2016 IEEE International Parallel and Distributed Processing Symposium Workshops

Proactive Cloud Management for Highly Heterogeneous Multi-Cloud Infrastructures

Alessandro Pellegrini
pellegrini@dis.uniroma1.it
DIAG – Sapienza, University of Rome

Pierangelo Di Sanzo
disanzo@dis.uniroma1.it
DIAG – Sapienza, University of Rome

Dimiter R. Avresky
autonomic@irianc.com
IRIANC – Munich, Germany

***Abstract*—Various literature studies demonstrated that the cloud computing paradigm can help to improve availability and performance of applications subject to the problem of software anomalies. Indeed, the cloud resource provisioning model enables users to rapidly access new processing resources, even distributed over different geographical regions, that can be promptly used in the case of, e.g., crashes or hangs of running machines, as well as to balance the load in the case of overloaded machines. Nevertheless, managing a complex geographically-distributed cloud deploy could be a complex and time-consuming task. Autonomic Cloud Manager (ACM) Framework is an autonomic framework for supporting proactive management of applications deployed over multiple cloud regions. It uses machine learning models to predict failures of virtual machines and to proactively redirect the load to healthy machines/cloud regions. In this paper, we study different policies to perform efficient proactive load balancing across cloud regions in order to mitigate the effect of software anomalies. These policies use predictions about the mean time to failure of virtual machines. We consider the case of heterogeneous cloud regions, i.e regions with different amount of resources, and we provide an experimental assessment of these policies in the context of ACM Framework.**

I. Introduction

The presence of software anomalies—such as memory leaks and/or unterminated threads— may be a major problem affecting performance and availability of computing applications. The study presented in [1] showed that software errors are the cause of around 40% of failures in web applications. Given the size and complexity of many modern application deployments, identifying and fixing software anomalies may be a long, costly, and burdensome task. To cope with this problem, some literature studies proposed techniques based on software rejuvenation [2], [3]. These techniques detect the effects due to accumulation of software anomalies by means of monitoring agents, which trigger specific actions to force software rejuvenation (e.g. process/system restart).

The modern cloud computing paradigm [4] offers the possibility to access virtualized computing resources on demand. Some studies demonstrated that virtualization and cloud computing can be exploited to improve availability and performance of applications subject to the problem of software anomalies [5], [6]. Basically, this can be done by means of smart strategies that use spare virtual machines to promptly replace active virtual machines before crashing, or whose performance has deteriorated, due to the accumulation of anomalies. However, when considering a large-scale application deployment on the cloud, the problem of managing accumulation of anomalies becomes more complex, given that additional factors, such as the presence of resources distributed over multiple cloud regions, are involved.

A recent literature study presented Autonomic Cloud Manager (ACM) Framework [7], a proactive machine learning (ML)-based framework that manages distributed deployments of client-server applications on multiple cloud regions. Advantages provided by ACM Framework are: (a) it automatically enforces software rejuvenation of a virtual machine (VM) which is approaching a failure and activates a spare healthy VM that takes its place; b) it allows replicas of VMs to be distributed over multiple cloud regions, while automatically handles and forwards incoming requests from clients to VMs of the different regions; c) it uses a proactive load balancing approach to distribute client requests to more-healthy regions on the basis of failure frequency of VMs.

We note that an application deployment over different geographical regions allows to improve availability (e.g. in front of a failure of an entire data center in a region). Even, in some case, applications might require to be instantiated on an hybrid cloud system, which typically involves multiple cloud regions. This may be required, e.g., when a portion of the data should not be disclosed on a public infrastructure. As the latter point, Gartner [8] foresees that in the near future, 51% of cloud-deployed applications will in fact exploit hybrid cloud infrastructures.

Nevertheless, we note that when using different cloud regions (notably when they belong to different cloud providers), they could be heterogeneous in terms of available resources (e.g. number of virtual machines, cpu type and memory size). These differences may have an impact on the effectiveness of workload distribution policy in ACM Framework. We emphasize that using different and heterogeneous cloud regions, as well as a multi-cloud infrastructures, could be as well a strategic decision. For example, different cloud providers offer various types of VMs at different costs. Also, the cost of VMs of the same cloud provider may change depending on the geographical region where they are located. Therefore, it could be more convenient to have more VMs in some regions, or of a given provider, rather than in/of other ones.

In this paper, we extend the previous results in [7] by studying different policies for the load balancing problem in ACM Framework. We remark that this problem arises from the need of distributing the workload in order to balance the effects of VM failures (and the subsequent overhead due to rejuvenation of VMs) over the different cloud regions where the application is deployed. We note that the heterogeneity of

978-1-5090-3682-0/16 $31.00 © 2016 IEEE
DOI 10.1109/IPDPSW.2016.124

regions is likely to exacerbate the load imbalance, thus leading to scenarios with (highly) overloaded/underloaded regions.

