Elasticity Management for Virtual Machine Fault Recovery

Kleber Divino, Carlos Kamienski

Universidade Federal do ABC (UFABC)
kleber.divino@gmail.com, cak@ufabc.edu.br

Abstract *In cloud computing, virtual machine level elasticity and fault tolerance are two mechanisms with different purposes but similar features, which may generate an overlapping of responsibilities, additional complexity and increased usage of resources. This paper advocates that a cloud environment with existing elasticity mechanisms does not need additional fault tolerance for recovering failures at a virtual machine level. We deployed a private cloud with elasticity and recovery mechanisms and undertook several experiments, both analyzed together and separately. Our results confirmed our hypothesis revealing that for different metrics the use of elasticity only is equivalent to the use of recovery only. Also, the use of both mechanisms brings only a small improvement.*

1 Introduction

In cloud computing, resources can dynamically adapt according to workload variations and users pay according to what they really use [Armbrust 2010]. Elasticity is the key feature in cloud environments for providing predictable performance by changing automatically the amount of resources allocated to an application to adapt to workload variations in order to match demand at any point in time [Mell 2011]. Also, fault tolerance techniques have been historically used for enabling systems to operate continuously, even in the case of failures. Fault tolerance is a mechanism used to avoid a service to fail to provide the established requirements in case of failure using redundant resources [Avizienis 2004]. Here we use the terms VM (virtual machine) recovery management or simply fault recovery for the specific class of fault tolerance systems that we deal with.

Elasticity and fault tolerance at the level of virtual machines are two mechanisms aimed at different purposes, but with similar features, which may generate an overlapping of responsibilities, additional complexity and increased usage of processing, storage and networking resources. Our main hypothesis in this paper is that elasticity for VM management is actually a superset of fault recovery at VM level and therefore there is no need of an additional fault tolerance mechanism to be put in place only for that purpose.

Elasticity mechanisms in cloud computing create new resources, e.g. virtual machines, whenever existing ones are not able to provide adequate performance to applications. Recovery mechanisms at the VM level within a fault tolerance framework create new resources for replacing existing ones that were subject to failure in order not to allow applications to experience reduced performance. In some environments, when a

component fails the remaining ones maintain the service working, but with graceful degradation. The monitoring system detects that a component failed and creates a new one. In other words, fault recovery fixes VMs in order not to incur performance penalties. For elasticity management, the trigger is performance, thus a VM failure generates an unbalance in performance that will be understood as an increase in demand and a new VM will be created. In conclusion, elasticity management may be considered a superset of VM recovery management because the former can deal with both failure and demand variation, whereas the latter can only deal with failure.

This paper advocates that a cloud environment with existing elasticity mechanisms does not need an additional fault tolerance mechanism when it comes to recovering failures at a virtual machine level, especially for stateless applications. A variety of well-known and studied different fault tolerance mechanisms might still be needed whenever state, context and history must be recovered integrally. As a matter of fact, we can state that elasticity may replace fault recovery mechanisms for applications that can deal with failures by starting a new VM.

We deployed a private cloud with dynamic elasticity and fault recovery mechanisms and undertook several experiments for understanding their behavior, both analyzed together and separately. Our results confirmed our hypothesis revealing that for different metrics the use of elasticity only (no recovery) is equivalent to the use of recovery only (no elasticity). Also, using both mechanisms together brings a small improvement. When it comes to deciding whether to use elasticity, recovery or both mechanisms, our experiments show that they are equivalent for our particular purpose. And since elasticity is already needed in any cloud-based environment for scaling up and down VMs and other resources, we can state that a specific recovery mechanism VM-level fault tolerance is not necessarily needed.

In the remainder of this paper, section 2 presents background and related work. Section 3 presents the methodology and section 4 presents our main results. Lessons learned are discussed in Section 5 and Section 6 draws conclusions and topics for future work.

2 Background and Related Work

Elasticity and fault tolerance have similarities in the sense they deal with performance problems, but they differ in purpose and scope. The use of elasticity in cloud computing empowers users to define policies for automating and enforcing the fulfillment of some pre-specified performance guarantees. Elasticity allows changing automatically the amount of resources allocated to an application to adapt to workload variations in order to match demand at any point in time [Herbst 2013]. On the other hand, fault tolerance is a mechanism used to avoid a service not to provide the established requirements in case of failure using redundant resources [Avizienis 2004]. Recovery in a redundant system may be performed by replacing the faulty component by a spare one or by dividing the workload among the other remaining components.

