Elastic VM for Cloud Resources Provisioning Optimization

Wesam Dawoud, Ibrahim Takouna, and Christoph Meinel

Hasso Plattner Institute,
Potsdam University,
Potsdam, Germany
firstname.lastname@hpi.uni-potsdam.de

Abstract. Rapid growth of E-Business and frequent changes in websites contents as well as customers' interest make it difficult to predict workload surge. To maintain a good quality of service (QoS), system administrators must provision enough resources to cope with workload fluctuations considering that resources over-provisioning reduces business profits while under-provisioning degrades performance. In this paper, we present elastic system architecture for dynamic resources management and applications optimization in virtualized environment. In our architecture, we have implemented three controllers for CPU, Memory, and Application. These controllers run in parallel to guarantee efficient resources allocation and optimize application performance on co-hosted VMs dynamically. We evaluated our architecture with extensive experiments and several setups; the results show that considering online optimization of application, with dynamic CPU and Memory allocation, can reduce service level objectives (SLOs) violation and maintain application performance...

Keywords: virtualization, consolidation, elasticity, application performance, automatic provisioning, optimization, cloud computing

1 Introduction

Later advance in virtualization technology software, e.g. Xen [2] and VMWare [16], enabled cloud computing environment to deliver agile, scalable, elastic, and low cost infrastructures, however, current implementation of elasticity in "Infrastructure as a Service" cloud model considers Virtual Machine (VM) as a scalability unit. In this paper, we developed an automated dynamic resources provisioning architecture to optimized resources provisioning in consolidated virtualized environments (e.g., Cloud computing). Unlike current implementation of elasticity in cloud infrastructure, we replaced the VM (as a coarse-grain scalability unit) with fine-grain resources units (i.e. %CPU as a share, Memory as

MB). Our Elastic VM is scaled dynamically in-place to cope with workload fluctuations, furthermore, the hosted application is also tuned after each scaling to maintain predetermined (SLOs). As a use case we implemented our approach into Xen environment and used Apache web server as an application, our SLO in this paper is to keep the response time of the web requests less than a specified threshold. Nevertheless, our architecture could be extended for any application that has tunable parameters such as Database applications. The key contributions of this work are as follow: First, we have studied Apache application performance under different configuration and different CPU and Memory allocation values. Second, we have developed a dynamic application optimization controller for Apache application to maintain the desired performance. Third, we built CPU and Memory controllers based on [6]. Fourth, we built elastic system architecture that join CPU, Memory, and application optimization controllers for elastic consolidated virtualized environments. Finally, the elastic system architecture has been evaluated with extensive experiments on several synthetic workload and experimental setups, experiments also have included real workload demand requests. Our results show that elastic system architecture can guarantee the best performance for application in terms of throughput and response time. The rest of the paper is organized as follow. Section 2 study the systems and concepts that drive our research. In section 3 we describe our elastic system architecture. Section 4 provides literature review for related work. In section 5, we describe our experimental setup and analyze results.

2 Overview

In this section, we give an overview of systems and concepts that drive our research; we will start with a detailed study of Apache server, then will discuss the complexity of enforcing SLOs into consolidated environments (e.g. clouds), and finally will explain concerns that accompany using feedback control systems in computing systems.

2.1 Apache server

Apache [1], is structured as a pool of workers processes that handle HTTP requests. Currently, Apache supports two kinds of modules, workers and prefork modules. In our experiments we use Apache with prefork module to handle dynamic requests (e.g., php pages). In prefork mode, requests enter the TCP Accept Queue where they wait for a worker. A worker processes a single request to completion before accepting a new request. Number of worker processes is limited by *MaxClients* parameter.