We present an experimental comparison of the load balancing policies in the case of two hybrid cloud infrastructures composed of two and three regions, respectively. In our experiments, we used two regions hosted in Amazon EC2 and one region hosted in a private infrastructure.

The remainder of this paper is structured as follows. In Section II we discuss related work. Section III presents an overview of ACM Framework. The load balancing policies for heterogeneous multi-cloud environments that we analyse are described in Section IV. Finally, the experimental data and the assessment of policies are discussed in Section VI.

II. Related Work

Load balancing in cloud computing [9], [10], [11] is a fundamental topic, and it has been studied in the literature from different aspects, such as scalability, generated overhead, energy consumption and carbon emission perspective. Our work, similarly to [12], specifically adds to all these aspects the issue of availability/dependability of applications hosted on a cloud environment. Differently from [12], we explicitly tackle the case of geographically-distributed multi-cloud (hybrid) environments.

The works presented in [13], [14], [15] specifically tackle the issue of load sharing from a SLA point of view. The work in [13] explicitly relies on a self-control loop to monitor the state of virtual machines. ACM Framework keeps the ability to control load balancing from a SLA-compliance perspective, as well to reduce the response time experienced by end users and to improve system availability.

In [16], [17] the authors propose to use a mixture of simulation and machine learning to study optimal deploys of cloud-based in memory applications. ACM Framework keeps the ability offered by these proposals and offers the possibility to modify the deploy at runtime in case the workload conditions change during the lifetime of the system.

The works in [18], [19] address the issue of resource allocation in a cloud environment trying to maximize the usage of available resources and to improve the efficiency. We keep these abilities, while adding the possibility to migrate incoming (remote) workload, so as to decrease response time and reduce and increase availability/dependability.

In [20], the authors present a middleware infrastructure to automate the deployment of large-scale service compositions. Our framework allows as well to transparently deploy applications in distributed (hybrid) clouds, while monitoring their runtime behavior to increase the availability/dependability.

In [21], [22], machine-learning techniques are used to learn workload indices to dynamically schedule jobs. The indices combine information from the key resources of contention: CPU, disk, network, and memory. We exploit machine-learning techniques as well to learn from the same features, but we used the learned information to determine what is the best configuration of the various cloud regions in terms of active VMs and incoming requests.

The works in [23], [24] target cloud computing environments using ML-based prediction models to self-tune the runtime configuration of the applications being run on the virtualized infrastructure. Differently from our proposal, these works do not explicitly consider multiple cloud regions. Moreover, their principal target is the performance of applications, rather than their availability and dependability.

In the work in [25], the possibility to add/remove VMs dynamically at runtime to account for load changes is explored in the context of MMOGs games. We have this same capability in our system, but we target generic applications in an agnostic fashion.

III. Overview of the ACM Framework

As we discussed, ACM Framework is designed for applications based on a client-server model, where the server can be replicated over different machines. ACM Framework builds on top of the F^{2}PM framework [26] and of the PCAM [6] framework. F^{2}PM is designed to build ML-based prediction models which are completely *agnostic* of the running application. During an initial phase, the system under monitoring (namely a VM running a server replica) runs the application and a thin software client which measures a large set of system features, such as memory usage, CPU time, and swap space usage. This information is transferred to a feature monitor agent. This agent builds a database of system features, for later usage by the ML algorithms. Under the accumulation of software anomalies, these measures are subject to change over time. The user of F^{2}PM can set several constraints which, altogether, define the failure point of the system. This failure point is not necessarily related to an actual crash of the system, rather it can describe as well the violation of one or more SLA (e.g. the average response time overcomes a given threshold). All measurements are fed into an automatic ML toolchain. The goal of this toolchain is to generate and validate alternative ML models for predicting the Remaining Time To Failure (RTTF), as well as to select (via Lasso regularization [27]) what are the most relevant system features to be used by these models. This selection allows to reduce the amount of information to be managed when the system is operational. The user of F^{2}PM is provided as well with a series of metrics which allow to select which is the most effective ML model to be used for predictions. F^{2}PM supports several ML models, namely Linear regression [28], M5P [29], REP-Tree [30], Lasso as a predictor [27], Support-Vector Machine (SVM) [31], and Least-Square Support-Vector Machine [32].

The ML-based prediction models generated by F^{2}PM are then used by PCAM to enforce proactive rejuvenation of a VM before it reaches a failure point. Indeed, these models allow PCAM to estimate the RTTF, i.e. the time after which a crash will happen or a SLA will be violated due to the accumulation of anomalies. PCAM keeps some VMs hosting server replicas in the ACTIVE state, while others VMs in the STANDBY state. The state of a VM is controlled by a Virtual Machine Controller (VMC). VMC maps a ML model to a given VM, and uses the system features selected by Lasso regularization for training the ML model to predict, at runtime, the RTTF of the VM. Whenever the estimated RTTF of an ACTIVE VM is less than a threshold (established by the user), VMC sends an ACTIVATE command to a VM in the STANDBY state and

a REJUVENATE command to the about-to-fail VM. In this way, availability of a server replica is ensured by prompt and proactive takeover of an anomaly-free VM.