A cloud service must offer different management options for controlling elasticity and procedures for analyzing the system efficiency with respect to avoiding over and

underprovisioning as much as possible. Here we focus on the ability of an elastic mechanism to provide fault recovery for VMs. Elasticity may be implemented as a rule-based system where conditions match the value of metrics against predefined thresholds to take positive or negative actions, whenever more resources are added and removed respectively.

Although elasticity is most commonly associated to public clouds, it can also be used by private and hybrid clouds. For example, Amazon AutoScaling[1] provides different metrics and thresholds for elasticity actions. Unfortunately this feature is not directly offered by private cloud platforms, e.g. OpenNebula[2] and OpenStack[3].

In a cloud computing environment failures may be divided up into three main groups [Tchana 2012]: failures in virtual machines (VMs), failures in hardware and failures in the applications. Usually fault tolerance techniques are comprised of two main phases, namely fault detection and fault recovery. Since a cloud service is run by a third party provider, there is the additional problem in implementing fault tolerance mechanisms, because it is not always easy to implement due to the inherent difficulty in defining responsibilities of detection and recovery between provider and clients. In this paper we deal only with VM failures and we assume that the responsibility is bound to the provider configured by the client using the same elasticity mechanism already offered for keeping up with workload variations.

Hypervisor based fault tolerance (HBFT) is an approach where the hypervisor is responsible for fault detection and recovery [Bressoud 1996]. The obvious advantage is that no changes in the operating system and/or hardware are required for the deployment of fault tolerance mechanisms. On the other hand, such solutions are dependent on a particular hypervisor, such as Xen, KVM or VMware. HBFT solutions are available for different hypervisors, such as Kemari or Remus [Zhu 2010]. In our work we did no consider this class of fault tolerance approach that recovers the state of a VM. In other words, our recovery mechanism creates a clean new VM whenever necessary.

As far as we are concerned, there is no other approach available in literature that deals with elasticity and failure recovery in the same way we do. Jhawar et. al [Jhawar 2012] propose an approach that delivers generic fault tolerance on applications running in virtual machines using a dedicated service layer, that does not require users to be aware of how the mechanism are implemented by the provider. Unlike this proposal, ours is focused on virtual machine recovery, but does not need any additional service or middleware layer since the existing elasticity mechanism assumes the responsibility of dealing with failures.

Yang et. al [Yang 2013] undertake a performance analysis study of a cloud service considering fault recovery of both processing nodes and communication links. Their work proposes some analytical models and methods and presents a numerical example. However, it is not related to evaluating a real system using elasticity and recovery mechanisms like our approach. Tchana et. al. [Tchana 2012] propose an approach for fault tolerance in a

[1] aws.amazon.com/autoscaling
[2] opennebula.org
[3] openstack.org

cloud computing environment based on the collaboration of providers and clients and autonomic repair measures. They compare their collaborative approach with exclusive (provider or client) approaches, leading to significant results. In a previous work, our research group studied elasticity management, but without considering faults [Simões 2014].

3 Methodology

3.1 Scenario and Environment

In order to obtain a better understanding about using elasticity for fault recovery, we conceived a simple scenario where clients download objects of different sizes from a Web Server. Although typical Big Data applications and frameworks are very popular these days in the cloud, such as those based on Hadoop or Spark, a large portion of applications running on the cloud that serve real users are based on Web Servers. We decided to use a very simple object download application in order to be able to exert more control over the environment and obtain a better understanding of the results.

Our scenario is composed of four key components specially developed for this project, represented in gray by Figure 1: request generator, failure injector, elasticity manager and recovery manager. Also, it uses OpenNebula cloud controller, Xen hypervisor and Apache server and load balancer.

Clients send requests for a group of 50 objects with an average size of 8MB following a Log-normal distribution [Downey 2001]. Requests are received by a load balancer and distributed to Apache Web servers deployed in a variable number of virtual machines (VMs). Even though there are different points of failure in a real computing system, we considered failures in VMs only, because they are subject to our elasticity manager that creates and teardowns them on demand. We used OpenNebula as a private cloud controller and Xen hypervisor. We developed a failure detection agent and a recovery manager to work together with OpenNebula. Its own built-in detector lacks flexibility for our purposes, since it forces the use of some pre-specified monitoring intervals, which delays the execution of recovery procedures. Finally, we developed a VM failure injector using the Weibull distribution. All components were deployed into 7 servers in a private cloud, where the experiments were conducted.

