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

PANACEA

Proactive Autonomic Management of Cloud Resources

Deliverable 3.3: Machine Learning solution for Intra and Inter Autonomic Cloud Managers (ACMs) for a proactive management of cloud resources

Version: v.1.0

Work package	WP 3
Task	Task 3.3
Due date	30/06/2015
Submission date	30/06/2015
Deliverable lead	IRIANC
Version	1.0
Authors	Dimiter R. Avresky
Reviewers	Michel Diaz, Eliezer Deckel

Abstract	The framework for Intra/Inter ACM for proactive cloud resources management is presented. By relying on it, cloud providers and application developers can deploy their application in a highly available fashion, with great ease, as well on a hybrid cloud infrastructure. Cloud regions can be even geographically distributed. The system is able to monitor at runtime what are the current system dynamics: if a given cloud region is overloaded, then the requests issued by remote users of the application can be redirected to different regions. The availability of the application is increased, the response time is kept below a given threshold.
Keywords	Machine Learning, Intra/Inter Autonomic Cloud Managers, proactive management

Document Revision History

Version	Date	Description of change	List of contributor(s)
V1.0	30.06.2015	First version of the Deliverable	Dimiter R. Avresky (IRIANC)

Disclaimer

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**	
RE	**Restricted to bodies determined by the PANACEA project**	
CO	**Confidential to PANACEA project and Commission Services**	

EXECUTIVE SUMMARY

In this Deliverable we present the framework for Intra/Inter ACM for proactive cloud resources management. This framework grounds on the ML prediction models which have already been presented in Deliverable 3.1, and the rejuvenation protocol which has been thoroughly discussed in Deliverable 3.2.

By relying on the framework for Intra/Inter ACM for proactive cloud resources management, cloud providers and application developers can deploy their application in a highly available fashion, with great ease. In particular, the application can be deployed as well on a hybrid cloud infrastructure, where cloud regions (namely, homogeneous sets of virtual machines hosted by a given cloud provider in a given location) can be several and even geographically distributed.

Our solution exposes several entry points for remote clients to connect to the application replicated on a distributed environment . Nevertheless, the system is able to monitor at runtime what are the current system dynamics: if a given cloud region is overloaded, then the requests issued by remote users of the application can be redirected to different regions. In this way, we are able to increase the availability of the application, to keep the response time below a given threshold, and to reduce the rejuvenation frequency of VMs composing the distributed virtualized environment.

TABLE OF CONTENTS

LIST OF FIGURES

ABBREVIATIONS

AM	Autonomic Manager
ACM	Autonomic Cloud Manager
ML	Machine Learning
VM	Virtual Machine
RT	Response Time
M5P	Rational Reconstruction of M5
SVM	Support-Vector Machine
SVM2	Support-Vector Machine to the power of 2
REPTree	Replication Tree
RTTF	Remaining Time to Failure

1 INTRODUCTION

The ML framework for rejuvenation has been extended in order to support the management of multiple, geographically distributed, cloud providers.

In particular, each homogeneous set of VMs hosted in a given geographical location by the same cloud provider is referred to throughout this document as a *cloud region*. Cloud regions can be either public or private. The interconnection between private cloud regions and public cloud regions allow to deploy the distributed application as well on hybrid cloud infrastructures. This is an important feature, considering that current trends [1] show that 51% of current cloud infrastructures are organized as hybrid cloud federations.

We have extended the ML framework in order to support the coordination between differentiated controllers hosted in different regions. This coordination is managed in a reliable way (explicitly relying on overlay networks) so that both the communication between the clients and the redirected connections from remote users can experience high availability and extremely reduced latency.

In our system model, remote users can connect to any cloud region. This mimics the scenario where application clients can select an entry point either relying on a list of IP addresses hard coded within the application, or can retrieve the IP address of a server through a separate directory service, such as DNS. Again, this gives high freedom of configuration/usage to our solution.

The extended framework allows at the same time to carry out differentiated tasks at various levels:

1. Intra ACM is handled at the single cloud region level. In particular, each cloud region is equipped with a controller and a load balancer. The controller monitors the state of VMs within the cloud region. If the RTTF of a VM approaches the configured threshold, the VM is rejuvenated and another VM is activated to take its place. This is done exactly according to the results in deliverables 3.1 [24] and 3.2 [25].
2. Inter-Autonomic management is handled by having the cloud region controllers exchange periodically state information about the controlled region. One controller is elected as the leader. This leader controller determines, on a global scale, what is the (current) best policy to redirect the incoming connections from remote users of the application to all the available cloud regions. In this way, the rejuvenation rate of all the cloud regions is levelled, which allows reducing the total number of rejuvenation operations, thus enforcing higher availability.
3. We additionally increase the control power offered by the controllers at the Intra-Autonomic level. In particular, each controller monitors as well the Mean Time to Failure (MTTF) of all the virtual machines which are in the region pool. If the MTTF goes below a specified threshold, the controller is able to activate additional VMs which join the pool. In its turn, this solution allows to reduce the rejuvenation rate of the cloud region, which enforces higher availability of the system.

We additionally enforce high availability by considering the possibility that cloud region controllers might fail. In particular, we rely on an efficient leader election algorithm, which is able to detect whether a controller has failed, and promptly elect a new leader controller.

The Distributed Architecture for Proactive Rejuvenation management implements an autonomic system [2] which relies on an Autonomic Manager (AM) to detect whether any VM in the distributed system is about to crash and enforce prompt operations to reduce at most the effect of this crash on the users of the system. At the same time, this autonomic system strives for increasing the overall availability of the system, which is deployed on a geographically-distributed (hybrid) cloud environment, and to reduce the response time as seen by the users. The overall general architecture of the elements of the proactive system is given in Figure 1.

It is interesting to emphasize that the same architecture described in Figure 1 is used in a threefold nature by our deliverable(s):

a) Intelligent collection of Training Data and usage of Machine Learning algorithms for creating prediction models for a proactive management of Intra cloud resources.

b) Intra ACM, using Training Data and creating prediction models by Machine Learning to determine rejuvenation instant for VMs within a single cloud region.

c) Inter ACM, to interconnect different cloud regions and redistributed the incoming workload to increase availability, keep response time below a given threshold, and to reduce rejuvenation frequency.

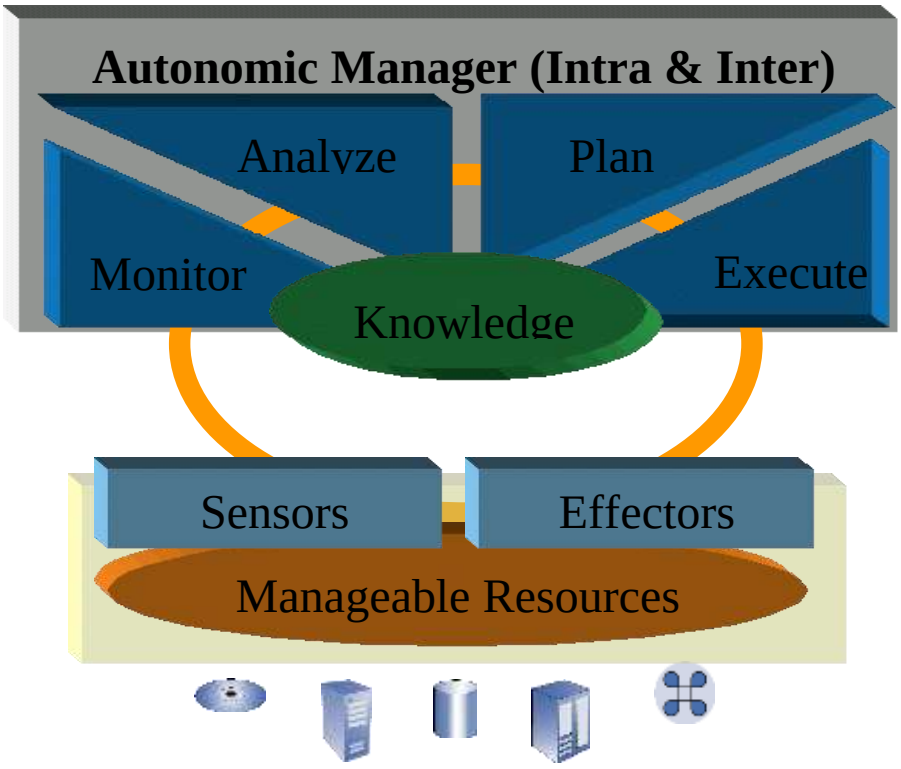

Figure 1: Architecture Elements: Autonomic Manager, Manageable Resources and Knowledge Resources

The same architecture in Figure 1, implemented within the Local Controller, is used in a twofold nature, namely an On-Line and an Off-Line mode. These modes of operations of the Inter/Intra–ACMs are presented for the realization of a control loop in their operations and Proactive Management of cloud resources. ML Framework & Global Architecture for Proactive Management are defined and the basic modules are implemented. Intra-ACMs forming Private Clouds and communicating via Inter-ACMs are capable of creating hybrid Clouds

As already mentioned in Deliverable 3.2 [25] (and summarized in Section 2), the Knowledge base is created from Training Data Base using the ML Framework, which generates ML prediction models. The chain of Manageable Resources (VMs installed in HP servers or geographically via Internet), Sensors and Monitor are used to construct the Knowledge Resources. Each Intra-ACM collects the features (physical parameters during the application execution), builds Data Base and utilizes the ML Framework or sends them to the Inter-ACM for forming the Knowledge Resources. The ML Framework is activated in order to analyse the Training Data Base and produce the Prediction Models for proactive management of cloud resources controlled by Intra-ACM/Inter-ACM (Plan module.) Dynamic Reconfiguration Flow Diagram based on ML prediction is activated by the Execute and Effectors. In this way, the control loop is closed and the Manageable Resources again are censored and monitored for performing proactive autonomic management of cloud resources. Based on the presented features of the Autonomic Manager (Inter/Intra) control loop the proactive management of the cloud resources can be accomplished and Self-rejuvenation property of autonomic self* management can be achieved.

The specified modules in Figure 1 realize different functions in each Off-Line or On-Line mode of operations of ACMs.

There are two levels for creating ACMs:

1) The first level (OFF-Line ACM) activates *Sensors* and *Monitor* modules to collect the features by Feature Monitor Client under anomalies (memory leaks and unterminated threads) in the Manageable Resources. Training Data base is created in the unit *Analyze*. Important decision are made by the module *Plan*. For example, determining the Remaining Time to Crash (RTTF) and Threshold for the Response time of web servers or Execution time of cloud applications, Web applications Availability requirements. The *Execute* segment determines the set of Machine Learning Algorithms that will be used for creating the ML Prediction Models and techniques for reducing the number of monitoring parameters (Lasso regularization), tools for automatically generating prediction models and defining the Machine Learning Framework. The pool of (joining and leaving) VMs is created and their state (active, standby) is determined in the *Effectors* unit. The unit "Effectors" starts the operation of ML Framework & Global Architecture for Proactive Autonomic Management.
2) In the second Level (On-Line ACM) *Sensors and Monitor* modules collect features (parameters) under injected anomalies from the Manageable Resources. There are two options: All parameters collected in Off-Line level or Reduced Number of Parameters determined by Lasso techniques. The module *Analyse* evaluates the RTTF of Web servers. An important function of *PLAN* unit is to compute the difference between the run time predicted RTTF and the selected interval for safe rejuvenation of

corresponding active VM. The same steps are taking by the *PLAN* unit for run time predicted threshold of the response (execution) time. The generated signal by PLAN unit will allow *Execute* to activate *Effectors* module. In this case, *Effectors* module triggers the safe rejuvenation step in the Dynamic Rejuvenation Flow diagram. If this signal is not generated then the control loop continues through all steps until the *Effectors* module is activated.

In our experimentation of the Intra/Inter ACM, we will rely on a geographically distributed hybrid cloud (which can be also regarded as a federation of different clouds, although throughout this document we will refer to it as a hybrid cloud). Each location will be referred to as a *cloud region*. Three different locations are used: location 1 is Amazon in Ireland, location 2 is Amazon in Frankfurt, location 3 is Munich on a private HP ProLiant server.

On each location there are, at the beginning, 3 couples of VMs, each one subject to a different kind of anomaly for generating different ML prediction models (memory leaks only, unterminated threads only, memory leaks plus unterminated threads).

We have used the ML Framework described in Deliverable 3.1 [24] to generate prediction models for all three regions, under the different kinds of anomalies – see Section 2 for a short summary on the generation of ML prediction models.

As the deployed application, we have used the Java version of the TPC-W standard benchmark application [3], [4].

The proposed solution allows enforcing high scalability of the deployed application. In fact, scalability can be controlled at different levels:

- Intra-ACM controllers can decide upon the number of VMs which are currently active. By controlling at runtime the execution dynamics of VMs into a cloud region (specifically, by controlling the Mean Time to Failure), the controller is able to make a prediction on the number of VMs that are necessary so as to keep the rejuvenation rate below a certain threshold. Considering that the rejuvenation rate is directly proportional to the current workload seen by the system, the controller can decide whether new VMs are required to fulfil the requests coming to the region. The new VMs will contribute for reducing the workload among the nodes (VMs) in a cloud region. In this way, the Intra-ACM controller is able to proactively scale up/down the number of VMs composing the region.
- Multiple cloud regions could be connected together, even at runtime. This gives the user of the framework an additional level of scalability. In fact, rather than adding single VMs to the system, entire regions, composed of any number of VMs, could be connected. This allows, for example, injection a higher amount of computing power, which could be possibly required only during certain operating periods of the system.

Overall, our solution gives great freedom in the composition of the system, and enforces autonomic and proactive scalability properties, with minor (or null) manual intervention.

2 ML-BASED PREDICTION MODEL FRAMEWORK

For the sake of clarity, we summarize here some of the results from Deliverable 3.2 [25]. The extensions to them will be described in the upcoming sections. The overall Intra ACM system is based on the ML Framework initially presented in Deliverable 3.1 [24], which is shown in Figure 2.

