, and related to prediction models generated using M5P, Lasso as a predictor and Linear Regression. Results show that the most-effective ML algorithm is M5P, providing a lower percentage of false negatives with respect to both Lasso and Linear Regression, confirming our previous prediction accuracy evaluation results in [30]. Nevertheless, in Figure 7, we can see that the higher the threshold T, the lower the number of false negatives. This is an expected result. In fact, if T is set to a too low value, the effect of even small prediction errors might bring the system to the failure point, preventing VMC from performing a rejuvenation action before the system failure. Based on the results, we can see that with different values of T the percentage of false negatives significantly changes. This demonstrates that the value of T can be used for reducing the number of false negatives (possibly for eliminating them at all) and, as a consequence, for increasing the overall availability of the system. The counterpart of high values of T consists of increasing the overall system overhead due to the increase of the VM switching frequency and rejuvenation actions. However, in all our experimental scenarios, we observed that, also with higher value of T (i.e. 540 seconds) the increase of response time due to the higher VM switching frequency was negligible with respect to the overall response time reduction achieved with PCAM with respect the case when VMs are switched after a failure occurs.

Results we discussed above show that PCAM can improve some system properties, including ones belonging to the category of self-* properties [28]:

- self-healing: this property is guaranteed by timely forcing the system to an anomaly-free state, relying on the self-rejuvenation capabilities of PCAM;
- self-optimizing: this property is guaranteed by ensuring a response time of the system below an acceptable threshold, even in the case of accumulation of anomalies;
- self-configuring: this property is guaranteed by allowing an automatic reconfiguration of the system, also when couples of VMs are added/removed to/from the managed pool of virtual resources.

As shown, these three properties enable PCAM to support seamless execution of client-server applications deployed on a cloud infrastructure.

The results obtained in [37] (Deliverable 3.3 – 2015-07-24.docx,) demonstrate that Inter Autonomic Cloud Manager (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). This allows for proactive scalability of Inter ACM on a larger scale. In fact, this configuration (exactly using Inter ACM) allows to proactively add entire regions of VMs (each region can be composed of any number of VMs) and promptly respond to the execution dynamics of the application.

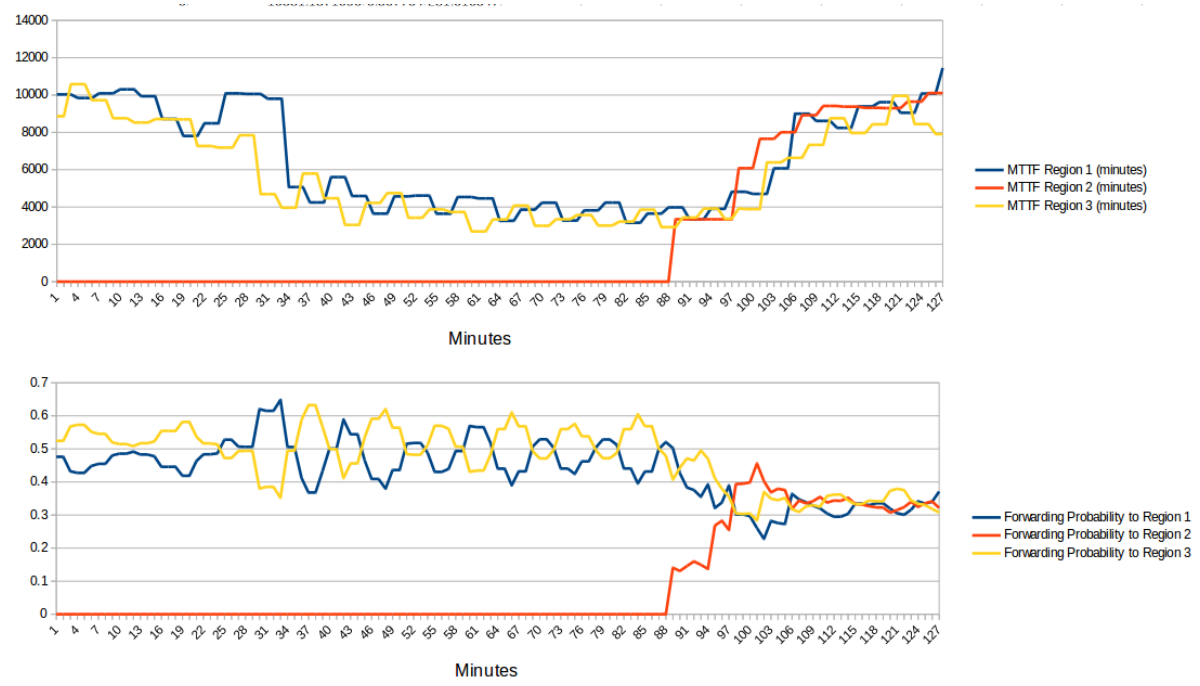


Figure 9 Request rate, MTTF, and Forward Probability, 3 regions, variable rate

In Figure 9 we report the results of controlled experiments using three regions: Munich (the leader, on a private infrastructure), Frankfurt (on a public Amazon infrastructure), Ireland (on a public Amazon infrastructure). Again, this controlled experiment is on a hybrid cloud.

The software configuration is the same as the previous experiments (as well, the injection rate of anomalies is the same), except for the fact that the load generated by TPC-W clients changes over time. In particular, until we reach minute 30 of the executed experiment, the TPC-W clients issue 300 requests per second. After that time, the load linearly grows towards 700 requests per second. This highest value is reached around minute 75.

Concerning the regions, we initially start with 2 regions (Munich and Frankfurt). The third region (Ireland) is added at time 90 minutes. By this controlled experiments, we are able to show, at the same time, how is the effect of increasing the workload when using two regions, and what is the effect on the decisions of Inter ACM (depending on the measured values) when adding a third region, with the highest workload.

By the result, we can see that when the workload starts to increase, Inter ACM promptly detects that the predicted MTTF decreases. Nevertheless, the distributed knowledge algorithm detects, as well, that this increased workload is affecting, at the same time, both

active regions. Therefore, Inter ACM correctly decides that the client requests should not be diverted to other regions. This shows that Inter ACM, relying on the prediction models which were built using ML algorithms correctly avoids making wrong decisions, in case of sudden bursts of load from remote clients.

Nevertheless, when we reach minute 90 and the third region is added, we can see that (similarly to the previous experiments) the MTTF of the three regions starts to increase, as long as the forward probability grow. As well, the forwarding probability for three regions reaches *equilibrium* at minute 127 i. e. 33% .

This is again an indication that Inter ACM is able to promptly cope with the variation of the conditions of the geographically-distributed deploy of the application, both when there is a variation on the conditions concerning the external clients connected to the region, and when there is an internal variation (due, e.g., to a scale up/scale down of the system i. e. , *elasticity property*).

3.3 Autonomic Task Allocation

Cloud computing enables the consolidation of diverse sets of workloads in shared infrastructures. The routing for the incoming jobs in the cloud has become a real challenge due to the heterogeneity in both workload and machine hardware and the changes of load conditions over time. Therefore, the cloud service provider must dispatch incoming tasks to servers with the assurance of the quality and reliability of the job execution required by end users while saving costs by improving resource usage efficiency.

Thus we focus primarily on designing and evaluating adaptive schemes that exploit on-line measurement and take decisions with low computational overhead for fast on-line decision making. With no priori knowledge of workload characteristics and the state of servers in terms of resource utilization and load conditions, our approach exploits on-line measurement related to the user required QoS and make judicious allocation decisions based on the knowledge learned from the observations, adapting to changes in workload and on-going performance of the cloud environment. This work can be relevant to Cloud service providers that use the SaaS, model where customers pay for the services, while the service provider sets up the VMs where the required software components are installed to deal with the service requests from the customer.

Our experimental evaluations are conducted on a multiple host test-bed, running with low to high loads that are achieved by varying the types and arrival rates of tasks. To conduct these experiments with greater ease, we have also designed and implemented a portable software module, the Task Allocation Platform (TAP), that is Linux based and easily installed on a Linux based machine. TAP will dynamically allocate user tasks to the available machines, with or without making use of on-line measurements of the resulting performance, and adapt to changes in workload and on-going performance of the Cloud environment, while optimising goals such as cloud provider' s profit while maintaining service level agreements (SLAs). TAP is flexible in that it can easily support distinct static or dynamic allocation schemes. It collects measurements on the test-bed, both to report on performance evaluation and also (for certain allocation algorithms) to exploit measurements for adaptive decisions. We will report on the performance observed with two well-known static allocation algorithms (Round-

Robin and a probabilistic " equal loading" scheme), and two dynamic algorithms (Sensible routing and Random Neural Network-based Reinforcement Learning).

3.3.1 Task allocation platform and test-bed

TAP carries out online monitoring and measurement constantly in order to keep track of the state of the Cloud system, including resource utilisation (CPU, memory, and I/O), system load, application-level QoS requirements, such as task response time and bandwidth, as well as energy consumption, and possibly also (in future versions of TAP) system security and economic cost. With knowledge learned from these observations, the system can employ the QoS driven task allocation algorithms that we have designed, to make online decisions to achieve the best possible QoS as specified by the tasks' owners, while adapting to conditions that vary over time.

Figure 10 shows TAP' s building blocks. The controller, which is the intellectual center of the system, accommodates the online task allocation algorithms, which work alongside the learning algorithm, with the potential to adaptively optimise the use of the Cloud infrastructure. TAP penetrates into the Cloud infrastructure by deploying measurement agents to conduct online observations that are relevant to the QoS requirements of end users, and send back the measurements to the controller. Three types of packets are used for communications between the components of the system: smart packets (SPs) for discovery and measurement, dumb packets (DPs) for carrying task requests or tasks, and acknowledgement packets (ACKs) that carry back the information that has been discovered by SPs. In this section, we present in detail the mechanisms that are implemented in the platform and the algorithms that are used.

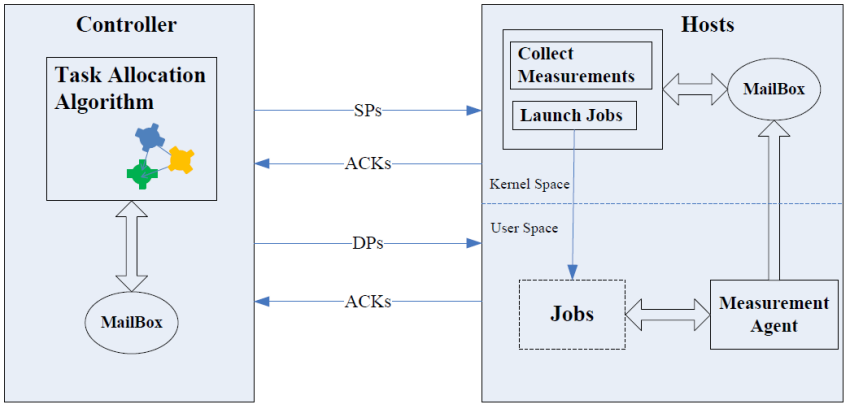


Figure 10 System Architecture showing the Task Allocation Platform (TAP), which is hosted by a specific computer that receives and dispatches jobs, and which interacts with the measurement system (at the right) which is installed on each host machine that executes jobs. The TAP communicates with each of the measurement systems at the hosts using SPs ("smart packets"), DPs ("dumb packets") and ACKs ("Acknowledgement Packets").

SPs are first sent at random to the various hosts in order to obtain some initial information and inform the measurement agents in the hosts to activate the requested measurement. The task allocation algorithm in TAP learns from the information carried back by the ACKs and makes adaptively optimised decisions, which are used to direct the subsequent SPs.

Thus, the SPs collect online measurements in an efficient manner and pay more attention to the part of the Cloud where better QoS can be offered, visiting the worse performing parts less frequently. The incoming tasks or task requests are encapsulated into the DPs, and exploit the decisions explored by SPs to select the host/Cloud sub-system that will to execute the task. Once a task (request) arrives at a host in the Cloud, its monitoring is started by the measurement agent which records the trace of the task execution until it is completed and deposits the records into a mailbox which is located in the kernel memory of the host. When an SP arrives at this host, it collects the measurements in the mailbox and generates an ACK which carries the measurements, and travels back to the controller where the measurement data is extracted and used for subsequent decisions of the task allocation algorithm. As soon as a task completes its execution, the agent also produces an ACK heading back to the controller with all the recorded data, such as the task arrival time at the Cloud, the time at which the task started running and the time at which the task execution completed. When the ACK of the DP reaches the controller, the task response time at the controller is estimated by taking the difference between the current arrival time at the node and the time at which the corresponding task arrives at the controller, which is used by the algorithm when the task response time is required to be minimised.

3.3.2 Experimental results

We conduct our experiments on a hardware test-bed as shown in Figure 11. The three hosts (with 2.8GHz, 2.4GHz, and 3.0GHz, respectively, dual-core CPU respectively) are used for task execution, while a dedicated host (2.8GHz dual-core CPU) accommodates the allocation algorithms.

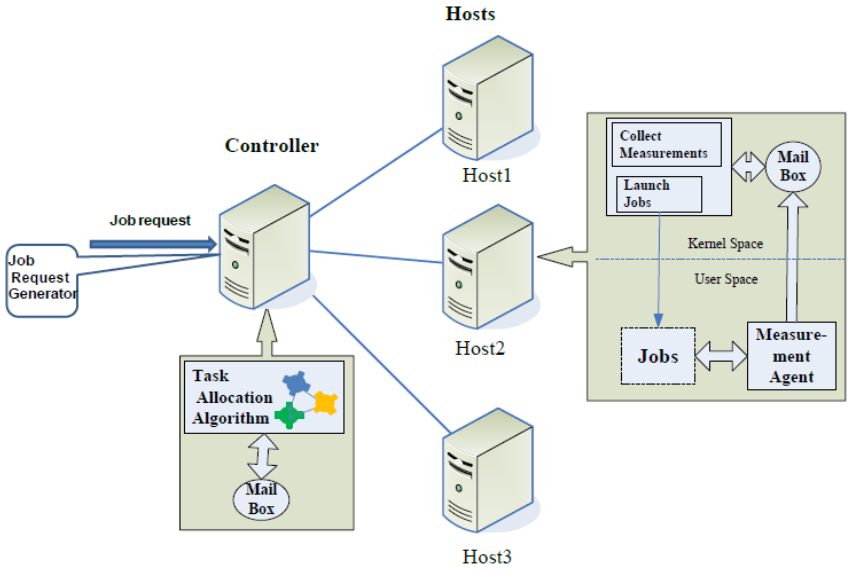


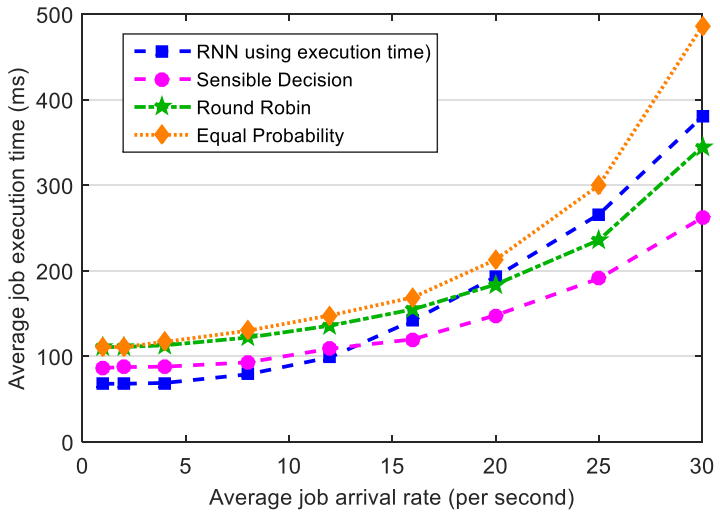
Figure 11 Schematic description of the task allocation test-bed. Jobs arrive at the controller machine for dispatching to the hosts. The TAP (Task Allocation Platform) software is installed at the controller and takes the dispatching decisions. TAP takes decisions based on data it gathers from each of the measurement systems (at the right) which are installed on each host machine that executes jobs.

The test-bed is in the small scale as we can easily vary the load of the system and evaluate the algorithms under low, medium and high (including saturation) load conditions. Actually, TAP is scalable because most SPs are sent to the hosts which are providing better

performance, avoiding ``flooding'' of SPs across the system.

The job used in the first set of experiments is a "prime number generator with an upper bound *B* on the prime number being generated" . The choice of *B* allowed us to vary the execution time resulting from both the CPU and memory requirements of the task. We installed the application in advance on the host, so that it actually provides the prime number generating services. TAP receives the job request which is packetized into an IP packet with the data fields {*task ID, QoS requirement, task size*} and makes the allocation decision which results in the arrival of a message from TAP which activates the application with specific value of *B* on the selected host. The QoS goal initially considered was the minimisation of the execution time on the host.

In order to study our proposed algorithms in a heterogeneous server environment, we introduce a *background load* on each host which stresses the CPU distinctly on the three host *i=1, 2, 3* with relative processing speeds of *2:4:1* for the CPU intensive services we provided. We compared the two dynamic approaches for task allocation to cloud hosts, the Sensible algorithm and the RNN-based approach, against round-robin scheduling and an equal distribution of load. All these experiments were repeated for a range of average task arrival rates equal to 1, 2, 4, 8, 12, 16, 20, 25, 30, 40 tasks/sec, in order to evaluate performance under load conditions that vary from light to heavy load, including saturation. Each experiment lasted 5 mins so as to achieve a stable state.



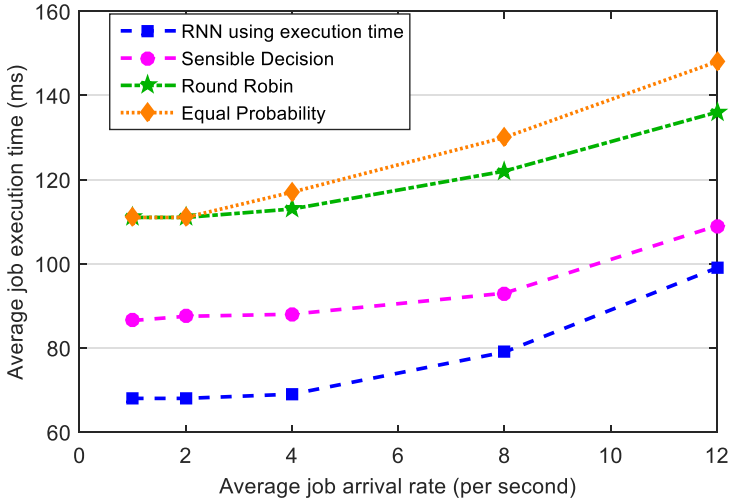


Figure 12. The average task execution time experienced under varied task arrival rates and different task allocation schemes in a cluster composed of hosts with non-uniform processing capacities.

The results of these experiments are summarised in Figure 12. We see that our proposed algorithm, the RNN and the sensible algorithm, benefits from their potential to detect the performance differences between the servers by leaning from the measurements, and direct tasks to the hosts which provide a better performance, while the two static allocation schemes, Round Robin and equal probability task allocation, perform worse as a whole. The RNN-based algorithm clearly outperforms the others, confirming that it is a fine-grained QoS-aware online task allocation algorithm.

In the second sets of experiments, we will study the effectiveness of TAP when there is greater diversity both in the types of tasks, and in the type of QoS criteria and the SLA that they request. To evaluate the allocation algorithms with two different classes of tasks, we used a web browsing workload generated with HTTPerf, which is a well-known web server performance tool.

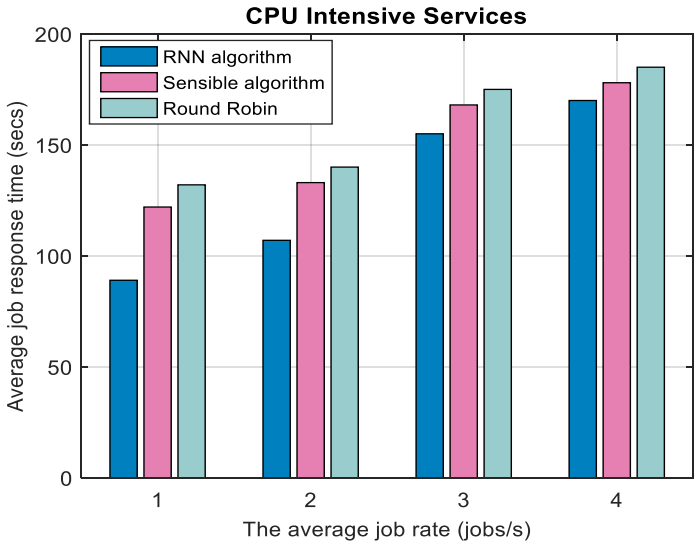
The first class corresponds to HTTP requests retrieve files from a web server, such as the Apache 2 HTTP server, whereby I/O bound workload is generated on the web server with very little CPU consumption, and the load on the I/O subsystem can be varied with the size of the retrieved files. In our TAP test-bed, the Apache server is deployed on each host in the cluster. HTTPerf generates HTTP requests at a rate that can be specified, while TAP receives the requests and dispatches them to the web servers.

On the other hand, the web services, which require a large amount of computation, mainly generate CPU load, are represented by CPU intensive tasks generated by the prime number generator.

In this case we compare the RNN based algorithms with the Sensible Algorithm, both using the Goal of minimising the response time. We also compare them to Round-Robin scheduling. The hosts themselves are stressed differently in terms of CPU and I/O in the cluster to provide different heterogeneous environments. The workload is generated so as to arrive at TAP following a Poisson process with different average rates of 1, 2, 3, 4 tasks/sec.

The different performance level offered by the hosts is implemented by introducing a background load which stresses I/O differently on each host, resulting in relative processing speeds of 6 : 2 : 1 for Hosts 1, 2, 3 with regard to I/O bound services, while a background load which stresses CPU distinctly on each host, resulting in the relative processing speed of

2 : 3 : 6 (corresponding to Hosts 1, 2, 3) is used for the CPU bound case.



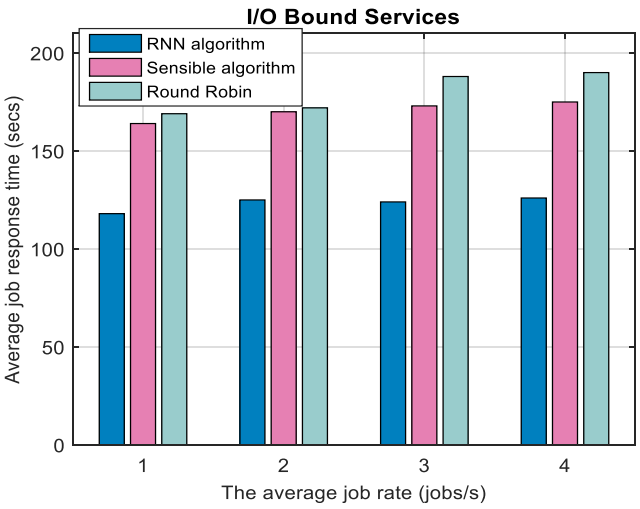


Figure 13 Average response time experienced by the CPU intensive web services and I/O bound web services in a heterogeneous cluster. We compare RoundRobin with RNN based Reinforcement Learning and the Sensible Algorithm.

The results in Figure 13 show that the RNN based algorithm performs better; the reason may be that it is able to detect the best possible host for the task based on its QoS requirement by effective learning from its historical performance experience and make more accurate decisions (compared with Sensible) which dispatch I/O bound tasks to the hosts where I/O is less stressed and dispatch CPU intensive tasks to the hosts which provide better CPU capacity. During the experiments, we reduced the background load in terms of both CPU and I/O stress on the Host 2 to the lowest level as compared with the Hosts 1, 3. It was found that the RNN based algorithm was able to detect the changes and dispatch the majority of subsequent tasks of both types to Host2, which also shows the host where CPU is heavily stressed to still provide good performance to I/O bound tasks. More generally, we also observe that Round-Robin performs worse than the two other algorithms.

3.4 Routing Overlay

SMART[4] is the open-source routing overlay developed by CNRS in PANACEA. SMART was designed for quickly detecting and recovering from path outages, as well as for discovering the optimal overlay routes for service-specific routing metrics. Last but not least, SMART was designed to control the path of data of an application through the overlay without the application even being aware that its data flows are routed over the overlay, so that it can work with off-the-shelf applications.