Figure 1 displays the result of experiments in which Apache is configured with different settings of Memory, traffic rate, and *MaxClients*. By monitoring the throughput, we notice that, there is a value of *MaxClients*, (e.g. 75), which gives the highest throughput (450 req/sec) for specific Memory settings (512MB). Before this value there is no enough workers to handle requests, and

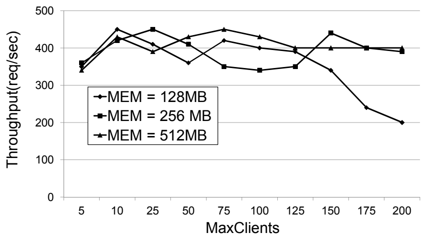


Fig. 1: Throughput vs. MaxClients under different hardware settings

after this value, performance regrades because of one of the following problems: CPU spend much time switching between many process or Memory is full so paging to harddisk consumes most of CPU time. Our heuristic Apache controller job is to find this optimum value dynamically.

2.2 SLOs enforcement complexity

Service-level agreement (or SLA) is a contract between a service provider and its customers. SLA consists of one or more service-level objectives (SLOs). An example of an SLO is: "The homepage should be loaded completely in no longer than 2 seconds". As seen, SLO consists of three parts: QoS metric (e.g., response time), the bound (e.g., 2 seconds), and a relational operator (e.g., no longer than). The violation of these objectives usually associated with penalties to the provider. The challenge is to map QoS metrics into low level resources (e.g. CPU and memory) dynamically.

2.3 Feedback Control of Computing Systems

Controllers are designed mainly for three purposes [5]: First, output regulation to be equal or near to the reference input; for example, maintaining Memory utilization always around 90%. Second, disturbance rejection which means if the CPU is regulated to be 70% utilized, then this must not affected by any other running applications like backup or virus scanning. Third, optimization which can be translated in our system as finding the best value of *MaxClients* that optimize Apache server performance. In terms of the feedback controllers, SLO enforcement often becomes a regulation problem where SLO metric is the measured output, and SLO bound is the reference input. The choice of control objective typically depends on the application. Indeed, with multiuse target systems, the same target system may have multiple controllers with different SLOs, unfortunately, identifying Input-output models for computing systems is not commonly used [19] because of the absence of the first-principle models. As a replacement, many research [18], [13], [6] [17] considered the black-box approach where the relation between the input and output is inferred by experiments. According to [19], to build a feedback controller able to adjust input-output of black-box's model you have to deal with many challenges: First, The controller may not converge to equilibrium, if the system does not have a monotonic relationship between a single input and a single output. Second, without an estimate of the sensitivity of the outputs with respect to the inputs, the controller may become too aggressive (or even unstable) or too slow. Third, the controller can't adapt to different operating regions in the input-output relationship, for example [19] shows that the mean response time is controllable using CPU allocation only when the CPU consumption is close to the allocated capacity and uncontrollable when the CPU allocation is more than enough. Here the notion of "uncontrollable" refers to the condition where the output is insensitive to changes in the input.

3 Elastic VM architecture

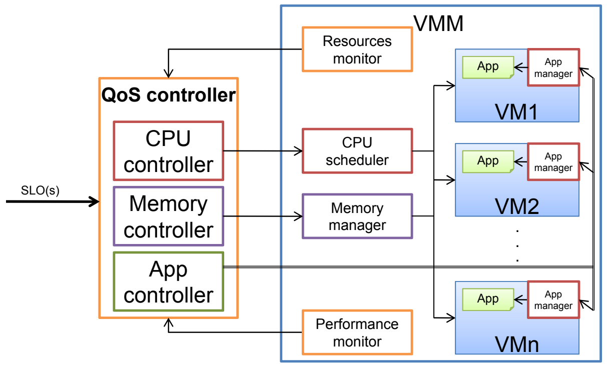


Fig. 2: Elastic VM architecture

Our architecture has main component "QoS controller" which communicates with many other modules implemented into the Virtual Machine Manager (VMM) and VMs levels as the following:

- Resources monitor module dynamically measures the resources consumption and updates the QoS controller with new measurements. The module depends on *xentop* tool to get CPU consumption of each VM.
- CPU scheduler is implemented to dynamically change the CPU allocation of the VMs according to determined values by QoS controller, this module depends on Xen credit scheduler as an actuator for setting the CPU shares for VMs. The credit scheduler has a non-work-conserving-mode which enables determining a limited portion of the CPU capacity for each VM. The credit scheduler prevents an overloaded VM from consuming the whole CPU capacity of the VMM and degrading the other VMs performance.
- Memory manger is implemented with help of balloon driver in Xen. This allows online changing of the VMs Memory. The driver doesn't allow VM

to exceed the determined variable *maxmem* at the domain creating time, so to have a wide range of the Memory size, we gave the variable *maxmem* an initial high value i.e. 500MB in all user domains configuration files then use the *mem-set* command to change the Memory size into the value determined by the controller.

– Performance monitor also keeps the controller up to date with performance metrics, i.e. the average response time and the throughput. The performance monitor is implemented on network device of the VMM, so it can monitor both the incoming and outgoing traffic.
– Application manager (App manager) is implemented into VM level, its job is to get new *MaxClients* value from the Application controller (App controller), to update the Apache configuration file, and then to reload Apache gracefully.

On the left side of figure 2 is the QoS controller; the controller has (SLOs) as inputs and proposed CPU capacity, proposed Memory allocation, and proposed *MaxClients* as outputs. In our approach the main SLO is to keep average response time of Apache web server into specific value regardless of the workload fluctuations, for this purpose we implemented three controllers to run in parallel, these controllers are as the following:

CPU controller: Which is a nested loop controller developed in [20]. The inner controller (CPU utilization controller) is an adaptive-gain integral (I) controller was designed in [17]:

$$a_{cpu}(k+1) = a_{cpu}(k) - K_1(k)(u_{cpu}^{ref} - u_{cpu}(k)), \tag{1}$$

Where

$$K_1(k) = \alpha . c_{cpu}(k)/r_{cpu}^{ref} \tag{2}$$

The controller is designed to predict the next CPU allocation $a_{cpu}(k+1)$ depending on last CPU allocation $a_{cpu}(k)$ and consumption $c_{cpu}(k)$, where the last CPU utilization $u_{cpu}(k) = c_{cpu}(k)/a_{cpu}(k)$. The parameter α is the constant gain which determine the aggressiveness of the controller. In our experiments, we set β=1.5 to allow the controller aggressively allocate more CPU when the system is overloaded, and slowly decrease CPU allocation in the under loaded regions. The disadvantage of this controller is that, it implies determining the reference utilization u_{cpu}^{ref} that will maintain the determined SLO (i.e. response time), whoever, this is not practical because, as seen in figure 3, the response time does not only depend on CPU utilization, but also on the request rate, which changes frequently. So, it is more realistic to have u_{cpu}^{ref} value automatically driven by the application's QoS goals rather than being chosen manually for each application. For this goal, another outer loop controller (RT controller) is designed [20] to adjust the u_{cpu}^{ref} value dynamically to ensure that the QoS metric, response time (RT), is around the desired value, this outer loop controller can be interpreted into the following equation:

$$u_{cpu}^{ref}(i+1) = u_{cpu}^{ref}(i) + \beta(RT_{cpu}^{ref} - RT_{cpu}(i))/RT_{cpu}^{ref} \tag{3}$$

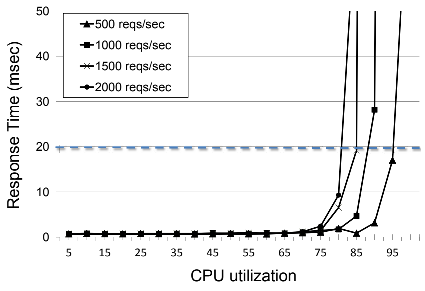


Fig. 3: Mean response time vs. CPU utilization under different request rates

Where $u_{cpu}^{ref}(i+1)$ is the desired CPU utilization, $RT_{cpu}(i)$ is the measured response time, and RT_{cpu}^{ref} is the desired response time determined by SLO. The outer controller (RT controller) ensures that the value fed to the CPU controller is always within an acceptable CPU utilization interval $[U_{min}, U_{max}]$.