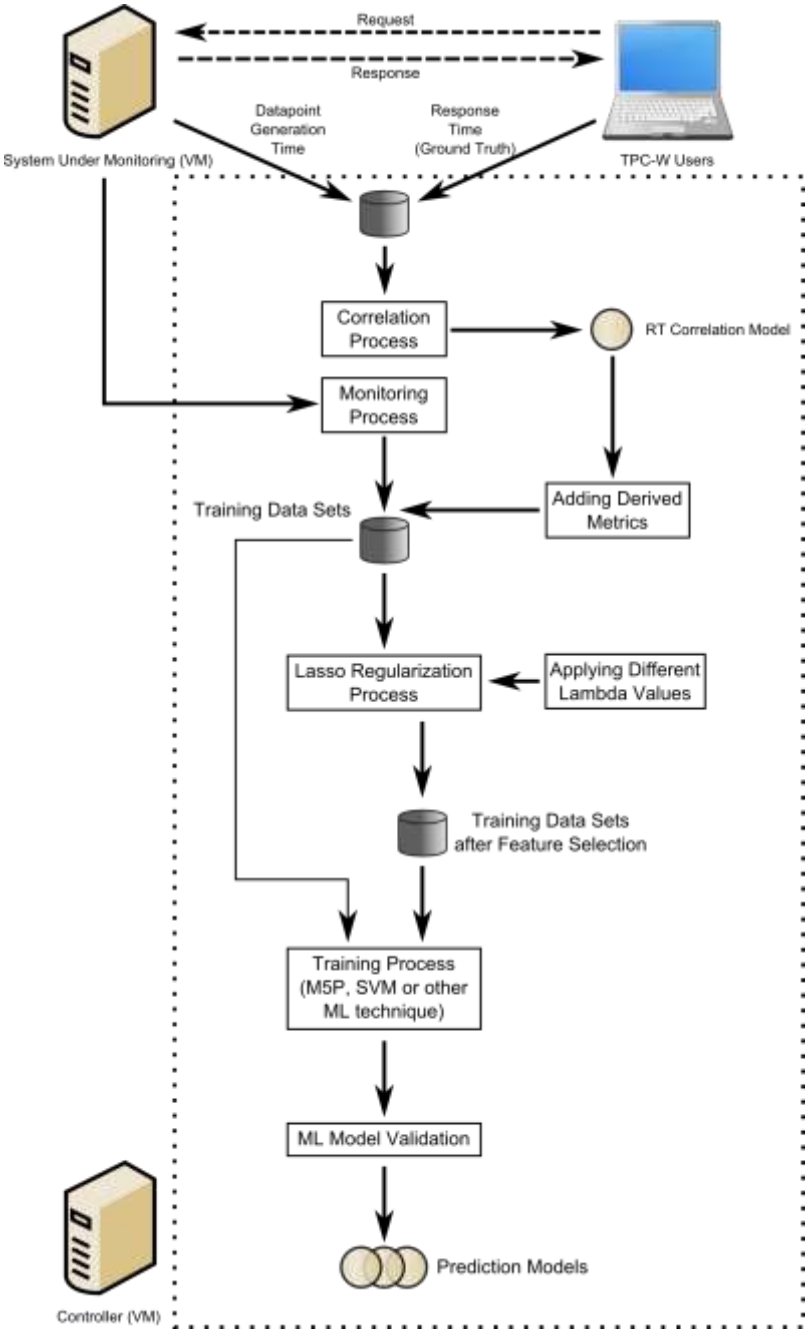

Figure 2: Machine Learning Framework

The ML Framework explicitly takes into account the Response Time, as seen by the end users, of any virtualized web application hosted on the machine under monitoring. During the offline learning process, we rely on a set of clients to generate the workload on the server. In this specific scenario, since clients are actually controlled by the system administrator carrying on the training process, it is possible to measure the actual response time as experienced by the clients.

2.1 Data Collection

As mentioned in Deliverable 3.1 [24], the initial step to build prediction models is to collect runtime data from the system under monitoring.

To build the initial database file (Raw Knowledge Base), namely the input file to the ML Framework, two different anomalies can be injected in monitored system to mimic a software environment which eventually becomes faulty, namely *memory leaks* and *unterminated threads*. To correctly mimic the behaviour of a system affected by software errors, anomalies are injected according to statistical distributions.

We rely on a client-server architecture to collect training data. The VM which is subject to the accumulation of anomalies runs the Feature Monitor Client (FMC), which periodically measures hardware parameters which can be specified by the user – namely, the system administrator configuring the architecture.

FMC sends the measured parameters to the Feature Monitor Server (FMS), which could be possibly installed in a separated VM. FMS stores the received datapoints, timestamped with the generation time, on a database file which is then used to generate the prediction model. We note that there is no a-priori suitable size of this database file, to generate accurate prediction models. In fact, the architecture of FMC/FMS allows to *incrementally* grow the size of this database file (generating multiple times the prediction models, from subsets of the datapoints), until the system administrator is confident that the achieved accuracy is suitable for the monitored application. To this end, the ML Framework provides a large set of indicators to determine what the obtained accuracy is. These indicators will be described in Section 2.3.

2.2 Steps to Generate a Prediction Model

The Machine Learning Framework is based on a set of utilities, which mostly just require a standard C compiler on a POSIX system, with `pthreads`.

Starting from the database file generated by FMC/FMS, each input datapoint (which is timestamped) keeps some information about the current state of the system. The Generation Time information is used to virtually place the datapoints on the original generation time axis. Then, a time window of size equal to the threshold is placed starting at $T_0 = 0$. All the original datapoints falling in this time window are used to generate an aggregated datapoint, meaning that all the values of the original datapoints are averaged in the aggregated datapoint.

Then, the aggregated datapoints are augmented with a set of derived metrics to be used as features for model building, namely slopes and Generation Time. Then, for each of the original measurements, slopes are computed according to the following formula:

$$slope = \frac{start - end}{n} \quad (1)$$

where *start* and *end* are the values (for each measurement) of the first and last original datapoint falling in the current time window.

To determine what are the most important parameters to build prediction models, we apply Lasso regularization [5]. This is of fundamental importance in case the initial set of parameters is very large, so that both the training time and the intrusiveness of monitoring while the application is deployed are reduced. We refer to Deliverable 3.1 [24] for a thorough description of this step.

We then rely on WEKA [6] to generate differentiated prediction models. In particular, the ML Framework generates out of the box prediction models according to the following regression techniques:

- Lasso as a predictor [5]
- Linear regression [7]
- M5P [8]
- REP Tree [9]
- SVM [10]
- LS-SVM [11]

2.3 Indicators to determine the accuracy of prediction models

The ML Framework output is complemented by a set of metrics which allow the system administrator to determine whether the accuracy of the generated prediction models is suitable for the current needs of the deployed applications. The following indicators are provided.

Maximum Absolute Prediction Error (MAPE)

This error represents the highest error (worst case) encountered during the prediction. It is expressed in seconds.

Relative Absolute Prediction Error (RAPE)

The relative absolute E_i error is relative to a simple predictor, which is just the average of the actual values. In this case the error is just the total absolute error instead of the total squared error. Thus, the relative absolute error takes the total absolute error and normalizes it by dividing by the total absolute error of the simple predictor.

$$E_i = \frac{\sum_{j=1}^{n}|P_j - T_j|}{\sum_{j=1}^{n}|T_j - \overline{T}|} \quad (2)$$

where P_j is the value predicted for sample j (out of n samples), $T_j$is the target (i.e., ground truth) value for sample j, and:

$$\overline{T} = \frac{1}{n}\sum_{j=1}^{n} T_j \quad (3)$$

It is expressed as a percentage of the total absolute error of the simple predictor.

Mean Absolute Error (MAE)

The Mean Absolute Error is the average of the differences between predicted and real remaining time to failure. Specifically, it is calculated as:

$$\frac{1}{n}\sum_{j=1}^{n}|P_j - T_j| \quad (4)$$

It is expressed in seconds.

Soft-Mean Absolute Error (S-MAE)

The Soft-Mean Absolute Error is calculated as the Mean Absolute Error except that when the value $|P_j - T_j|$ is below a given threshold it is assumed to be equal to 0.

3 CONFIGURATION OF ONE COMPUTING NODE

The final goal of this research is to show the effectiveness of a Global Architecture for Proactive Management in a (geographically) distributed environment. We discuss here the software used on each VM implementing a copy of the web server, which is the essential building block of our distributed virtualized architecture. In fact, every computing node in our distributed environment is organized as in Figure 3.

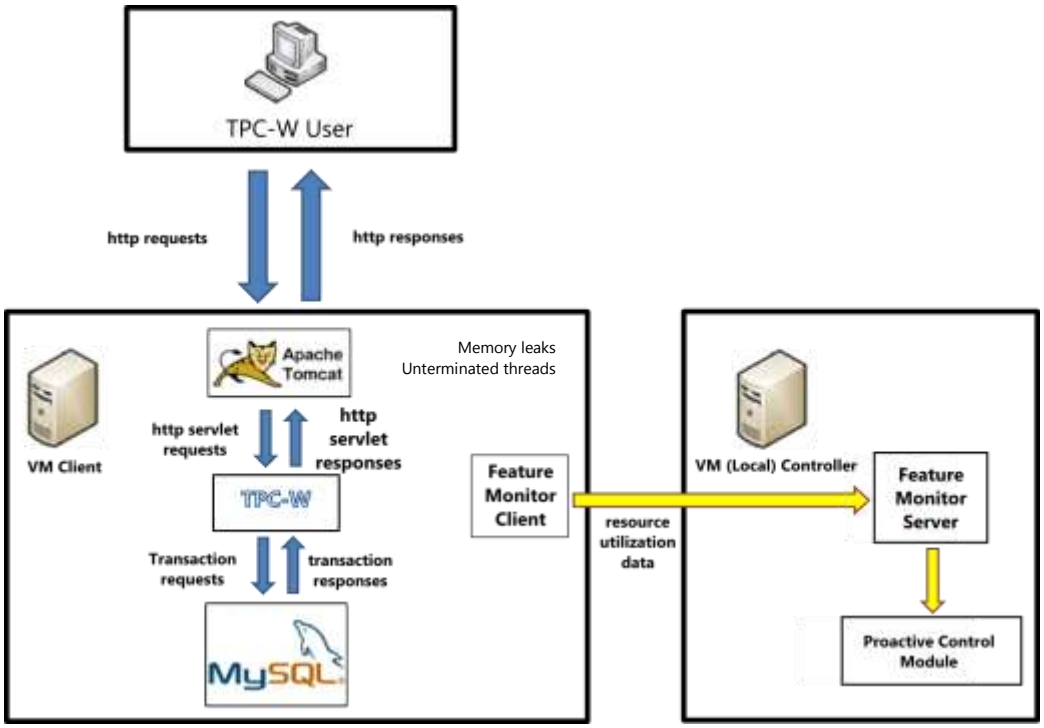

Figure 3: Software Configuration at Node Level and Local Controller

The general node-level architecture is shown in Figure 3. Each web server hosts the TPC-W Web Application standard benchmark [12], which requires several pieces of software to run correctly:

- Apache Tomcat 6.0.41
- A Java implementation of TPC-W [4]
- MySql Server version: 5.1.41-3 [13]
- A *Feature Monitor Client*, which is a proprietary software package aimed at collecting resource usage statistics of the VM Client machine and send it to the Feature Monitor Server

The Java implementation of TPC-W relies on Java HTTP Servlet, a standard for implementing java classes for handling HTTP requests, allowing to manage dynamic contents in a web server through the Java platform. Multiple TPC-W users, implemented as emulated browsers (see Deliverable 3.1 [24] for a complete description of the behaviour of the clients) are used

to simulate the traffic generated by end users. Given the fact that TPC-W Clients can be (geographically) distributed, and provided that their location is of no constraint to the generality of our approach, they are depicted as a single box in Figure 3.

The TPC-W workload configuration parameters are: the number of users; the client average think time (time between the http response and subsequent http request); the workload interaction mix (type and percentage of interactions executed by users). While we have used one single interaction mix for our experimentation (see again Deliverable 3.1 [24]), the number of TPC-W Clients has been varied in between 8 and 128 during the training phase carried out for this deliverable.

Figure 4 reports the RT as seen by the Clients when no anomalies where injected. The plot shows the Inverse Cumulative Function, along with confidence intervals. This function tells (for a given time value on the x axis in seconds) what is the amount (in percentage) of web interactions that take more than that amount of time to complete.

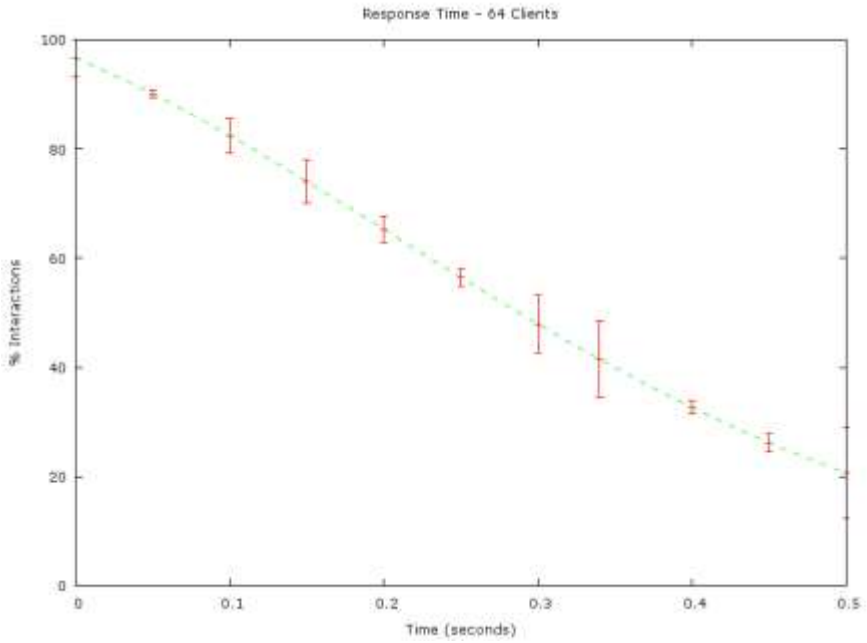

Figure 4: TPC-W Response time with 64 Users

It is important to emphasize that we have selected as the plotting range for the x axis the interval [0, 0.5]. In fact, we consider 0.5 seconds to be still a sustainable response time for application users who are using the web application, since it allows having anyhow timely responses. If the green line converges quickly to zero, the system is not overloaded. Using 64 Clients is motivated by the fact that in this configuration the workload is non-minimal (80% of web interactions require 0.5 seconds or less to complete), yet the system is not yet thrashing, therefore allowing us to capture the dynamics of injected anomalies independently of anomalies created by a too-high workload generated by the application.

The goal of the Feature Monitor Client is similar in spirit to that of the data collection phase. In fact, the Feature Monitor Client collects resource utilization data of the VM client and sends them to the Feature Monitor Server. The Feature Monitor Server is installed on a different virtual machine, where the actual controller (targeted at proactively deciding for the

rejuvenation of VMs) is running.