The design objectives, the architecture and the implementation of SMART have been fully described in deliverable D2.3. As discussed in this report, one of our main design goals was to build a routing overlay that can be widely deployed over a sizable population of routers. An all-pair probing approach implies a costly O(n2) probing overhead as the number of participating nodes n increases, and evidence indicates that it is able to scale to approximately 50 overlay nodes. As discussed in deliverable D2.3 [32], the problem of learning the optimal route between two given nodes can be cast as a multi-armed bandit problem, for which many efficient algorithms have been proposed. At the time deliverable D2.3 was written, we were using the EXP3 algorithm proposed by Auer et al. [33]. In collaboration with Imperial College London, we have developed a new approach inspired from Cognitive Packet Network (CPN) [36]. This new approach uses Reinforcement Learning to adjust the parameters of a Random Neural Network (RNN), acting as an adaptive critic.

In this section, we present the empirical results obtained with this new approach using data collected over the Internet. These results demonstrate that it is possible to significantly improve over native IP routing with a modest monitoring effort. Empirical validation with live experiments using the emulation/simulation environment developed by QoS Design will be presented in Section 3.5. We also present experimental results obtained with live experiments over the Internet in Section 3.6.2.

3.4.1 Latency minimization

We now describe the results that were obtained with the proposed algorithm during an Internet-scale experiment done in spring 2014, where we used 19 nodes of the NLNog ring[5] shown in Figure 14. Note that these overlay nodes are interconnected by literally hundreds of Internet nodes which are unknown to us or the overlay, and which support the overlay itself.

[4] **Self-MAnaging Routing overlay**

[5] **The NLNog ring is a network of 293 nodes scattered over 46 countries (see https://ring.nlnog.net).**

Figure 14: Geographical location of the 20 nodes selected in the NLNog ring.

We first measured the latency between all pairs of nodes every two minutes, communicating through the Internet, for a period of one week using the ICMP-based ping utility. Furthermore, when five consecutive packets were lost between a specific pair of nodes, we considered that the particular source was disconnected from that destination. We thus collected some 1.7×10^6 measurement data over the week, from which we can compute the weighted adjacency matrix of the overlay graph at each measurement epoch, and hence compare the round trip delay of the IP route with that of the optimal overlay route.

The analysis of collected data confirmed the deficiencies of Internet routing observed in previous studies. There was an outage of the IP route at least once in the week for 65% of OD pairs, and 21% of these outages lasted more than 4 minutes (and more than 14 minutes for 11% of them). This analysis also revealed that, as shown in Figure 15, in 50% of the cases it is possible to improve over the latency of the IP route by adding one or more intermediate overlay nodes to the path. Surprisingly enough, in 30% of the cases, the minimum latency path is a path with only one intermediate overlay node, that is, a two-hop path. This shows that a limited deviation from IP actually produces much better QoS than IP itself. Interestingly, even though in 20% of the cases the optimal path is a 3 or 4 overlay-hop path, there is on the average no significant gain (only 5.4%) in considering overlay paths of more than two hops. This suggests that we can restrict ourselves to paths with at most one intermediate overlay node (this is true only on average, since, for instance, the RTT between Narita/Paris can be more than halved if we use two intermediate nodes instead of at most one).

As we will now show, SMART allows a significant decrease in round-trip delay, with a very modest monitoring and computational effort. We consider a fixed OD pair, the other overlay nodes serving just as relays. We restrict ourselves to the N=18 overlay paths of at most two hops and assume that the routing algorithm probes K=3 paths (including the direct IP route) at each time slot, that is, every two minutes. The algorithm therefore measures 5 links per measurement and decision round (to be compared to the 342 links monitored in the all-pairs probing approach). Our results are summarized in Table 2, which shows the average relative gap to the minimum latency that can be achieved with two-hop routing (the averaging is over time and over the 342 OD pairs).

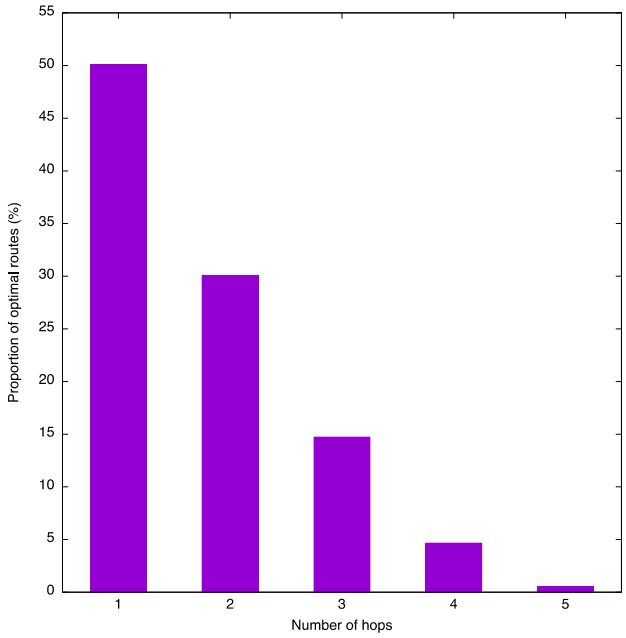


Figure 15: Percentage of instances when the optimal path includes 1, 2, 3 or 4 hops.

These results demonstrate that SMART uses the optimal two-hop route in 84% of the cases, and that it provides near-optimal latencies, with a clear improvement over native IP routing. However, these average values do not truly measure the gains obtained in the pathological routing situations we seek to improve. In Table 3, we present the results for some OD pairs, for which our system allows a huge decrease in round-trip delay.

	IP route	SMART
Non-optimal instants (%)	50.1	16.2
Gap to optimal latency (%)	11.1	4.2

Table 2: Performance of native IP and SMART routings on the whole set of NLNOG traces compared to optimal two-hop routing.

	IP route	SMART
Melbourne/Gibraltar	390	274
Narita/Santiago	407	254
Moscow/Dublin	180	82
Honk Kong/Calgary	267	132
Singapore/Paris	322	155
Tokyo/Haifa	323	181

Table 3: Average RTT (ms) for some pathological OD pairs.

On the other hand, Figure 16.a shows the RTT between Narita (Japan) and Santiago (Chile) over 5 successive days. The RTT of the direct IP route is about 400 ms, whereas the RTT of the minimum latency path is about 250 ms. As can be seen, SMART learns quickly which is the minimum latency path and tracks this path until the end of the 5 days. Figure 16.b shows the same results over the first 3 hours. We notice that it takes only 12 measurement epochs (24 minutes) for SMART to learn the optimal route.

3.4.2 Throughput maximization

We now describe the results obtained in an experiment involving 9 AWS (Amazon Web Services) data centres located as shown in Figure 17. In summer 2015, we measured the available throughput between all pairs of data centres every five minutes, communicating through the Internet, for a period of four days. We thus collected some 8.3 x104 measurement data over the 4 days period. Assuming that the available throughput over a path is the minimum of the throughputs of its constituent links, the analysis of these data revealed that the IP route is the maximum throughput route only in 23% of the cases, and that most of the time, the maximum throughput overlay route passes through 1 or 2 intermediate nodes (see Figure 18).

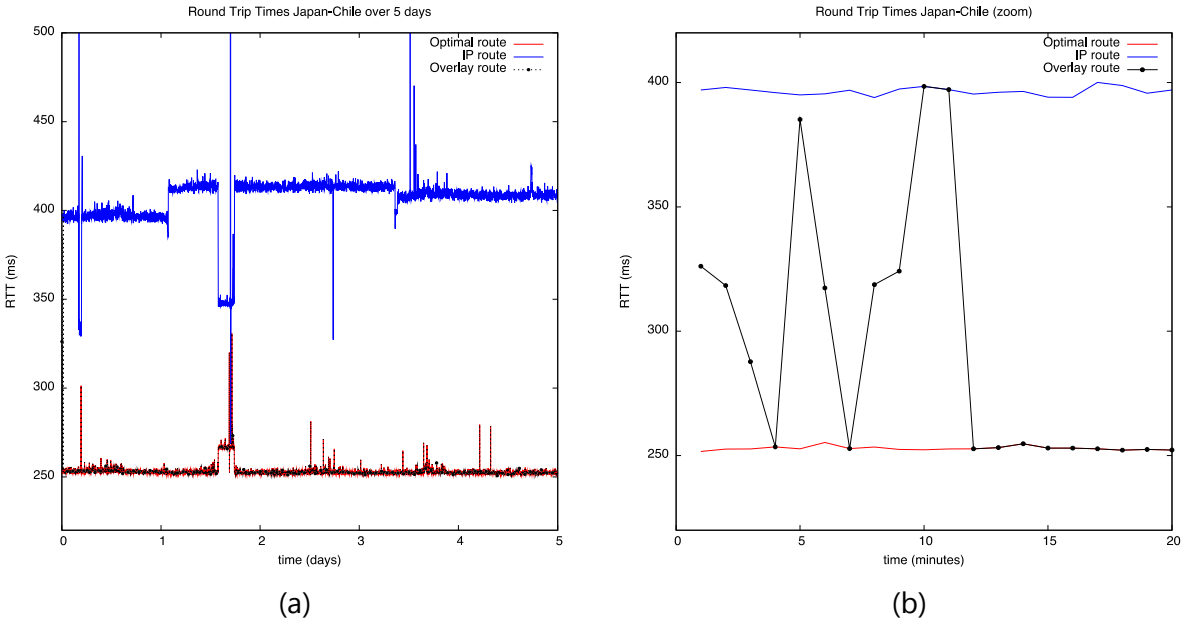


Figure 16: (a) RTT (ms) measured for the Narita(Japan)-Santiago(Chile) connection in an experiment lasting 5 consecutive days, and (b) RTT over the first 3 hours .

Figure 17: Locations of the Amazon EC2 data centres used in our experiment.

As for delay minimization, we consider only the N=8 overlay paths of at most two hops and monitor K=3 paths at each measurement epoch. The monitoring effort is therefore limited to 5 links, whereas the all-pair probing measures the throughput of 72 links at each measurement epoch. We present in Table 4 the results obtained for some pathological OD pairs, for which the available throughput is at least doubled.

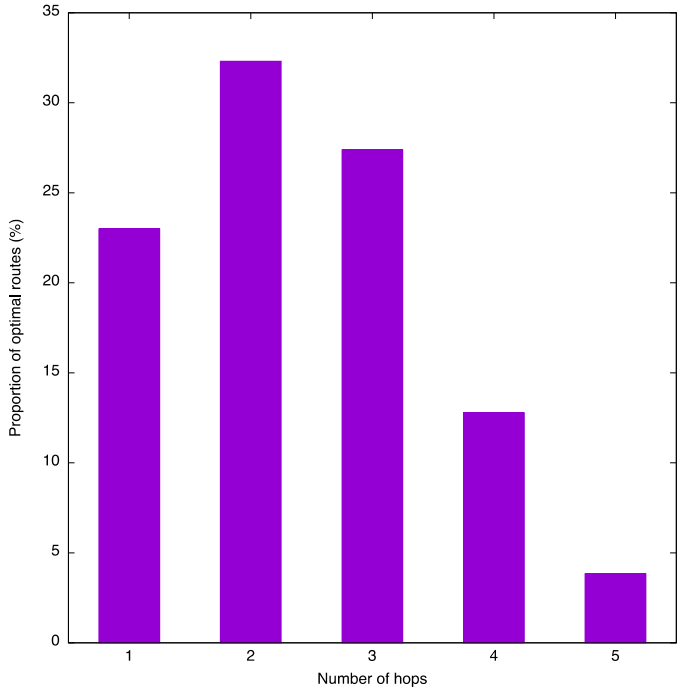


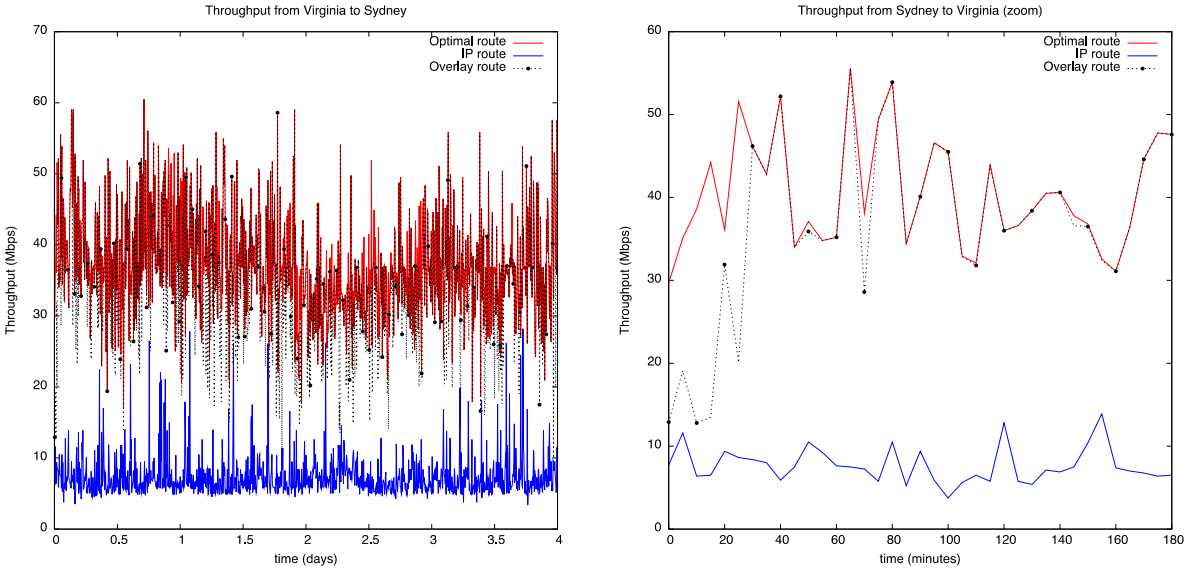
Figure 18: Percentage of instances when the optimal path includes 1, 2, 3, 4 or 5 hops.

	IP route	SMART
Dublin/Sydney	11.5	37.5
Singapore/Sao Paulo	12.8	42.0

Sydney/Virginia	8.5	52.3
Virginia/Singapore	7.4	33.8
Virginia/Sydney	6.9	35.0
Virginia/Tokyo	10.3	39.7

Table 4: Average throughput (Mbps) for some pathological OD pairs.

On the other hand, Figure 19.a shows the available throughput between Sydney (Australia) and Virginia (USA) over the 4 successive days. The average throughput of the direct IP route is 8.5 Mbps, whereas the average throughput of the optimal path is 55.3 Mbps. Figure 19.b shows a zoom over the first 3 hours.



(a) (b)

Figure 19: (a) Throughput (Mbps) measured from Sydney to Virginia over 4 consecutive days, and (b) over the first 3 hours.

3.5 Offline Design and Planning of Cloud-hosted Web Services

3.5.1 Introduction

The overlay simulation module of PANACEA project operates closely with the overlay network system as it enables the design and optimization of large-scale overlay networks. The simulator enables accurate modelling of the underlying physical network, including the routers, links, data centres, and network protocols.

The architecture the different modules have been fully described in deliverable D4.2[1]. This module I used hereafter to assess the self-properties for the SMART system. It can also be

used for any distributed application that has to be tested in near-real life conditions before a production deployment.

3.5.2 Simulation approach

Abilene Network was a high-performance backbone network created by the Internet2 community in the late 1990s. In 2007 the Abilene Network was retired and the upgraded network became known as the "Internet2 Network".[1], [1]

Figure 20: Abilene network

What is interesting with this network, from a simulation perspective is that along the topology we have not only the network configuration but also real traffics recorded during long periods (say months).

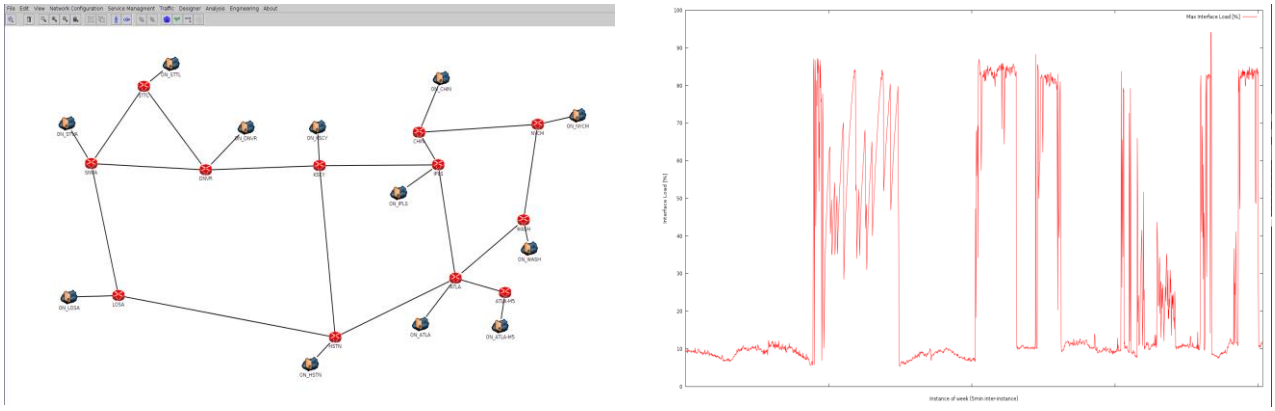


Figure 21: Abilene network in NEST IP/MPLS and it's maximum interface load

We analysed the historical data for Abilene network for a period spanning over 6 months. We have run several simulations using different periods until we found a single period (a week) with significant variations. All the following results are extracted from the simulations we did on that period.

From the analysed data, we exhibited three sub-scenarios. The first one is an overlay path for

which the SMART path is always performing better than the IP path. The second one is an overlay with a SMART path performing better than the IP route during a perturbation phase. The third one is where introduce a link failure and compare the IP and SMART paths during the failure.

3.5.2.1 The first case

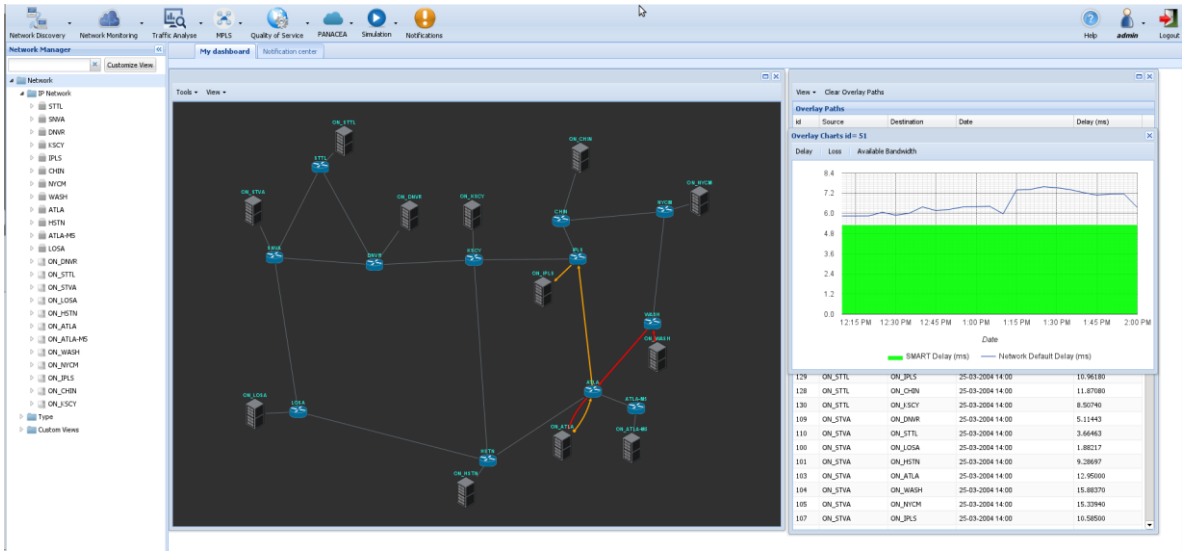


Figure 22: IP vs SMART Path case 1

The IP path used from Washington to Indianapolis goes through New York and Chicago and has a performance (delay) worse than the route proposed by SMART depicted in Figure 22.

The chart on Figure 23 shows in blue the evolution of the delay for the IP path and in green the evolution of the delay for the SMART Path.

As can be identified in the chart below, it is obvious that the SMART path is always better than the IP path.

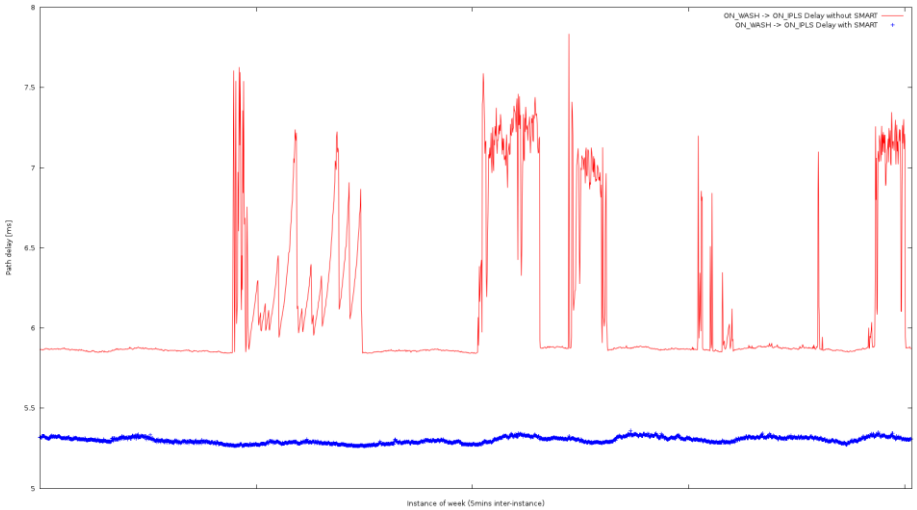


Figure 23: IP vs SMART Path delay comparison (case 1)

3.5.2.2 The second case

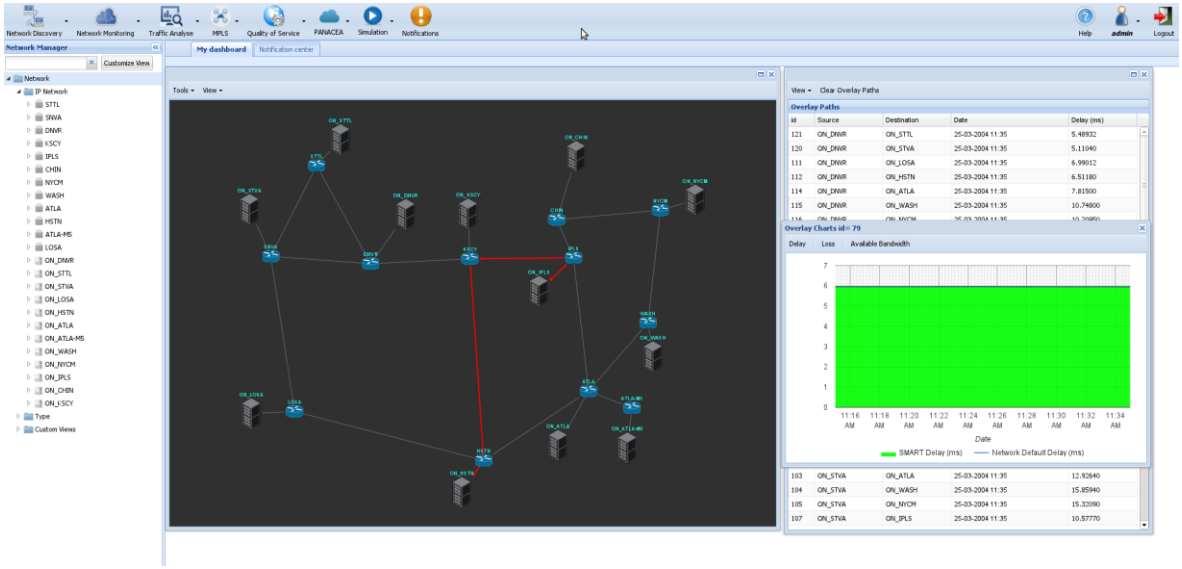


Figure 24: IP vs SMART Path case 2

For this particular case, IP and SMART paths are the same most of the time. This is not the case during the perturbations phase. During several periods, congestion impacts the IP path leading to severely degraded performances. At the same time, SMART manages to find better routes.