PCAM targets as well transparency towards the user of the application at the level of a single cloud region, namely the virtualization infrastructure where all the VMs managed by a VMC are hosted. In fact, all the requests issued by remote clients of the system are directed to VMC, which hosts a load balancer. The goal of this component is to balance the load associated to client requests to VMs in the ACTIVE state.

ACM, as shown in Figure 1, brings all the capabilities of PCAM to a geographically-distributed network of VMs. In particular, several VMCs instances are in charge of monitoring a cloud region each. As mentioned before, a cloud region includes a set of VMs hosted by a single cloud provider or a single virtualization infrastructure in a given geographic location. To maximize the dependability and to reduce the response time, the interconnection among the various controllers is actuated via an overlay network, which selects the path with the smallest latency among two given controllers, and is able to reroute connections in case of a network link failure. Among all the regions VMCs, a leader VMC is automatically elected using the algorithm in [33], which has been shown to be tolerant to multiple nodes and link failures.

As shown in [7], the failure and rejuvenation rate of MVs can have a non-negligible impact on the performance of a region, and consequently on the response time experienced by end-users. Further, some cloud regions could be more overloaded than others. This may be due to: a) the different number of clients than could be connected to any region, and b) the heterogeneity of regions in terms of available resources. Under these circumstances, some regions could be overloaded with respect to others, and the rate of anomaly accumulation of these regions could be therefore likely higher. For these reason, ACM Framework requires smart policies to determine how to balance the load across regions.

IV. Policies for Load Balancing in Highly Heterogeneous Environments

In this section, we present the policies that we evaluate in our study. These policies aim at performing efficient proactive load balancing across cloud regions in order to avoid that different failure and rejuvenation rates in different regions lead to overloaded and underloaded regions. Ultimately, the goal of these policies is to ensure that all active VMs in all regions show the same Mean Time To Failure (MTTF) in front of the heterogeneity of regions in terms of number and computing power of VMs.

In ACM Framework, the MTTF of VMs hosted in a cloud region is estimated by the ML models. The VMC of a region i periodically sends to the leader VMC the last average value of the Region Mean Time To Failure (RMTTF), say $lastRMTTF_i$, calculated as the average MTTF of all active VMs in the region i. When the leader VMC receives $lastRMTTF_i$ at time t, the current RMTTF of the region i, say $RMTTF_i^t$, is (re-)calculated by using the following weighted average:

$$RMTTF_i^t = (1-\beta) \cdot RMTTF_i^{t-1} + \beta \cdot lastRMTTF_i, \quad (1)$$

where $RMTTF_i^{t-1}$ is the previous value of RMTTF and $0 \leq \beta \leq 1$.

The goal of the policies is therefore to decide the fraction f_i of global incoming requests to be forwarded to a cloud region i to ensure that the different values of the current RMTTF of all regions converge (fast) to the same value.

A. Policy 1: Sensible Routing

The first policy that we study is called *sensible routing*, and is based on the work presented in [34].

Assuming to have N cloud regions, the fraction f_i of global incoming requests to be forwarded to cloud region i is calculated as:

$$f_i = \frac{RMTTF_i^t}{\sum_{j=1}^{N} RMTTF_j^t}. \quad (2)$$

Intuitively, by using this policy, the fraction of requests forwarded to a region i is proportional to the weight of the current RMTTF of the region over the sum of the last RMTTF of all regions.

B. Policy 2: Available Resources Estimation

This policy uses a single numeric parameter as an abstraction to quantify the amount of available resources in a region. It assumes that resources are linearly consumed by the accumulation of anomalies over time (therefore by the incoming requests). Accordingly, the *estimation* of the amount of available resources a in region i is calculated as:

$$Q_i = RMTTF_i^t \cdot f_i \cdot \lambda \quad (3)$$

where λ is the global incoming request rate, thus $f_i \cdot \lambda$ is the incoming request rate of region i. The above estimation is based on the idea that if a region shows a higher RMTTF in front the same amount of received requests, then the amount of available resources in that region is higher. Similarly, if the region receives more requests in front the same RMTTF, then the amount of available resources in that region is higher.

The fraction of requests to be forwarded to region i is calculated as as:

$$f_i = \frac{Q_i}{\sum_{j=1}^{N} Q_j}. \quad (4)$$

Basically, with this policy, the fraction of requests forwarded to a region i is proportional to the current amount of estimated resources of the region over the sum of the amount of estimated resources of all regions.