Within the Proactive Control Module installed on the controller VM, all the prediction models described in Section 2 are available, and upon its startup the user/system administrator is able to select, which ML model should be evaluated upon the receipt of one new set of hardware features. The ML Framework generates automatically a large set of curves for ML prediction models, as shown in Figure 12 and Figure 13. Also, their indicators for a prediction accuracy are accessible in Section 2.3. This information assists the user/system administrator to select the most suitable ML prediction model. In this way, when on VM Client the occurrence of anomalies alters the measured features, VM Controller is able to correctly predict the RTTF of VM Client.

Actually, more than one VM Controller could be present. In fact, given that we are explicitly targeting distributed rejuvenation, theoretically speaking multiple sets of VMs (hosted on different physical hosts) could rely on multiple local controllers. Even more generally speaking, when dealing with overlay networks (where multiple geographically-distributed VMs share a virtualized local network), multiple virtual clusters of VMs could rely on one single (virtually) local controller.

This flexibility of the approach comes directly from the simplicity of the node-level architecture, as described in Figure 3. In fact, since we are only relying on a client/server architecture, it is possible to provide the interaction of VMs with their respective local controller, independently of the physical location of the VM.

Concerning the single hardware host on which VMs are installed (in the case of the distributed environment, multiple *heterogeneous* hardware hosts will be used, distributed in Europe). All virtual machines of the experimental environment are equipped with Ubuntu 10.04 Linux Distribution (kernel version 2.6.32-5-amd64).

All our infrastructure has been developed in C technology [14]. We have specifically relied on multithreading (using POSIX threads) and our algorithms have been synchronized using locks [15]. All protocols have been extensively tested for a long period of time and under different input conditions (workloads and anomalies injections) in one or several regions in the cloud.

4 INTRA-AUTONOMIC MANAGEMENT

The Intra ACM architecture is reported in Figure 5. In its most general form, it includes a VM acting as a controller (VMC) and k couples of VMs (slave VMs) acting as (replicated) servers. The slave VMs of a couple c_x (with $x \in \{1, \dots, k\}$) are named $VM1_x$ and $VM2_x$, respectively. VMC and the slave VMs communicate via message exchange.

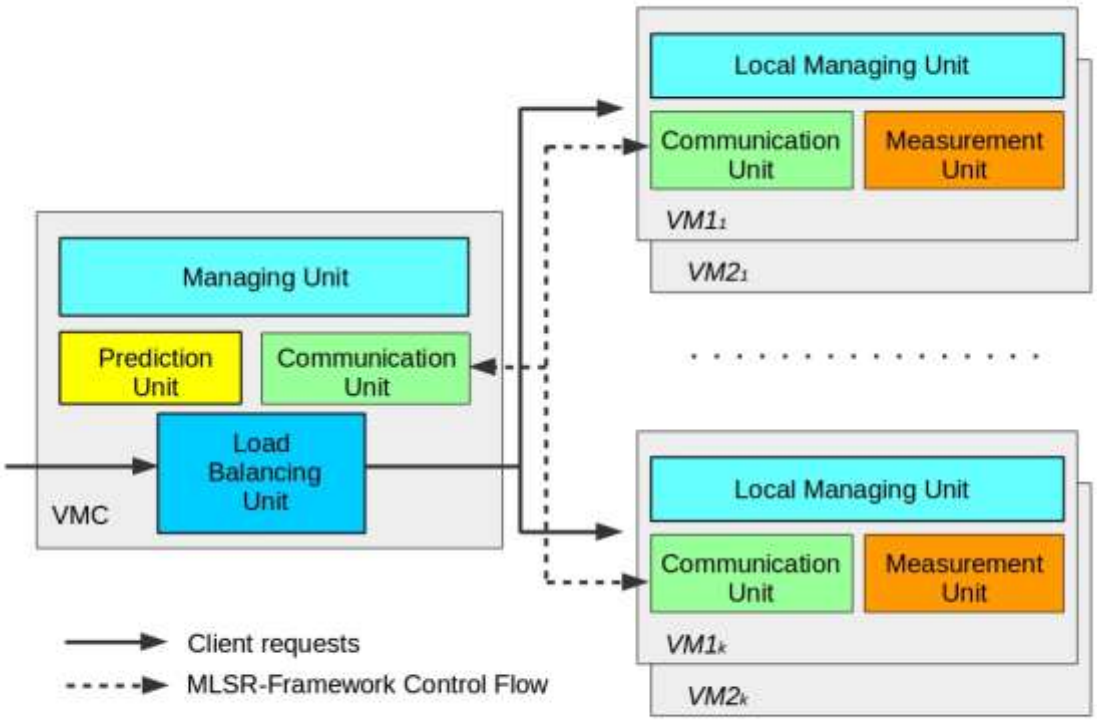

Figure 5: Intra ACM Architecture

On the VMC side, the following components of Intra-Autonomic Autonomic are installed:

- A **Communication Unit** (CU), which is in charge of communicating with all slave VMs;
- A **Prediction Unit** (PU), which provides RTTF predictions of slave VMs;
- A **Load Balancing Unit** (LBU- explained in details in Figure 8), which forwards (remote) clients' requests to slave VMs();
- A **Managing Unit** (MU), which manages the set of VMs, processes incoming messages from VMs and decides when to trigger the rejuvenation of a slave VM.

On all slave VMs, the following components are installed:

- A **Communication Unit** (CU), delegated to communicating with VMC;
- A **Measurement Unit** (MeU), which collects local measurements of the system

features;

- A **Local Managing Unit** (LMU), which sends collected measurements to VMC and receives commands from VMC to start rejuvenating the (local) slave VM.

MU keeps a list of the couples of slave VMs. When the system starts up, the MU activates one slave VM for each couple of slave VMs, then it marks as *active* the activated slave VMs and as *stand-by* the other. The LBU forwards client requests only to the active slave VMs. The pool of VMs can be re-sized at run-time by adding or removing new couples of VMs. The Intra ACM architecture has more cost-effective solution, where a set of Active VMs can utilize lesser number Standby VMs then the Active VMs. This feature is realized in the protocol for implementing the Dynamic Reconfiguration Flow Diagram, shown in Figure 7.

In the Intra ACM architecture, as shown in Figure 5, dashed lines represent information exchanged among VMs and VMC. In particular, they enable VMC to implement the *On-line control loop*, by receiving values of monitored features by VMs, and sending to them the *rejuvenate* command.

Solid lines represent requests coming from remote clients. These requests pass through the LBU, which forwards them to active VM, as shown in details in Figure 8.

4.1 Machine Learning-based RTTF Prediction

As we pointed out, the PU of VMC provides the predicted RTTF of a slave VM as a response to a query executed by the MU. The PU leverages on ML-based models, which are generated by using the prediction models described in Section 2, to predict the RTTF. The ML-Based Prediction Model Framework builds predictions model by exploiting a dataset of system feature measurements collected while monitoring VMs running in the presence of anomalies.

After generating the RTTF prediction models, the ML-Based Prediction Model Framework provides a number of indicators for each model, also including the mean and the relative absolute prediction error. These indicators, which have been thoroughly discussed in Deliverable 3.1 [24], are used to allow the user to select the most accurate model to be exploited for RTTF estimation. The selected model is therefore fed into PU of Intra-Autonomic Manager to be used at run-time.

We have configured the ML-Based Prediction Model Framework to monitor the following features of the computing nodes described in Section 3:

- n_{th}: the number of active threads in the system,
- m_{used}: the amount of memory used by applications,
- m_{free}: the amount of free memory in the system,
- m_{shared}: the amount of used memory in buffers shared by applications,
- m_{buff}: the amount of memory used by the underlying operating system to buffer data,
- m_{cached}: the amount of memory used for caching data,
- SW_{used}: the amount of used swap space,
- SW_{free}: the amount of free swap space,

- CPU_{user}: the percentage of CPU time spent by normal processes executing in user mode,
- CPU_{ni}: the percentage of CPU time spent by high priority processes executing in user mode,
- CPU_{sys}: the percentage of CPU time spent by processes executing in kernel mode,
- CPU_{iow}: the percentage of CPU time spent by processes waiting for I/O to complete,
- CPU_{st}: the percentage of CPU time spent by processes waiting for services of other processes,
- CPU_{id}: the percentage of CPU idle time.

The trend of these features is reported in Figure 6, for the case of memory leaks. The Remaining Time to Hit the Response Time Threshold (RTTH)/Remaining Time to Crash (RTTC) are depicted on the abscissa in Figure 6. The feature selection process and analysis have been reported in Panacea Deliverable D3.1-ML_Framework and D.3.2-Proactive Management [24], Deliverable D4.1 of PANACEA project: Description of feasible use case [22]. March 2014, Deliverable D2.1 [23].

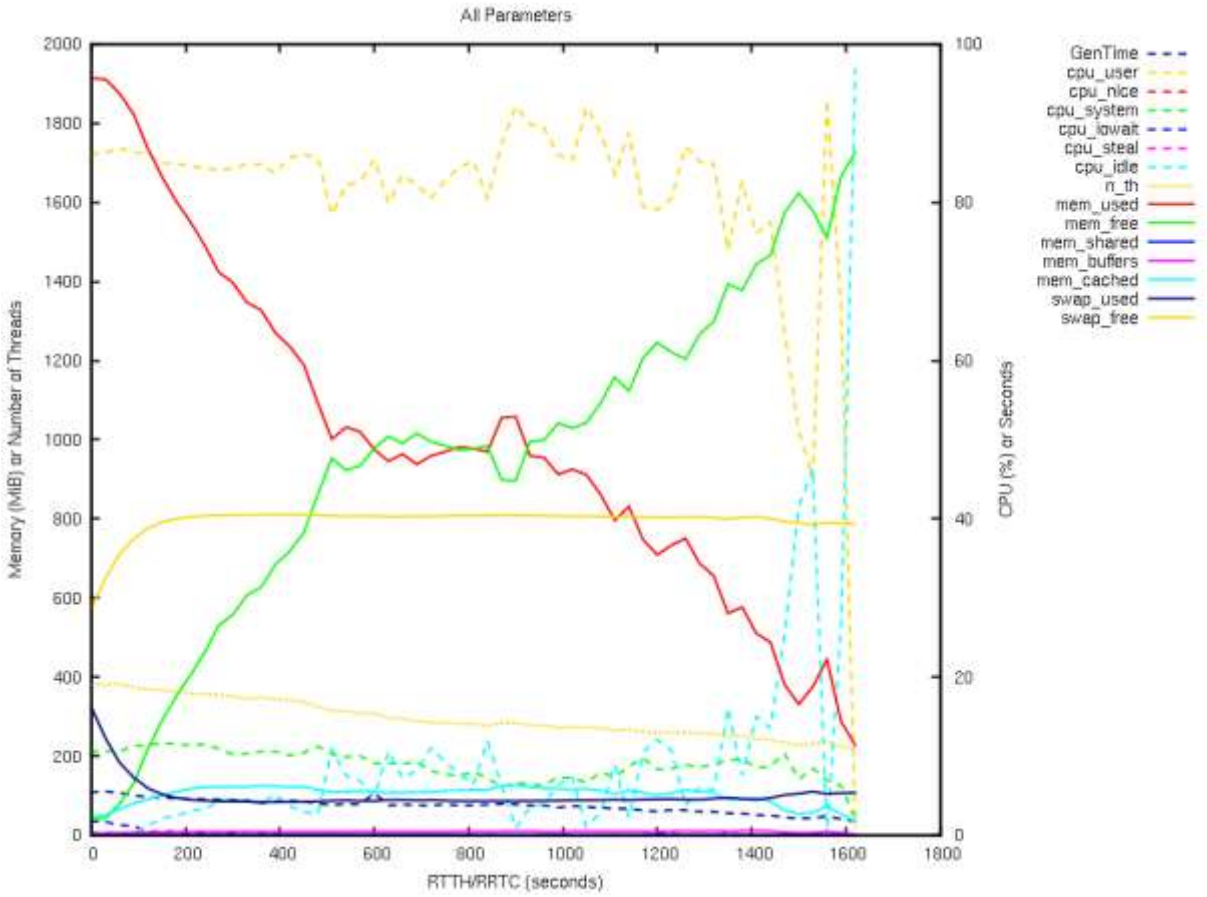

Figure 6: System parameters (memory leaks)

The Intra ACM therefore uses the same features to perform online estimation of the RTTF prediction model.

4.2 On-line Control Loop

Once the selected RTTF prediction model has been fed in PU, Intra ACM can perform the on-line control loop. Specifically, for each couple x of VMs, Intra ACM executes the following steps:

- The LMU of the active slave VM, say $VM1_x$, collects local measurements of the set of system features and sends them to VMC.
- Upon receiving measurements from the LMU of $VM1_x$, the MU of VMC queries the PU by using measurements as input, and retrieves the predicted RTTF of $VM1_x$. If the predicted RTTF is smaller than a (tunable) threshold T, then the MU performs the following actions:
    - In the list of VMs, it marks $VM1_x$ as *stand-by* and the other slave VM of the same couple, i.e. $VM2_x$, as *active*;
    - It communicates to the LBU the new list of active slave VMs;
    - It sends to $VM1_x$ the *rejuvenate* command in order to start the rejuvenation procedure;
    - It starts to receive measurements from $VM2_x$.

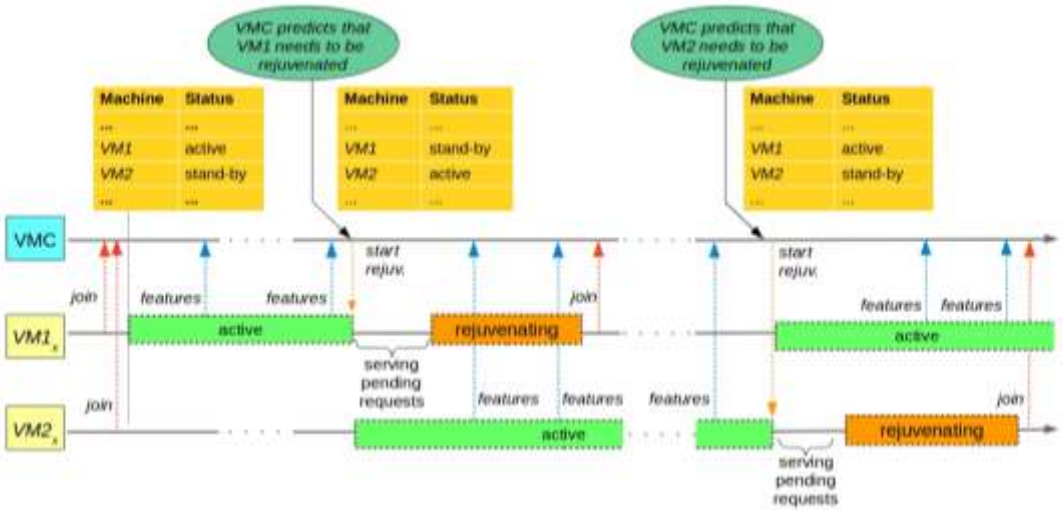

Figure 7: Dynamic Reconfiguration Flow Diagram

The diagram in Figure 7 provides an example of control flow to illustrate the on-line control loop control of Intra ACM in the case of one couple of VMs.