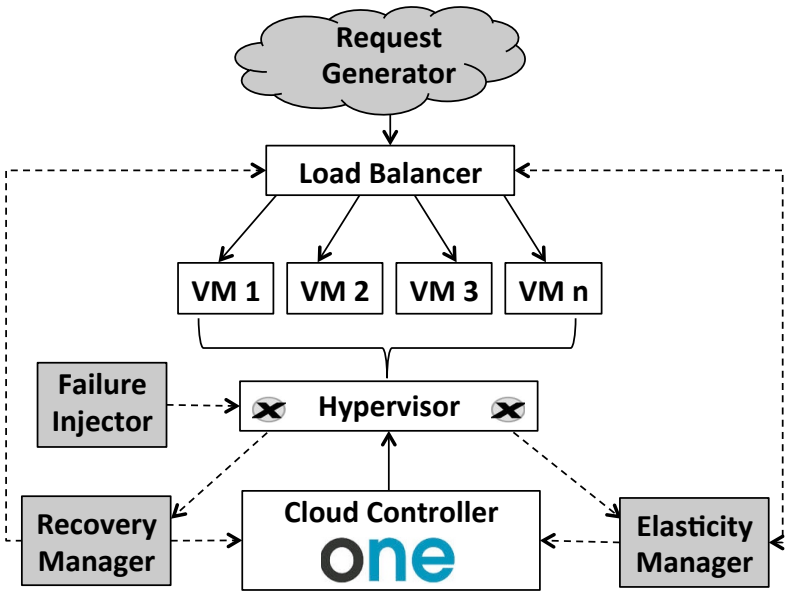


Figure 1 – Evaluation Scenario

3.2 Metrics

The basic metrics used in our experiments are:

- Response time per MB (RTMB): the total download time divided by the object size in MB. We compute the mean, median and 90th percentile every 10 seconds;
- Error rate: percentage of objects whose download does not finish successfully;
- SLA Violations: percentage of requests that violates the SLA, defined as the mean RTMB plus three times the standard deviation.
- Number of VMs: number of VMs allocated by the elasticity to deal with client requests;
- Workload: number of simultaneous requests being handled by the load balancer;
- MTTD: mean time to detect a fault.

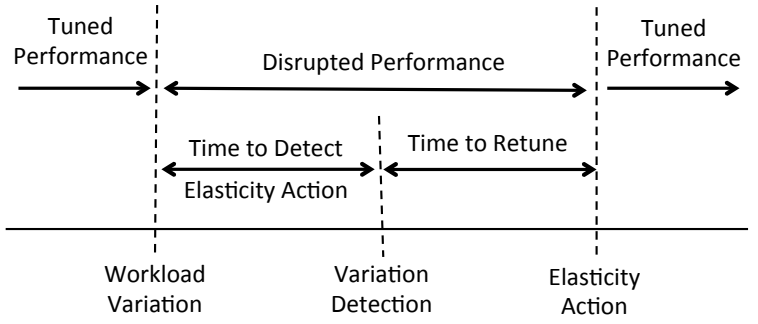
3.3 Elasticity and Recovery Management

We implemented both elasticity and recovery mechanisms that can be used together or separated. Elasticity actions are positive, when a new VM must be created because the average load of all active VMs is causing longer response times, and negative, when active VMs are idle. We used workload as the metric for controlling elasticity with two thresholds (higher and lower) defined in preliminary experiments for scaling up and down. The higher and lower thresholds were configured to 270 and 180 simultaneous requests per VM in the load balancer, respectively. We evaluated different thresholds and all confirm our main hypotheses, so they are not shown here. We developed a customized elasticity manager since there is no general purpose one available with the required flexibility.

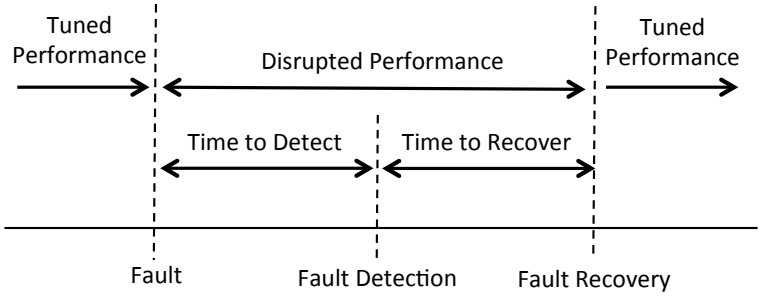
We adopted an application metric (number of requests) for managing elasticity, according to our previous findings where we found out that using application-level metrics is more precise than machine-level ones [Simões 2014]. Although there might be some dispute over this issue, our previous experience showed that in real settings, system-level metrics (such as CPU and memory) are not adequate for elasticity, since there is no clear mapping between variations in CPU and memory usage and response times. Also, we only considered the number of requests in the load balancer as the elasticity metric, because that is what usually the user care in this particular class of application.