C. Policy 3: Exploration

The third policy uses an exploration strategy, as it is inspired to the hill climbing [28] search algorithm. This policy calculates the Average RMTTF (ARMTTF) over all regions, i.e.:

$$ARMTTF = \frac{\sum_{i=1}^{n} RMTTF_i^t}{N}. \quad (5)$$

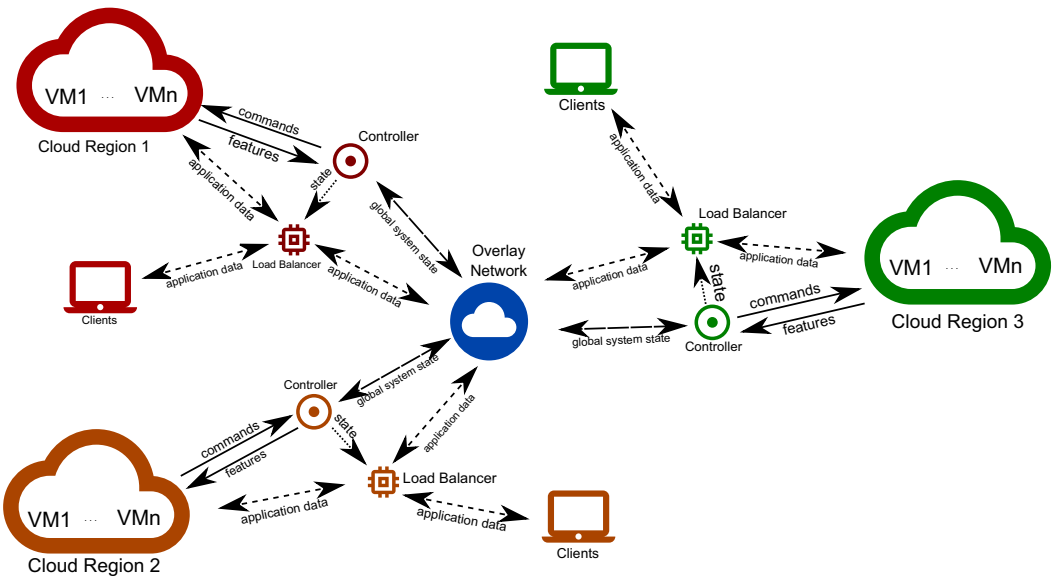

Fig. 1. ACM framework organization (Figure taken from [7])

Then, all regions for which $RMTTF_i^t > ARMTTF$ get their current value f_i decreased, while the regions with $RMTTF_i^t < ARMTTF$ get their value f_i increased.

The policy therefore selects all the regions such that $RMTTF_i < ARMTTF$ (which we call the set of overloaded regions $OL = \{i : RMTTF_i < ARMTTF\}$), and for each of these regions it computes the new value of the fraction f_i, say f_i^{next} as:

$$f_i^{next} = \frac{RMTTF_i}{ARMTTF} \cdot f_i \cdot k \tag{6}$$

where k is a constant scaling factor. Of course, the equality $\sum_{i=1}^{n} f_i = 1$ must hold. To this end, it must be ensured that any portion taken out of some f_i must be added to some f_j, $i \neq j$. Then, the policy computes the *total variation of the flow of overloaded regions*:

$$\Delta f^{<} = \sum_{i \in UL} (f_i^{next} - f_i) \tag{7}$$

Then it selects all the regions such that $RMTTF_i > ARMTTF$ (which we call the set of underloaded regions $UL = \{i : RMTTF_i > ARMTTF\}$), and for each of these regions it updates the workload fraction as:

$$f_i^{next} = \frac{\Delta f^{<}}{\sum_{i=1}^{N} RMTTF_i} \cdot f_i \cdot k \tag{8}$$

where k is the same scaling factor.

Summarizing, all fractions are calculated as:

$$f_i^{next} = \begin{cases} \dfrac{RMTTF_i}{ARMTTF} \cdot f_i \cdot k & \text{if } RMTTF_i < ARMTTF \\ \dfrac{\Delta f_{<}^t}{\sum_{i=1}^{n} RMTTF_i} \cdot f_i \cdot k & \text{otherwise} \end{cases} \tag{9}$$

V. The ACM Framework Closed Control Loop

ACM Framework adopts a control strategy based on a *closed loop*. ACM Framework assumes that a user can arbitrarily connect to whichever cloud region. Each region has a load balancer (LB) to which users send requests. In order to achieve that any region i processes the established fraction of request f_i over the global incoming requests, ACM Framework uses a *global forward plan*. After that the fraction f_i of requests that each region should process has been calculated, this plan establishes the fractions of requests that are sent from users to the LB of a region that have to be forwarded to the local region and to be forwarded to LBs of other regions. The plan is updated at each step of the closed control loop.

The closed control loop includes the four states reported in Figure 2. Initially, the system enters the Monitor state. In this state, the system features are collected by each VMC in a region according to the distributed organization of the F^{2}PM framework. These results are then fed to the ANALYZE() routine, which is depicted in Algorithm 1.