When a slave VM (active) receives the *rejuvenate* command, it completes pending requests and starts the rejuvenation action. We remark that the MU switches the active VM of a couple before to send the *rejuvenate* command, thus new incoming request are immediately

forwarded to the other VM of the couple. We further note that, when the active VM of a couple changes, it may be required, depending on the application, to execute some specific procedures before forwarding requests to the other VM, such as migrating data sessions, flushing in-memory data to shared databases, etc. Solutions for addressing these issues without affecting system availability as perceived by users have been discussed in [16]. We do not further deal with these issues, because they are orthogonal to the scope of this work.

The controller carries out all these steps in a completely automatic way, with no human intervention at all.

The threshold T used by MU can be specified by the user. Essentially, T defines a *safety value*, which is used by Intra ACM to determine the time instant when a VM has to be rejuvenated before the predicted failure time. On the one hand, this safety value allows ensuring that the VM can still execute pending requests before the actual failure time. Consequently, T has to be set at least in the same magnitude order of the request response time. On the other hand, we note that the value of T can be used to reduce the effect of MTTF prediction error on system availability. As an example, a low value of T entails that a VM is rejuvenated a small time before the predicted failure time. In such a scenario, even a very small overestimation of MTTF could prevent Intra ACM from rejuvenating a VM before the actual failure time.

Of course, if (due to a false negative) a VM fails before the controller rejuvenates it, the VM is definitely lost. In this case, the controller removes it from the pool of available VMs after some time (in fact, it receives no data from the VM). Nevertheless, destroying the VM or reactivating the VM requires manual intervention.

5 INTER-AUTONOMIC MANAGEMENT

As mentioned, the goal of the proposed framework is to maximize the availability of a given application hosted on a geographically-distributed (hybrid) cloud infrastructure, and at the same time to keep the response time as seen by remote clients of the application under a given threshold.

The available cloud resources can be distributed over the services offered by different cloud providers, and/or over services offered by the same cloud provider in different geographical locations, and/or over private cloud infrastructures. Each set of VMs, namely VMs hosted in a given region by one single cloud provider (or private infrastructure) is referred to as a *cloud region*. Each cloud region hosts at least the following set of VMs:

- A **Controller** (CON): this is a VM which is in charge of supervising the execution of the VMs in the cloud region;
- A **Load Balancer** (LB): this is a VM which receives connections from the remote clients and redirects them to specific VMs in the cloud region for processing. This is the actual entry point of the distributed system from the outside world, and receives from CON information about which are the currently available VMs;
- **Computing nodes** (CN): these are the VMs which host a copy of the application. Each CN in a couple cx (with $x \in \{1,\cdots,k\}$), named VM1_x and VM2_x, (shown in Figure 5), can be either in the ACTIVE state, or in the STANDBY state. A CN currently in the ACTIVE state is processing requests from the remote clients, while a STANDBY CN is a spare node of the cloud region which, upon receiving an activation command from CON, switches to the ACTIVE state so that it can process incoming requests. This command is received whenever a currently ACTIVE VM has to undergo the rejuvenation procedure (due to the accumulation of too many anomalies), or whenever CON decides that a higher number of VMs are necessary to process the incoming requests, given the current load of the system.

The overall distributed architecture is composed of any number of cloud regions. They can be hosted by both public providers and private infrastructures, thus making the solution viable in the case of public, private and hybrid cloud environments. Additionally, they can be geographically distributed, thus making the overall distributed system resilient as well to partitioning or disasters.

We note that having multiple LBs in the system is representative of several real-world scenarios, thus making our approach of general applicability. In fact, a given web-based application usually has multiple entry points which can be reached by the clients either by having the IP addresses of the entry points hard-coded within the application, or having additional directory services, such as DNS servers, providing the IP addresses on demand. In both cases, the LB IP addresses could be both hard-coded or provided upon request by a DNS service, thus making our solution compliant with current development standards.

In Figure 8, we present a sample architectural organization with three Controllers and Load Balancers when the number of cloud regions is set to three.

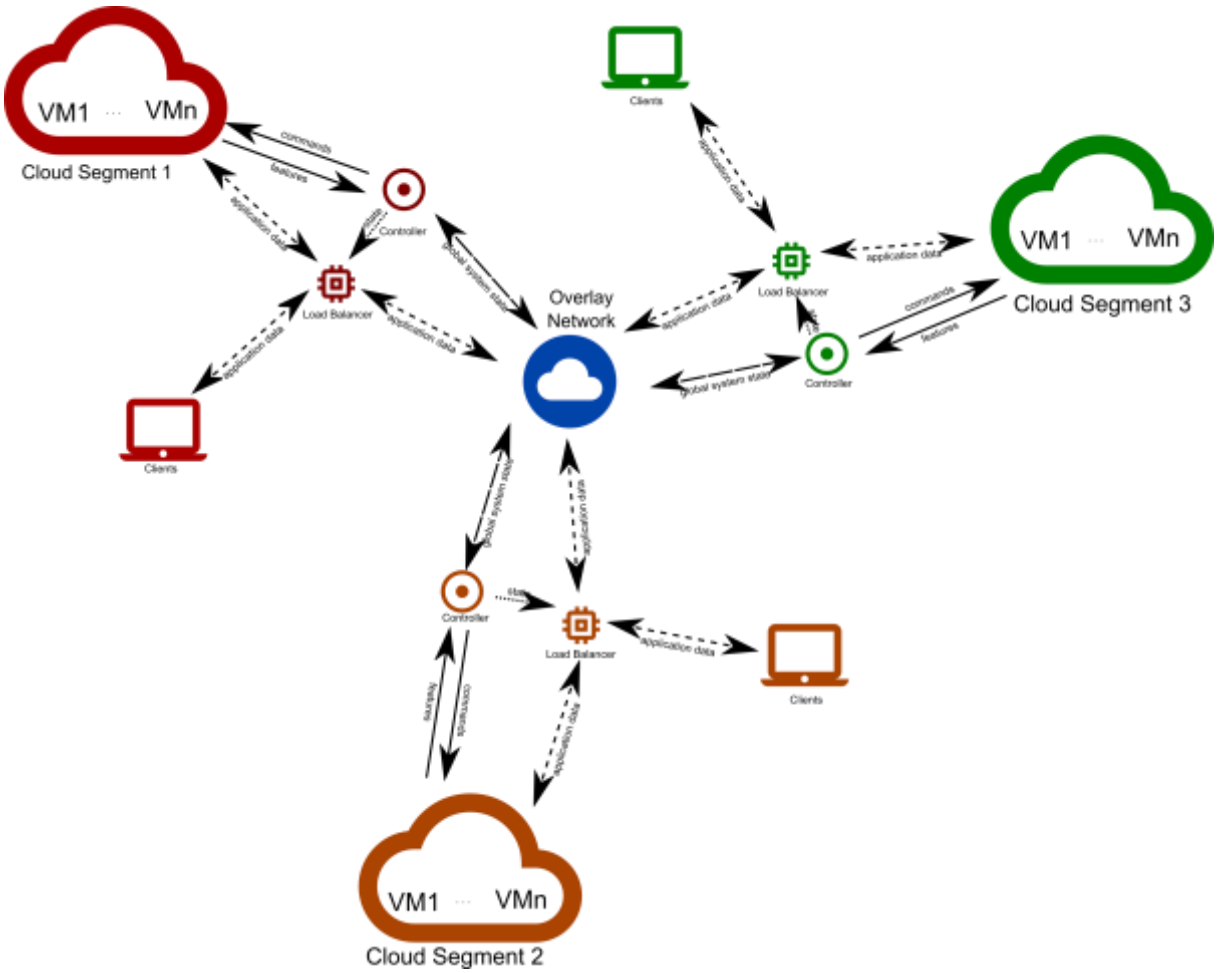

Figure 8: Global Architecture of the Distributed Environment with 3 cloud regions

Of course, if a single cloud region could be easily managed via Intra ACM, as described in Section 4, having multiple (possibly geographically distributed) cloud regions demands for an increased coordination capability by our proposed solution. In the following sections we describe the principles and the software components which we have exploited and developed to this end.

We note that in this architecture, both the load balancers and the controllers could be regarded as single point of failures for the single cloud regions. Concerning the load balancers, we do not address this issue for a simple reason. In fact, IP addresses of load balancers, as mentioned before, are recovered by the remote clients by using either external services, or by using internal (hard-coded) tables. Therefore, if a load balancer fails, devising any method to replace it could be proven unfeasible, as there is no possibility of intervention on the client side. Moreover, a correct implementation of the client application will likely retrieve a different IP address for a different load balancer (hosted in another cloud region) if the application fails to respond from the crashed balancer. Therefore, after a certain (small) amount of time, availability of the system will be restored by design by the fact that the client

application will try again to connect to the service accessing it via a different entry point (a different load balancer in our scenario). Nevertheless, we note that, due to their simplicity, load balancers VM are unlikely to fail.

On the other hand, if the controller fails, we might lose the entire region. In fact, while the system would behave correctly for a certain amount of time – in fact the load balancer is still active, and VMs are serving requests – if some VM in the region fails, the load balancer cannot be notified of this, and requests will not be redirected to other (active) VMs. The feasibility of this approach is demonstrated with only 3 cloud regions in our experiments, as shown in Figure 8.

To solve this issue, a replication approach for the controllers can be envisaged. The replication method and software will be provided by IBM, presented in Deliverable 2.4 [26]. It will be used to tolerate the faulty leader and deliver the communications between the Controllers of different cloud regions. It will provide replication services, leader election and distributed locking in distributed environment.

In fact, having a spare controller (which constantly monitors over the currently active one) would be enough to detect whether the primary has failed. In this case, the spare controller would be able to notify the load balancer of its takeover as the primary controller. Possibly, an additional virtual machine serving as the spare controller could be activated. This solution is quite trivial, and has already been addressed in literature (see, e.g., [17]–[19]), so we consider it orthogonal to our solution.

5.1 Communication among Cloud Regions

To allow for the exchange of knowledge among the controllers, they must rely on a reliable communication system. Additionally, due to the fact that remote clients' requests are redirected towards VMs hosted in differentiated cloud regions, our framework must ensure that the decision of redirecting a request does not affect its response time.

To this end, we have decided to rely on a specifically-targeted overlay network which is designed to operate between different cloud regions. The overlay network that will be used with required features is reported in Deliverable 2.3-Autonomic Overlays of Panacea project [26].

Each cloud region runs two proxy agents (transmission and reception agents) handling overlay traffic of all virtual machines of its cloud. These agents are hosted within the LB VM, which is the connection point among the remote users and the CNs in the system, and act as "connection proxies" among all the different LBs, as shown in Figure 8.

The overlay network is formed of software routers deployed at the cloud sites and possibly in other locations in the Internet.

They constantly monitor the information used to determine the optimal path in the network to each possible destination, namely all the other LBs in the distributed environment. Whenever a transmission agent is requested to deliver a packet to any LB in the system, the information is sent to the destination agent following the optimal path.

Therefore, each CON-CON communication is directed towards this overlay network. In this way, we are sure that the communication among controllers is guaranteed even in the case of

network faulty links. Furthermore, we ensure that the requests issued by remote users of the applications are redirected to other regions with the minimal latency offered by the current state of the network. To demonstrate the feasibility of communications between Inter ACM and Intra ACM, within only 3 cloud regions, in our experiment we relied on the standard TCP_IP protocol.

5.2 Determination of the Global State

To let the controllers take coherent decisions, we rely on a *leader election* algorithm in order to elect, among all the CON nodes, one leader. We assume that the overlay is such that it can maintain the delays under a given value consistent with the values of the RTT of less than a given value. This leader is the one in charge of collecting usage data from other controllers, and to decide what is the best (global) policy to keep the MTTF of the system above a certain threshold, to keep the response time seen by the end users below a given threshold, and to minimize the rejuvenation rate of all the VMs forming up the distributed application.

As the leader election algorithm, we rely on the one presented in [20]. This is an efficient algorithm which can scale to any number of participants (thus, ensuring scalability of our proposal to any number of cloud regions) and which is highly resilient to changes in the network of participants. Specifically, by relying on this algorithm, our framework is able to enforce dynamic network reconfiguration, even in case of multiple node and link failures in high-speed networks with arbitrary topology. In particular, CON nodes keep routing tables which allow to store information about the participating nodes (namely, the other controllers) and they asynchronously update their content, namely these routing table are incrementally generated/updated during the lifetime of the distributed application.

Each CON uses its external IP address as its ID number. At application initialization (or whenever controllers suspect the current leader to be failed) each CON node must exchange information about its ID only with neighbor CONs, in order to determine what is the (non-failed) CON node with the highest ID. This is the leader, which is determined through a distributed election.

More complex implementation of the Leader Election with several features that allow handling complicated cases (replication service for a high availability and partitioned-tolerant leader election) will be exploited that is provided by IBM in Deliverable 2.4 [26] .

The ultimate goal of the proposed approach is to distributed the (new) incoming connections from remote clients of the (general) application hosted by the virtual cloud infrastructure to the cloud regions, which are currently less loaded.

For the sake of simplicity, let us now consider one single cloud region. In this region, among the several available virtual machines (slaves), several are in the *active* state, others are in the *standby* state. At this point, we exploit the same predictors generated by the framework described in Section 2 to compute the (current) MTTF, rather than the RTTF. We note that we consider that the MTTF can change at runtime because the number of connected remote users of the system can be any number, and the distribution of anomalies can change over time, which is the most general application case.

The predictors generated as described in Section 2 rely both on measured features (i.e., the current state of some measures of interest, such as used memory or user CPU usage) and added derived metrics.

The added derived metrics are used in this step of our proposed solution in a way different with respect to the actual prediction of the RTTF by the controllers. In fact, the slopes are representative of the effect of the current clients' request rate λ, due to the fact that they express how far the system is approaching the rejuvenation point given both the current workload and the anomaly accumulation rate of a given (set of) virtual machines.