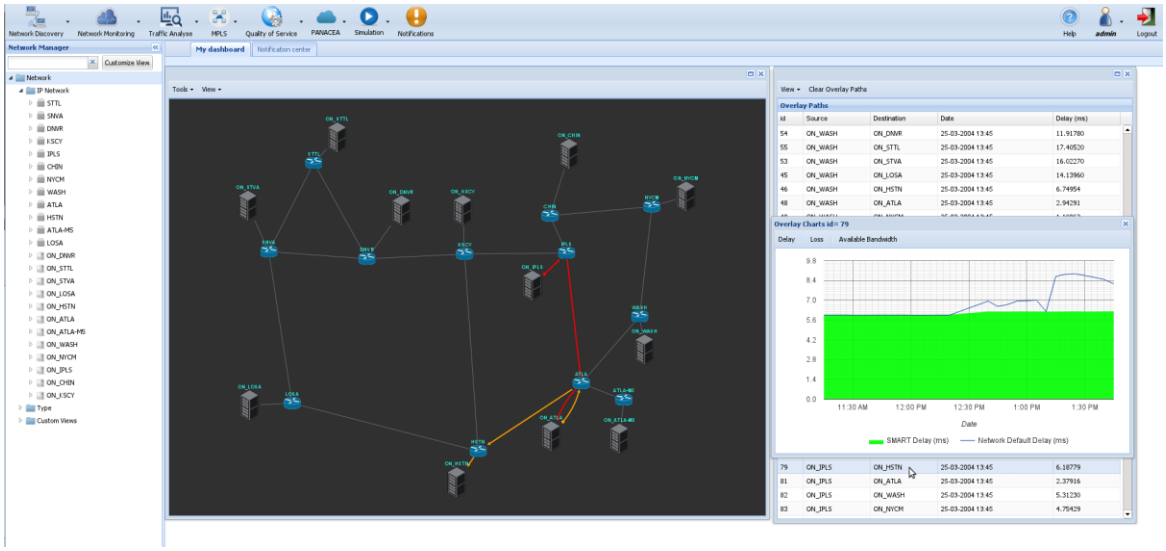


Figure 25: Example of SMART Path outperforming the IP path

The performance comparison is illustrated in the chart of Figure 26 here after.

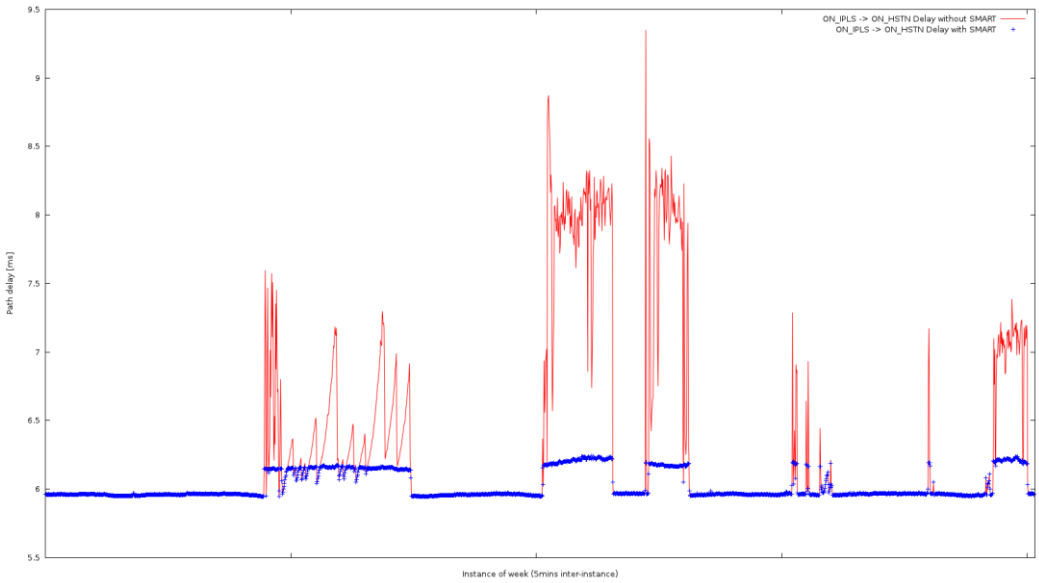


Figure 26: IP vs SMART Path delay comparison (case 2)

3.5.2.3 The third case

In the nominal case the path from Los Angeles to Houston is depicted below

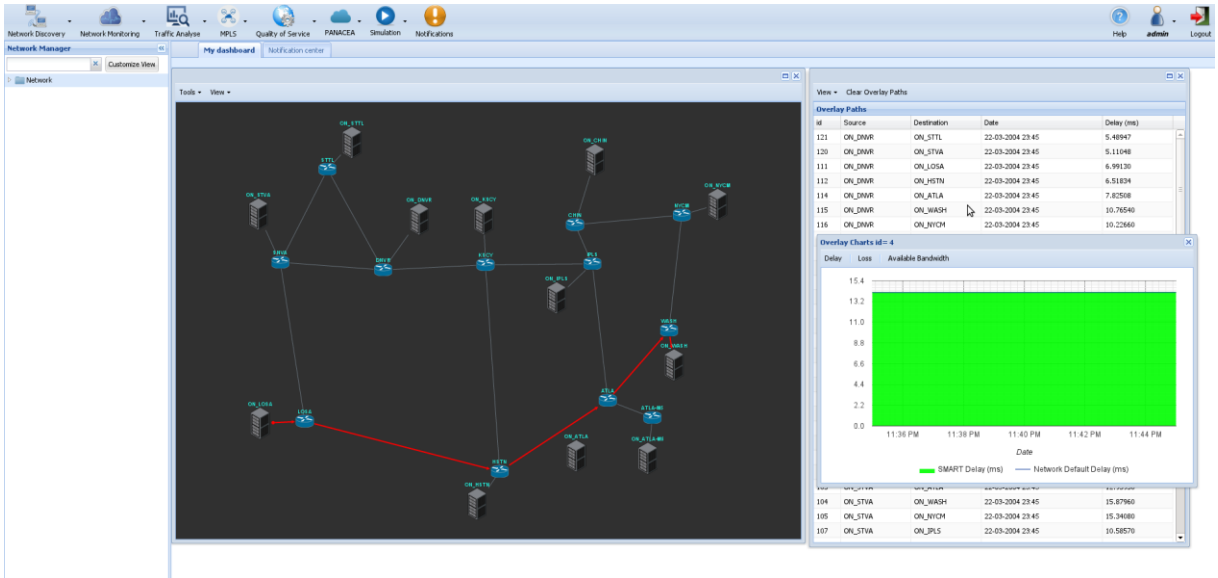


Figure 27: IP path before failure

Then we introduced a failure on the link Los Angeles->Houston. First, the IP path is recovered using the alternate path depicted hereafter:

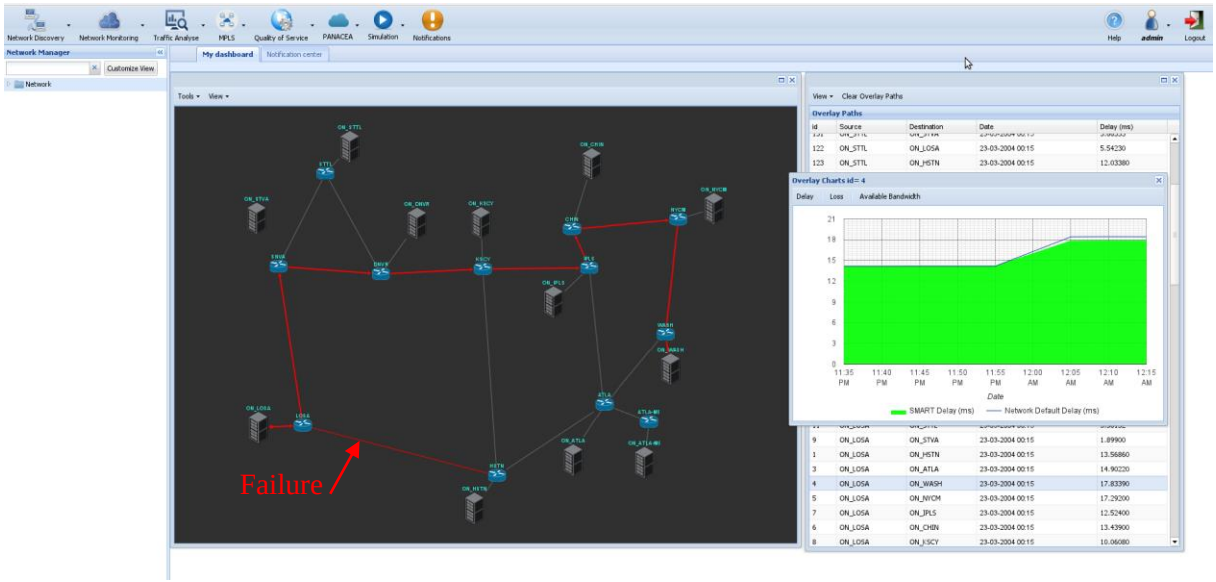


Figure 28: IP path after the failure

Then SMART computed a better route by using the path depicted bellow

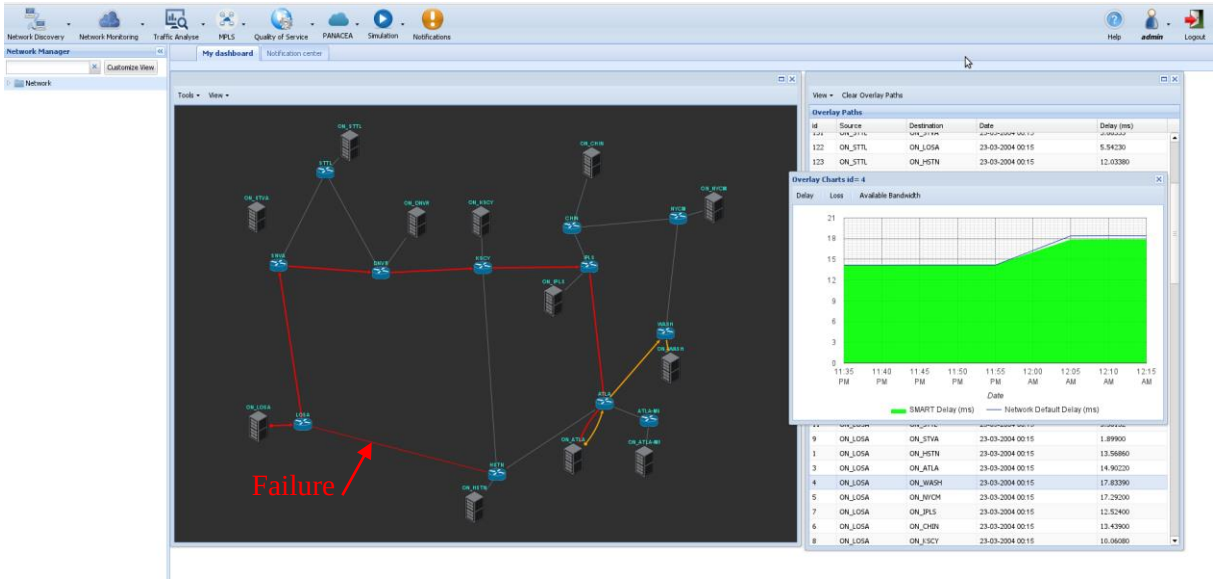


Figure 29: SMART path after the failure

The difference here, is that SMART is using proxy node Atlanta rather than passing through Chicago and New York.

A summary of performance comparison is shown in the Figure 30 below.

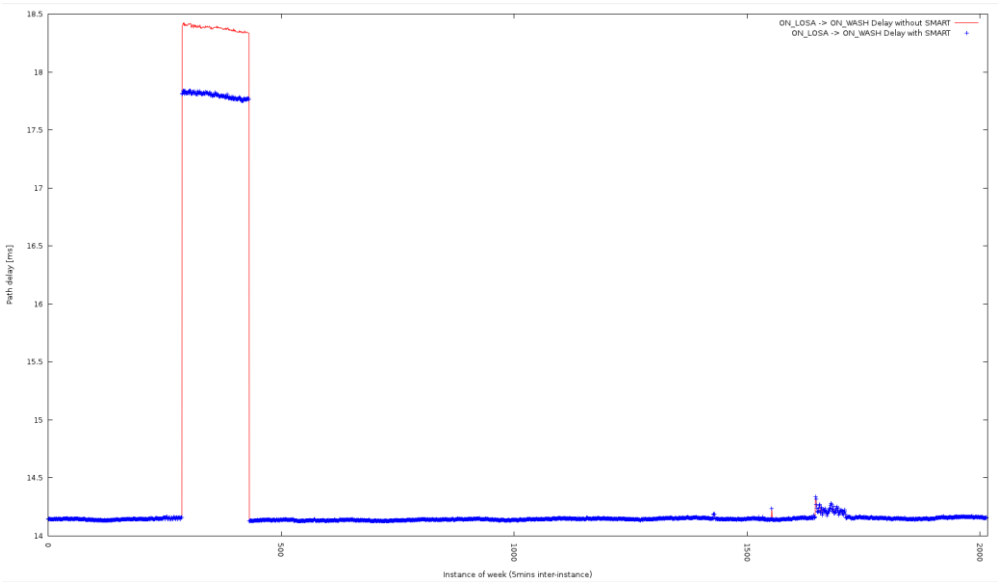


Figure 30: IP vs SMART path delay comparison (case 3)

3.5.3 Hybrid approach

3.5.3.1 The setup

For this approach, we used an Intel server with two Xeon E5-2699v3 processors (18 Core each) and 64GB of RAM. We have deployed 12 VMs as follows,

- One for the Core emulator,
- 9 for the proxies,
- One for the Client and one for the application server.

NEST and the simulation kernel were deployed on the Intel server itself.

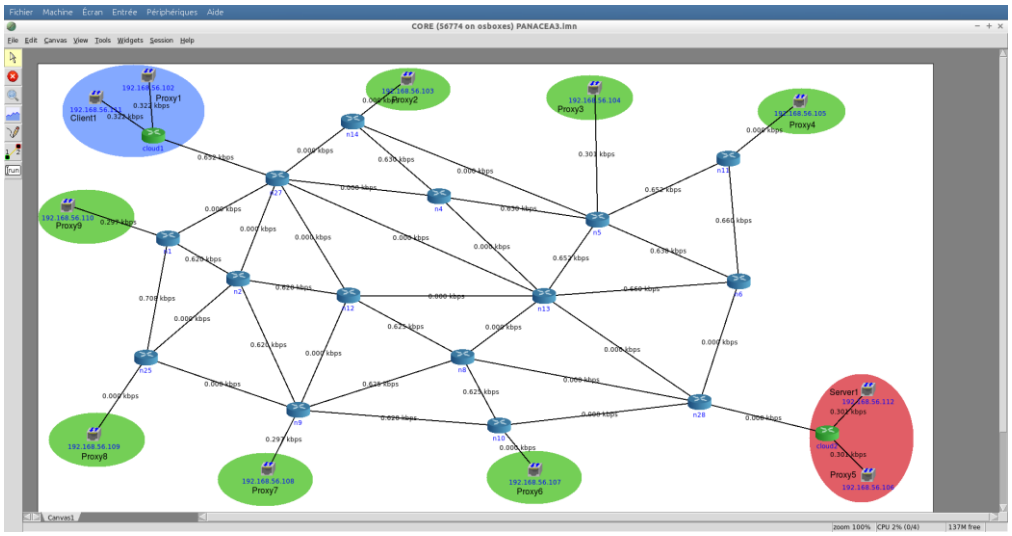


Figure 31: Emulation scenario with nine proxies

As a reminder the emulation process is driven by NEST (Simulation). It means that at each step of the emulation, the simulation kernel is generating stochastic background traffic. The kernel is then computing relevant metrics for the overlay paths (delay, bandwidth,⋯). These metrics are set on the links inside the Core emulation.

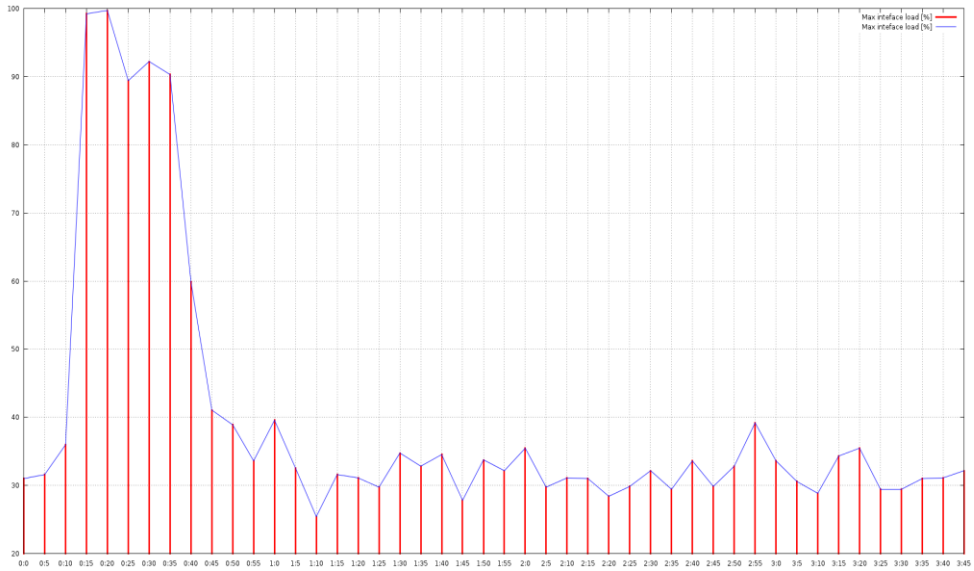


Figure 32: The profile used for background traffic

SMART uses these metrics to potentially discover new paths offering better performances. If it's the case the kernel updates the used paths for the overlay networks.

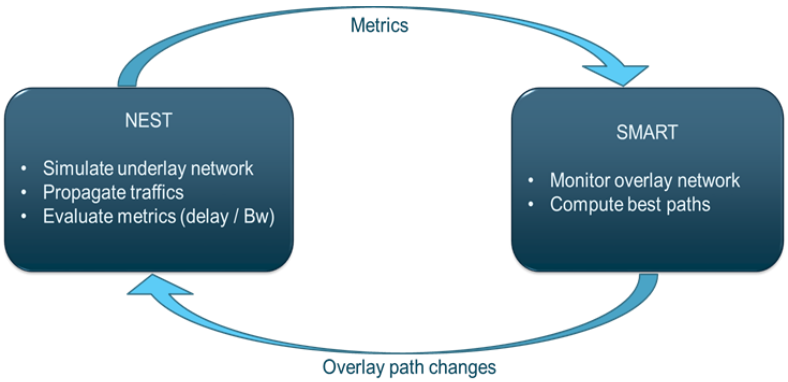


Figure 33: Emulation driven by NEST

We conducted two types of experimentations. First considering the delay metric and then considering the bandwidth metric. For each experimentation, we performed the tests with and without activating SMART.

3.5.3.2 The delay metric

The experimentation illustrate clients injecting http requests toward a web server. The idea is

to evaluate the impact of the network on the response-time of each request. After activating SMART, we check whether it can alleviate this impact or not.

We ran several simulations that span over a 24H period. We extracted a window of 3 hours (cf Figure 32) containing the most significant results. During this window a huge perturbation that lasts for one hour occurs.

We compared the response times of http requests sent during this period with and without SMART. The results are compiled in the following chart.

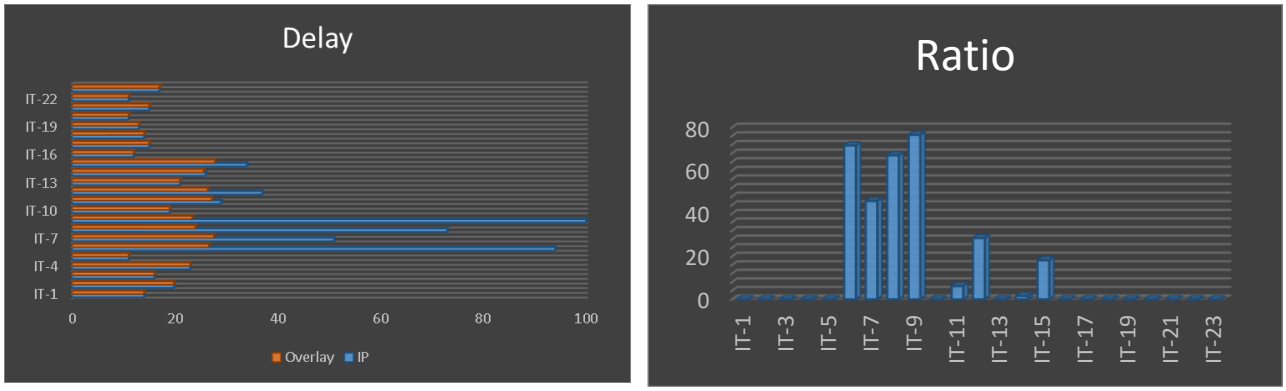


Figure 34 response time comparison

What is significant in this case is the improvement ratio. While the IP path can deliver good performance in general, sometimes SMART can achieve ~70% improvement during perturbations and maintain good performances for the end user.

We also analyzed the paths that were used by SMART to achieve those improvements. It appears that proxies 3, 7 and 8 were alternately used depending on network conditions.

In the particular case of a checkout phase, we have injected requests toward a special web page hosted by the server. This web page performs a computation that lasts 2.7s. According to the checkout time limit of 3s, this leaves us with 300ms for the network delay.

Figure 35: E-commerce checkout phase dilema

In this demonstration, we measured the delay before the perturbation occurs and we

obtained 2705.3 ms.

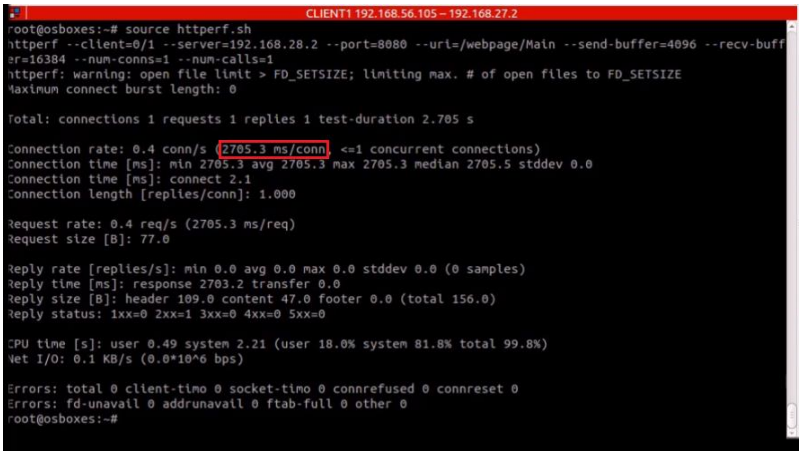


Figure 36: httperf measurement before perturbation

During the perturbation the time response exceeded the 3s threshold (3025.7 ms). This would have lead to the clients leaving the page without finalizing the transaction.

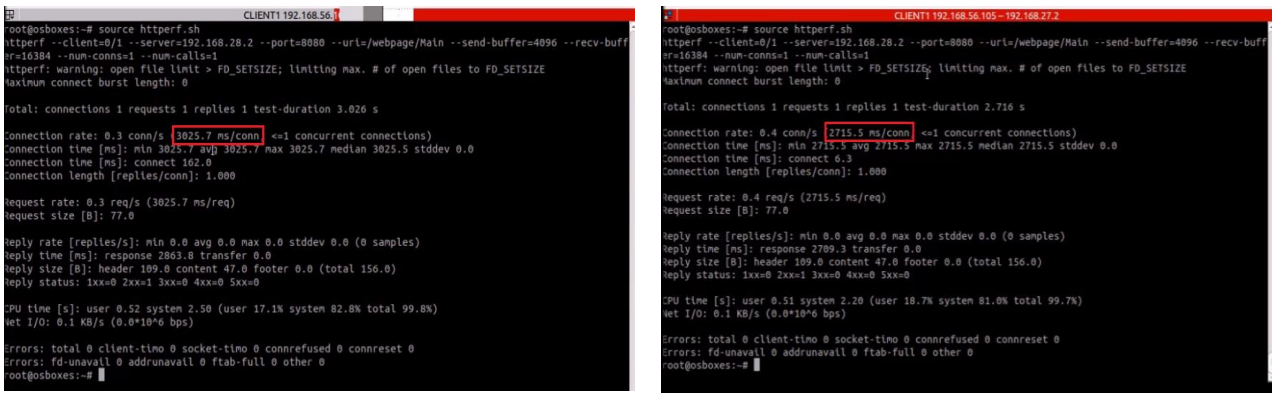


Figure 37: httperf measurement during perturbation

After few ms, SMART detected a better path (going through proxy 3) that achieves a response time below the 3s limit (2715.5 ms). To achieve this improvement, SMART used an alternate path passing through proxy 3.