In our experiments, we set β=1.5, the CPU allocation is limited to the interval [10, 80], and the CPU utilization is also limited to the interval [10, 80]. The desired response time (RT) in all our experiments is 20 milliseconds.

Memory controller: In our experiments we noticed that increasing the number of Apache processes can increase the throughput, but at some level, the performance is degraded drastically when the Apache processes consumed the whole available Memory, at this point, system starts to swap the Memory contents into the hard-disk, this behavior add more workload to the CPU which typically already overloaded by the big number of the processes. To keep the system away from bottlenecks, we implemented the Memory controller designed in [6] to keep the CPU controller run in an operating region away from the CPU contention:

$$a_{mem}(i+1) = a_{mem}(i) + K_2(i)(u_{mem}^{ref} - u_{mem}(i)) \tag{4}$$

Where

$$K_2(i) = \lambda . u_{mem}(i)/u_{mem}^{ref} \tag{5}$$

The controller aggressively allocates more Memory when the previously allocated Memory is close to saturation (i.e. more than 90%), and slowly decreases Memory allocation in the under-load region. Along our experiments, we set u_{mem}^{ref}=90%, λ=1, and the limits of the controller to be [64, 512], where the 64 is the minimum allowed Memory allocated size, and the 512 is the maximum allowed allocated Memory size.

Application controller: after extensive of experiments and monitoring Apache behavior, we found that there was a specific value of *MaxClients* which gives the best throughput and the minimum response time as seen in figure 1, finding the optimum value of *MaxClients* was examined by former research e.g. [8], unfortunately, these optimization methods are not applicable to our case for many reasons: First, we have a dynamic resources, so it will be difficult to dynamically determine the new optimum *MaxClients* value for each new resources allocation. Second, we don't have the chance to run an active optimization using our generated traffic, since it may influence the real service performance. Third, the optimum value is affected by traffic type and CPU utilization.

In the light of the mentioned problems, we designed our heuristic Apache controller to find the best *MaxClients* value passively (depending on the real traffic). The Apache controller monitors four measured values to determine the best *MaxClients*: response time, throughput, CPU utilization, and number of running Apache processes. The controller saves the best record of these values. The best record is calculated by finding the record which satisfies the QoS response time metric and gives the highest throughput with less CPU utilization. With each new measurement of monitored values, Apache compares the current record with the best record, if it is better; the current record will be saved as the best record. While it is running, if the Apache noticed a violation of QoS metrics (response time in our case) it tries to predict the problem by the following rules:

Rule1: Apache processes starving problem: Apache processes starving problem occurs when Apache server runs big number of processes, as a result, CPU spends most of the time switching between these processes while giving small slot of the time to each process, such behavior causes requests to spend longer time in application queue, which end up with high response time and many timed-out requests. To eliminate this problem, the Apache controller reloads the Apache server with the last best record, this reload is supposed to reduce the number of running processes, reduce CPU utilization, and consequently reduce response time.

Rule2: Resources competition problem: The competition on resources is predicted by Apache controller as response time increases, number of running apache processes reaches *MaxClients* value, and at the same time CPU utilization decreases (i.e. less than 90%). The reason behind the low utilization in competition case is that, CPU controller, according to the high response time, suggests allocating more CPU, while the fair share which gives each co-located VM on the same core the same capacity of the CPU (e.g., 50% in case of two VMs) prevents the VM from exceeding this limit. As seen above, with both rules, the proposed Apache controller will not only look for the optimum *MaxClients* value, but also will eliminate performance bottlenecks by keeping a history of the last best running configurations.