As mentioned, the same models used by the controllers to proactively rejuvenate virtual machines in a given cloud region, according to the description given in Section 4, are used to compute the current MTTF. Specifically, given a certain prediction model, we evaluate its value passing as input the current slopes and the values of the other features as measured immediately after the rejuvenation of a given virtual machine. In this way, we actually ask the ML-based prediction model to estimate how much time it would take for a VM to reach the rejuvenation point from the beginning of its (current) execution, if the workload and the anomaly rate were the same as the current ones.

This evaluation of the prediction model allows us to have an estimation of the Mean Time to Failure (MTTF) of a given VM. The controller carries out all these steps in a completely automatic way, with no human intervention at all.

5.3 Proactive Activation of new VMs

Due to high workload and anomalies rate a given region can be overloaded and the balancing algorithm should redirect the requests to less loaded region(s).

As a countermeasure to this issue, each region's controller monitors the MTTF. If this value becomes less than a given threshold (which can be specified at configuration time), then the controller decides to activate new spare VMs. Specifically, this can be easily done by sending a control message to a VM in the pool which is currently in the *standby* state, so that this VM is brought to the *active* state even if no rejuvenation action is taking place.

We note that if the number of spare VMs is low, this solution can be complemented with services offered by cloud service providers, which allow to create and start *any number* of new VMs from the system image. Newly instantiated VMs in our framework, according to the node configuration described in Section 3, immediately connect to the region's controller (Intra ACM) to notify their presence, and they join the pool in the *standby* state. In this way, a set of regions can be controlled autonomously by their Intra ACM for activating new cloud resources (VMs). In the following, we describe the steps required to start up additional VMs in Amazon.

Amazon is taken here as an example because of its wide use in cloud applications. Others solutions, as OpenStack and OPEN NEBULA, will have to be handled in a similar way.

The other possibility is that the user manually activates a set of virtual machines, via a user interface. This is described, as well, in the following paragraphs. The difference is that the automatic activation requires seconds to have a new VM joining the pool, while the manual procedure require minutes. These two possibilities are given to demonstrate the interest of the automatic mode.

5.3.1 Automatic Proactive Activation of new VMs

To let the controller launch additional VMs, the controller (Intra ACM) must have access to

Amazon AWS command line tool. To install it, a working installation of Python 2 version 2.6.5+ or Python 3 version 3.3+ must be available.

The preliminary step (which must be run only when the controller is installed for the first time) is to download and run the installation script:

```
$ curl "https://bootstrap.pypa.io/get-pip.py" -o "get-pip.py"
$ sudo python get-pip.py
```

Then, the AWS command line can be installed:

```
$ sudo pip install awscli
```

Then, the AWS command line tool must be configured:

```
$ aws configure
AWS Access Key ID [None]: AKIAIOSFODNN7EXAMPLE
AWS Secret Access Key [None]: wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
Default region name [None]: us-west-2
Default output format [None]: json
```

From now on, several additional installation steps are required, specifically to create a security group, a key pair and a role to be used when creating/accessing VM on Amazon. After these steps, the controller is able to launch additional instances.

The run time MTTF prediction, based on the ML prediction models (shown in Figure 12 and Figure 13 – Frankfurt and Ireland Amazon Region) created by the ML framework in each region, will autonomously trigger Intra ACM to launch an additional VM(s) through the following commands. In this way, proactive scaling of VMs in a given region can be realized in few seconds. As a result, the response time threshold and seamless application execution will be achieved.

1. This command can be used to launch an additional VM, from a given AMI:

```
$ aws ec2 run-instances --image-id ami-29ebb519 --count 1 --instance-type
t2.micro --key-name devenv-key --security-groups devenv-sg --query
'Instances[0].InstanceId'
"i-ec3e1e2k"
```

1. The instance will take few seconds to launch. Once the instance is up and running, the following command will retrieve the public IP address that you will use to connect to the instance.

```
$ aws ec2 describe-instances --instance-ids i-ec3e1e2k --query
'Reservations[0].Instances[0].PublicIpAddress'
"54.183.22.255"
```

Using the Panacea AMI will make this VM immediately try to connect to the controller in the current region.

The controller carries out all these steps in a completely automatic way, with no human intervention at all.

5.3.2 Manual Proactive Activation of new VMs

Differently, the manual activation of new VMs from the Panacea AMI requires a human being access the web console, shown in Figure 9.

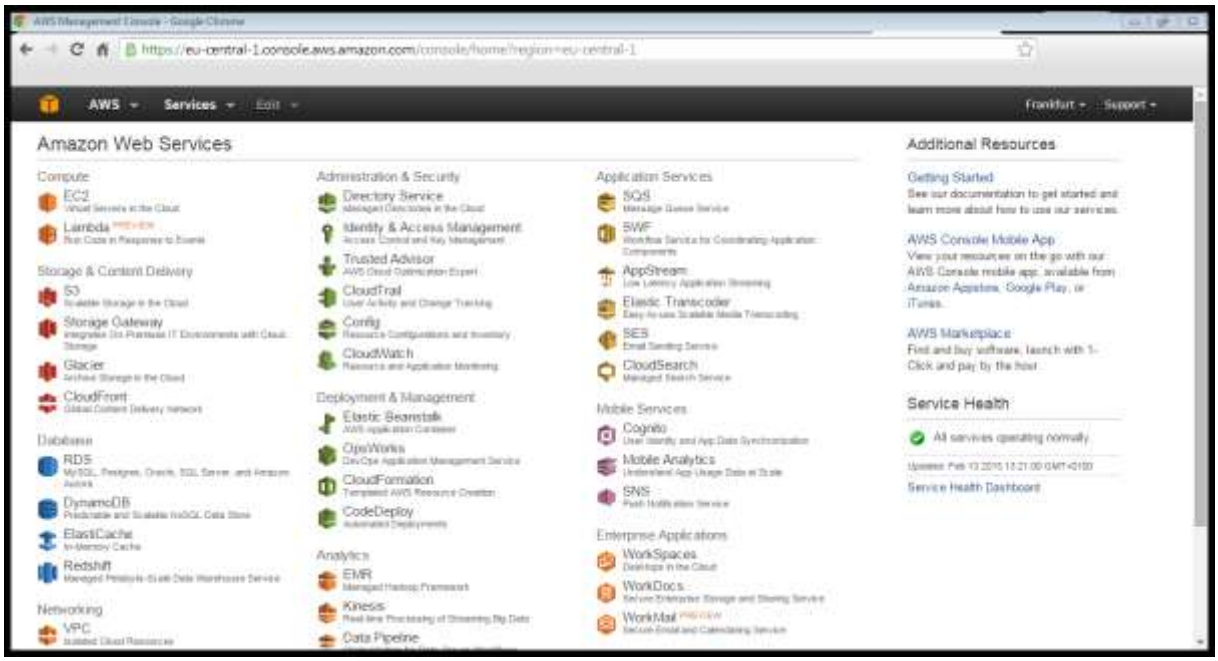

Figure 9: Amazon AWS Web Console after login

Then, the user has to access the EC2 Virtual Servers in the Cloud, and select the AMI, as shown in Figure 10.

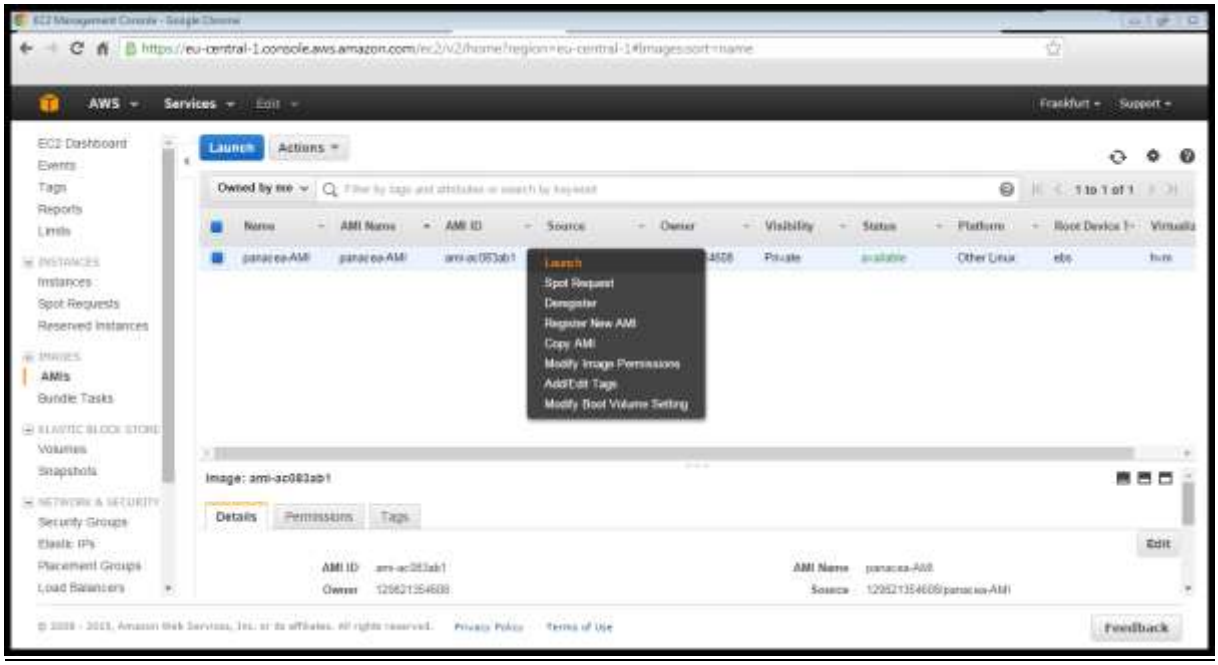

Figure 10: Selection of the Panacea AMI in the Amazon web interface

Then, the user must go through the configuration of a large set of parameters, which we do not report for brevity, but which are related to all the parameters that the command line interface is able to pass directly to Amazon in the single last command. This long procedure is an additional reason why the automatic procedure is to be preferred to the human intervention.

5.4 Proactive Scalability of Cloud Resources via Intra/Inter ACM

The proposed solution allows to enforce high scalability of the deployed application. In fact, scalability can be controlled at different levels:

- Intra-ACM controllers can decide upon the number of VMs which are currently active. By controlling at runtime the execution dynamics of VMs into a cloud region (specifically, by controlling the Mean Time to Failure), the controller is able to make a prediction on the number of VMs that are necessary so as to keep the rejuvenation rate below a certain threshold. Considering that the rejuvenation rate is directly proportional to the current workload and accumulated anomalies seen by the system, the controller can decide whether new VMs are required to fulfil the requests coming to the region. In this way, the Intra-ACM controller is able to proactively scale up/down the number of VMs composing the region.
- Multiple cloud regions could be connected together, even at runtime. This gives the user of the framework an additional level of scalability. In fact, rather than adding single VMs to the system, entire regions, composed of any number of VMs, could be connected. This allows, for example, injecting a higher amount of computing power, which could be possibly required only during certain operating periods of the system.

Overall, our solution gives great freedom in the composition of the system, and enforces autonomic and proactive scalability properties, with minor (or null) manual intervention.

This scalability is guaranteed by design due to the nature of the protocol diagram which is reported in Figure 7. In fact, new VMs can be added asynchronously, in any order, without the risk for deadlock or starvation.

5.5 Overlay Network Resilient to Link Failures

The Overlay network is composed of several Intra overlays that are interconnected by an Inter overlay network. In this task, we will mainly focus on application-specific inter-cloud overlays. The proposed overlay support will use the machine-learning algorithms developed in WP3 in order to proactively reconfigure the application-specific overlays when an adverse event is predicted. It will also be able to derive the most adequate topology from the application requirements given by the user. To this end, we will have to solve the following problems: Topology creation and reconfiguration; Dynamic routing within the overlay network.

Agents are used for creating overlay topologies (between controllers) for selecting minimum paths and delays between them, based on Panacea Deliverable D2.3: Autonomic Communication Overlay [27]. It is required agents to provide unique IDs of all VMS on both sites. TCP-IP protocol or UDP protocol can be used for communications.

In Figure 11 the overlay network topology is shown. The network includes 8 nodes. Each node is connected with other 4 nodes in the network. The virtual topology is scalable (Up and Down to any size) and resilient to partitioning in presence of multiple node and links failures.

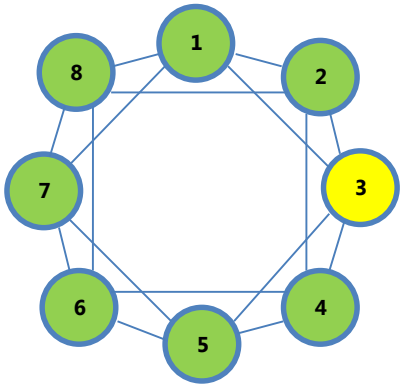

Figure 11: Virtual Network Topology with 8 Nodes

5.6 Proactive Management and High Availability based on Intra/Inter ACM

Intra ACM provides distributed proactive management of the cloud resources for given Workloads and accumulated anomalies.

Intra ACM autonomously can increase the availability of its region by proactively minimizing the fault negatives and also by proactive self-rejuvenation.

In the case of Faulty Leader and New Leader is elected without interrupting the services - keeping the response time and availability of the hybrid cloud by redirecting the flows to the available regions.

Inter ACM can coordinate and proactively to manage the resources of the hybrid clouds.

In the case of hybrid cloud, the datapoints (which can be regarded as sensible data) coming from all the cloud regions – specifically, as well from the public cloud – can be stored in the private region of the controller. In this way, the important data are always kept in the most secure region, which is the private one. This is exactly why our experiment has been set up in such a way that the collection of the data is carried out in our private server in Munich, which is a ProLiant Server equipped with 32 cores (offering enough computing power to generate the ML models) and 1 TB of disk storage (to store the database files of an extremely large number of VMs – this is important as well for scalability).

Heavily relying on a hybrid cloud in this way is not a problem. In fact, Gartner [1] states that, in the near future, more than 51% of cloud environments will be organized as hybrid infrastructures, making our solution perfectly suitable for this trend.

6 EXPERIMENTAL RESULTS

In our experimental study we use the well-known TPC-W Benchmark [12]. TPC-W is a transactional web e-Commerce benchmark. The system workload is generated by a number of users, which interact with the web shop through web browsers for searching, browsing and ordering books. TPC-W, in addition to mimic user behaviour through a web page navigation graph, also defines the structure of all web pages, including images, and the database schema. The system workload can be configured by changing the number of concurrent users, and other parameters, such as the transaction execution mix or the database size. A comprehensive description of the benchmark can be found in the TPC-W specification document [3].