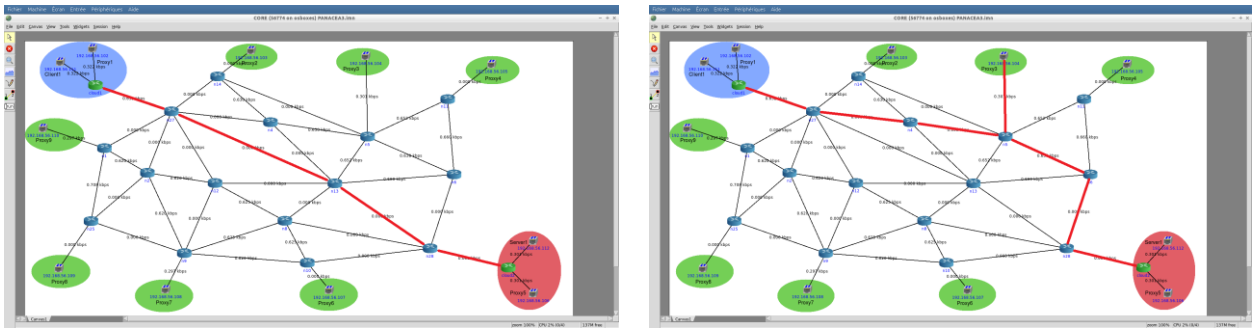


Figure 38: IP vs SMART path during perturbation

3.5.3.3 The bandwidth metric

We also conducted a test campaign using the appropriate version of SMART that optimizes

the bandwidth metric. This version of SMART uses a bandwidth related probe that measures the minimal residual bandwidth along the overlay paths. The goal is to exhibit the paths with the best residual bandwidth for user applications.

During this experimentation, we used the same scenario and the same environment. Rather than injecting http requests, we injected new requests that have to download 10MB files.

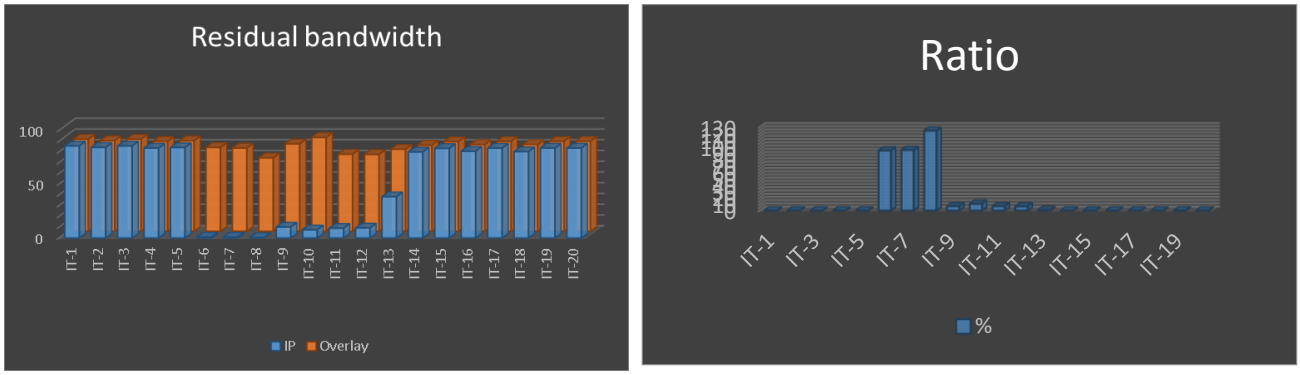


Figure 39: Comparison of residual bandwidth over the IP and SMART Path

It appears that while the IP path suffers congestion SMART was able to find better paths using intermediate proxies. As for the delay metric, what is significant is the achieved improvement ratio.

From a user perspective, on the iteration "IT8" where an improvement of 124% is achieved using SMART. We measured the time needed for a user to download a 10MB file.

The results that we obtained are:

- On the IP path : 1m52s (737.4 Kb/s)
- Over SMART Path : 33s (2.42Mb/s)

This means a 70.5% improvement on the user side.

3.6 Integrated Validation

In this section we validate the integration of the individual PANACEA innovations.

3.6.1 Task allocation across multiple clouds

In Section 3.3, we have focused on the autonomic task allocation within a single cloud. However, in this section, we present an extension of the TAP designed for workload distribution across multiple clouds over wide area networks.

As shown in Figure 40, a user's web requests are routed to the "local" – perhaps the geographically closest cloud -- if the cloud has enough capacity. The dispatcher at the local cloud receives the incoming workload and adaptively selects the best possible server based on the user-defined QoS (e.g. the response time).

If the workload in the local cloud increases, the dispatcher could decide whether to forward the subsequent requests to the remote clouds in order to balance the load and offer better

QoS to all the requests it receives. The decision would rely on a variety of considerations, such as security, cost, QoS and energy consumption.

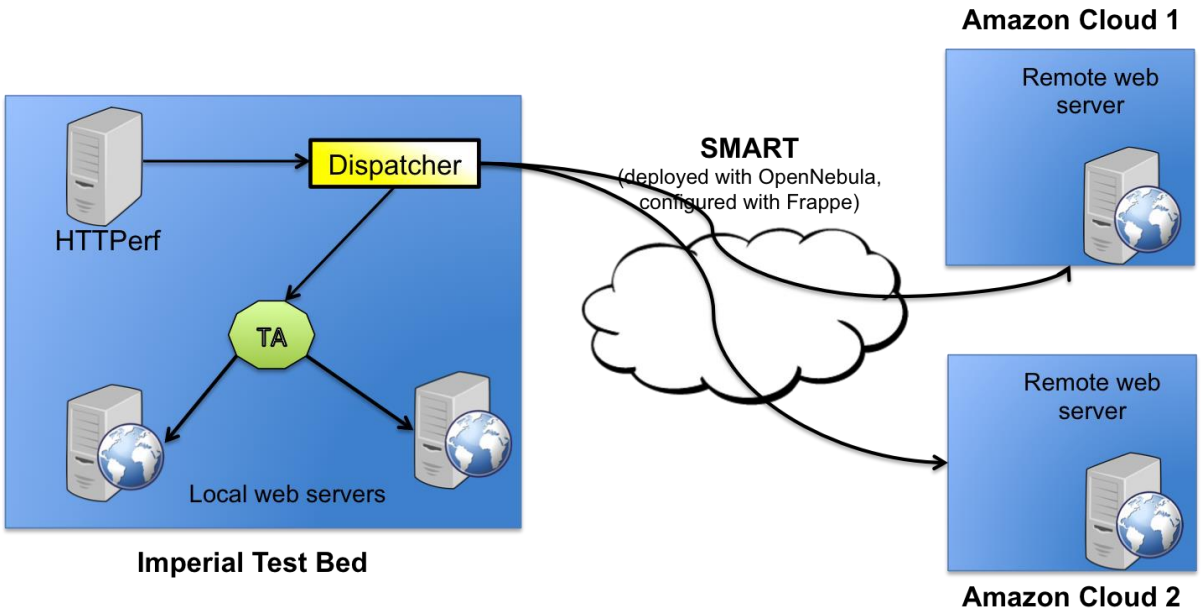


Figure 40 The architecture of the experimental system in multiple cloud scenario.

We currently only concern ourselves with the QoS that tasks receive, in particular the response time observed by tasks. Obviously this response time will be determined by the network delay incurred by the access to local or remote clouds, which includes the network delay to process the request, the network delay to forward the task with its possible code and data, plus the time it takes to return the results to the sender after execution, plus the waiting time and service time inside the clouds. That is to say the dispatcher would select a remote cloud to share the workload of the local cloud by considering the response time which consists of the data transfer latency which depends on the traffic conditions on the connections to the remote clouds and the proximity of the remote cloud to the local cloud as well as the response time within the cloud which is related to the viability or health state of the cloud. It should be noted that the data transfer delay for the local cloud, though not zero, may be negligible. We assume that each cloud will report its health state regularly, including the response time within the cloud. So we only need to measure the network delay on the connections to all the external clouds in order to predict the data transfer latency. This of course is easier said than done because it depends on the nature of the transfers and on the other traffic in the connections.

To simplify matters, we measure the network delay constantly via pinging, whereby we can obtain the round-trip delay (denoted by T_n^j for the j-th cloud) and the packet loss (denoted by L^j) on each connection. The packet loss also needs to be considered because HTTP requests utilize TCP to transfer the data and require retransmissions for the lost packets, which results in extra delay. Therefore, the network delay on the connection to the j-th cloud can be obtained by

$$D_n^j = T_n^j/(1 - L^j)$$

The data transfer time of a request to a remote cloud and its response traveling back to the local cloud can be approximated using the corresponding network delay which is obtained from the ongoing measurements and updated using the weighted average on the same

connection:

$$D^j = \frac{[S/M]T_n^j}{(1-L^j)},$$

where M is the maximum packet size (bytes), and S is the total data size (bytes).

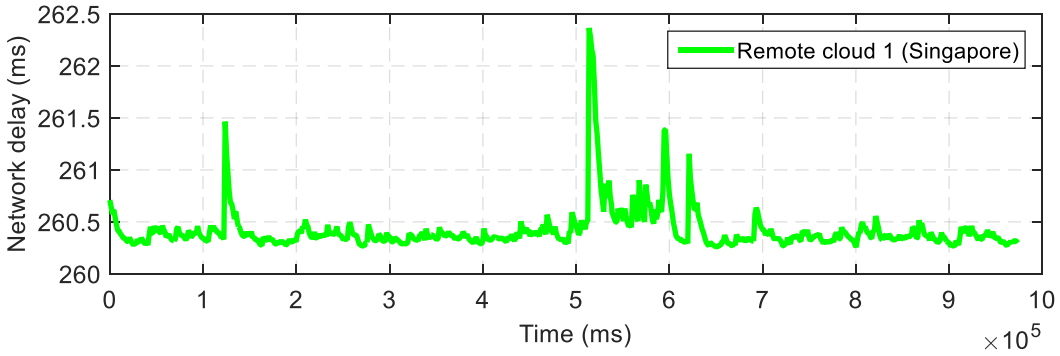
In the subsequent experiments, we apply a simple greedy algorithm which selects the cloud offering the minimal response time for the incoming web requests. Inside the local cloud, a more sophisticated algorithm, RNN-based Reinforcement Learning, is applied to optimize the workload distribution among different servers. If a remote cloud is selected, our proposed routing overlay (namely SMART) is used to optimize the network delay while routing the web requests and responses on the connection between the local and remote cloud.

3.6.1.1 The experimental results

To validate our proposed system, we built an experimental system which includes the local cloud in the test-bed of Imperial College London and the three remote clouds located in Ireland, Virginia and Singapore using Amazon AWS.

The web requests are originated using Httperf for retrieving a file of size 128K from the web servers, which generates I/O bound workload on the web servers. In the local cloud, the TAP is deployed for optimizing the web request allocation across the three web servers with distinct I/O capacity. In the remote clouds, there are other web servers deployed for load balancing. Between the local and remote clouds, the routing overlay-SMART, which is deployed using OpenNebula and configured using Frappe, is used for routing the web requests and responses with the optimized network delay.

The measurement of the network delay on the connections to all the remote clouds is carried out every one second and the average response time for the Httperf requests inside each cloud is also reported every one second. As shown in Figure 41, the network connection to the cloud located in Ireland has the lowest network delay because this cloud is closer to the local cloud in London than the others, and the traffic on the connection appears to be low.



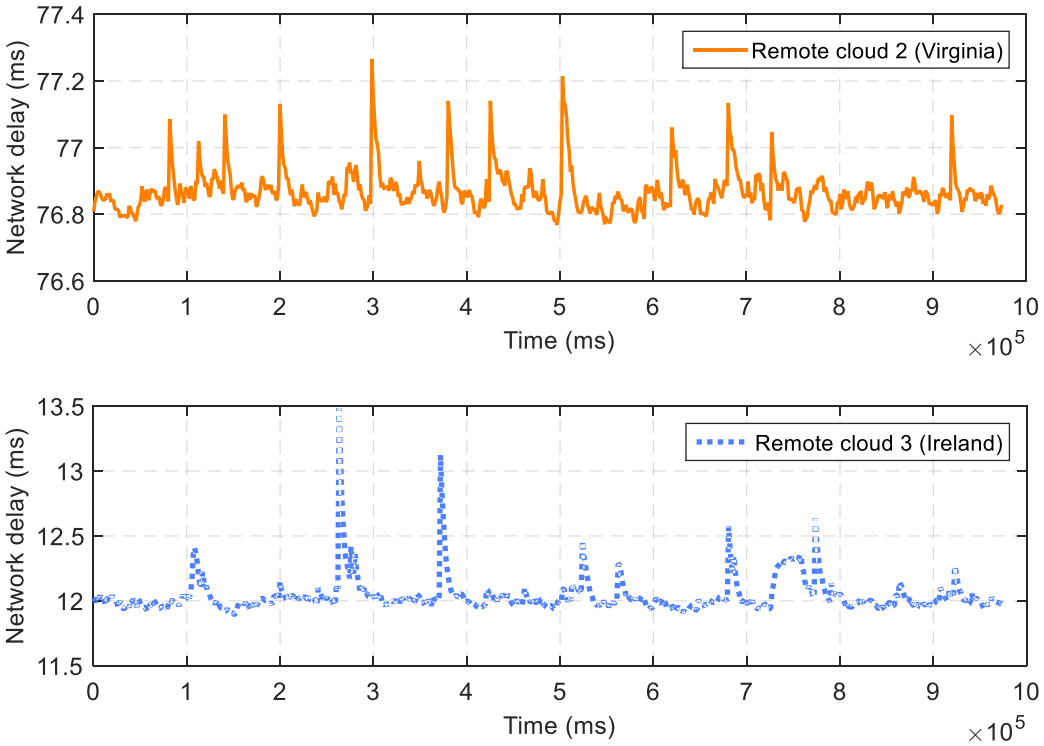


Figure 41 The weighted average network delay over time on the connections to the three remote clouds; it is derived from the measurement of the round-trip delay and loss via pinging.

The web browser requests were originated from a client inside the local cloud at the rate of 0.5 requests per second. We varied the background workloads, which consume I/O capacity on the web servers in the local cloud over time in order to observe whether our TAP is able to adapt to the changing load conditions. Therefore, the local cloud is loaded lightly, modestly, heavily and finally lightly during 20 seconds for each of these successive conditions. As shown in Figure 42, the incoming web requests were dispatched to the local cloud when it was under light and modest load conditions. Our autonomic TAP learned the optimal request allocation based on the online measurement and directed the requests to the web server which provided the fastest response. As the workload in the local cloud increased to a certain "high" level which resulted in a response time that was significantly greater than that of one of the remote clouds (including network delays), the TAP selected the remote cloud for the subsequent web requests until it detected that the local cloud' s time had dropped significantly due to the offloading of workload from background tasks.

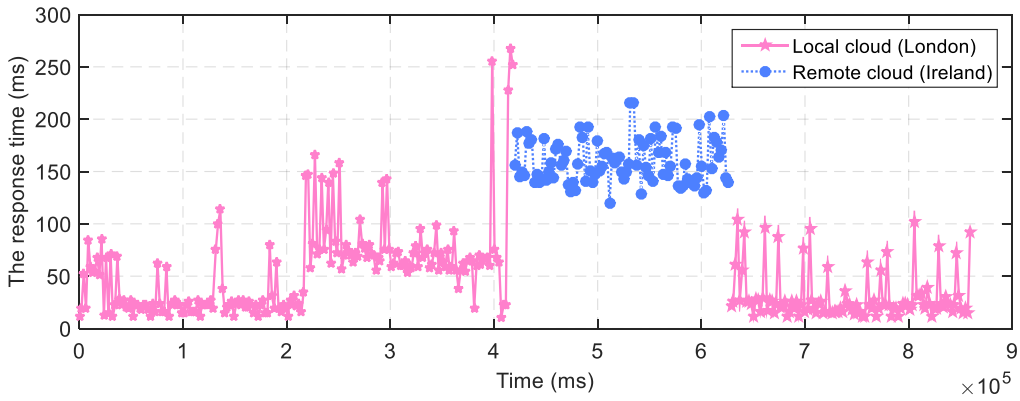


Figure 42 The measured response time from our experiment, for each web request as time elapses; the different colours represent the different clouds where the requests are allocated and processed. We observe that as long as the local cloud's response time is low, tasks are processed locally. When the local response time increases significantly the tasks are sent to the remote cloud, and then when things improve at the local cloud, they are again executed locally.

3.6.2 Live Experiment with SMART

Unfortunately, the AWS platform is not well suited to demonstrate latency improvements with SMART, merely because the IP route between any pair of data centres almost always corresponds to the minimum latency route. We were thus unable to reduce latencies with SMART in the integrated experiment described in Section 3.6.1. In order to demonstrate improvements with SMART, we had to consider bandwidth maximization instead. We describe below the results obtained in a live experiment using the AWS data centres located in Sao Paulo, Tokyo and Oregon (see Figure 43). The experiment consists in a client running in the Tokyo data centre and downloading large files (100 MB) from a web server located in the Sao Paulo data centre. SMART was deployed using the consistency service based on Frappe and developed by IBM. SMART autonomously decides to forward packets either on the IP route, or via the Proxy running in the Oregon data centre.

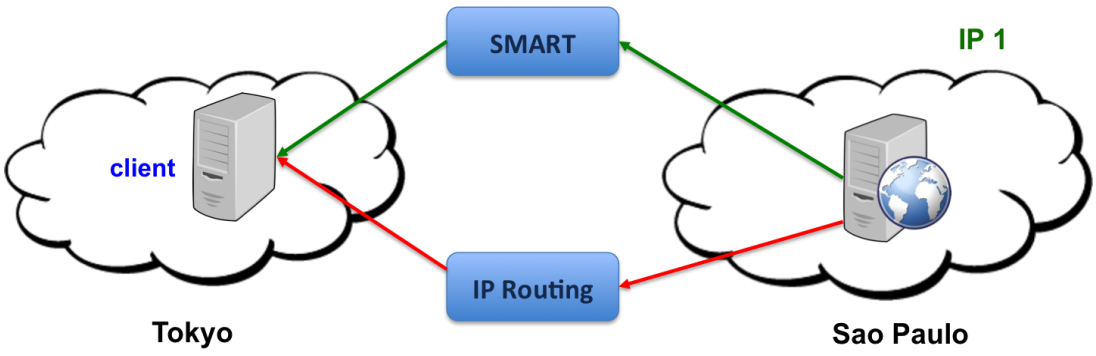


Figure 43: Live experiment with SMART over the Internet.

The experiment was run on April 5th, 2016 between 10:26 and 13:54 (approximately three hours). We downloaded 50 files (a file download every 250 seconds). Every 250 seconds, we send a request to the web server to download a file using a port that is intercepted by SMART, so that the data are routed either directly over the IP route, or via the Proxy in

Oregon. A few second after, another request is sent to the web server to download the same file, but using a port that is not intercepted by SMART, so that the data are sent over the IP route. Results are presented in Figure 44.

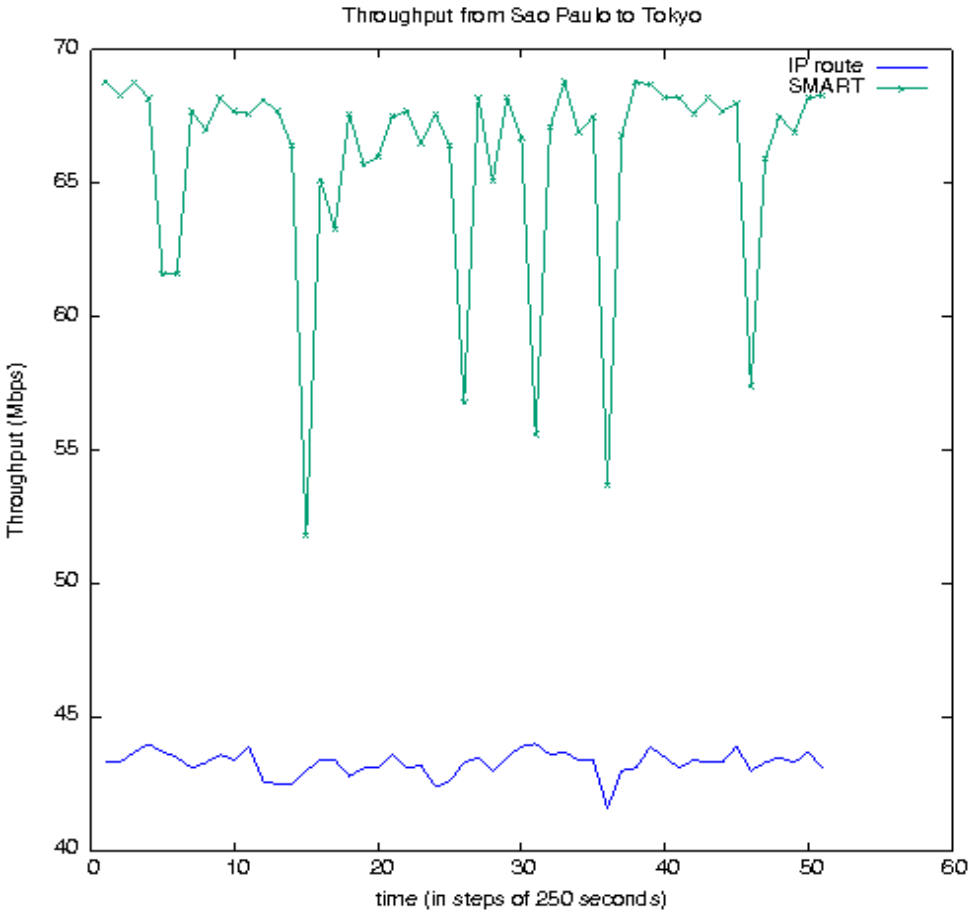


Figure 44: Results obtained in the live experiment with SMART.

3.6.3 Integrated Validation of Machine Learning (ML) framework with Autonomic Overlay Network

The SMART overlay network has been integrated with the Intra/Inter ACM. Intra/Inter ACM uses SMART to enable reliable communication among Controllers and Load Balancers of different Cloud Regions. A *Transmitting Agent* (TA) and a *Receiving Agent* (RA) of SMART are installed and executed on each VM where there is an instance of the Controller and of the Load Balancer. Additionally, in each Cloud Region, a *Router* (R), a *Monitor* (M) and a *Forwarder* (F) of SMART are installed. Finally, a specific VM is dedicated to the *Central Unit* (CU) of SMART.

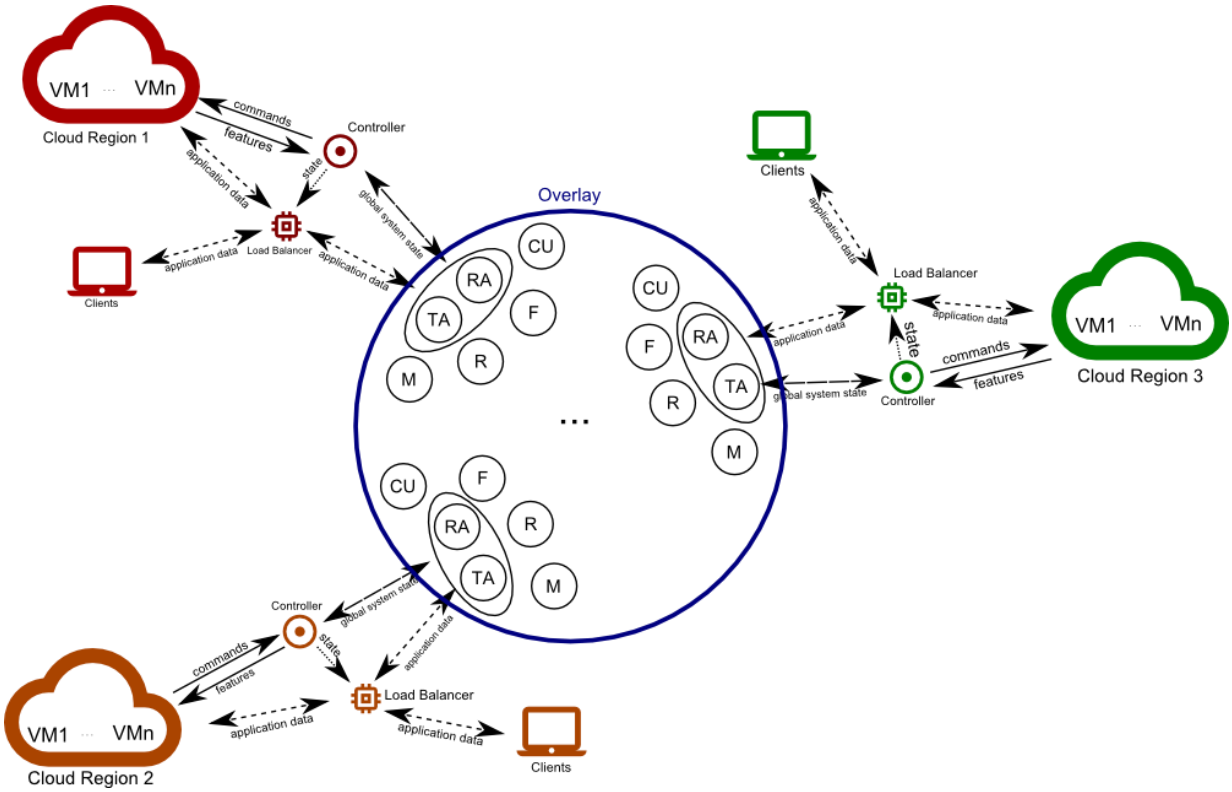


Figure 45: System Architecture for a proactive management of cloud resources based on Machine Learning with SMART.