The execution of Algorithm 1 brings the system into the Analyze state. The operations associated with this state differentiate between an execution on the leader VMC and on the slave VMCs. In particular, every VMC—both the leader and the slaves—apply the ML-based prediction models offered by F^{2}PM to determine the $RMTTF_i$ of the local region. Then, all slave VMCs send their $RMTTF_i$ values to the leader

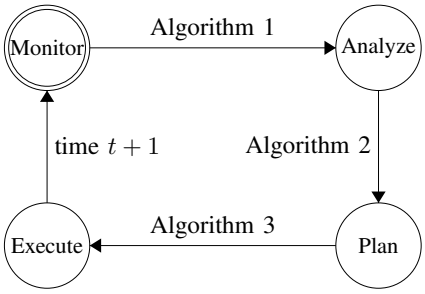

Fig. 2. Control Loop Flow Diagram driven by ML prediction models for Hybrid Clouds

Algorithm 1 Analyzing the Distributed Deploy

```
procedure ANALYZE( )
    Predict local RMTTF using ML-based models
    if current VMC is leader then
        collect all RMMTF_i from slave VMCs
    else
        send local RMTTF to leader VMC
    end if
    Actuate PCAM policies
end procedure
```

VMC. Additionally, all VMCs actuate PCAM policies locally which depend on the ML-based prediction models.

When all the values of $RMTTF_i$ are received by the leader VMC, the system transitions to the Plan state. This state is associated with the execution of Algorithm 2, which takes place only at the leader VMC. The goal of this algorithm is to use all the information gathered from the distributed regions to determine a global current state of the system. In particular, Algorithm 3 issues a call to a POLICY() function, which is in charge of using one of the three policies presented in Section IV. The selected policies can be specified at configuration time. Therefore, the POLICY() function determines the new value of f_i^t, for all the regions i in the system. The new values f_i^t are then sent back to the local slave VMCs.

The system then moves to the Execute state. The goal of this state, which is associated with Algorithm 3, is to make the workload forward plan devised in Algorithm 2 persistent on all the geographically-distributed cloud regions.

Moreover, during the execution of this Algorithm, each local VMC controller uses the ML-based prediction models offered by F^{2}PM to determine, via correlation analysis, whether the clients directly connected to the region are experiencing a Response Time which is over a pre-defined threshold. In this case, the system adds new VMs to the pool, so as to reduce it. In particular, when the global workload increases, the failure rate of VMs in one or multiple cloud regions may increase, so that excessive performance loss and low availability may be experienced by clients. As a countermeasure to this issue, ACM can proactively change the number of active VMs in each cloud region. If the RMTTF of a cloud region becomes less (more) than a given threshold, then the local controller can activate news VM (deactivate some active VMs) by using MTTF prediction models to evaluate the expected RMTTF as a result of the VM activation (deactivation).

Algorithm 2 Planning Autonomic Actions on the Deploy

```
executed only by leader VMC
procedure PLAN( )
    for i ∈ CloudRegions do
        f_i^t ← POLICY(f_i^{t-1}, RMTTF_1, ..., RMTTF_n)
    end for
    send to all slave VMCs the associated f_i^t
end procedure
```

Algorithm 3 Executing Autonomic Actions on the Deploy

```
1: procedure EXECUTE( )
2:     if slave VMC then
3:         receive f_i^t from leader VMC
4:     end if
5:     install new f_i^t in the load balancer
6:     if Predicted Response Time > threshold then
7:         ADDVMS( )
8:     end if
9: end procedure
```

After the execution of EXECUTE() completes, the system enters again the Monitor state, and the time era t is incremented to the next one.

VI. EXPERIMENTAL RESULTS

A. Benchmark Setup

To assess the presented policies, and to compare their effectiveness, we conducted an experimental study using a hybrid cloud architecture. We used three cloud regions: Region 1, hosted in the Ireland Region of Amazon EC2, Region 2, hosted in the Frankfurt Region of Amazon EC2, and Region 3, privately hosted in a 32-cores HP ProLiant server with 100 GB RAM, located in Munich (Germany). We used 6 `m3.medium` Amazon EC2 instances in Region 1, 12 `m3.small` Amazon EC2 instances in Region 2, and 4 VMs equipped with 2 virtual CPU cores, 1 GB or RAM, and 4 GB of virtual disk space in Region 3. The HP ProLiant server was equipped with VMware Workstation 10.4 as the hypervisor. All VMs were equipped with Ubuntu 10.04 Linux Distribution (kernel version 2.6.32-5-amd64).

The test-bed application was the TPC-W benchmark [35], a multi-tier e-commerce web application that simulates an online store. We used a Java implementation of TPC-W [36] developed using servlets, and relying on MySQL [37] as a database server. Clients were emulated using emulated web browsers to generate requests according to TPC-W specifications. We modified the TPC-W implementation to randomly generate software anomalies at run-time, including memory leaks and unterminated threads. Specifically, anomalies were generated with different probabilities on each VM when receiving a client request—10% of requests generate a memory leak, 5% of requests generate an unterminated thread. This led to scenarios where each VM (thus each cloud region) showed different anomaly occurrence patterns. We varied the number of active clients (towards each cloud region) in the interval $[16, 512]$, ensuring that the clients connected to each cloud region (thus, to the VMC on each cloud region) where significantly different in number. Based on our previous results

in [26], we selected REP Tree as a ML model for predicting the MTTF.