In our experimental setting, slave VMs are equipped with Java SE Runtime Environment version 1.6. Each slave VM runs a Java implementation of the TPC-W benchmark [4], hosted by Apache Tomcat version 6.0. As for the database server, we use MySql version 5.1. The workload is generated via emulated web browsers. An emulated web browser simulates the presence of a user accessing web pages through a web browser.

In this section, we first study the viability of our solution on a single cloud region, so as to assess the feasibility of the Intra ACM. Then, we explicitly rely on a hybrid cloud setup, composed of two regions hosted by Amazon (one in Ireland, one in Frankfurt) and one additional region hosted on a private HP ProLiant server in Munich. This second experimental assessment allows us to evaluate the feasibility of the Inter ACM.

The experiments with Munich (Leader) and Frankfurt and Ireland use 3 nodes from the Scalable topology in Figure 11. As well , if the Leader fails (Munich) then the new leader is elected (let us say Frankfurt). The TPC-W users load at Munich will be forwarded to (Ireland and Frankfurt) in real time. As well, the Dynamic reconfiguration Flow Diagram can handle the TPC_W load on Munich , while the New Leader is elected. As well, the new elected leader might add new VMS to keep the response time under the threshold and seamless execution.

Moreover, this experimental setup shows that our solution is scalable to any number of virtual machines hosted in any number of cloud regions, at a geographical scale. In fact, each cloud region will host a number of virtual machines which varies over time, as well according to the number of connected clients (thus, depending on the current load). Moreover, the possibility to add at runtime new cloud regions (as we will show in Section 6.3) gives the possibility of an additional scalability level. As it will be shown, the distributed controllers' architecture will promptly detect the joining of a new cloud region, proving our proposal to be viable even in case of churn.

6.1 Experiments Repeatability

Our experiments are fully repeatable. In fact, we have generated multiple times prediction models on the same virtualized infrastructures, and the outcoming prediction models have always been comparable. As an example, we report in Figure 12 and Figure 13 the M5P prediction models built on Amazon for the Ireland and Frankfurt cloud regions. Numerically, the S-MAE error (with 10% threshold) for the Ireland region is 79.182, while for Frankfurt region is 82.193. By these results, we can see that the prediction models, when built on similar (virtualized) architectures, provide almost the same results. Of course, a small

variability is related as well to the fact that the experiments are carried out on a virtualized environment, without any knowledge about the underlying hardware used to provide the virtualized service, and by the fact that the hardware is shared among different users at the cloud provider level.

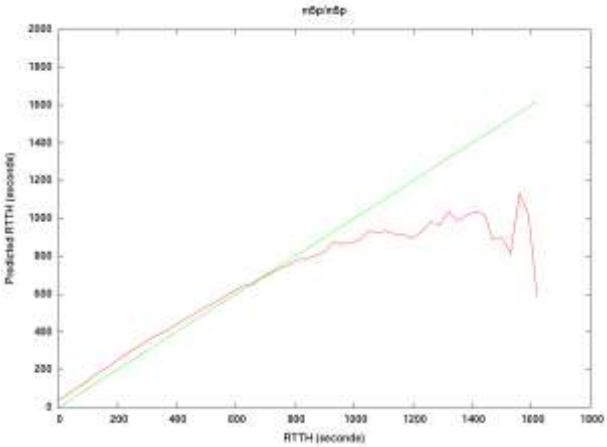

Figure 12: M5P prediction model for Frankfurt Amazon Region

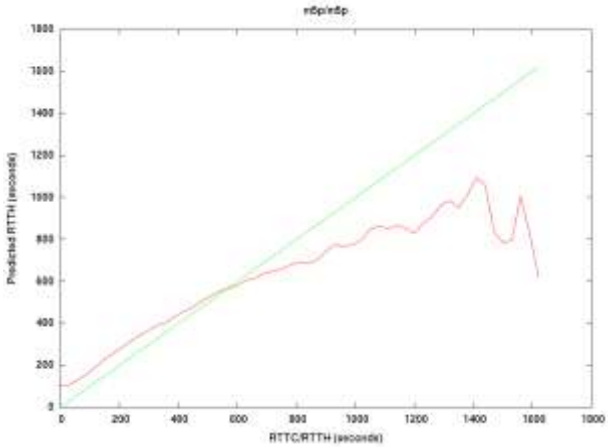

Figure 13: M5P prediction model for Ireland Amazon Region

6.2 Experimental Results for the Intra-ACM

In this section, we present an experimental study we carried out to evaluate the effectiveness of Intra ACM. We first describe the experimental environment used in our tests. Then, we discuss the used prediction models. Finally, we show and discuss results of experiments carried out in the case of two different system configurations in terms of number virtual machines and combination of injection of anomalies.

For this preliminary experimentation, we have relied on Amazon EC2, in the Frankfurt region. In particular, we use `m3.large` instances, which offer 2 virtual CPUs, and 7.5 GB of RAM. Ubuntu Server 12.04 LTS is used as the virtualized operating system.

We injected two kinds of anomalies in slave VMs, i.e. memory leaks and unterminated threads. To this aim, we modified the Java implementation of the TPC-W. Specifically, when a user request was received by a slave VM, a new (dummy) Java object and/or a thread entering an infinite loop was generated with a given probability. In order to evaluate the ability of the framework to cope with scenarios with variable anomalies' injection rates, probabilities of generating a new object/unterminated thread in each VM, as well as their size, were randomly changed every time the VM was restarted after a (predicted) failure. This led to an execution scenario where VMs have been showing different anomaly occurrence patterns. The organization of our experimental setting is depicted in Figure 14.

The worst-case scenario for this experiment is when both memory leaks' and unterminated threads' injection rates are the highest. This means that the anomaly probability is set to 100%, and therefore each time a remote client issues a request, a new object is created and a new thread is spawn.

In this experiment, since the number of clients is fixed to 64 users, the incoming workload is as well fixed. We will deal with variable workload in Section 6.3.

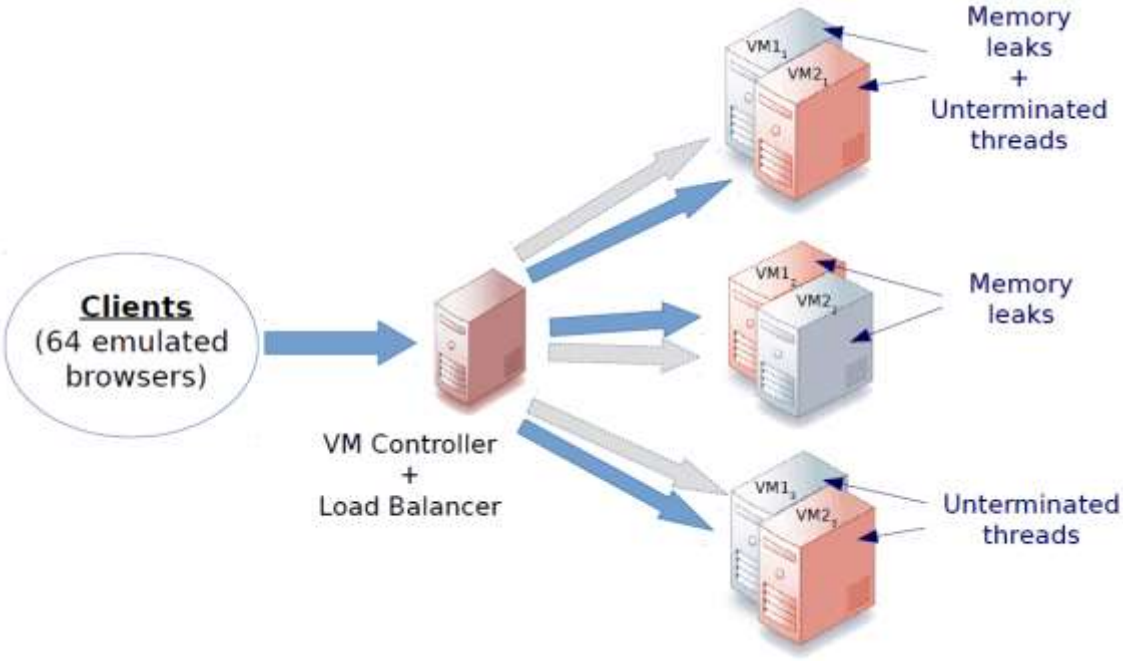

Figure 14: Intra ACM test-bed scenario

In our experiments, the failure condition of a VM was true if at least one of the following conditions was true:

1) both free memory amount and free swap space amount were less than 5% of the total memory and total swap space, respectively,
2) the average response time was higher than 4.5 seconds along 10 consecutive measurements by emulated web browser, and
3) the VM had not been sending measurements to FMS for more than 1 minute (which might suggest that the VM has crashed).

We evaluated three supervised learning models for implementing MLP, i.e. Linear Regression, M5P (decision tree with the possibility of linear regression functions at the leaves), and the regression model achieved through Lasso regularization as a predictor. By our results, M5P has been proven as the most effective ML algorithm (both in terms of training time and prediction accuracy).

We present experimental results for the cases of two scenarios. In the first one, we used only one couple of slave VMs, say c_1. In this scenario, client requests are processed by one active VM. Both memory leaks and unterminated threads are injected in active VM. 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 $VM1_1$ to $VM2_1$ or viceversa (as related to the flow diagram in Figure 7). In the second scenario, we used three couple of VMs, say c_1, c_2 and c_3. 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 $VM1_1$ and $VM2_1$. Conversely, we injected only unterminated threads in $VM1_2$ and $VM2_2$, also memory leaks in ($VM1_3$ and $VM2_3$). 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 15, 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 15, 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 Intra ACM ensured an upper bound value of the average response time equal to 4.5 seconds (as we set in the VM failure condition).

The distance between two consecutive red lines changes due to the fact that the injection rate of anomalies changes. In fact, some red curves are more close to each other, while other are more far. It is interesting to note that when the red lines are closer, the curves associated with features are steeper, because the VMs approach the failure point more quickly. This distance is representative as well of the MTTF (related to the current load and the current anomalies injection rate), which we are able to estimate as described in Section 5.2.

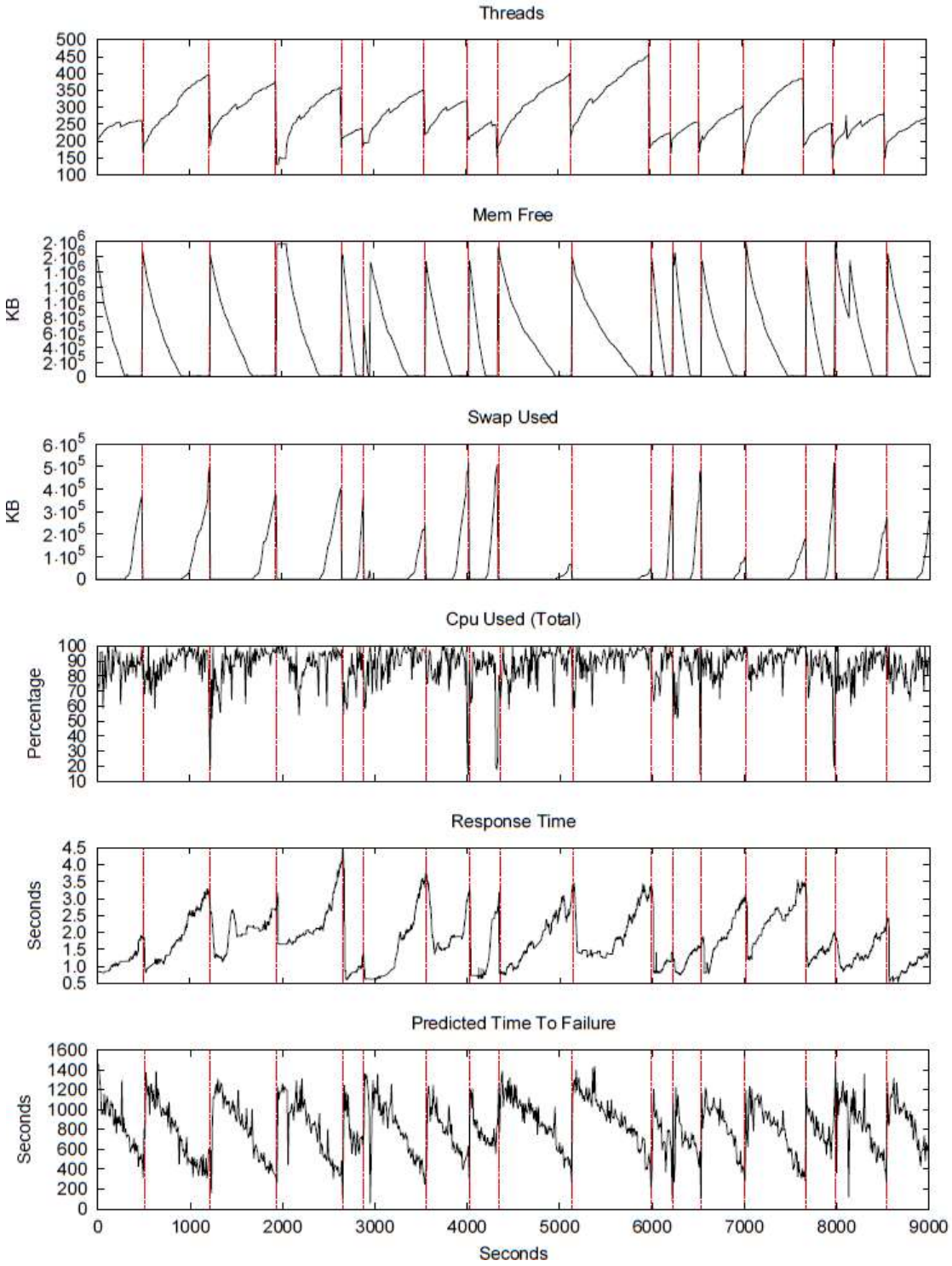

Figure 15: System features, response time and predicted RTTF for the scenario with 2 VMs and Lasso (with reduced parameters) as a predictor.