3.6.3.1 Experimental Results

In the following, we show results of an experimental study, where we run the Intra/Inter ACM both with SMART and without SMART. Data demonstrate results of the integration and the effects on the response time and on the Region Mean Time to Failure (RMMTF). Experiments have been executed with three Cloud Regions located in Frankfurt, Ireland and Munich. The Frankfurt and the Ireland Cloud Region belong to Amazon EC2 service (public cloud service), while the Munich Cloud Region is hosted within a private cloud environment. This configuration gives rise to a hybrid cloud system. For each Cloud Region there were 8 VMs, each one running an instance of TPC-W.

The goal of these experiments is to demonstrate the integration of Intra/Inter ACM with SMART and to evaluate the overhead introduced by SMART. Each experiment has been executed along 24 hours.

The following picture shows a screenshot where we can see a number of terminal windows for running and monitoring components of SMART. The terminal window on left monitors the CU. The three terminal windows on the central column control the router on each Cloud Regions. The terminal windows on right control two example Transmitting Agents (on Frankfurt and Ireland Cloud Regions).

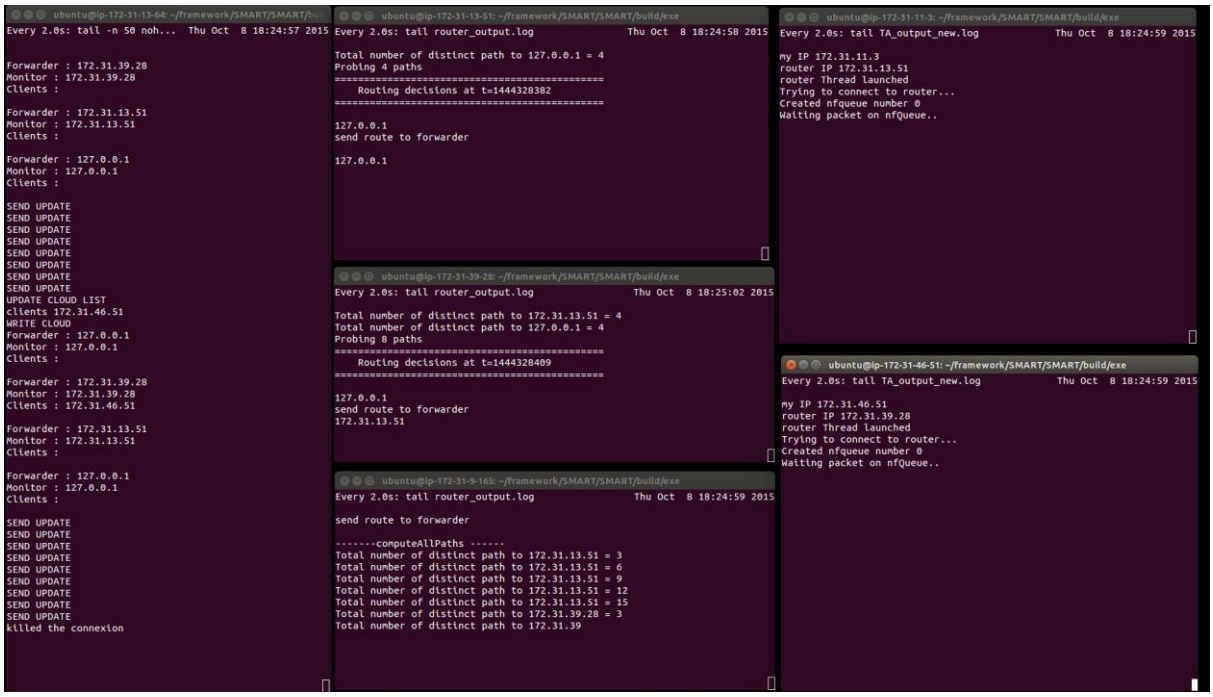


Figure 46: A Screenshot of Terminal Windows for Running and Controlling SMART's components

The two graphs below show the Average Response Time, as perceived by TPCW users, without SMART and with SMART.

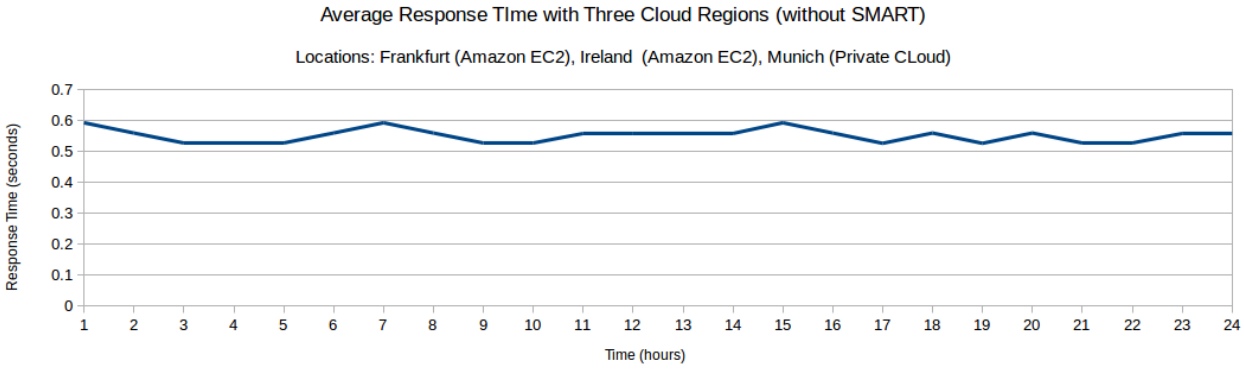


Figure 47 Response Time without SMART

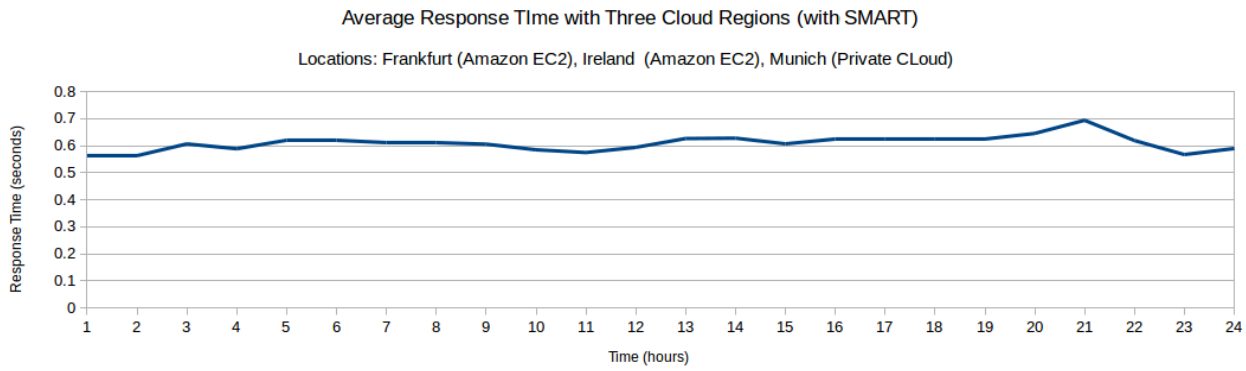


Figure 48 Response Time with SMART

By results, we can see that the Average Response Time with SMART is slightly higher (about 9%) with respect to the case where SMART is not used. This result is due to the overhead introduced by SMART. Particularly, with SMART, TCP-IP packets are forwarder from one load balancer to a load balancer in another cloud region through the SMART Router, which, in turn, forwards them to other cloud region. Hence, there is an additional hop along the path of TCP-IP packets.

The two graphs below show the RMMTF for each Cloud Region, both without SMART and with SMART.

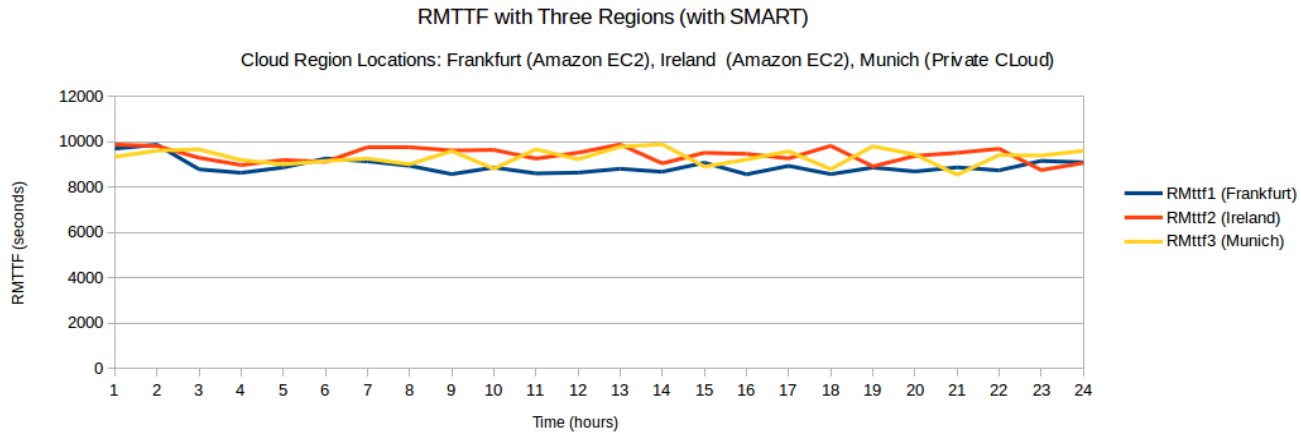


Figure 49 RMTTF without SMART

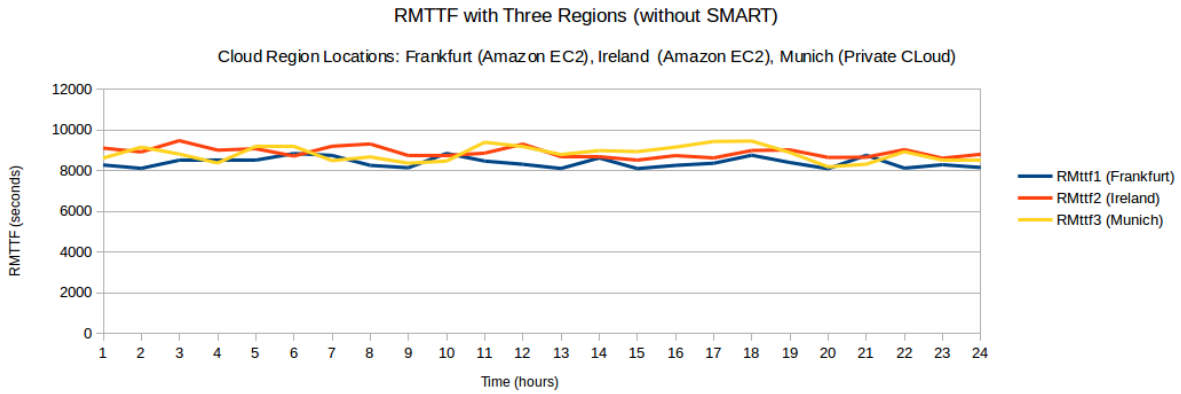


Figure 50 RMTTF with SMART

Graphs show that, in both cases, the RMMTF of the three regions are quite similar. However, with SMART, values of RMMTF are slightly higher (about 7%).

The presented results demonstrate the capability of the system, after the integration with SMART, to handle the cases where congestion or errors may occur within the Internet. As a result, the response time and the RMTTF will be maintained at the expected level.

3.6.4 Metrics validation

Innovation	Technical metrics	Validation
Proactive Service Management Using Machine Learning	• Improvement in availability offered by the PANACEA innovation compared to an unmanaged service. • Prediction buffer time: time interval between a reliable anomaly prediction is made and when it effectively occurs. • Number of parameters that need to be monitored. • False positive and false negative ratios • The strategy used by Inter Autonomic Cloud Manager Framework (I-ACMF) to ensure that all available regions show the same RMTTF. • I-ACMF to guarantee scalability (up/down) of (private/public) regions EC2 • Workload automatically to be balanced with respect to different computing power of regions, and to different anomaly occurrences in different regions by Load Balancers.	The experiments have performed with the following key parameters: • Requested application response time, according the SLA, is kept less than *average 3 sec.* • The Remaining Time to Failure (RTTF) of VMs in a cloud region is retained within the interval *(1,400 sec. – 400 sec.)* • The Region(s) Mean Time to Failures (RMPTTF) is preserved inside the interval *(10,000sec. – 4,000sec).* • Choosing the safety value, (*Threshold) equal to 540sec,* for a safety proactive rejuvenation of VMs in order to achieve *0.16 % false negatives.* • Selecting initially *2 Cloud regions (can be more regions)* with *scheduled numerous VMs* and adding in an autonomous way a *third cloud region (can be more regions)* for scalability (up/down) of (private/public) regions i.e. *elasticity property* realized. • *Reaching Equilibrium 33%* of forwarding probability via *Load Balancing* due to adding proactively *third cloud region* (can be more regions and the *equilibrium to be reached.* • Parameters reductions via Lasso Regularization in experiments from total number 25 to 6 monitored for building the ML prediction models. • Achieving *0.16 % false negatives,* using *10% false positives* and the rest of variables specified within the gain formula in [25], the availability can be increased: a) without ML prediction A_{orig}=0.9999 b) with a prediction to obtain A_{pfh} =0.999983. In this case, for one year, the Outage is reduced significantly: Availability Outage 0,9999 52mn 34s; 0,99999 5mn 15s .

Online QoS-driven Task Allocation	• Improvement in response times compared to other typical load balancing algorithms (e.g., round-robin)	• Improvement in response times of user requests, which can be as high as 35% for I/O bound jobs with respect to round robin. • Capability to dynamically allocate jobs to the most appropriate servers in multi-cloud environments.
Routing Overlay	• Typical network metrics: throughput, packet latency and loss rate. • Path discovery time, i.e., the time it takes to find a new optimal path between two clouds, • Network overhead of the overlay system due to monitoring and control.	• Experimental results with data collected over the Internet show reduction of latency by 2 in some cases, and throughput multiplied by 5 in some other cases. In a live experiment over the Internet, the throughput between Sao Paulo and Tokyo was increased by 51%. • Short discovery time (e.g., 12 measurement epochs between Narita and Santiago in NLNog) • Significant reduction of the monitoring effort with respect to the all-pairs probing approach, e.g., only 5 links are monitored per measurement epoch instead of 342 in NLNog. • The per-packet overhead is 40 Bytes (20 Bytes for SMART Header and 20 Bytes for UDP tunnel). The extra time per packet was measured to be less than 3ms.
Frappe	• High availability • Leader election	• Achieve High availability for Central Unit (CU) of SMART system
Open Nebula	• Cloud manager	• Achieve the deployment of VMs and software components over hybrid cloud environment

Table 5: USE CASE 1 - Validation of PANACEA innovations metrics.

4 EXPERIMENTS IN USE CASE 2

As it was introduced in deliverable D4.1 [8], Data Analytics as a Service (DAaaS) is a Cloud platform designed at Atos to help our costumers to easily build data processing workflows from different big data sources. Although the platform has been continuing its developing course inside Atos, to prove the validity of PANACEA and the utility it could have for internal Atos products, Atos brought the code of the initial proof of concept, developed initially by Atos Research and Innovation department.

The main focus of the experiments detailed in the following sections was done around Apache Hadoop 2 [9]. Hadoop, one of the most successful open source big-data analysis tools, is a key component in the commercial solution of DAaaS, so the learnings and experience collected with the following experiments will be directly beneficial for Atos product portfolio. At the same time, since we are using an open source tool, the findings published here could be also employed by other third party companies or entities.

The rest of the section goes as follows: Section 4.1 introduces the experiment setup for this UC both in inter and intra Cloud deployments. Section 4.2 details the integration with the Pervasive Monitoring solution of PANACEA. Section 4.3 details the integration of UC2 with the PANACEA intercloud manager, in particular it demonstrates the usefulness of the Self-Configuring and Self-Adapting mechanisms of PANACEA with UC2. Finally Section 4.4 demonstrates the integration with all the previous PANACEA technologies plus the Overlay Networks and Machine Learning Framework with UC2.

4.1 Experiment setup

Three different types of deployment have been performed for this use case to validate the PANACEA innovations. The first one in a local Cloud testbed at Atos facilities, to validate the self-configuring and self-adapting technologies of PANACEA, and the other two in Amazon EC2 [10], one to validate the multicloud testbed deployment and overlay network (using in part the local Cloud testbed infrastructure), the other one to validate the machine learning framework technologies. In this section we describe each one of those deployments with more detail.

Local Cloud testbed:

As commented before, to validate the self-configuring and self-adapting technologies of PANACEA we deployed the second use case and part of the PANACEA developed technologies in a local testbed at Atos facilities. This testbed was composed of the following resources:

- A Virtual Machine with 1 virtual core and 1 GB of memory. It runs Ubuntu 14.04 LTS [15] and is configured as the frontend of OpenNebula 4.14 [11], including the specific modifications for PANACEA [12].
- One physical server with 2 Intel Xeon ES-2620 processors (24 logical cores in total), 32 GB of memory and 1 TB of disk. It runs Ubuntu Linux 14.04 LTS [15] and is configured as a physical host for OpenNebula.

The figure 51 details the typical deployment of the DAaaS use case in this testbed, although Sections 4.2 and 4.3 will provide more details about how this is configured, lets see now a

short description of each one of the detailed components per VM (represented by the grey boxes):

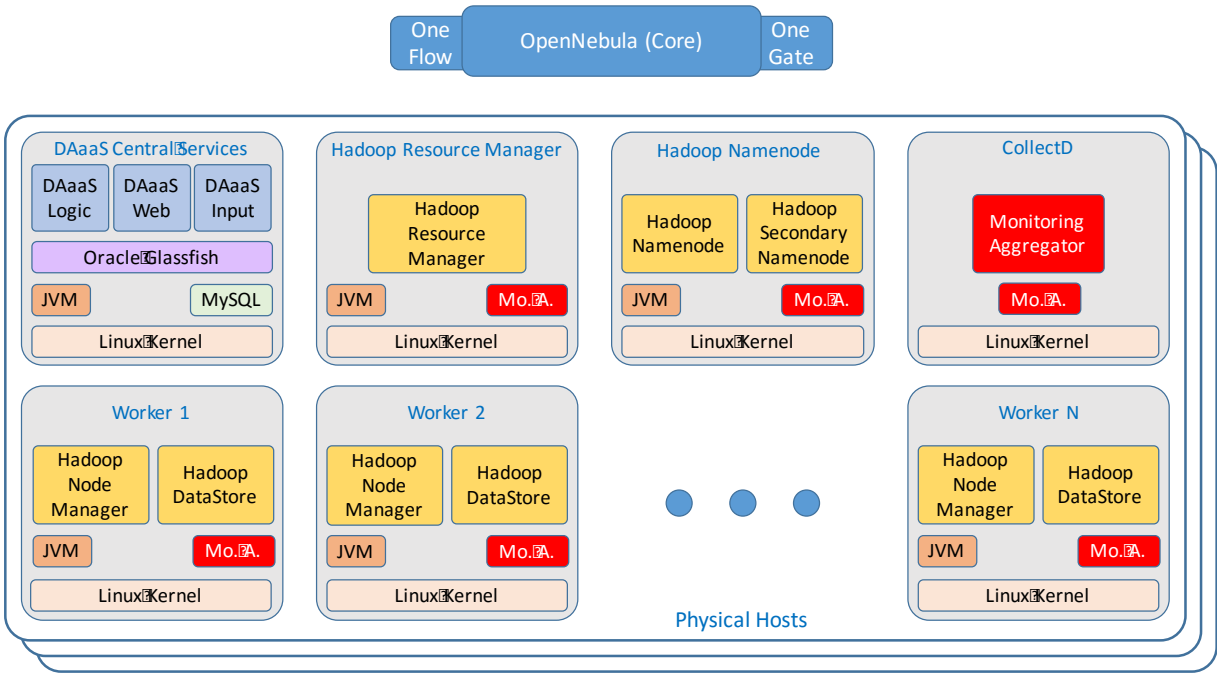


Figure 51: The initial deployment schema for the DAaaS UC.

- DAaaS central services: The DAaaS central services consist of the following components:
    - DAaaS sensor input: Web services that collect the data incoming to the platform.
    - DAaaS web: The interface for the users to configure the different analysis algorithms and to read results.
    - DAaaS logic: The coordinator of the recollection of the results of the analysis using the algorithms configured by the user.
    - Oracle Glassfish [13]: The container for the three DAaaS components presented above.
    - MySQL [14]: The relational database used to store user information and their specific configurations.
    - JVM [16]: The Oracle Glassfish server runs in a Java Virtual Machine.
- Hadoop Resource Manager (YARN) [17]: Since the version 2 of Hadoop, the Resource Manager lives as independent project. YARN is the responsible of allocating different processing task to the different Hadoop Worker Nodes. Hadoop employs Java, so you see there a JVM component as in the previous VM. Also, there is a Monitoring Agent (Mo. A.), more details about it in the bullet point that talks about the CollectD VM.
- Hadoop NameNode [18]: By default, the storage solution for a Hadoop cluster installation is based on the Hadoop File System (HDFS). The HDFS has a central component, the Namenode. The Namenode is responsible of keeping a register indicating where a piece of data is storage (in a replicated way) in the Hadoop working nodes. An installation of HDFS also requires as backup a Secondary Namenode, it is recommended to do it in a separate host, but since we did not use it

for any experiment, it was decided to install it together with the NameNode. Also, as in the case of the Resource Manager, the Namenode uses Java internally, for that reason you see a JVM box there. Also, there is a Monitoring Agent (Mo. A.), more details about it in the next bullet point.

- CollectD [19]: The PANACEA Pervasive Monitoring solution is based on CollectD, for developing proposes, we added a standard CollectD server for the self-configuring and self-adaptation scenarios. Little changes are necessary to adapt it to the Pervasive Monitoring solution. This VM is the responsible to aggregate all the metrics from all VMs. To do so, it is necessary to install a monitoring agent (Mo. A.) in each VM, this agent automatically registers to the CollectD VM and sends the metrics there to be queried at any time.
- Hadoop worker node: The worker node is responsible for performing the different tasks in a Hadoop cluster. Since each worker node is a VM, it is possible to dynamically change the number of worker nodes in a Hadoop cluster, limited by the availability of physical resources. Each worker node hosts the following components:
    - Hadoop Nodemanager: This component is responsible of getting all the requests from the YARN component. It will create independent containers for all the applications running in the Hadoop Cluster. This containers will be isolated environments where the application could perform their own Map or Reduce actions.
    - DataNode: Stores a part of the data in the HDFS volume.
    - Other: As commented before, Hadoop needs a Java installation. Also, a Monitoring Agent is installed to send metrics to the CollectD service.
- Linux (Guest OS): All VMs of the DAaaS platform run Linux as the guest OS, in particular for all our experiments we employed Ubuntu 14.04 LTS [15].

The size of the Hadoop deployment is variable, and depends on the experiment procedures. The most significant variable in the deployment is the number of worker nodes. We discuss the details of the central components in the DAaaS platform below:

DAaaS central services:

The DAaaS application has been developed using J2EE technologies. They require an installation of JDK 7 [16] or higher. The application has been developed under the Oracle Glassfish [13] application server; version 4 of this server is necessary. The DAaaS services make use of an Oracle MySQL [14] database, which requires an installation of version 5 or higher. In order to access this database, it is necessary to get a compatible Connector/J [20] library.