B. Experimental Data

To assess the validity of our policies, we run two different experiments, one with two regions, and one with three regions. The first experiment evaluates all the three policies on a geographically-distributed hybrid cloud environment composed of Region 1 and Region 3, namely using Amazon VMs in Ireland and privately-hosted VMs in Munich.

For each policy, Figure 3 shows the variation over time of: a) the RMTTF of each region, b) the calculated fraction f_i for each region, and c) the average response time measured by all clients. By the results, we can see that the three policies show different behaviours. In particular:

- With Policy 1, the values of the RMTTF of the two regions do not converge. This can be seen by the fact that the two RMTTF stabilize to different values. Further, the values of f_i are subject to oscillations.
- Policy 2 performs better. The values of the RMTTF converge quite quickly, and f_i shows less-oscillating values. We remark that Policy 2 explicitly takes into account an estimation of the amount of available resources on each region.
- Policy 3 is able to converge better than Policy 1, however the values of RMTTF and f_i are less stable with respect to Policy 1.

In all cases, the average response time measured at the clients is kept below the threshold of 1 second, and its variations are not highly affected by some policy more than others.

A more complex scenario is reported in Figure 4, where all three regions are used. This experiment confirms that with Policy 1 the RMTTF does not converge. In particular, the values of the RMTTF continue to oscillate, and so do the values of f. This, in turn, causes many redirections of the request flow between regions, which generates additional overhead in the system. Contrarily, both Policy 2 and 3 are able to cope with the heterogeneity of regions, given that the RMTTF converges in both cases. Policy 2 converges more quickly, although it produces values of f_i that are slightly more oscillating than Policy 3. For the sake of brevity, we do not report the response time measured at the clients for the case of 3 regions, because it is similar to the results shown in Figure 3.

Overall, we can conclude that Policy 1, based on the sensible routing, is more suitable for less-heterogeneous environments, as already reported in [7]. On the contrary, when heterogeneity is very high, the quickest convergence and the most stable results are provided by Policy 2, which is based on explicit available resources accounting. Exploration approaches, such as Policy 3, are similarly valid, yet they can suffer more from their intrinsic randomness.

VII. Conclusions

In this paper we analyzed different policies to balance the workload across heterogeneous cloud regions. Particularly, we focused on the case of a large-scale deployment on heterogeneous cloud regions of applications subject to software anomalies. We study the different policies in the context of ACM Framework, which uses ML models to predict the MTTF of machines that run server replicas of an application. All load balancing policies that we studied rely on MTTF predictions, and they aim at ensuring that all active VMs in all regions show the same MTTF in front of the heterogeneity of regions. By the results, the policy which explicitly takes into account the estimation of the amount of resources in all cloud regions has been proven to show the fastest convergence and the highest stability.

Acknowledgements

The research presented in this paper has been supported by the European Union via the EC funded project PANACEA, contract number FP7 610764.

References

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

[2] K. Vaidyanathan and K. Trivedi, "A comprehensive model for software rejuvenation," *IEEE Transactions on Dependable and Secure Computing*, vol. 2, no. 2, 2005.

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

[4] B. Hayes, "Cloud computing," *Communications of the ACM*, vol. 51, no. 7, p. 9, jul 2008.

[5] 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. [Online]. Available: http://dx.doi.org/10.1109/TC.2009.119

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

[7] D. R. Avresky, P. Di Sanzo, A. Pellegrini, B. Ciciani, and L. Forte, "Proactive Scalability and Management of Resources in Hybrid Clouds via Machine Learning (short paper)," in *Proceedings of the 14th IEEE International Symposium on Network Computing and Applications*. Boston, MA, USA: IEEE Computer Society, 2015.

[8] J. Fenn, "Gartner' s Hype Cycle Special Report for 2011," Tech. Rep. August, 2011. [Online]. Available: http://www.gartner.com/technology/research/hype-cycles/index.jsp

[9] R. Rajan and V. Jeyakrishnan, "A Survey on Load Balancing in Cloud Computing Environments," *Ijarcce.Com*, vol. 2, no. 6, pp. 4726–4728, 2013.

[10] A. Khiyaita, H. E. L. Bakkali, M. Zbakh, and D. E. Kettani, "Load balancing cloud computing: State of art," *Network Security and Systems (JNS2), 2012 National Days of*, pp. 106 – 109, 2012.

[11] N. J. Kansal and I. Chana, "Cloud Load Balancing Techniques : A Step Towards Green Computing," *IJCSI International Journal of Computer Science Issues*, vol. 9, no. 1, pp. 238–246, 2012.

[12] C. Zenon, M. Venkatesh, and A. Shahrzad, "Availability and Load Balancing in Cloud Computing," *International Conference on Computer and Software Modeling IPCSIT vol.14 (2011) IACSIT Press, Singapore*, vol. 14, pp. 134–140, 2011.