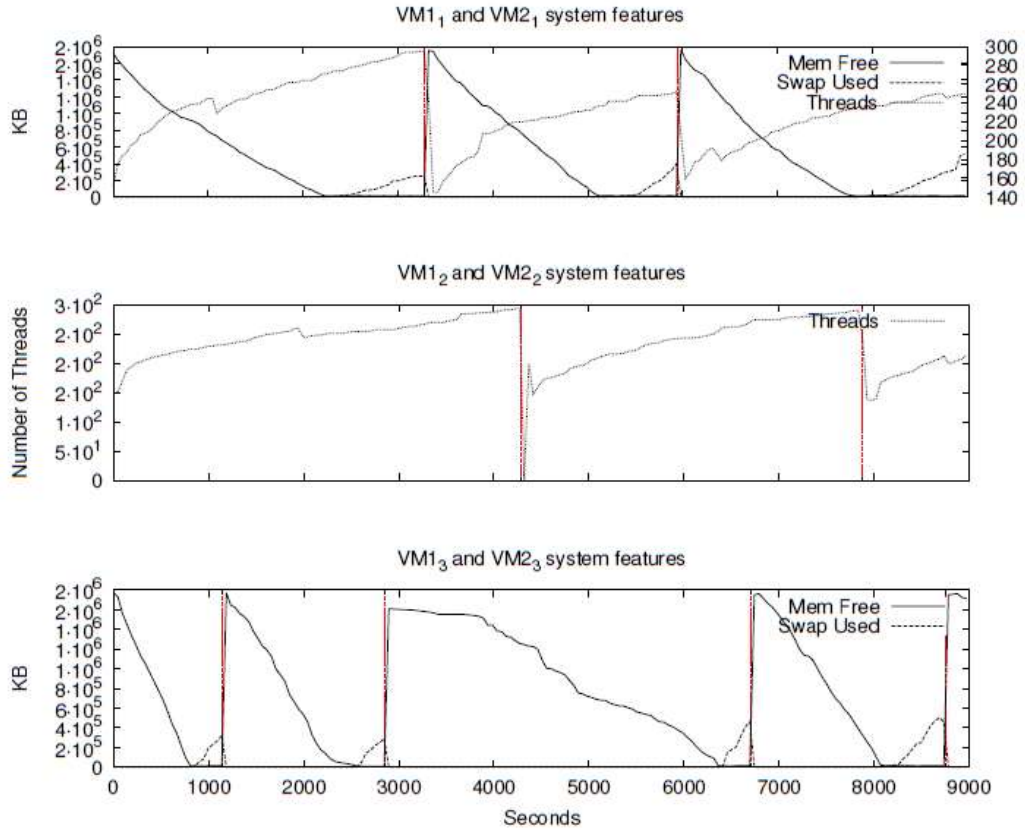

Figure 16: System Features for the scenario with 6 VMs and Lasso (with reduced parameters) as a predictor.

In Figure 16, 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 16, 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.

To complete the study, and to show the accuracy of Intra ACM, we measured the number of *false negatives*. As mentioned in Section 2, our framework 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. This scenario is shown in Figure 17. On the top of the figure, we can see the effect of a prediction error. Adopting the threshold T, as shown in the bottom figure, allows preventing the occurrence on the failure also in the case of prediction errors.

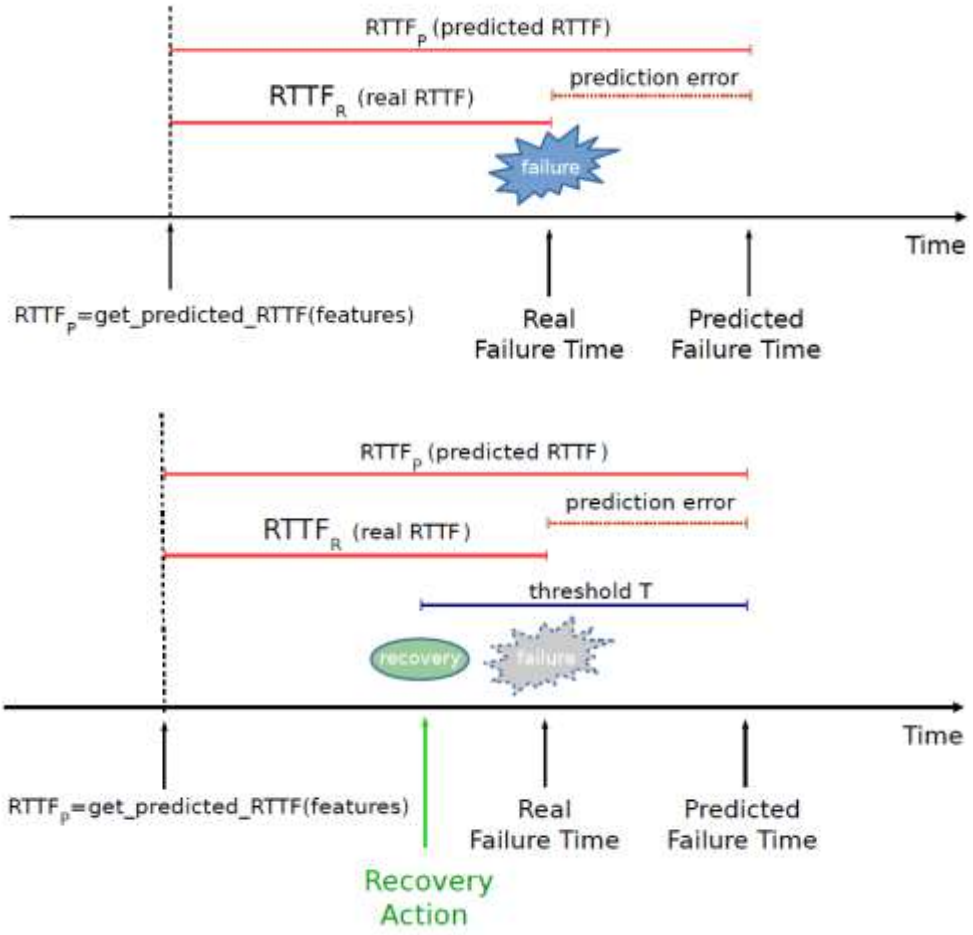

Figure 17: False Negative Occurrence

We note that availability A is determined by measuring and estimating MTTF and Mean Time to Repair (MTTR):

$$A = \frac{MTTF}{(MTTF + MTTR)} \tag{5}$$

We introduce the *gains formula*, expressed as in [21]:

$$A_{pfh} = A_{orig} + k * \frac{MTTR}{MTTF + MTTR} \tag{6}$$

where, $k = 1 - \left(1 + r * (rf - 1)\right) * \left(1 - P_p * r + P_e * \frac{r}{p}\right)$, and:

- A_{orig} is the availability of the system without proactive failure handling;

- A_{pfh} is the availability with proactive failure handling.

k depends on:

- Precision (# correct alarms/all alarms): p
- Recall (# correct alarms/ all failures): r
- Probability that a failure can be avoided: P_p (failure prevention)
- Probability that failures occur due to actions: P_e (extra failure)
- Mean improvement in MTTR: rf

Therefore, the overall availability of the system directly depends on the number of false negatives.

In Figure 18, 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. These values have been selected according to the results in Figure 15. In fact, by considering as well a confidence interval (which was already thoroughly discussed in Deliverable 3.1 [24] and Deliverable 3.2 [25]), we select the smallest rejuvenation threshold to 300 seconds – remember that the experiments were carried out under the same conditions (injection rates and workload). Then, we increased this value to study its effect on the number of false negatives.

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. Nevertheless, in Figure 18, we can see that the higher is the threshold T, the lower the number of false negatives. This is an expected result. In fact, as already mentioned in Section 4, 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. It was negligible with respect to the overall response time reduction achieved with Intra ACM with respect the case when VMs are switched after a failure occurs.

The results additionally show that by simply tuning the rejuvenation threshold, the system can increase significantly availability, according to Equation (6).

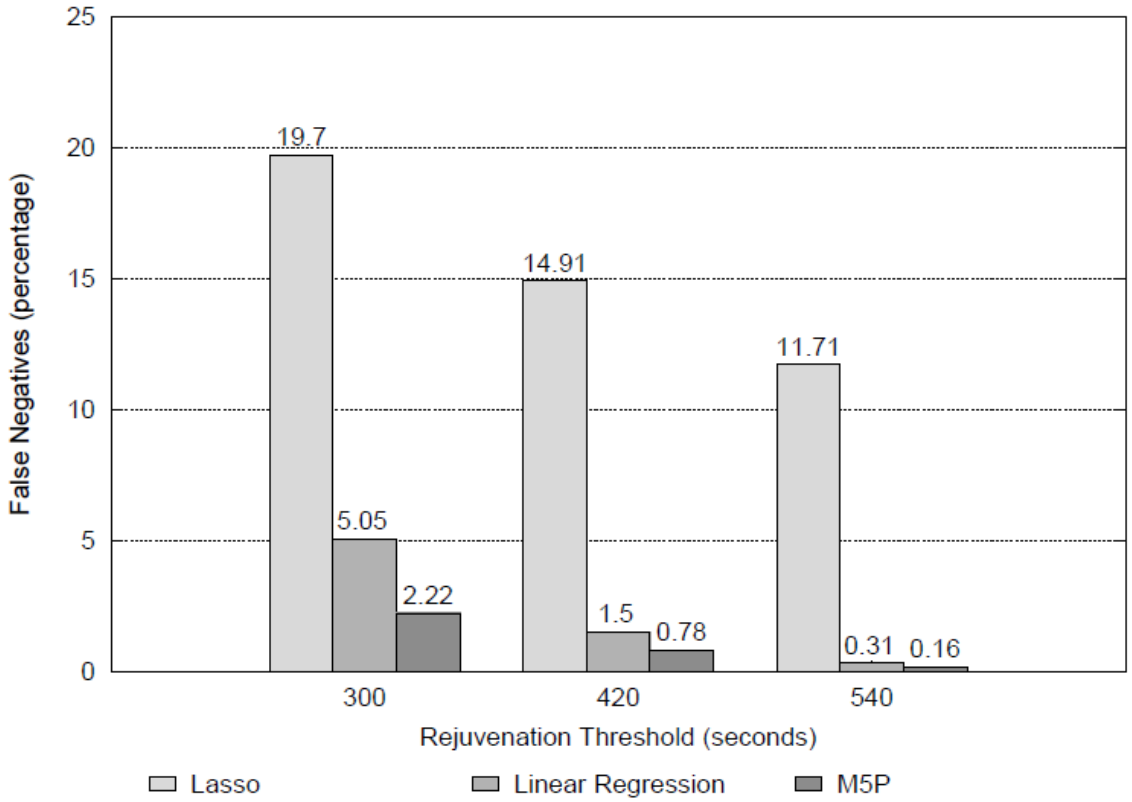

Figure 18: Percentage of false negatives using different thresholds in the case of memory leaks and unterminated threads.

6.3 Experimental Results for the Inter-ACM

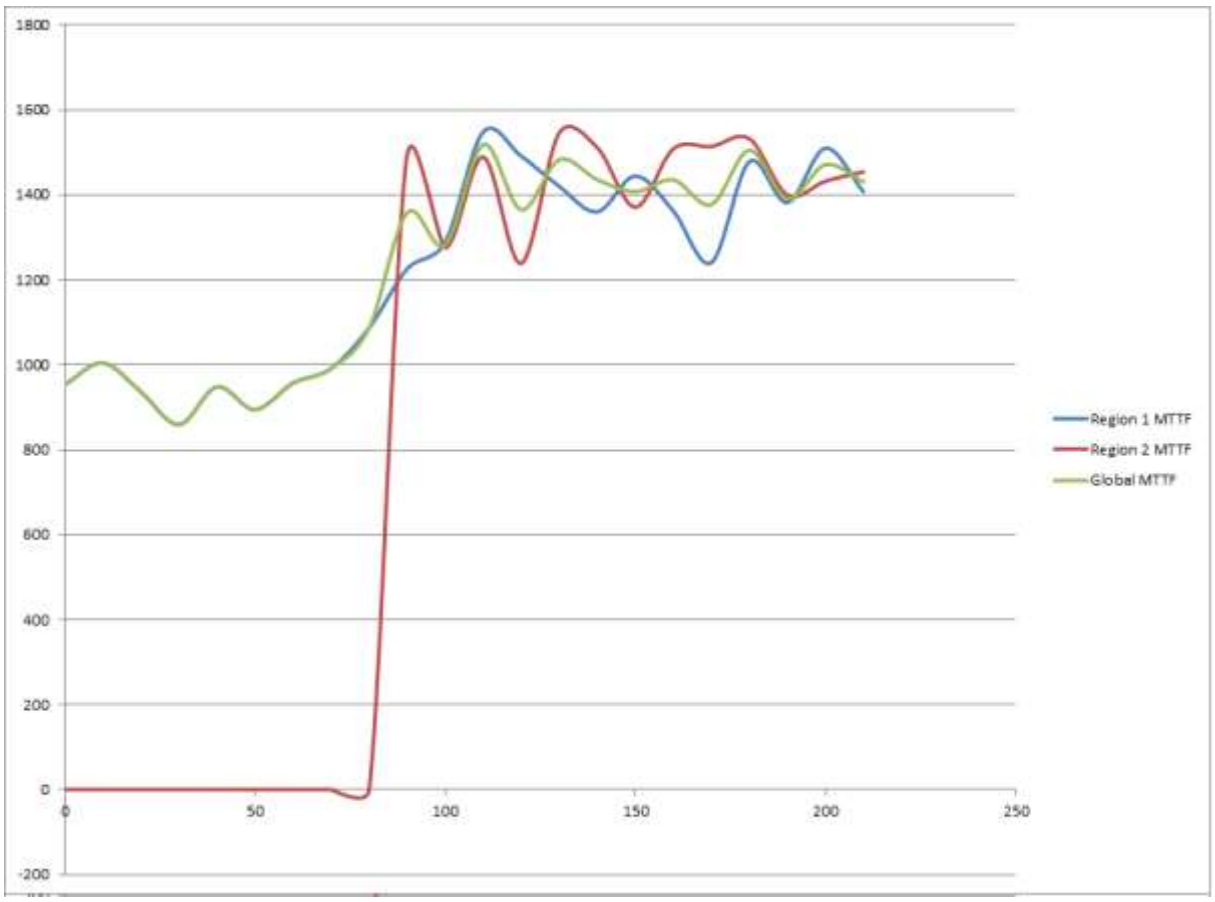

Figure 19: Global MTTF variation when switching from 1 to 2 cloud regions.

As a preliminary assessment, Figure 19 reports the variation of the MTTF measured by the system. In particular, in this phase of the experimentation, we have used only the Amazon Ireland and the Munich regions. At system startup, only the Amazon Ireland region is active, so only the Amazon Ireland controller is working and serving users' requests. After 70 seconds of execution, the Munich region is activated, and the Munich's controller connects to Ireland's. At this point, the leader election algorithm is run, and the two controllers start exchanging information. Before the second region connects, the global MTTF is the same as the Ireland's region.