All DAaaS central services have been implemented using Java technologies such as Struts 2, Spring, or Hibernate. The code already compiled and packaged will be provided by Atos. This entire platform has been tested under the Linux OS. We do not require any particular version of Linux.

Apache Hadoop:

The DAaaS platform has been developed using Apache Hadoop version 1.x.x, but for the scenarios presented here, we adapted it to Hadoop 2.x (in particular, for all the scenarios, Hadoop 2.6 was employed).

OpenNebula Services:

As it is depicted in the figure 51, the management of the Cloud infrastructure is performed by OpenNebula. The OpenNebula Core controls the physical and virtual resources of the infrastructure, monitoring hosts to know if they have available resources and automatically managing the lifecycle of VMs on them.

OneFlow allows users and administrators to define, execute and manage services composed of interconnected VMs with deployment dependencies between them. Each group of VMs is deployed and managed as a single entity, and is completely integrated with the advanced OpenNebula user and group management.

DAaaS service is deployed via the OneFlow service. Thanks to this it is possible to define the DAaaS collection of VMs as a unique service in OpenNebula, this information will provide the relations between VMs and dependencies between them. More details about this are provided in Section 4.3.

OneGate provides a REST interface for VMs to interact with OpenNebula. OpenNebula assigns an individual security token to each VM instance, and makes it accessible to applications running inside the VM through the contextualization mechanism. Applications use this token to contact OneGate in order to request operations to OpenNebula. OneGate checks the VM ID and the token sent and, if valid, the operation is forwarded to OpenNebula through its XML-RPC interface.

OneGate allows VMs to pull information from OpenNebula. For VMs that are part of a service, they can also retrieve information from other VMs or retrieve service information. This can be used to automate the deployment of services and VMs. OneGate also allows VMs to push information to OpenNebula. This can be used to provide configuration information or performance metrics.

Moreover, OneGate allows VMs to request operations on VMs, like `shutdown`, `resched`, `stop`, `suspend`, `resume`, `hold`, `release`, `resize`, `migrate`…. Also, the `scale` operation on the service allows a VM to modify the cardinality (number of VMs) of a given role, which is the basis for autonomic elasticity.

OneGate is going to be used by the contextualization scripts for self-configuration, and by the elasticity logic of the DAaaS application. Again, more details about this are provided in Section 4.3.

MultiCloud testbed:

The following figure represents the components deployed for the MultiCloud Testbed scenario:

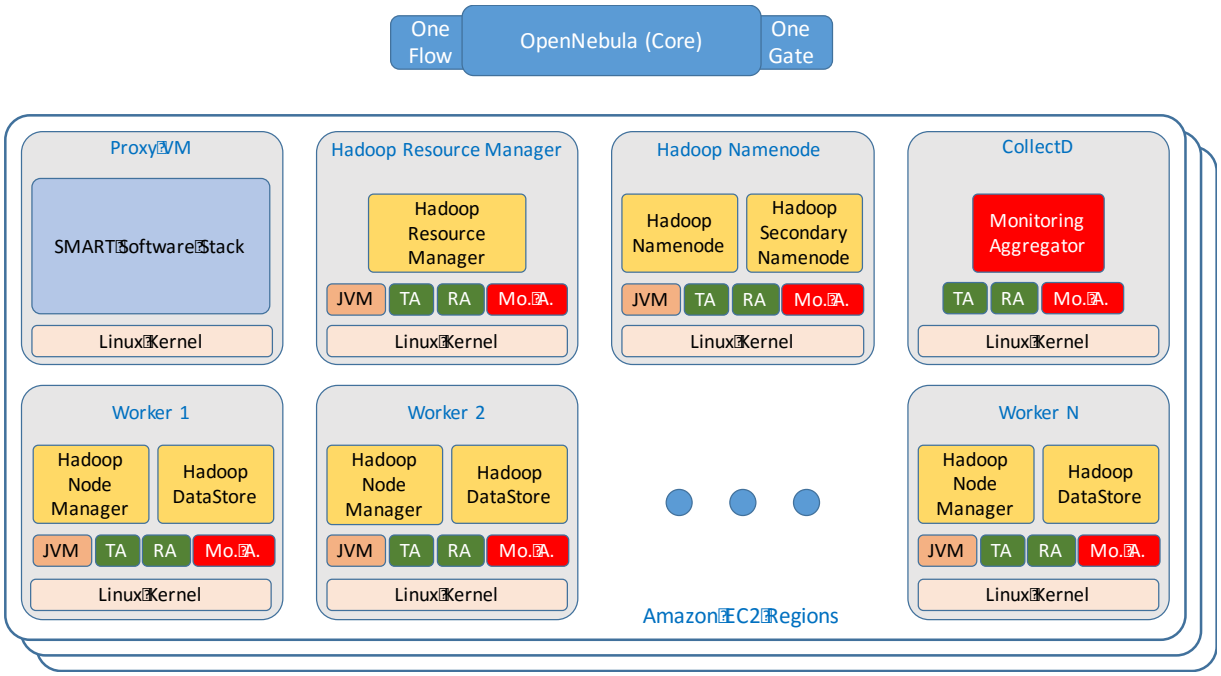


Figure 52: MultiCloud testbed deployment for DAaaS.

As you can see, the scenario is quite similar to the Local Cloud Testbed explained before. Here we are going to only focus in the changes.

In this scenario we have configured OpenNebula to be able to deploy VMs in 7 Amazon EC2 regions: Ireland, Tokyo, Sidney, Singapore, California, Oregon, and Virginia. From the OpenNebula point of view, each region acts as an independent Cloud testbed. This is the same OpenNebula configuration as in the previous case, so deployment at Atos facilities in Spain is possible, acting as the eighth testbed.

The main difference with the previous deployment is the inclusion of the SMART software in all the VMs. The new components are:

- Proxy VMs: It is necessary to deploy one of these VMs per Amazon region. These VMs will contain the necessary proxy SMART software to intelligently redirect the traffic. One of the proxy VMs will act as *leader*, and will have the CU server configured (see section 4.4 for more information).
- TA/RA software – Each VM associated to the Hadoop deployment need to have installed the SMART Transmission Agent (TA) and Reception Agent (RA), each one of them configured to connect to its local Proxy. More details about this in section 4.4.

4.2 Pervasive Monitoring

As commented in the previous section, to support the Pervasive Monitoring solution developed in PANACEA, we have implemented a series of CollectD monitoring metrics for Hadoop. These metrics were later used by the Machine Learning Framework and the Elasticity scenario to take different decisions based on the values.

Apart from the typical and standard metrics provided by CollectD, such as: CPU usage, RAM usage, disk usage, IOPS, etc. That are also necessary to know the health of Hadoop, extra metrics were implemented based on the internal Hadoop metrics system. Those metrics can

be sent by Hadoop to a log file or to a Ganglia monitoring system. For this project the first solution was selected, making use of the CollectD Exec plugin to retrieve then. The following metrics collector were implemented for Hadoop:

- **Hadoop Cluster Metrics –** These metrics report of the status of the different nodes in the Hadoop cluster from YARN perspective.
- **Hadoop JVM Metrics –** These metrics report on the status of the JVM that Hadoop uses for its own execution. This script collects this metric per node of the Hadoop cluster.
- **Hadoop YARN Metrics System –** These metrics report if the internal system of Hadoop for collecting metrics is working as expected.
- **Hadoop Queue Metrics –** These metrics reports in the work queue of Map-Reduce jobs.
- **Hadoop RPC Metrics –** Hadoop offers a series of RPC APIs for users to interact programmatically with it. These metrics reports in the status of all the RPC ports.
- **Hadoop UGI Metrics –** Hadoop offers several graphical user interfaces. These metrics reports the status of those systems.

4.3 Self-Configuration and Autonomic Elasticity

Objectives:

This section describes the different experiments for Self-Configuration and Autonomic Elasticity from the point of view of Hadoop cluster, but first it is important to note why this kind of solutions are interesting from Atos point of view.

As it is going to be discussed later on in the scenario workflow description, deploying an Hadoop cluster is not an easy task, and to really save in OPEX costs, it needs to be automated to the maximum. Automation has been one of the key revolution in the latest years of what has become known as DevOps. The OneFlow/OneGate technologies help to a great extent to automate the deployment of a Hadoop cluster. In a few minutes it is quite easy to have several VMs already configured, with its masters and working nodes. Performing these kinds of actions usually takes hours if done manually.

From the other side, the Autonomic Elasticity will help to size the number of worker nodes in the cluster for the amount of tasks in the queue. This will optimize the number of resources used for a given and variable amount of work in a system like DAaaS, saving from one side costs in the amount of resources running at a giving moment and, from other side, the user satisfaction, since when the resources are really necessary to give a quick result, they are automatically created on the fly.

Automatic Deployment and Self-Configuring:

From the figure 51 it is possible to observe that any DAaaS/Hadoop deployment requires several VMs. As in any Cloud application deployment, the network configuration information is not available before starting the deployment. It is then quite important to think a bit about the deployment workflow and which VMs need to be deployed first and which VMs later.

In this particular case, we thought that the best solution was to deploy first the initial set of workers, followed from the NameNode, CollectD and then the ResourceManager. The

ResourceManager will then execute a series of contextualization scripts that will configure all initial deployment VMs.

This workflow in more detail:

1. The service is defined in OneFlow format, an example can be seen in the Annex A. It is clear to see from this document that the first set of VMs deployed by OpenNebula are the Hadoop Worker nodes. Once those VMs are running and contextualized, OneFlow will deploy the NameNode, CollectD, and ResourceManager.
2. Once the ResourceManager VM is running, it will start a long script that will perform the following steps to fully configure the Hadoop cluster via the OpenNebula OneGate service to self-configure itself (network configuration, starting DAaaS services, starting monitoring services, etc.).
3. Once all this is done, the Hadoop cluster is ready to be used. Users can submit map-reduce jobs to it. At the same time the ResourceManager starts the elasticity monitoring scripts. We will talk in detail about then in the next pages.

Using the previous workflow based on the OneFlow and OneGate technologies it is possible to deploy and entire Hadoop cluster in a matter of minutes.

Elasticity:

To discuss the topic of elasticity in the scope of Apache Hadoop 2.x, it is necessary to divide the problem into two different parts. From one side web have a workflow process, where it is necessary to see how exactly add or remove nodes from a Hadoop cluster on the fly. From other side, it is necessary to understand which parameters are relevant in deciding when to increase or decrease the number of working nodes in Hadoop. Let's see these two problems one by one:

Elasticity workflow:

Each one of the worker nodes of Hadoop acts in a two-way fashion, from one side it holds a copy of the data for the Hadoop File System that it is controlled by the NameNode, from other side it is used by the ResourceManager as computation power to calculate the different Map-Reduce applications of the users. When we add or remove a worker node from the cluster it is necessary to consider both aspects.

As it was commented before, after the cluster is started there is a process, running in the background in the ResourceManager, that starts monitoring the stress levels of the cluster (the parameters used to take this decision are explained later on in the text). This process follows the next loop:

1. Monitor the status of the cluster, if it is detected that the cluster it is using less or more resources than necessary:
    a. Request via OneGate to delete a specific node in the cluster. In the case the cluster is using more resources than necessary.
    b. Request via OneGate to OneFlow to increase the number of worker nodes by one. In the case that the cluster is using less resources than necessary.
2. Wait until the new worker node is created or deleted to continue monitoring the status of the cluster. The action of deleting a node or adding one takes time, several minutes, so it is necessary to give time to the cluster to see if the adaptation action takes effect before allowing it to adapt another adaptation action.

3. Once the adaptation actions have been performed, go back to 1.

The workflows to add or to delete a worker node are quite different. The simple one being the process of adding a node:

1. Once the request has been made via OneGate to create a new worker node, a script is started, monitoring each few seconds via OneGate the deployment of the node. This script will do the following:
    a. Wait until the service moves at OneFlow level from the state: SCALE to COOLDOWN to RUNNING[6].
    b. Once the service is back to RUNNING state, it will determine the IP address of the new VM to connect to it.
2. After the new VM is ready, the ResourceManager and NodeName are configured to make use of it, adding it to their slave list and refreshing both services.

The process of deletion of a node it is more complicated for two reasons:

- This worker node could contain a replica of data of the HDFS. The NameNode for the Hadoop used in this case it is configured to keep at least three copies of each piece of data in different worker nodes. If the node to be deleted has a copy of any data, the NameNode needs to replicate this in the other worker nodes to make sure no data is in risk of being lost. For this reason, the elasticity logic it is configure to always have a minimum of three nodes.
- This worker node could be running a map or a reduce operation. Deleting the worker node in the middle of any of them, will force the ResourceManager to rollback part of the calculations and lose time having to recalculate all or part of the map-reduce application.

Considering the two previous points, the deletion of a worker node elasticity algorithm works in the following way:

1. When it is necessary to decrease by one the number of worker VMs of the cluster, the one selected to be deleted is the last one to be added. The main reason it is that is less probable that this VM has a big amount of HDFS data. HDFS works in quite an static way, it does not balances the data when a new worker node is added, although this could be beneficial from a security point of view (less risk of losing several worker nodes at the same time that will lose definitively data), balancing the data could create a heavy increase of the network traffic that will slow the performance of the whole cluster.
2. Before deleting the VM, it is marked as decommissioned at the ResourceManager and NameNode level. It is necessary to wait until those two Hadoop components mark that VM as unused, this happens when:
    a. All the map or reduce tasks running in that VM are finished. When marked as

[6] The explanation of these states is the following one: SCALE – OpenNebula is creating a new worker node, it selects a physical host/Amazon region to deploy the node, copying the disk image to it; COOLDOWN – OpenNebula is waiting for the machine to boot up and the for its internal startup process to finish; RUNNING – After the two previous actions finished, OpenNebula via OneFlow reports that all VMs inside the defined service are operating normally.

decommissioned the ResourceManager stops assigning any job to that worker node, but it needs to wait for the actual running it to give their results to the rest of the worker nodes.

b. From the NameNode point of view, a VM after decommissioned it is not liberated until the NameNode makes sure all the date replicated on it, it has at least 3 copies in the rest of the worker nodes[7].

3. Once the VM has been completely decommissioned by both NameNode and ResourceManager. The elasticity engine requests OpenNebula, via OneGate, to delete the VM. In this case OneFlow it is not involved in the loop[8].

Elasticity parameters:

As it has been implied in the previous text, the focus of the elasticity study was performed around the Apache Hadoop YARN resource manager [17]. Topics related to storage where not touched in this work. But there are several parameters that need to be considered in the configuration of YARN before starting to implement an elasticity policy:

- **Scheduling policy** – YARN can be configured to different scheduling policies to assign resources to incoming map-reduce jobs. YARN allows one to create their own scheduler, but in this case, we decided to progress with the standards ones provided by Apache:
  - **Fair Scheduler -** This scheduler bases its scheduling policy into trying that all map-reduce jobs get an equal share of resources over time. By default, the Fair Scheduler bases scheduling fairness decisions only on memory. It can be configured to schedule with both memory and CPU.
  - **FIFO Scheduler –** This scheduler bases its scheduling policy in the First-In-Firt-Out strategy. Applications that arrive first are giving preference over resources. If, after assigning resources to the previous application, there is still space for new applications, other can be deployed and executed in parallel.

The fair scheduler in terms of CPU and Memory usage was configured following the Apache Hadoop recommendations [23]. For an environment where the number of worker nodes change over time, these optimal configuration instructions could be a problem, since it depends in the mean amount of resources. To solve this issue, all worker nodes are created with the same amount of virtual CPUs and RAM memory. In this way, whatever the number of worker nodes, the optimal values for the configuration of the fair scheduler will be the same.

During all our experiments, we monitored the following Hadoop metrics to know the status of the Hadoop cluster:

[7] This operation not always work, leaving the worker node in a deadlock. After reading a lot of documentation about HDFS our conclusion is that to make Hadoop perfectly elastic the best solution it is to switch to a complete object-storage based solution like Amazon S3 or Swift and stop using HDFS, that it is was not designed at the beginning for this kind of elastic operation. But this is out of the scope of this project.

[8] It would be possible to ask OneFlow to decrease the order of worker VMs to one less. This works quite easily in stateless services like webservers behind a Load Balancer, but for state applications like Hadoop, this decommissioned process is more clean. In this way, it is made sure that no job has to be restarted or no data enters in the risk of only having two copies for a period instead of three. The downside it is that takes more time to downsized the cluster, this has an economical impact for the usage of more resources for a longer period.

- AppsSubmitted – Number of map-reduce applications that have been submitted for a period of time to be processed by the Hadoop Cluster.
- AppsPending – Number of map-reduce applications that are still waiting to be executed.
- AppsCompleted – Number of map-reduce applications that have been successful completed
- AllocatedMemoryGB – Total amount of RAM memory allocated by Hadoop for all the cluster while running the scenario.
- ReservedMemoryGB – Total reserved amount of RAM memory allocated by Hadoop for all the cluster while running the scenario.

The parameter selected in the study to see if it is possible to scale up or down the number of worker nodes in a Hadoop Cluster is the number of AppsPending and number of AppsCompleted.

Elasticity results:

All the results presented here were calculated following the experimenting methodology:

- An entire Hadoop cluster composed of 6 different VMs: Monitoring, ResourceManager, NameNode, and 3 workers, was deployed to the same physical host.
- Before the deployment to the physical host and, to minimize external conditions that could affect the results, it was make sure no other competing set of VMs were being executing in that physical host.
- In all the cases the same collection of map-reduce jobs were executed. It was composed of 32 map actions and 1000000 reduce actions. This amount of jobs for the hardware experiment make sure that it will take several hours to finish the jobs, minimizing in part external or temporary influences in the results.
- Those map and reduce jobs were sent to the Resource Manager in a set of small applications, a total of 100. Those applications were submitted to the Resource Manager in sets of 10 applications each 10 minutes with the objective to stress it bit by bit to fire up the elasticity policies.

A continuation we present several graphs that characterise the types of experiments performed.

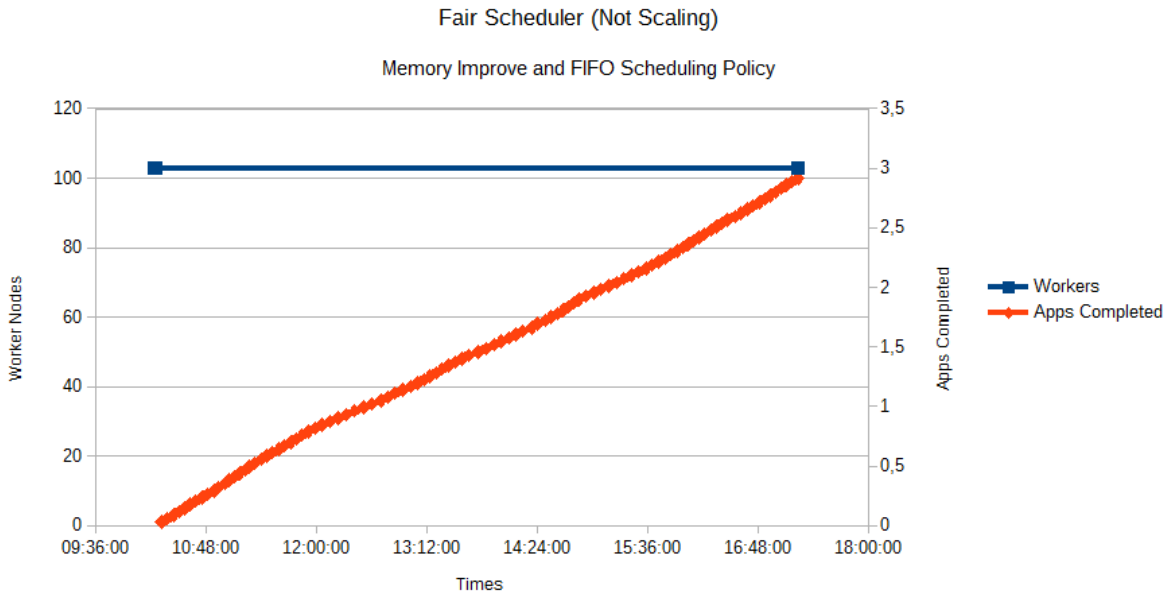


Figure 53: No allowing scaling result.

The figure 53 represents the time of executing the set of previously mentioned applications without allowing the cluster to scale. In this case it takes nearly 7 hours (6 hours, 59 minutes), for the cluster to finish all map-reduce jobs.

The next figure allows the system to scale, using the same configuration parameters:

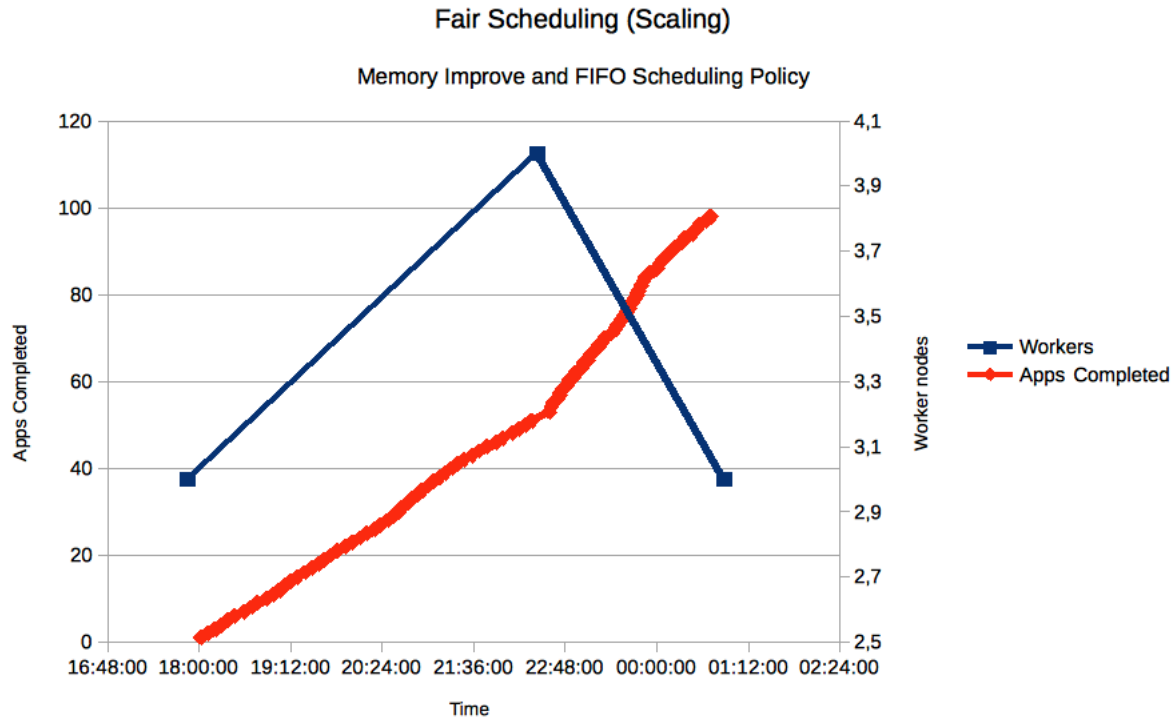


Figure 54: Allowing scaling results.

In this occasion the process took 10 minutes less than in the previous case. Using exactly the same configuration. In the middle of the process it created a new node, and as it can clearly

be seen in the figure 54, when adding the new node the frequency of finishing apps increases, that it also decreases a bit the pending apps queue, the metric that it is used to scale up and down the nodes. When that pending queue decreases under a certain level, the extra worker node is deleted, from 4 to 3, and the frequency of finishing apps also decreases, but since there are not more incoming applications to the queue, it is not problematic.