We also developed our own fault recovery mechanism, since it gave us higher flexibility when compared to the one provided by OpenNebula. OpenNebula provides a fault detection feature that has been also evaluated in some experiments. However, we do not show its results since the performance was consistently worst compared to our own detection and recovery management mechanism.

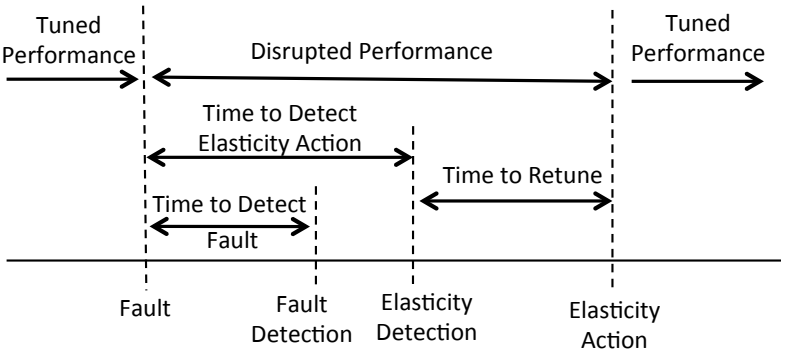
Figure 2 depicts the management mechanisms for both elasticity and recovery. Figure 2a presents a scenario with elasticity, but no failures, where the elasticity management mechanism detects workload variations and takes elasticity actions as a response. In this example, we assume an increased workload that causes a momentary performance disruption up to the point where the positive elasticity action is taken (a VM is created) and the performance is well tuned again.



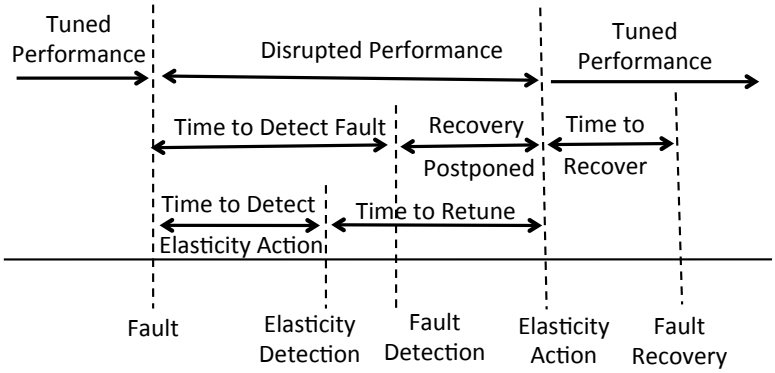
a) No Failures



b) Recovery Only



c) Elasticity Only



d) Elasticity & Recovery

Figure 2 – Elasticity and Recovery Management

Figure 2b depicts a scenario with recovery but no elasticity where a fault is detected and recovered and in-between the performance suffers some disruption. In Figure 2c, an elasticity mechanism manages alone both performance and fault problems. In this scenario, there are two detection events, for the fault itself and the resulting unbalance from workload and system resources. The most complex scenario is depicted by Figure 2d, combining elasticity and recovery. In this example, when a fault happens the system quickly detects a performance degradation and starts a positive elasticity action. Next, the fault itself is detected but the recovery action is blocked until the elasticity mechanism finishes the creation of a new VM. This situation may create two new VMs as the outcome of a single fault. Also, when the fault detection happens first than the elasticity detection, we do not postpone the elasticity action but leave it to the next round to detect.

3.4 Experiments: Elasticity and Recovery

We undertook a series of experiments where we managed elasticity and recovery together or separated. Table 1 summarizes the factors and levels used in our experiments, i.e. the parameters varied during different experiments and the values they assumed. Failure rate follows a Weibull distribution with average interarrival times of 15, 10 and 5 seconds, for low, medium and high failure rate respectively. However, due to space constraints we only present results for high failure rate, since for medium and low failure rates results followed intuition, i.e. the lower the failure rate, the better the system performance in general. As it

can be noticed, we forced an unusual high failure rate, so that we could perform experiments in a realistic platform. Otherwise, experiments could take months to complete, waiting for failures to happen.