When Munich region is connected, then the total number of VMs active in the system is higher, so the global MTTF increases. This proves that our framework is correctly able to estimate the MTTF. Also, the Intra/Inter ACM can provide efficient proactive management of the cloud resources. As a result of this, the response time threshold and the seamless execution is guaranteed.

It demonstrated that our system is resilient to a high variability in the number of users, and to changes in the workload.

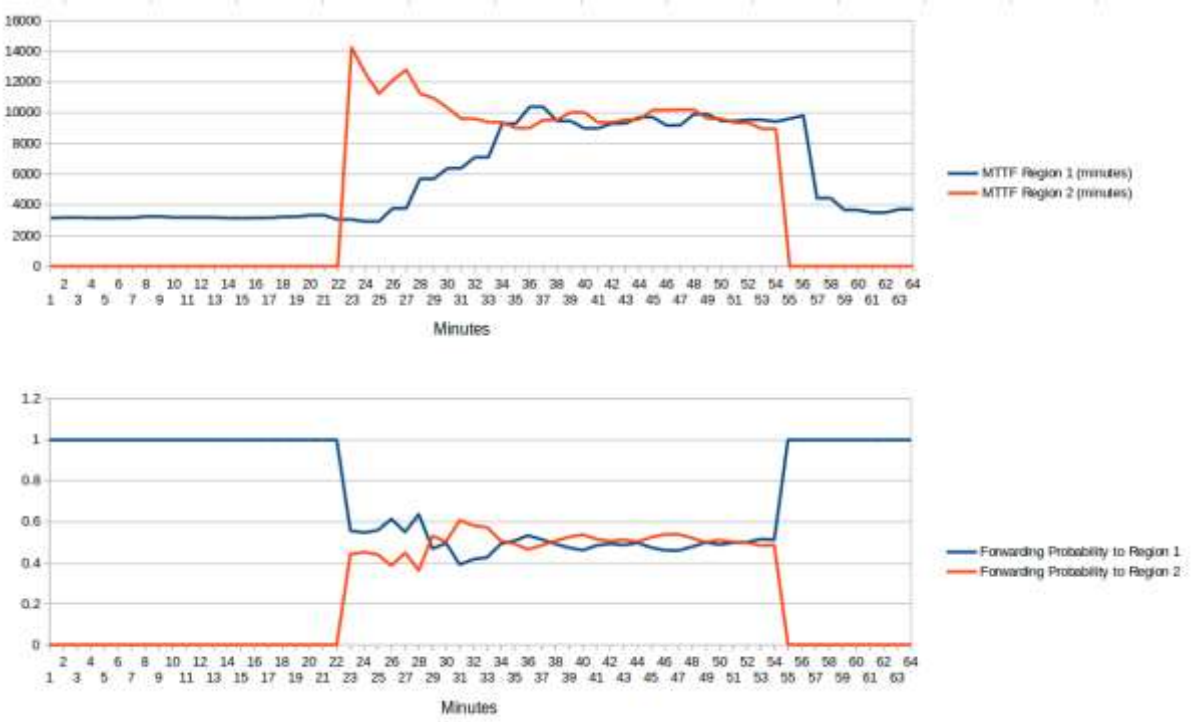

Figure 20: MTTF and Forward Probability, 2 regions, Constant rate (500 requests per sec)

In Figure 20 we report the experimental results with two regions (Munich and Frankfurt). Munich is on a private cloud and is the leader, Frankfurt is on public cloud (Amazon EC2). By this configuration, we are assessing the behaviour of Inter ACM on a hybrid cloud.

The top plot shows how the MTTF of the two regions varies over time, while the bottom plot shows the forward probability of the two regions. The forward probability is the probability that, given a certain workload and a certain accumulation of anomalies, at a given time the Inter ACM decides that another region is less loaded, and therefore clients' requests are redirected to them.

The load is generated by the same TPC-W clients which have been used in Section 6.2. The anomaly rates are the same as Section 6.2. We have configured TPC-W clients in order to issue 500 requests per second towards the transparently distributed application.

In this controlled experiment, we start with only one region (Munich, the leader) which is active. There are 3 couples of VM in this region, one subject to memory leaks, one subject to unterminated threads, one subject to both anomalies. The MTTF of the region is constant, because the workload is constant (we recall that the accumulation of anomalies, if the workload is constant, has an effect on the RTTF, not on the MTTF).

After 22 minutes of execution – during this time, VMs are rejuvenated, according to the results shown in Figure 15 – a second region is added (this region is Frankfurt, which is not the leader). Given that the workload is constant, in this controlled experiment the MTTF of the two regions increases, because now there are 6 couples of VMs (3 per each region) which are serving requests. Nevertheless, in the beginning the MTTF of Region 2 (Frankfurt) is higher than Munich' s, because fewer requests are coming to them. This is reflected in the forwarding probability: Inter ACM decides that fewer requests should be issued to Region 1,

while always more requests should be issued to Region 2. Around minute 34, the system reaches equilibrium: the forwarding probability is constant (around 50%, exactly because we have a constant workload and 2 regions), and the MTTF of the two regions becomes almost the same.

Around minute 55, the second region is removed from the system. This controlled experiment shows that the system is as well resilient to failures of entire regions, because the overall execution of the application does not change. The MTTF of Region 1 (Munich, the leader) converges to the initial value, again because the load is constant. The forwarding probability becomes 100% because Inter ACM knows that the only available region is Munich at that time.

These results show that Inter ACM is able to properly coordinate the behaviour of multiple regions. In particular, Inter ACM is able to promptly detect the state of entire regions, and its internal metrics quickly converge to the actual values, even in case of strong variations of the system (an entire region is removed). This allows for proactive scalability of Inter ACM on a larger scale than the experiments shown in Section 6.2. 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.

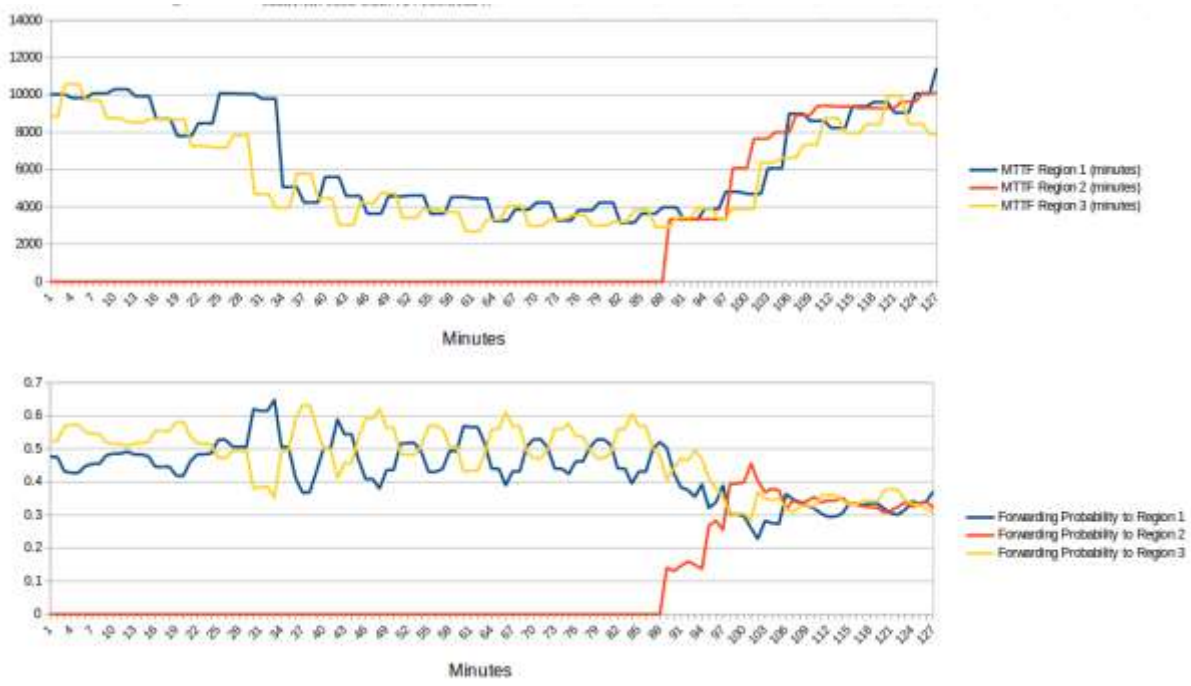

Figure 21: request rate, MTTF, and Forward Probability, 3 regions, variable rate

In Figure 21 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 experiment (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 experiment) the MTTF of the three regions starts to increase, as long as the forward probability grows. 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).

7 CONCLUSIONS

In this report, we present the implementation of Virtualization and Machine Learning Frameworks for enhancing the availability and performance of web based applications. We automatically generated Machine Learning (ML) models, based on monitoring a set of system features, during the off-line training phase. Given the large number of system features to be monitored, we use Lasso regularization techniques for selecting a subset of system features, while preserving low values of prediction errors.

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

Intra ACM controllers are interconnected over a distributed/geographical overlay network. In this way, multiple cloud regions can work together to increase the availability of the application, and to reduce the response time seen by the users of the application. In particular, the Intra ACM infrastructure allows to build public/hybrid distributed cloud architectures which are able to self-tune themselves so as to cooperate for reducing the response time and increase the availability. By relying on the same ML prediction models, the various Intra ACM controllers can redistribute the incoming load among the various cloud regions, so as to reduce at the same time the rejuvenation frequency.

We created a working prototype of a system that allows self-* properties (healing, rejuvenation, reconfiguring) and seamless application execution to be achieved. We realized a highly reliable web server by using a set of virtual machines.

The reported analytical analysis in Deliverable 3.1 [24], regarding the increasing the availability, which relies on reducing the Mean Time To Repair (MTTR) of server via proactive management of cloud recourses, is validated by the obtained results in cloud environment.

The results which have been provided in this deliverable emphasize that the experiments are perfectly repeatable, with prediction models (under the same workload and the same anomalies injection rate, on the same virtualized hardware) showing only small variations.

As well, Leader Election, during a run time, in a resilient to partitioning network in the presence of failures of Inter ACM is realized. This feature will permit the user requests to be redirected dynamically by the corresponding LB to the functioning regions. As a result, the response time for the users will be maintained under the required threshold and seamless execution of cloud applications will be provided.

This framework can be used for any cloud installation (not only web servers) that requires a large number of resources and has a high probability to fail during the run time.

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

8 REFERENCES

[1] **J. Fenn, "Gartner ' s Hype Cycle Special Report for 2011," 2011.**

[2] **IBM Corporation, "An architectural blueprint for autonomic computing .," *Quality*, vol. 36, no. June, p. 34, 2006.**

[3] **Transaction Processing Performance Council, *TPC Benchmark W, Standard Specification, Version 1.8*. Transaction Processing Perfomance Council, 2002.**

[4] **T. Bezenek, T. Cain, R. Dickson, T. Heil, M. Martin, C. McCurdy, R. Rajwar, E. Weglarz, C. Zilles, and M. Lipasti, "Characterizing a Java implementation of TPC-W," in *Proceedings of the Third Workshop On Computer Architecture Evaluation Using Commercial Workloads*, 2000.**

[5] **R. Tibshirani, "Regression Shrinkage and Selection Via the Lasso," *Journal of the Royal Statistical Society, Series B*, vol. 58, pp. 267–288, 1994.**

[6] **I. H. Witten, E. Frank, L. E. Trigg, M. A. Hall, G. Holmes, and S. J. Cunningham, "Weka: Practical machine learning tools and techniques with Java implementations," 1999.**

[7] **K. P. Murphy, *Machine Learning: a Probabilistic Perspective*. MIT Press, 2012.**

[8] **Y. Wang and I. H. Witten, "Inducing Model Trees for Continuous Classes," in *Procedings of the 9th European Conference on Machine Learning*, 1997, pp. 128–137.**

[9] **H. A. Chipman, E. I. George, and R. E. Mcculloch, "Extracting Representative Tree Models From a Forest," in *IPT Group, IT Division, CERN*, 1998, pp. 363–377.**

[10] **C. Cortes and V. Vapnik, "Support-Vector Networks," *Machine Learning*, vol. 20, no. 3, pp. 273–297, 1995.**

[11] **J. A. K. Suykens and J. Vandewalle, "Least Squares Support Vector Machine Classifiers," *Neural Processing Letters*, vol. 9, no. 3, pp. 293–300, 1999.**

[12] **W. D. Smith, "TPC-W: Benchmarking an ecommerce solution." 2000.**

[13] **MySQL, "MySQL database server." 2004.**

[14] **B. W. Kernighan and D. M. Ritchie, *The {C} Programming Language*, 2nd ed. Prentice Hall Professional Technical Reference, 1988.**

[15] **W. Stallings, *Operating Systems, Internals and Design Principles*. Prentice Hall, 1998.**

[16] **L. M. Silva, J. Alonso, and J. Torres, "Using Virtualization to Improve Software Rejuvenation," *IEEE Trans. Comput.*, vol. 58, no. 11, pp. 1525–1538, 2009.**

[17] **M. Patiño-Martínez, R. Jiménez-Peris, B. Kemme, and G. Alonso, "Scalable Replication in Database Clusters," in *Proc. of the 14th International Conference on Distributed Computing*, 2000, pp. 315–329.**

[18] **J. Gray, P. Helland, P. O' Neil, and D. Shasha, "The Dangers of Replication and a Solution," in *Proc. of the SIGMOD*, 1996, pp. 173–182.**

[19] **F. B. Schneider, *Replication management using the state-machine approach*. ACM Press/Addison-Wesley Publishing Co., 1993.**

[20] **D. R. Avresky and N. Natchev, "Dynamic reconfiguration in computer clusters with irregular topologies in the presence of multiple node and link failures," *IEEE Transactions on Computers*, vol. 54, no. 5, pp. 603–615, 2005.**

[21] **F. Salfner, M. Lenk, and M. Malek, "A Survey of Online Failure Prediction Methods," *ACM Computing Surveys*, vol. 42, no. 3, pp. 10:1–10:42, 2010.**

[22] **Deliverable D4.1 of PANACEA project: Description of feasible use case. March 2014**

[23] **Deliverable D2.1 of PANACEA project: Principles of Pervasive Monitoring. March 2014.**

[24] **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. Octover 2014.**

[25] **Deliverable 3.2: Machine Learning Framework and Global Architecture for Proactive Management. December 2014.**

[26] **Deliverable 2.4 : Architecture of the autonomic cloud with pervasive monitoring , June 2015.**

[27] **Deliverable D2.3: Autonomic Communication Overlay, March 2015.**