As a matter of completeness in the results, it is necessary to also comment about the next case. Here we are going to show directly the result after allowing the algorithm to scale:

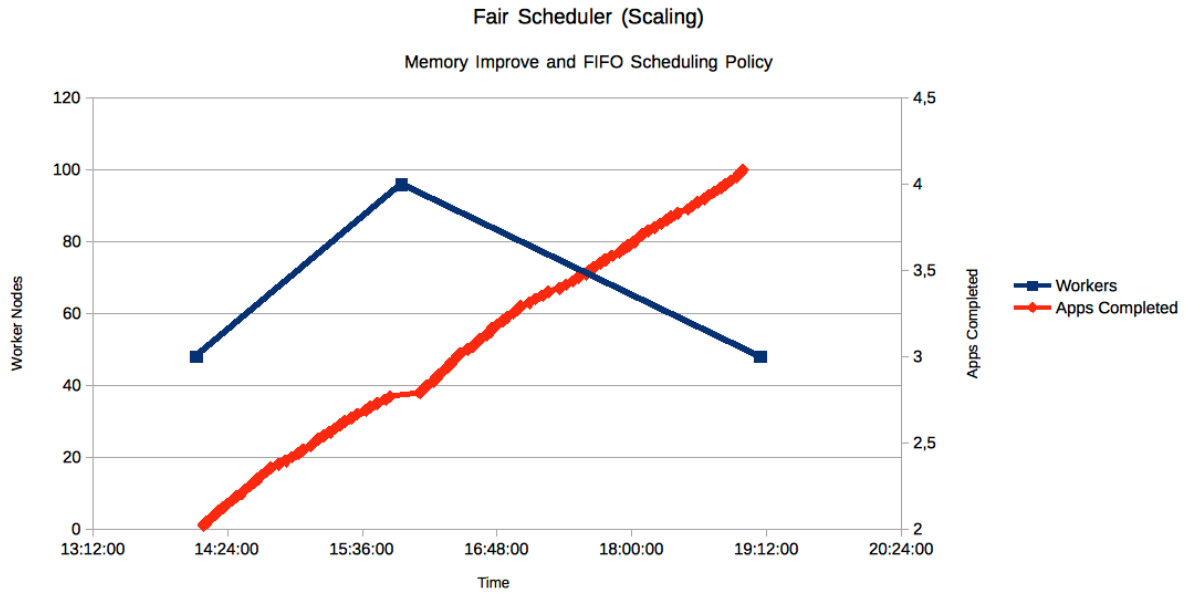


Figure 55: Allowing scaling results, with optimal memory configuration.

This is the result of several tests, allowing the system to scale, and also, optimizing the memory configuration to the maximum (each worker node is allowed to use a bit more of memory than before for map-reduce jobs, to the cost of giving less memory to OS and HDFS). In this case we observe that it only takes nearly 5 hours to finish the job, that is 2 hours less than before. In this case the system also scales, even sooner than before.

Elasticity conclusions:

The work in understanding elasticity in Hadoop has just started internally at Atos. Our findings are in line with other commercial solutions [24], and the objective now it is to learn how to correlate those metrics with the type of incoming applications coming to the cluster.

Also the study of the parameters indicates that other possible solution it is to create clusters adhoc. Thanks to the self-configuration techniques in PANACEA it is quite quick to deploy a complete Hadoop cluster in matter of minutes. Depending on the type of map-reduces jobs that comes to our system, it will be easy to deploy and adhoc Hadoop cluster, optimize in configuration for that type of job, and when it finish, delete it completely. **Saving both OPEX and CAPEX costs.**

In both cases, it is important to note the necessity to separate Hadoop computation resources and HDFS resources, with the possibility to use object-storage solutions instead of HDFS ones. This decoupling will allow to have independent lifecycles for data part, that it would probably have a longer lifetime, to the computation part where, each computation resource, has an smaller lifetime, even in the scale of few hours.

4.4 Integrated Validation

As indicated in the introduction of this section, it was decided to use the second use case to validate several of the PANACEA technologies solutions at the same time. For this proposes we devised the following scenario: an inter-cloud deployment of Hadoop, using OpenNebula PANACEA technologies as inter-cloud manager, including the previously exposed monitoring and elasticity features. Together with this, we added the overlay network technologies, that allow better communications over Internet. The experimental platform in this case is the Amazon EC2 platform, with data centres located in various continents. Each region of Amazon will act as a cloud provider, all managed with OpenNebula.

In this section we first explain a bit the complexity of Hadoop deployment in a multicloud scenario. Later on we present the implemented solution and the test procedure, together with the results.

Inter-cloud deployment of Hadoop

Deploying Hadoop into a multicloud/inter-cloud environment rises two main problems, both of them related to data locality:

- HDFS spread across several multidomains. Whereas communications between nodes located inside the same Cloud infrastructure have low latency, external communications give rise to far greater latencies. Synchronization tasks between nodes are going to take longer time than usual, creating performance impacts.
- Worker nodes need to communicate constantly between each other and to get data from HDFS nodes. If the network part is not performing accordingly, it will be a performance hit to the time to perform the calculations.

For these two previous reasons, typical deployments of Hadoop clusters tend to be deployed into the same domain. When multidomains are necessary, the Hadoop cluster is divided into regions. Well, it is not exactly regions, since Hadoop supports the concept of rack, so it knows if a piece of data, for replication purposes is replicated into several racks, so if a rack fail (for example the switch that connects all those nodes fails), no data is lost. We could extend this rack concept to Amazon regions. In this way all data will be replicated to all the regions by HDFS, making then possible for worker nodes to access quickly the data without too much penalty.

The main problem of this approach is the communication between HDFS nodes located in different regions. Each time users upload data to one of the regions, thousands of MBs of data need to be replicated all over the world. So, improving communications between different regions at HDFS level seems to be the interesting approach, and this is the one that we are going to validate in this integrated version of PANACEA solutions.

Inter-cloud architecture for use case 2

The architecture is presented in the figure 52, and the deployment procedure is quite similar to the one introduced in the section 4.3. The main difference it the necessity to deploy first the SMART proxies and later on TA/RA agents on any Hadoop machine. Each of the Hadoop machines needs to be configured to the respective region to where they are deployed. In the page 87 an example of the OneFlow service description for this inter-cloud deployment can be seen.

For this set of experiments we deploy the following architecture:

- A SMART proxy in Amazon Virginia region. This SMART proxy will also contain the central unit (CU) that will synchronize all communications between proxies.
- A SMART proxy in the other regions (California, Oregon, Ireland, Tokyo, Singapore and, Sydney).
- The rest of the Hadoop installation: One ResourceManager, One NameNode, One Monitoring VM and, several worker nodes.

Experimentation

For this integrated experimentation, the objective was to see if the SMART software could help the Hadoop Filesystem to reduce the times of transmitting data between nodes. To do so, we designed the following experiment:

- We evaluated in which two Amazon regions the SMART software was giving better performance. In the case of this experimentation, those were Tokyo and Sao Paulo.
- The Hadoop cluster was deployed in Tokyo. This included the Resource Manager, the NameNode, and three worker nodes.
- We uploaded 300MB of data to the HDFS of Hadoop.
- We then created a new worker node in Sao Paulo region and added it to the HDFS (the Hadoop filesystem by default is not proactive, it is not going to replicate the data from the nodes in Tokyo to the node in Sao Paulo).
- We then decommissioned one of the worker nodes in Tokyo. Since HDFS is configured to have at least 3 copies of data in all nodes, this forces it to copy the 300MB of data from Tokyo to the Sao Paolo node.

This infrastructure was deployed employing OneFlow/OneGate technologies from OpenNebula. This deployment had the possibility of also deploying and configuring SMART at the same time.

The procedure was to deploy the cluster with and without SMART and to measure the time that it took to transmit the data from Tokyo and Sao Paulo, comparing at the end in which case the transmission was faster, with or without SMART.

The transmission time is written in the logs from the NameNode and working nodes. We only look at transmission time. Note that Hadoop takes an extra time just waiting idle for a moment to decommission the node. That time was ignored.

During these tests we found two types of situations, the ones were there was not benefit from SMART and we were getting the same performance than the cluster without SMART deployed. But, **when SMART was able to find a route with more bandwidth, we observed an improvement of nearly 23% with respect to native IP routing** (1 minute 15 seconds to transmit 300MB vs just 55 seconds).

Conclusions

An entire cluster of Hadoop was deployed with the benefits of better latency and network communications between nodes, in a multi-cloud environment, employing several PANACEA technologies: Self-Configuring, Self-Adapting, Overlay Networks, and Monitoring. Overlay networks in particular could enhance the performance of Hadoop in a multi-cloud environment since the dependency of Hadoop with network performance, reducing the time to deliver the results to the clients of the DAaaS solution. This demonstrates several

technologies of PANACEA working at the same time for a specific application.

At the same time we observed that Hadoop could benefit from SMART in a multicloud environment. Although a reduction of just 20 seconds could seem poor, it is necessary that in big-data type, transmission of big amounts of data between nodes in the distributed filesystem, such as HDFS, happens constantly. Even a few seconds of improvement here and there could have a big benefit in the long run for the number of resources used to run an application.

4.4.1 Hadoop/IRIANC Autonomic Cloud Management (ACM) Integration

This section describes the steps to be taken so as to allow an integration between Hadoop Cluster and IRIANC Autonomic Cloud Management (ACM) System, based on Machine Learning for a proactive management of cloud resources. The Hadoop Cluster installation/configuration is first discussed, then conclusions regarding the integration are provided. The Hadoop cluster is installed by IRIANC in Frankfurt region by sharing the Hadoop configuration, provided by ATOS, and using Amazon AMI (public cloud). The core of the ACM System is installed in Munich Private Cloud for generating the Machine Learning models. Since, the integration is performed in Hybrid Cloud.

4.4.1.1 Hadoop Cluster

The architecture of the deployment of the Hadoop Cluster consists in two master nodes and three slaves.

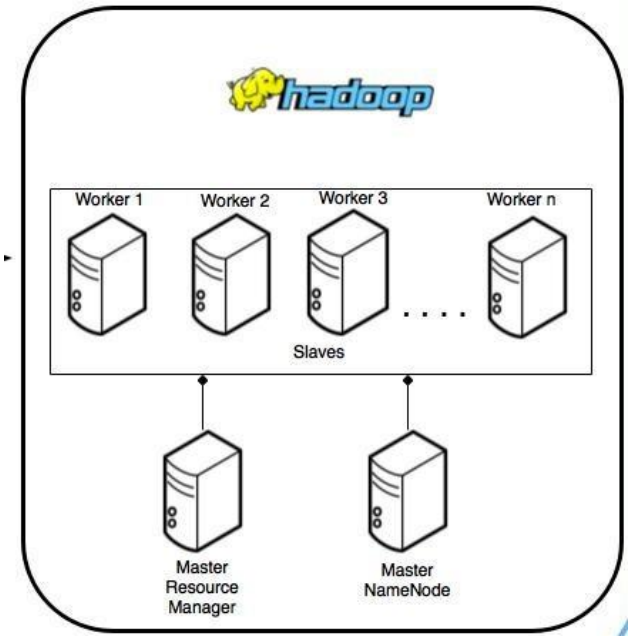


Figure 56 Hadoop cluster.

The configuration to be done in the master nodes (ResourceManager and NameNode) and slaves nodes (Workers) is as follow; for the testbed we must have two **nodes and three workers nodes** at least configured.

4.4.1.2 Considerations regarding the integration

The ACM Framework enables to increase the availability of applications and keeps their response time below a certain threshold by exploiting several techniques in a joint manner:

1. **Duplication of virtualized resources**: at a single cloud region, multiple VMs host the *same* application. These multiple VMs should be seen by the distributed application as *the same virtualized node*
2. **Software rejuvenation:** when a VM is approaching its failure point, a new VM is joined to the pool of active VMs of the application (taking the place of the about-to-fail one), and the failing one is restarted. The application sees the new VM as if it is the *same* as the rejuvenated one

The Hadoop cluster does not allow the join/leave pattern required for software rejuvenation. ACM could be therefore, only used to predict the Remaining Time to Crash (RTTC) of the application. Nevertheless, in order to do so, machine learning-based approaches are used. To generate reliable models the application under study must have the following characteristics:

1. **Software anomalies**: without anomalies, the framework has nothing to learn. Moreover, in case the response time is a measure of interest, the anomalies must be *proportional to the workload of the system*. In the opposite case, it is impossible to reliably predict the response time.

2. **Long execution time**: this is related to a *single instance* of the application. Restarting the same application multiple times is not an option, as anomalies (e.g., memory leaks) are destroyed by the application process' destruction.

If at least these two characteristics are not offered by the application, then it is not possible to use ACM to build prediction models, as either there is nothing to learn (as for point 1) or noise in the process is too high (as for point 2)

4.4.1.3 Status of the integration

So far we have been able to set up 5 VMs using the provided AMI on Amazon. The application according to the steps provided at the top of this document is running. In particular, by using 5 VMs on Amazon, the application lasts seconds without memory leaks and is able to correctly compute an approximation of the mathematical constant π.

Concerning this execution of the experiment, the following are the results:

```
Starting Job
16/04/01 15:29:29 INFO client.RMProxy: Connecting to
ResourceManager at Hop-ResourceMan/172.31.0.200:8050
16/04/01 15:29:29 INFO input.FileInputFormat: Total input
paths to process : 2
```

```
16/04/01 15:29:30 INFO mapreduce.JobSubmitter: number of
splits:2
16/04/01 15:29:30 INFO mapreduce.JobSubmitter: Submitting
tokens for job: job_1459524203763_0001
16/04/01 15:29:30 INFO impl.YarnClientImpl: Submitted
application application_1459524203763_0001
16/04/01 15:29:30 INFO mapreduce.Job: The url to track the
job: http://Hop-
ResourceMan:8088/proxy/application_1459524203763_0001/
...
Job Finished in 28.182 seconds
Estimated value of Pi is 3.14118000000000000000
Number of Maps  = 2
Samples per Map = 100000
```

4.4.1.4 Next steps for the integration

In case the application to compute the mathematical constant π could be modified to last longer and to artificially produce memory leaks, by using AMC framework it will be possible (as shown in previous experimentation) to build prediction models to determine when the application is about to fail.

4.4.1.5 ML-Based Prediction Models for Hadoop π computation

We have installed our ACM System on one of the Hadoop nodes dedicated to computation. In particular, on top of that node, we have installed our *Feature Monitor Client (FMC)*, which was connected to the *Feature Monitor Server (FMS)* running on the *namenode server* of the Hadoop setup.

We have varied the incidence of memory leaks on the π application. This has been done by varying the inter arrival time (namely, the sleep period among two consecutive leak generations) so as to have the application last a different amount of time per each run. The inter arrival time was selected using a random uniform distribution in between 15 and 35 seconds.

Since Hadoop, in the current configuration, does not allow restarting a single node while a map/reduce task is running, nor it allows adding a node during the execution. Therefore, to collect the training data, we have configured FMC to consider a node crashed when the limit for Java Heap was reached for the current configuration of Hadoop. At this point, we manually restarted the VM, killed all other Hadoop-related nodes. When the VM was fully restarted, we restarted as well the Hadoop cluster and a new execution of the software computing π was started. We have repeated this process until we have collected 16 different runs, with the application crashing after around 20 minutes of executions (with variations related to the different inter arrival times). The parameters measured by the FMC module of ACM are reported in Figure 57.

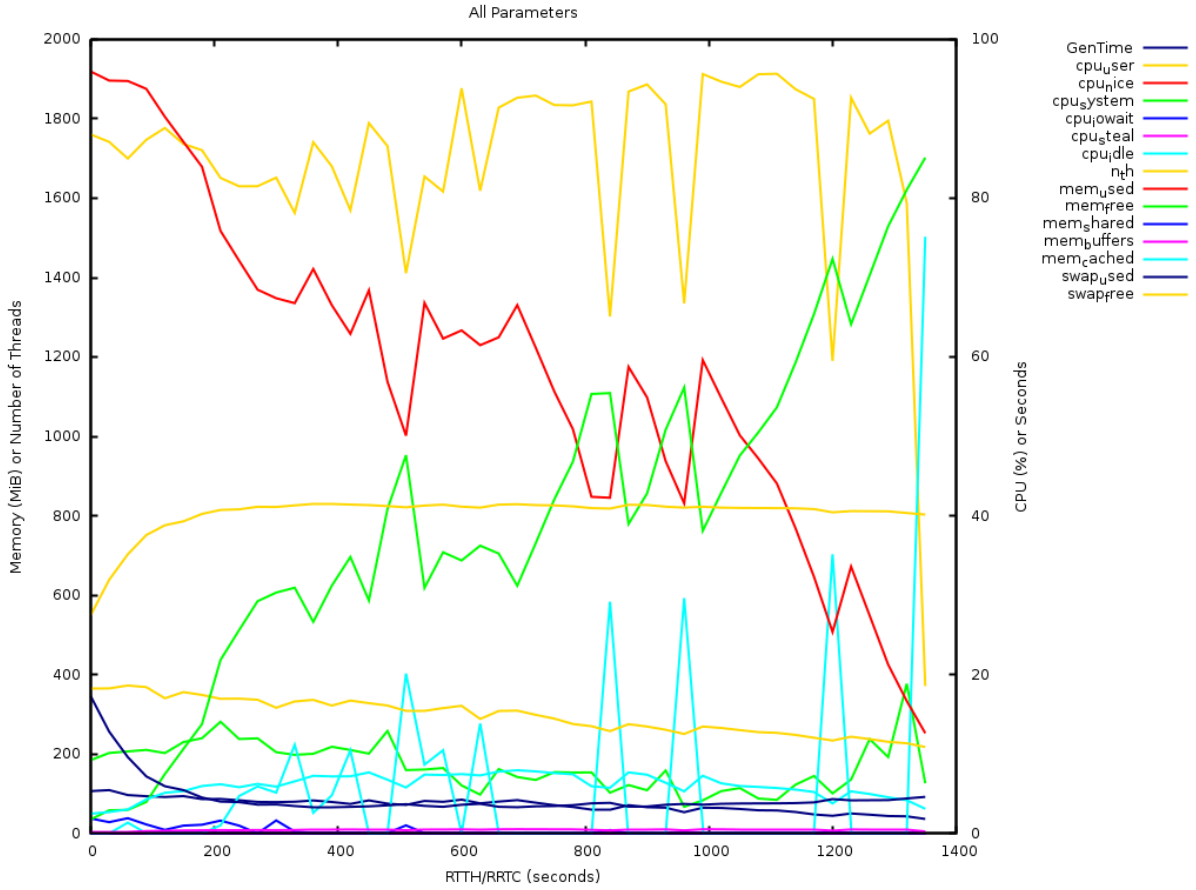


Figure 57 System Parameters measured by FMC

We have then run the ACM ML Framework to build the ML-based prediction models. The first step entails running Lasso Regularization to select only the most relevant parameters to the construction of prediction models. In Figure 58, we report the number of parameters selected by Lasso. As it can be clearly seen, Lasso is correctly able to reduce the number of parameters, which are considered as relevant to the construction of effective ML-based prediction models for the application, which computes π using Hadoop.

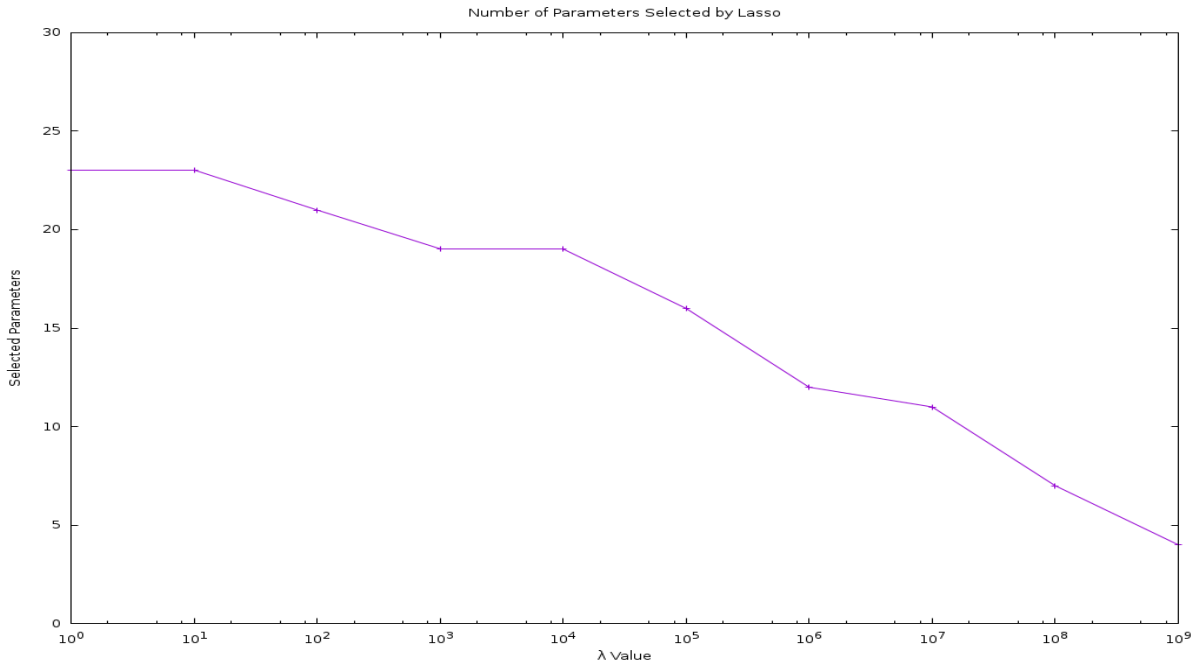


Figure 58 System Parameters selected by Lasso Regularization

In particular, the parameters which have been selected by Lasso are reported in the following table, along with the corresponding associated weight. For the sake of simplicity we only report the results for $\lambda = 10^9$.

Parameter	Weight
mem_free	0.000300275597967
mem_cached	0.001127350080723
swap_used	-0.000132390998576
swap_free	0.000152799780556

Table 6 *: Parameters selected by Lasso with* $\lambda = 10^9$

This result is important because it tells that Lasso correctly identifies that in this particular experiment the system fails only due to memory reasons. In fact, the only type of anomaly injected in the system is memory leaks, and Lasso correctly discards any parameter not necessarily related to memory.

We have measured different types of errors to account for the precision of our predictions models. We report in the following tables the errors associated with all the prediction models that have been generated. For the sake of conciseness, concerning Lasso we report only results for $\lambda = 10^9$. The measured errors are:

- Maximum Absolute Prediction Error – MAPE
- Relative Absolute Prediction Error – RAPE
- Mean Absolute Error – MAE
- Soft-Mean Absolute Error with 10% tolerance Threshold – SMAE-10%
- Soft-Mean Absolute Error with 300 seconds tolerance Threshold – SMAE-30

The minimum errors for all ML techniques achieved for SAME 10 % and SAME 300 are shown in Table 7 and Table 8, respectively. REP Tree is the most accurate in a prediction.

Algorithm	Error (seconds)
Linear Regression	127.631104
M5P	130.2017152
REP Tree	69.677976
SVM	126.5796032
SVM2	126.7674368
Lasso ($\lambda = 10^9$)	407.475510129

Table 7 *SMAE-10% Error*

Algorithm	Error (seconds)
Linear Regression	37.3593872
M5P	40.0017744
REP Tree	15.8972928
SVM	48.2772736
SVM2	47.2935232
Lasso ($\lambda = 10^9$)	360.027964992

Table 8 SAME-300 Error

The training and validation times required by each ML algorithm are reported in Table 9.

Algorithm	Training (seconds)	Validation (seconds)
Linear Regression	0.79	0.17
M5P	1.51	0.10
REP Tree	0.11	0.09
SVM	1.62	0.16
SVM2	1.40	0.16

Table 9 Training and Validation times.

Finally, we report the models fitted using the various machine learning algorithms. Note that the set of parameters used for training is different: Lasso uses only a subset of parameters, namely the ones associated with non-zero weights in the β vector. Therefore, Lasso is more efficient in real-time critical applications. On the other hand, the other algorithms rely on the whole set of input parameters.