Table 1 – Factors and Levels

Factor	Levels
Elasticity	Yes, No
Failures	Yes, No
Recovery	Yes, No
Arrival Rate	Constant, Variable
Failure Rate	Low (15), Medium (10), High (5)

We conducted experiments with two types of workloads, in order to represent situations similar to reality. A first series of experiments used a constant (fixed) workload for being able to directly compare the behavior of elasticity and recovery techniques. Even though elasticity is fit to situations of variable workloads for adapting resources according to demand, a constant rate was needed in order to observe whether its performance is similar to a traditional recovery mechanism. In a second series of experiments we increased the workload up to a certain point and then started to decrease it little by little. This is done by configuring the client start and stop times. Clients start every 5 seconds generating requests using a Poisson distribution. For both elasticity and recovery management, the monitoring interval was 1 minute.

Each experiment lasted for 40 minutes and was replicated 15 times with 99% asymptotic confidence intervals being computed. Considering processing times, experiments took approximately 170 hours.

4 Results

4.1 Constant Workload

Figure 3 presents the typical behavior of the allocated number of VMs when a constant workload is used. For the case with no failures the same number of VMs remains from the beginning to the end. With failures, the most effective way of dealing with them is using both elasticity and recovery mechanisms, which improves performance only slightly.

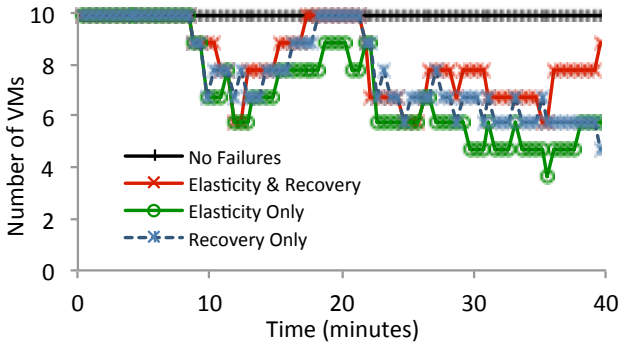
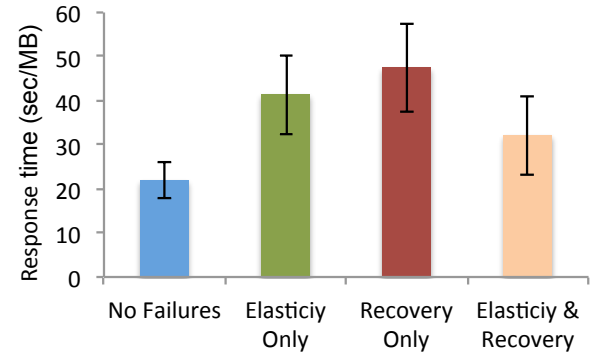
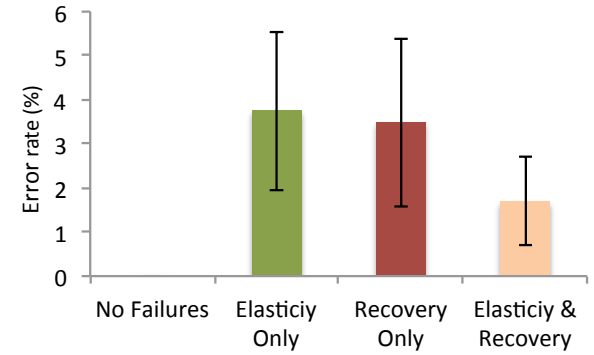


Figure 3 – Elasticity and recovery – time series

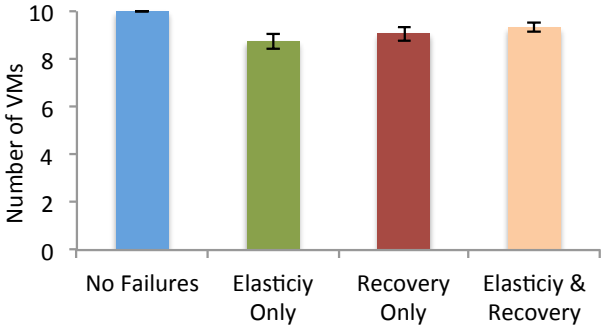
Figure 4 depicts the average and confidence interval of 15 replications of each experiment and confirms the first impression of Figure 3. Figure 4a shows the response time per MB (less is better). It can be observed that without failures it varies around 20 seconds, where for the elasticity only and recovery only mechanisms it varies around 40 seconds, with a higher confidence interval. Using both mechanisms, the value of this metric varies around 30 seconds. Figure 4b presents the error rate with an equivalent conclusion. For both elasticity and recovery mechanisms the error rate is lower and using only one of them yields similar values. Figure 4c presents similar information for the number of allocated VMs, which is proportional to the other metrics.



a) Response Time per Megabyte



b) Error Rate



c) Number of Virtual Machines

Figure 4 – Comparison of different mechanisms

Since the confidence intervals in Figure 4 overlap, we cannot conclude that one approach is superior to the others. In other words, when it comes to deciding whether to use

elasticity, recovery or both mechanisms, our experiments with constant rate show that they are equivalent. Also, since elasticity is already needed in any cloud-based environment for scaling up and down VMs and other resources, we can state that a specific recovery mechanism VM-level fault tolerance is not necessarily needed for that specific purpose.