[13] T. Aubonnet and N. Simoni, "Self-Control Cloud Services," in *Network Computing and Applications (NCA), 2014 IEEE 13th International Symposium on*, 2014, pp. 282–286.

[14] R. Lee and B. Jeng, "Load-balancing tactics in cloud," *Proceedings - 2011 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery, CyberC 2011*, pp. 447–454, 2011.

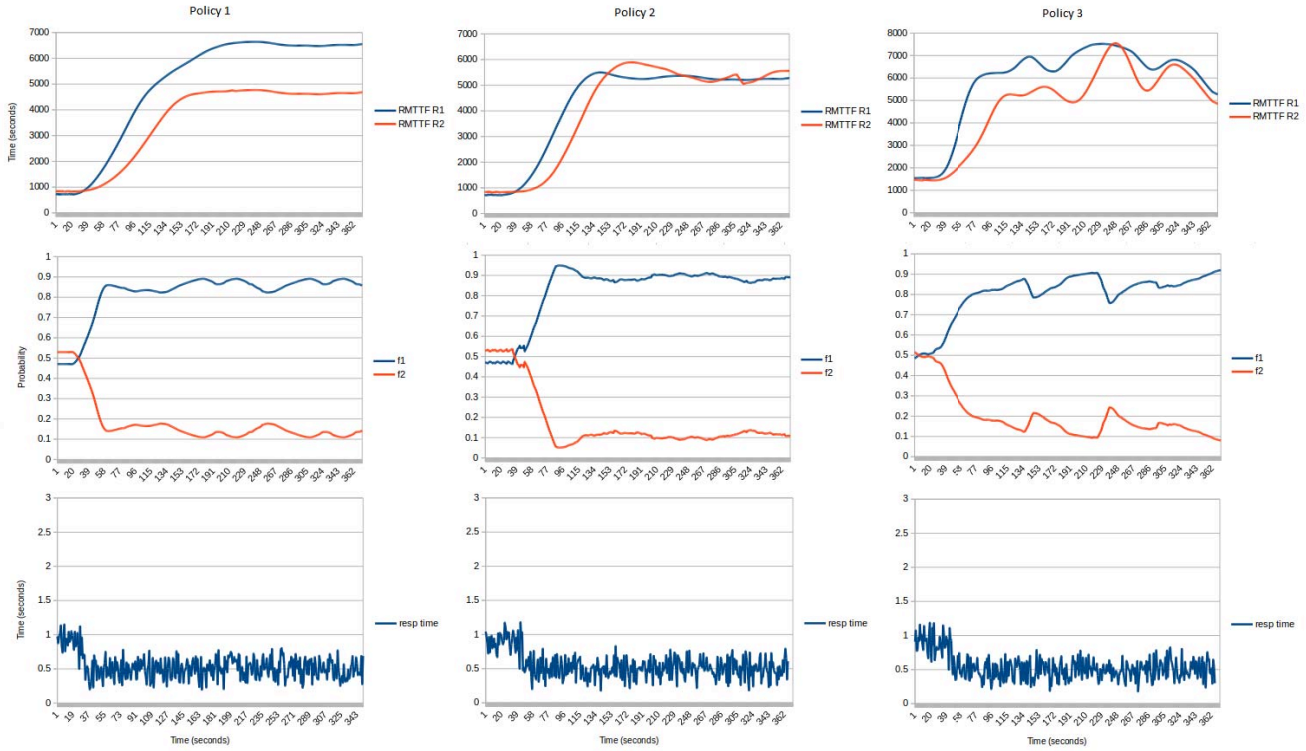

Fig. 3. Results using 2 regions. First row shows RMTTF, second row shows the workload factor f_i, third row shows the response time measured by the clients of the system.