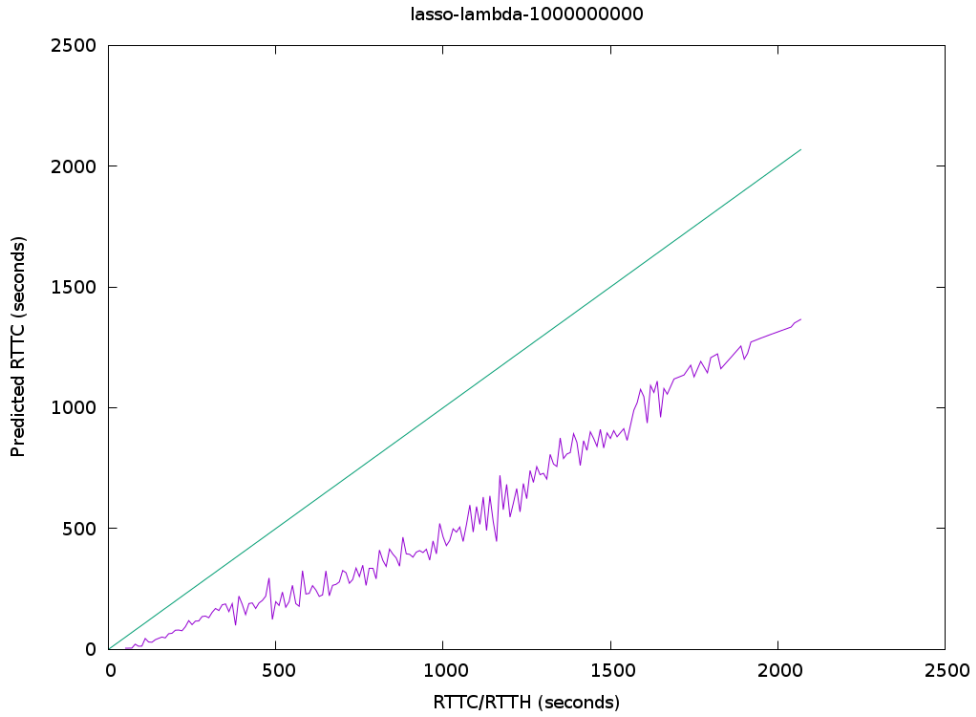


Figure 59 Model fitted using Lasso (with $\lambda = 10^9$)

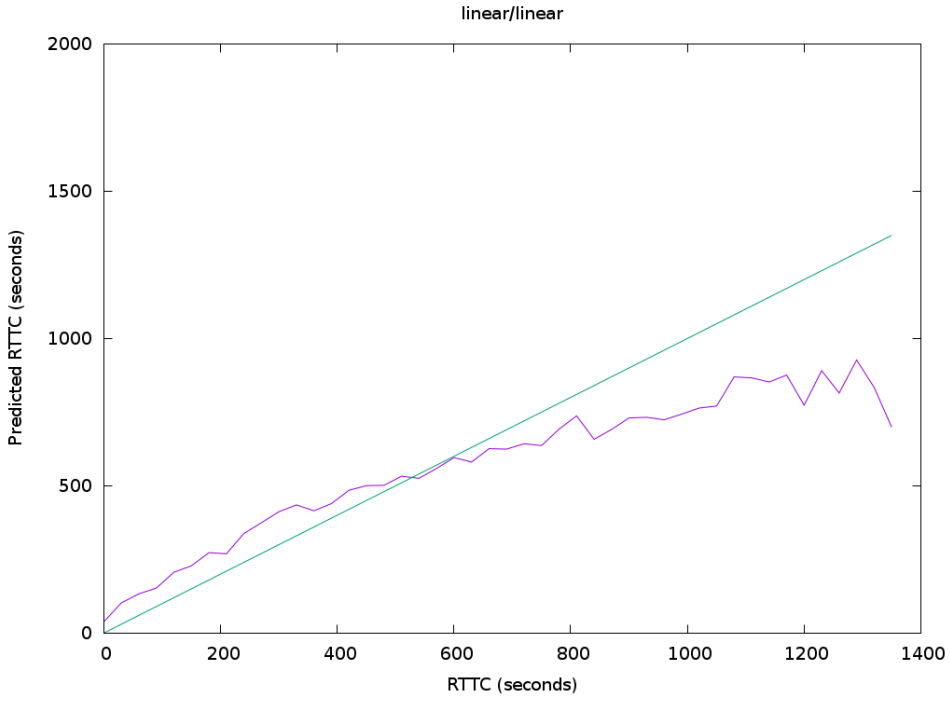


Figure 60 Model fitted using Linear Regression

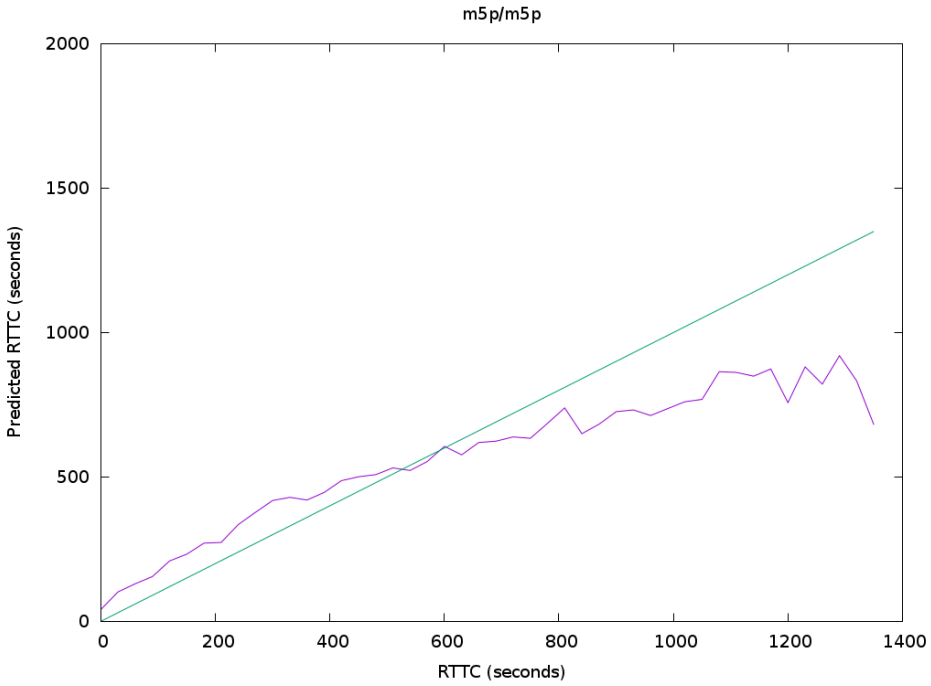


Figure 61 Model fitted using M5P.

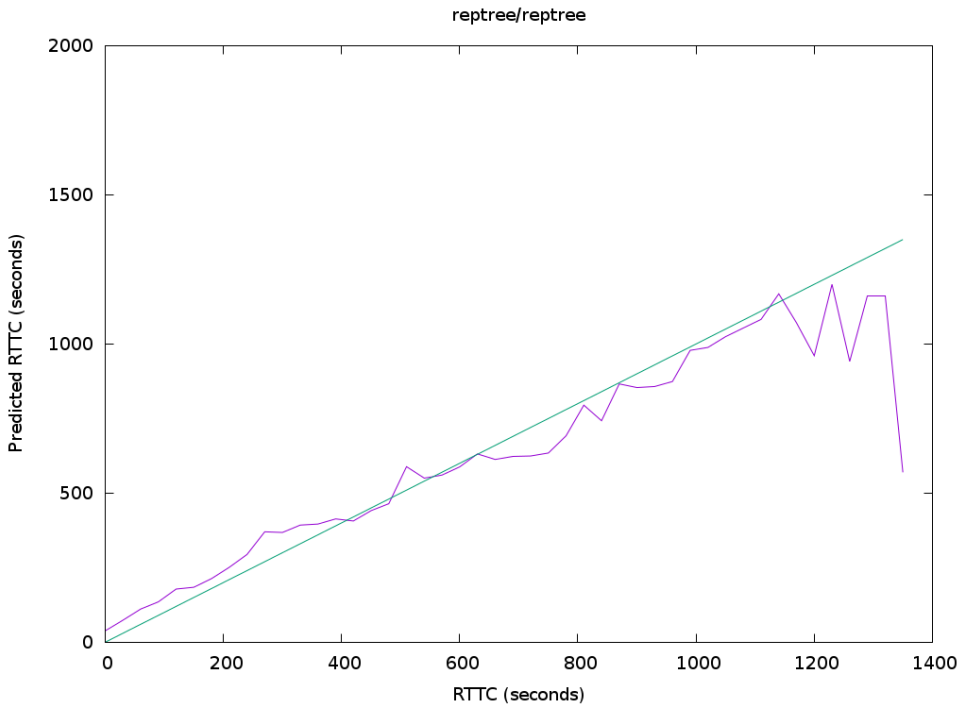


Figure 62 Model fitted using REPTree.

4.4.1.6 Benefits from the Integration

In this section, we discuss what are the benefits on application availability in case of applications built on top of Hadoop when integrated with the ACM Framework.

ACM, by relying on ML-based prediction models, is able to detect that an application in Hadoop 2.x is about to crash due to memory leaks (as well, ACM can make prediction for another anomalies i.e. unterminated threads, etc.. .) In the process of integration, an integration middleware can be designed. This middleware can be notified by ACM about what is the about-to-fail node (VM – Worker.) Therefore, the middleware can ask to the *Hadoop Resource Manager* to decommission it. Therefore, no new application or task of actual running applications will be scheduled to that node. At this point ACM can provide a new VM (worker node). By using the middleware, the new node is assigned to Hadoop and it can start assigning jobs to it. Therefore, we are able to provide a new VM (worker node) to be added to an existing Hadoop cluster. If a running application has a subtask mapped to the node, which has been rejuvenated, the application can simply remap the task to the Hadoop cluster again. In fact, the Hadoop cluster has already received a new (rejuvenated) worker node.

The obtained results of the Integration of Hadoop with ACM System for generating ML prediction models can significantly contribute for increasing availability of applications running in Hadoop 2.x.

4.4.2 Metrics validation

Innovation	Technical metrics	Validation
Proactive Service Management Using Machine Learning	• Improvement in availability offered by the PANACEA innovation compared to an unmanaged service. • Prediction buffer time: time interval between a reliable anomaly prediction is made and when it effectively occurs. • Number of parameters that need to be monitored. • False positive and false negative ratios	We were not yet able to validate the improvement in availability, but we were able to build the prediction model for Use Case 2. Our results show that the number of parameters that need to be monitored can be as low as 4. They also show that the prediction buffer time is large enough to reconfigure the system.

Routing Overlay	• Typical network metrics: throughput, packet latency and loss rate. • Path discovery time, i.e., the time it takes to find a new optimal path between two clouds, • Network overhead of the overlay system due to monitoring and control.	• Most of the times, the IP route was optimal and we obtained the same transfer time with and without SMART. But, when SMART was able to find a route with more bandwidth, we observed an improvement of nearly 23% with respect to native IP routing. • The per-packet overhead is 40 Bytes (20 Bytes for SMART Header and 20 Bytes for UDP tunnel). The extra time per packet was measured to be less than 3ms.
Open Nebula	• Cloud manager •	• Achieve the deployment of VMs and software component over hybrid cloud environments • Enable the deployment of threshold-based auto-scaling policies.

Table 10: USE CASE 2 : Validation of PANACEA innovations metrics.

5 CONCLUSION

PANACEA-enabled services are able to manage themselves without direct human intervention, recovering from many inevitable anomalies and autonomously optimizing their performance in changing conditions. The main expected impact of these new capabilities is improved availability and QoS of cloud services, as well as reduced OPEX since autonomic services will require less supervision and manual reconfiguration and intervention.

In this deliverable, we have evaluated to what extend PANACEA objectives have been reached using two use cases, related to cloud web hosting and to big data analytics. We have presented results for the individual validation of PANACEA components as well as results for the validation of the global PANACEA solution, evaluating a number of technical metrics to measure the impact of PANACEA.

Although we were unfortunately not able to run all components in a single experiment, our results show that these components can work together. They show that:

- The Machine Learning Framework developed in PANACEA empowers cloud services with self-* (healing, optimizing and configuring) properties. The self-healing property is guaranteed by timely forcing the system to an anomaly-free state, relying on the self-rejuvenation capabilities of the system; The self-optimizing property is guaranteed by ensuring a response time of the system below an acceptable threshold, even in the case of accumulation of anomalies; Finally, the self-configuring property is guaranteed by allowing an automatic reconfiguration of the system, also when couples of VMs are added/removed to/from the managed pool of virtual resources.

- The measurement-based decision algorithms (Sensible routing and Random Neural Network-based Reinforcement Learning) of the autonomic task allocation platform allows a significant reduction of response times with respect to well-known static allocation algorithms (e.g., round-robin or probabilistic " equal loading" scheme).

- The autonomic routing overlay allows in some cases significant improvements of latency and bandwidth over native IP routing. This was shown both with offline experiments with real data collected over the Internet and with live experiments using Amazon data centres in Use Cases 1 and 2. The Frappe coordination service allows simplifying and automating the deployment of the SMART routing overlay, while easing the achievement of high availability.

- The emulation/simulation environment allows the offline planning of the deployment of new services in multi-cloud scenarios. In our experiments, this environment was used to provide additional validation results for the autonomic routing overlay.

- The cloud management solution of PANACEA eases the deployment of services in hybrid cloud scenarios, and enables services to autonomously grow and shrink resources allocated to them as and when needed, as demonstrated by our work on the autonomic elasticity of Hadoop in Use Case 2.

REFERENCES

[1] Zack Whittaker. *Amazon Web Services Suffers Partial Outage* http://www.zdnet.com/article/amazon-web-services-suffers-partial-outage/

[2] *10 Devastating Outages and Failures of Major Brands in 2011* http://www.evolven.com/blog/2011-devastating-outages-major-brands.html

[3] D. Avresky, O. Brun, E. Dekel, D. Garcia Perez, G. Gorbil, E. Huedo and A. Rachdi, "D4.1: Description of Feasible Use Cases", public deliverable of FP7 PANACEA project, February 2015.

[4] I. Febles, "D5.1: Market Analysis, Business Models and Value Chain", public deliverable of FP7 PANACEA project, February 2015

[5] A. Al Sheikh, A. Rachdi "D4.2: Overlay Simulation Environment", public deliverable of FP7 PANACEA project, March 2015.

[6] Robert H. Zakon. "Hobbes' Internet Timeline 10.1". Retrieved September 16, 2011.

[7] "Abilene Network" (PDF). Internet2. February 2, 2005. Retrieved September 16, 2011.

[8] PANACEA Consortium "D4.1: Use cases definition", public deliverable of the FP7 PANACEA project. March 2015 (2nd edition).

[9] Apache Hadoop. http://hadoop.apache.org/, last visited on March 15th 2016.

[10] Amazon Elastic Cloud Computing (EC2) https://aws.amazon.com/ec2 , last visited on March 15th de 2016.

[11] OpenNebula 4.14 'Great A' Tuin' http://opennebula.org/opennebula-4-14-great-atuin-is-out/ , last visited on March 15th 2016.

[12] OpenNebula modifications for PANACEA project: https://github.com/dsa-research/opennebula-panacea . Last visited on March 15th 2016.

[13] Oracle Glassfish: http://www.oracle.com/technetwork/middleware/glassfish/overview/index.html . Last visited on March 15th 2016.

[14] MySQL: https://www.mysql.com/ . Last visited on March 15th 2016.

[15] Ubuntu 14.04 Long Term Support (LTS): http://releases.ubuntu.com/14.04/ . Last visited on March 15th 2016.

[16] Java Standard Edition (SE): http://www.oracle.com/technetwork/java/javase/overview/ index.html . Last visited on March 15th 2016.

[17] Apache YARN: https://hadoop.apache.org/docs/r2.7.1/hadoop-yarn/hadoop-yarn-site/YARN.html. Last visited on March 15th 2016.

[18] Hadoop File System (HDFS) https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html . Last visited on March 15th 2016.

[19] CollectD – The system statistics collection daemon: https://collectd.org/ . Last visited on March 15th 2016.

[20] MySQL Java Connector: https://dev.mysql.com/downloads/connector/j/ . Last visited on March 15th 2016.

[21] Hadoop Metrics: https://hadoop.apache.org/docs/r2.7.2/hadoop-project-dist/hadoop-common/Metrics.html . Last visted on March 15th 2016.

[22] CollectD exec plugin: https://collectd.org/wiki/index.php/Plugin:Exec . Last visited on March 15th 2016.

[23] Determine YARN and MapReduce Memory Configuration Settingshttp://docs.hortonworks.com/HDPDocuments/HDP2/HDP-2.0.6.0/bk_installing_manually_book/content/rpm-chap1-11.html , Last visited on March 15th 2016.

[24] SLA Policies for autoscaling Hadoop clusters. http://blog.sequenceiq.com/blog/2014/09/01/sla-samples-periscope/ . Last seen on March 15th 2016.

[25] D. Avresky, D. Garcia Perez, "Deliverable 3.1: 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" , FP7 PANACEA, January 2014.

[26] D. Avresky, "Deliverable 3.2: Machine Learning Framework and Global Architecture for Proactive Management" , PANACEA, December 2014.

[27] S. Pertet and P. Narasimhan, "Causes of Failure in Web Applications" , Carnegie Mellon University, Tech. Rep. CMU-PDL-05-109, 2005.

[28] P. Di Sanzo, A. Pellegrini, and D. R. Avresky, "Machine Learning for Achieving Self-* Properties and Seamless Execution of Applications in the Cloud," in Proceedings of the Fourth IEEE Symposium on Network Cloud Computing and Applications, ser. NCCA. IEEE Computer Society, 2015.

[29] D. Cotroneo, R. Natella, R. Pietrantuono, and S. Russo, "Software aging and rejuvenation: Where we are and where we are going," in Proceedings - 2011 3rd International Workshop on Software Aging and Rejuvenation, WoSAR 2011, 2011, pp. 1–6.

[30] A. Pellegrini, P. Di Sanzo, and D. R. Avresky, "A Machine Learning based Framework for Building Application Failure Prediction Models" , in Proceedings of the 20th IEEE Workshop on Dependable Parallel, Distributed and Network-Centric Systems, ser. DPDNS. IEEE Computer Society, 2015.

[31] H. Zhang, G. Jiang, K. Yoshihira, and H. Chen, "Proactive workload management in hybrid cloud computing," Network and Service Management, IEEE Transactions on, vol. 11, no. 1, pp. 90–100, March 2014.

[32] U. Ayesta, O. Brun, H. Hassan, B.J. Prabhu, "D2.3: Autonomic Communication Overlay" , PANACEA, March 2015.

[33] P. Auer, N. Cesa-Bianchi, Y. Freund, and R. Schapire, "The nonstochastic multi-armed bandit problem," SIAM Journal on Computing, vol. 32, no. 1, pp. 48–77, 2002.

[34] E. Gelenbe, R. Lent, A. Montuori, and Z. Xu, "Towards networks with cognitive packets," in Proc. 8th Int. Symp. Modeling, Analysis and Simulation of Computer and Telecommunication Systems (IEEE MASCOTS), San Francisco, CA, USA, August 29-September 1 2000, pp. pp 3–12.

[35] E. Gelenbe and Z. Kazhmaganbetova, "Cognitive packet network for bilateral asymmetric connections," IEEE Trans. Industrial Informatics, vol. 10, no. 3, pp. 1717–1725, 2014.

[36] O. Brun, L. Wang, and E. Gelenbe, "Big data for autonomic intercontinental overlays," to appear in IEEE Jour. Selected Areas in Communications (special Issue on Emerging Technologies in Communications - Big data), 2016.

[37] D.Avresky, Deliverable 3.3: Machine Learning solution for Intra and Inter Autonomic Cloud Managers (ACMs) for a proactive management of cloud resources" FP7 PANACEA 2015-07-24.

APPENDIX A – ONEFLOW CLUSTER DESCRIPTION

The following code snippet represents the Hadoop service in OpenNebula OneFlow format:

```
{
  "name": "Panacea Hadoop Cluster",
  "deployment": "straight",
  "description": "",
  "roles": [
    {
      "name": "Worker",
      "cardinality": 3,
      "vm_template": 375,
      "elasticity_policies": [

      ],
      "scheduled_policies": [

      ]
    },
    {
      "name": "NameNode",
      "cardinality": 1,
      "vm_template": 373,
      "parents": [
        "Worker"
      ],
      "min_vms": 1,
      "max_vms": 1,
      "elasticity_policies": [

      ],
      "scheduled_policies": [

      ]
    },
    {
      "name": "ResourceManager",
      "cardinality": 1,
      "vm_template": 372,
      "parents": [
        "Worker"
      ],
      "min_vms": 1,
      "max_vms": 1,
      "elasticity_policies": [

      ],
      "scheduled_policies": [

      ]
    },
    {
      "name": "CollectD",
      "cardinality": 1,
      "vm_template": 374,
      "parents": [
        "Worker"
      ],
      "min_vms": 1,
      "max_vms": 1,
      "elasticity_policies": [

      ],
      "scheduled_policies": [

      ]
    }
  ],
  "ready_status_gate": false
}
```

The following code snippet represents the OneFlow Service deployment in an inter-cloud

environment employing the overlay network functionalities:

```
{
  "name": "EC2 Panacea Elasticity Hadoop Cluster",
  "deployment": "straight",
  "description": "",
  "roles": [
    {
      "name": "Worker",
      "cardinality": WORKER_CARDINALITY,
      "vm_template": WORKER_TEMPLATE_ID,
      "parents": [
        "SmartProxyVirginia",
        "SmartProxyIreland"
      ],
      "elasticity_policies": [

      ],
      "scheduled_policies": [

      ]
    },
    {
      "name": "NameNode",
      "cardinality": 1,
      "vm_template": NAMENODE_TEMPLATE_ID,
      "parents": [
        "Worker"
      ],
      "min_vms": 1,
      "max_vms": 1,
      "elasticity_policies": [

      ],
      "scheduled_policies": [

      ]
    },
    {
      "name": "ResourceManager",
      "cardinality": 1,
      "vm_template": RESOURCEMANAGER_TEMPLATE_ID,
      "parents": [
        "Worker"
      ],
      "min_vms": 1,
      "max_vms": 1,
      "elasticity_policies": [

      ],
      "scheduled_policies": [

      ]
    },
    {
      "name": "CollectD",
      "cardinality": 1,
      "vm_template": COLLECTD_TEMPLATE_ID,
      "parents": [
        "Worker"
      ],
      "min_vms": 1,
      "max_vms": 1,
      "elasticity_policies": [

      ],
      "scheduled_policies": [

      ]
    },
    {
       "name": "SmartProxyVirginia",
       "cardinality": 1,
       "vm_template": SMART_TEMPLATE_ID_VIRGINIA,
       "min_vms": 1,
       "max_vms": 1,
       "elasticity_policies": [
```

```
      ],
      "scheduled_policies": [

      ]
   },
   {
      "name": "SmartProxyIreland",
      "cardinality": 1,
      "vm_template": SMART_TEMPLATE_ID_IRELAND,
      "parents": [
         "SmartProxyVirginia"
      ],
      "min_vms": 1,
      "max_vms": 1,
      "elasticity_policies": [

      ],
      "scheduled_policies": [

      ]
   },
   {
      "name": "SmartProxyTokyo",
      "cardinality": 1,
      "vm_template": SMART_TEMPLATE_ID_TOKYO,
      "parents": [
         "SmartProxyVirginia"
      ],
      "min_vms": 1,
      "max_vms": 1,
      "elasticity_policies": [

      ],
      "scheduled_policies": [

      ]
   },
   {
      "name": "SmartProxySydney",
      "cardinality": 1,
      "vm_template": SMART_TEMPLATE_ID_SYDNEY,
      "parents": [
         "SmartProxyVirginia"
      ],
      "min_vms": 1,
      "max_vms": 1,
      "elasticity_policies": [

      ],
      "scheduled_policies": [

      ]
   },
   {
      "name": "SmartProxyOregon",
      "cardinality": 1,
      "vm_template": SMART_TEMPLATE_ID_OREGON,
      "parents": [
         "SmartProxyVirginia"
      ],
      "min_vms": 1,
      "max_vms": 1,
      "elasticity_policies": [

      ],
      "scheduled_policies": [

      ]
   },
   {
      "name": "SmartProxyCalifornia",
      "cardinality": 1,
      "vm_template": SMART_TEMPLATE_ID_CALIFORNIA,
       "parents": [
         "SmartProxyVirginia"
      ],
      "min_vms": 1,
```

```
        "max_vms": 1,
        "elasticity_policies": [

        ],
        "scheduled_policies": [

        ]
    },
    {
        "name": "SmartProxySingapore",
        "cardinality": 1,
        "vm_template": SMART_TEMPLATE_ID_SINGAPORE,
      "parents": [
          "SmartProxyVirginia"
        ],
        "min_vms": 1,
        "max_vms": 1,
        "elasticity_policies": [

        ],
        "scheduled_policies": [
        ]
    },
     {
        "name": "SmartProxySaoPaulo",
        "cardinality": 1,
        "vm_template": SMART_TEMPLATE_ID_SAOPAULO,
        "parents": [
           "SmartProxyVirginia"
        ],
        "min_vms": 1,
        "max_vms": 1,
        "elasticity_policies": [

        ],
        "scheduled_policies": [

        ]
    }
  ],
  "ready_status_gate": false
}
```