Figure 5 depicts results for VM fault detection and recovery time for the constant workload scenario. It is possible to observe that elasticity responds faster when it comes to fault detection, which is due to synchronism caused by the particular way the failure detection script is scheduled by the Linux cron mechanism. Also, the averages with the confidence interval do not differ significantly.

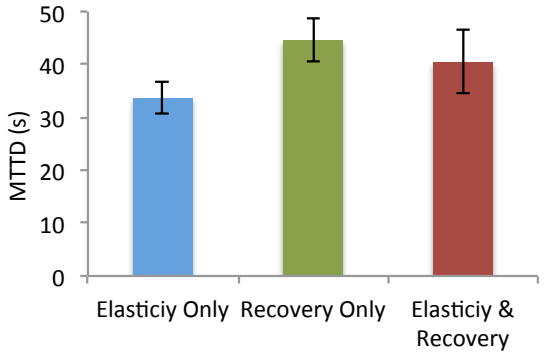
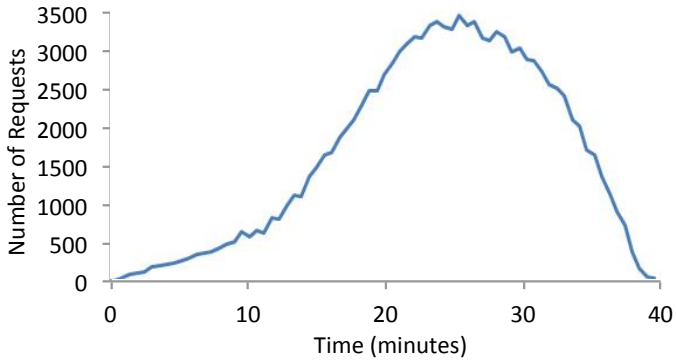


Figure 5 – Fault Detection time for constant workload

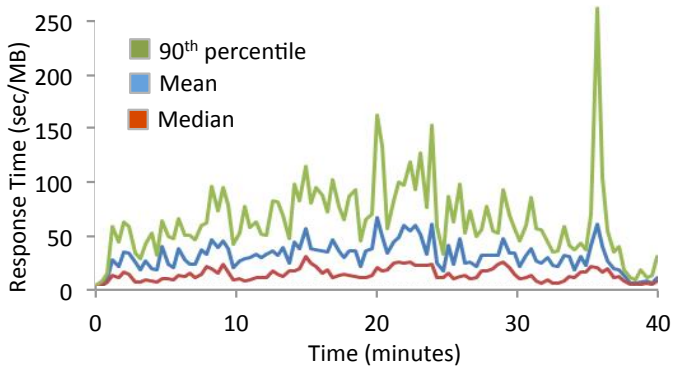
4.2 Variable Workload

Further, we conducted experiments with a typical workload typical that cloud services will face with dynamic resource allocation and deallocation. As shown by Figure 6a the workload is increased up to a certain point and than decreased, by carefully configuring clients start and stop times and request arrival rates. In Figure 6b we can observe that the response time per MB is kept around 20 seconds, similar to the constant workload and no failures experiment showed in Figure 3b, which is our benchmark. Figure 6c shows that we start with only one VM and scale it up to 10 and than back to one, according to the demand represented by requests. This result corroborates with our previous results from analyzing elasticity in private and public clouds [Simões 2014].