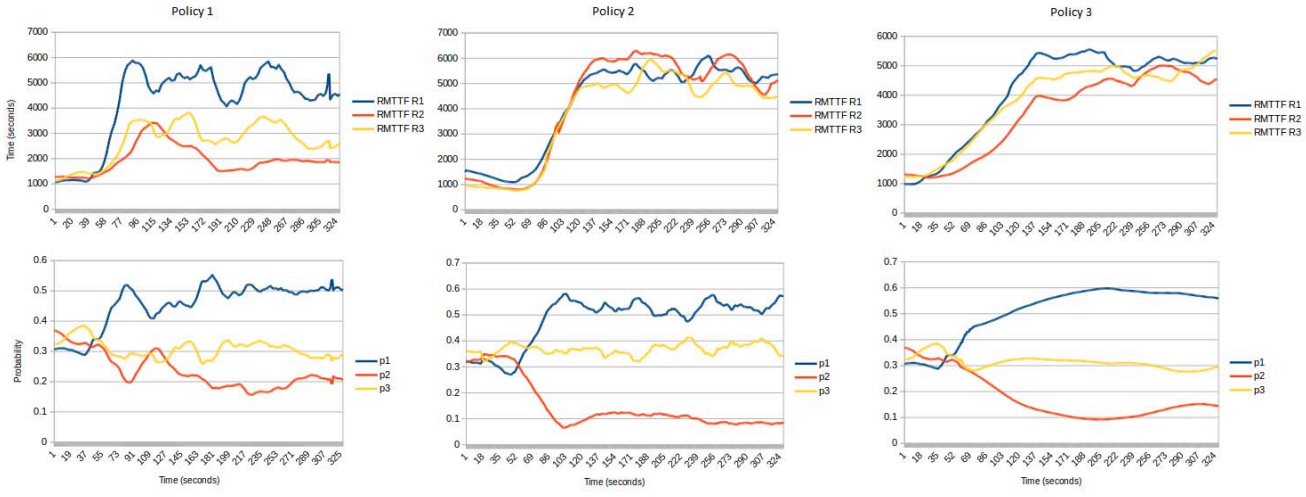

Fig. 4. Results using 3 regions. First row shows RMTTF, second row shows the workload factor f_i, third row shows the response time measured by the clients of the system.

[15] S. Goswami and A. D. Sarkar, "A Comparative Study of Load Balancing Algorithms in Computational Grid Environment," in *2013 Fifth International Conference on Computational Intelligence, Modelling and Simulation*, vol. 1, no. 2. IEEE, sep 2013, pp. 99–104.

[16] P. Di Sanzo, F. Antonacci, B. Ciciani, R. Palmieri, A. Pellegrini, S. Peluso, F. Quaglia, D. Rughetti, and R. Vitali, "A Framework for High Performance Simulation of Transactional Data Grid Platforms," in *Proceedings of the 6th International ICST Conference on Simulation Tools and Techniques*, ser. SimuTools '13. ICST, Brussels, Belgium, Belgium: ICST, 2013, pp. 63–72.

[17] P. Di Sanzo, F. Quaglia, B. Ciciani, A. Pellegrini, D. Didona, P. Romano, R. Palmieri, and S. Peluso, "A flexible framework for accurate simulation of cloud in-memory data stores," *Simulation Modelling Practice and Theory*, vol. 58, pp. 219–238, nov 2015.

[18] S. Rampersaud and D. Grosu, "A Sharing-Aware Greedy Algorithm for Virtual Machine Maximization," in *IEEE 13th International Symposium on Network Computing and Applications*, ser. NCA, 2014, pp. 113–120.

[19] G. Xu, J. Pang, and X. Fu, "A load balancing model based on cloud partitioning for the public cloud," *Tsinghua Science and Technology*, vol. 18, no. 1, pp. 34–39, 2013.

[20] L. A. F. Leite, C. E. M. D. Santos, D. Cordeiro, M. A. Gerosa, and F. Kon, "Deploying Large-Scale Service Compositions on the Cloud with the CHOReOS Enactment Engine," in *IEEE 13th International Symposium on Network Computing and Applications*, ser. NCA, 2014, pp. 121–128.

[21] P. Mehra and B. W. Wah, "Automated Learning of Workload Measures for Load Balancing on a Distributed System," *1993 International Conference on Parallel Processing - ICPP'93*, vol. 3, 1993.

[22] M. Randles, D. Lamb, and A. Taleb-Bendiab, "A comparative study into distributed load balancing algorithms for cloud computing," *24th IEEE International Conference on Advanced Information Networking and Applications Workshops, WAINA 2010*, pp. 551–556, 2010.

[23] P. Di Sanzo, D. Rughetti, B. Ciciani, and F. Quaglia, "Auto-tuning of Cloud-Based In-Memory Transactional Data Grids via Machine Learning," in *2012 Second Symposium on Network Cloud Computing and Applications*, ser. NCCA. IEEE, dec 2012, pp. 9–16.

[24] P. Di Sanzo, F. M. Molfese, D. Rughetti, and B. Ciciani, "Providing Transaction Class-Based QoS in In-Memory Data Grids via Machine Learning," in *2014 IEEE 3rd Symposium on Network Cloud Computing and Applications*, ser. NCCA. IEEE, feb 2014, pp. 46–53.

[25] A. P. Negralo, M. Adaixo, L. Veiga, and P. Ferreira, "On-Demand Resource Allocation Middleware for Massively Multiplayer Online Games," in *2014 IEEE 13th International Symposium on Network Computing and Applications*, 2014, pp. 71–74.

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

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

[28] E. Alpaydin, *Introduction to Machine Learning*, 3rd ed., 2014.

[29] 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.

[30] 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.

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

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

[33] 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.

[34] L. Wang and E. Gelenbe, "Adaptive Dispatching of Tasks in the Cloud," *IEEE Transactions on Cloud Computing*, vol. pp, no. 1, pp. 1–1, jan 2015.

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

[36] 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.

[37] MySQL, "MySQL database server," http://www.mysql.com, 2004.