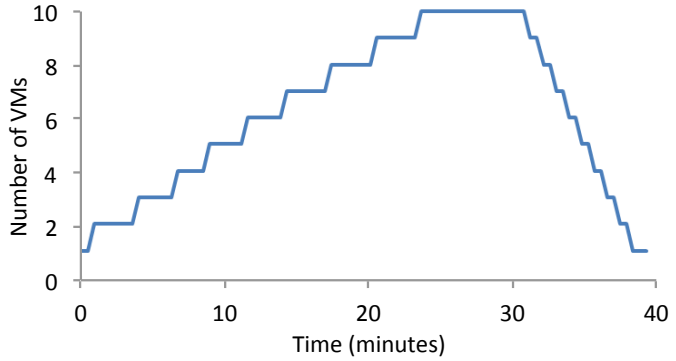
For these experiments, elasticity is always used, since there will not be possible for a system to deal with such increase in the workload without a feature for dynamically increasing the number of VMs. However, in order to get a deeper understanding of the need of a recovery mechanism for VM management, some experiments use recovery together with elasticity and others use elasticity only.



a) Number of Requests



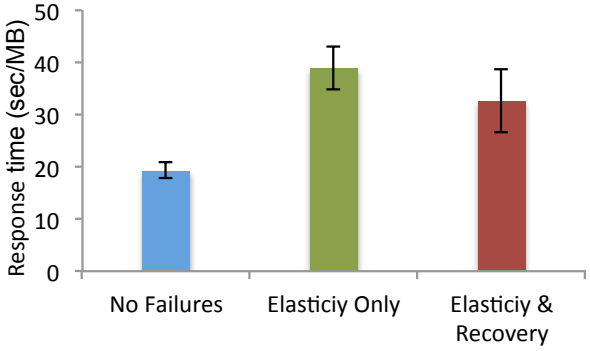
b) Response time per Megabyte



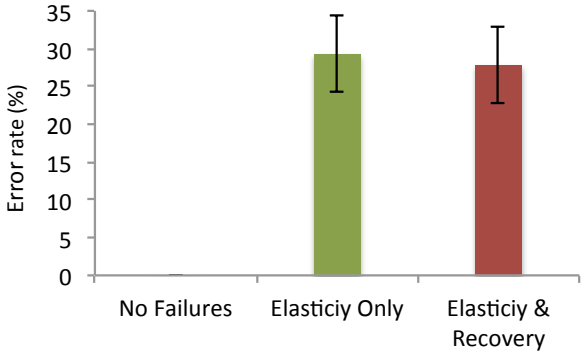
c) Number of VMs

Figure 6 – Elasticity/recovery: Variable workload

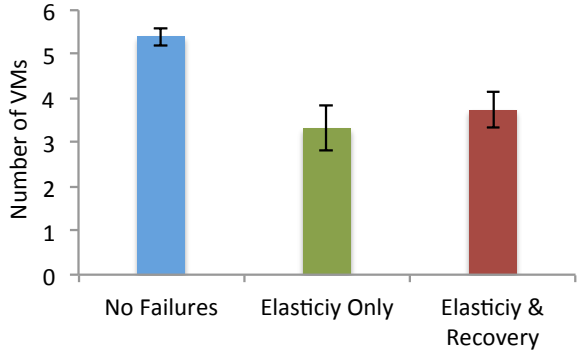
Once more, we can observe that for a variable workload, the use of elasticity can play both roles of scaling up and down resources according to demand and replacing VMs in case of failures. The key results for variable workload are presented in Figure 7. We can observe that there is a significant difference when no failures are injected. The use of recovery together with elasticity yields better results in average, i.e., they are closer to the no failures scenario. However, there is no statistical significance since the confidence intervals overlap. Also for these experiments we computed the SLA violation metric, Figure 7d, which corroborates the other results. SLA is less violated when no failures are injected and violation increases for the two scenarios with failures (namely elasticity only and elasticity & recovery).



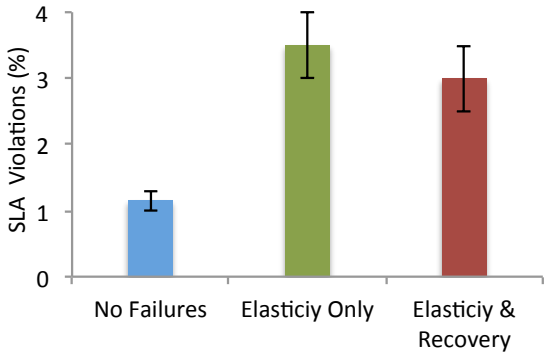
a) Response Time per Megabyte



b) Error Rate



c) Number of VMs



d) SLA Violations

Figure 7 – Elasticity with variable workload

Figure 8 depicts results for VM fault detection and recovery time for the variable workload scenario. One can observe that MTTD is similar for both cases (elasticity only and elasticity & recovery) and also similar to the values observed for the constant workload scenario presented in Figure 5.

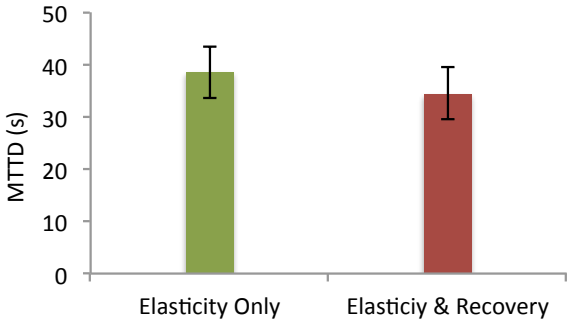


Figure 8 – Fault Detection Time – variable workload

5 Discussion

Our results show that for the particular scenario we evaluated, i.e. VM failures in a cloud environment, there is no evidence that using elasticity-only, recovery-only or both mechanisms is superior to the others. In other words, when it comes to deciding whether to use elasticity, recovery or both mechanisms, our experiments show that they are equivalent. And since elasticity is already needed in any cloud-based environment for scaling up and down VMs and other resources, we can state that a specific recovery mechanism VM-level fault tolerance is not necessarily needed, when no other requirements are present.

We showed that this is valid for a typical pre-cloud scenario (no elasticity) and more targeted to fault recovery mechanisms, i.e. a scenario with constant workload. Also, it is true for a scenario of variable workload, where the use of elasticity has proven to be able to play both roles of scaling up and down resources according to demand and also replacing VMs in case of failures.

Please notice that we are not proposing a complete solution for fault tolerance in cloud computing, but only an elasticity-based solution for VM failure recovery in those environments.

6 Conclusions

Elasticity is a key mechanism for providing the ability to adapt server behavior to variable workload in cloud computing. Fault recovery is a traditional mechanism for avoiding failing to provide a service with unexpected performance levels. We advocate in this paper that when it comes to VM management in a cloud environment, only elasticity management is able to perform both roles.

Particularly, we showed that the performance of a management mechanism based on fault recovery achieves similar results to a mechanism solely based on elasticity management, or even similar results when both mechanisms are used. Also, we can observe this is consistent over constant and variable workload scenarios.

As future work we intend to analyze the effect of using different workloads, such as by Hadoop/Spark applications or even cloud benchmarks 0. Also, we intend to analyze the effect of using predictive elasticity management 0.

References

Armbrust, M. et al., "A View of Cloud Computing", Communications of the ACM, 53(4), pp. 50-58, 2010.

Avizienis, A. et. al, "Basic concepts and taxonomy of dependable and secure computing", IEEE Trans. on Dependable & Secure Comp., 1(1), pp. 11-33, 2004.

Bressoud, T. C., Schneider, F. B., "Hypervisor-based Fault Tolerance". ACM Transactions on Computer Systems (TOCS), 14(1), pp. 80-107, 1996.

Cooper, B., Silberstein, A., Tam, E., Ramakrishnan, R., Sears, R., "Benchmarking Cloud Serving Systems with YCSB", ACM SoCC 2010, pp. 143-154, June 2010.

Downey, A. B., "The structural cause of file size distributions", 9th Modeling, Analysis and Simulation of Computer and Telecommunication Systems, 2001.

Herbst, N. R., Kounev, S., Reussner, R., "Elasticity in Cloud Computing: What It Is, and What It Is Not", ICAC 2013, June 2013.

Jhawar, R., Piuri, V., Santambrogio. M., "Fault Tolerance Management in Cloud Computing: A System-Level Perspective", IEEE Systems Journal, 7(2), pp. 288-297, November 2012.

Mell, P., Grance, T., "The NIST Definition of Cloud Computing", NIST Special Publication 800-145, 2011.

Simões, R., Kamienski, C., "Elasticity Management in Private and Hybrid Clouds", IEEE Cloud 2014.

Tchana, A., Broto, L., & Hagimont, D, "Approaches to Cloud computing fault tolerance", IEEE Intl Conf. on Computer, Information and Telecom. Systems (CITS), May 2012.

Yang, B., Tan, F., Dai, Y.-S., "Performance evaluation of cloud service considering fault recovery", Journal of Supercomputing, 65(1), pp. 426-444, July 2013.

Zhu, J., Dong, W., Jiang, Z., Shi, X., Xiao, Z., Li, X., "Improving the Performance of Hypervisor-based Fault Tolerance", IEEE IPDPS, pp. 1-10, April 2010.

Roy, N., Dubey, A., Gokhale, A., "Efficient autoscaling in the cloud using predictive models for workload forecasting," IEEE Cloud 2011, pp. 500-507, July 2011.