4 Related work

Dynamic provisioning of resources - allocation and de-allocation of the resources to cope with workload - had much interest especially after the widely usage of consolidation environments such as virtualized datacenters and cloud. Significant prior research have been sought to map the (SLOs) such as QoS requirements into low level resources requirements such as CPU, Memory, and I/O requirements. All the studied approaches considered the mean response time (MRT) as their SLO and accordingly developed the suitable controllers for resources management e.g. [4], [17], [15] and [6]. To this end, previous related works can be divided into three main folds: dynamic resources provisioning using controllers, resources management using migration of VM feature and multi-instances provisioning, and application optimization.

Research in [4], [17] and [15] considered only CPU controllers to automate the dynamic resources provisioning, while [6] designed parallel CPU and Memory controllers to be sure that consolidated applications can have access to sufficient CPU and Memory resources, with the help of Memory controller [6] keeps the whole system away from the high levels of utilization that can drastically degrade the performance [12]; nevertheless, applications optimization with dynamic resources provisioning is the common missing issue. Unlike aforementioned works, [15] has developed a multi-tier dynamic provisioning system; it presents novel provisioning technique based on combination of predictive and reactive mechanisms. The application behavior and workload characteristics are analyzed offline depending on history monitoring, but the provisioning is completely automated. The provisioning of the resources in web server tier is implemented by running more VMs instances. In some productive environments such as Amazon Elastic Load Balancing, the quality of service metrics (e.g., request count and request latency) is watched by Amazon Cloudwatch. Amazon scalability mechanism depends on initiating a VM instance as a load balancer routing the traffic into many similar VMs instances, this approach have many limitations: First, it is limited to specific application like web servers and not applicable to the other applications like Databases. Second, it depends on a VM as a load balancer, which can be a single point of failure. Third, it admits VM as a scaling unit.

Several researches have leveraged VMs migration mechanism for coping with dynamic workload fluctuation as well as providing scalability and load balancing models, for example, [7] and [18] propose migration to handle dynamic workload changes and resource overloads in production systems to avoid application performance degradation. But, migrating VM consumes I/O and CPU and network resources which might contribute at performance degradation of other VMs, furthermore, using migration with applications that have long-running in-memory state or frequently updated data such as database and messaging applications might take too long time causing service level violations during migration. Additionally, security restrictions might increase overhead during migration process [11].

Towards application optimization, [8] have implemented three controllers to optimize the configuration parameters of the Apache web server (i.e. MaxClients) online, the Newton's method optimizer which is inconsistent with the highly variable data, the Fuzzy controller which is more robust but converges slowly, and finally, the heuristic controller which works well under specific circumstances and requires former knowledge of bottleneck resources. [3] developed an agent-based solution to automate system tuning, the agents do both controller design and feedback control, however, slow converges of the system (i.e., 10 minutes for MaxClients), makes it unsuitable for sudden workload changes.

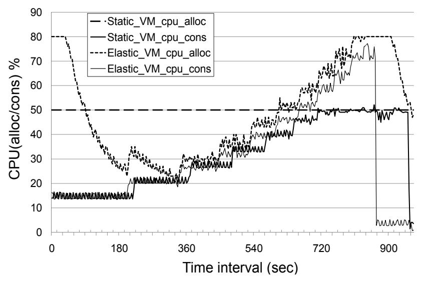
5 Experimental Setup

Our experiment conducted on a testbed of two physical machines (Client and Server) connected by 1 Gbps Ethernet. Server machine has Intel Quad Core i7 Processor, 2.8 GHz and 8GB of Memory, it runs Xen 3.3 with kernel 2.6.26-2-xen-686 as hypervisor. On the hypervisor are hosted VMs with Linux Ubuntu 2.6.24-19. These VMs run Apache 2.0 as a web server in prefork mode. For workload generation, *httperf* tool [10] is installed on client machine. In the following experiments we deal with three VMs setup: First, Static VM, which is a virtual machine initialized with 512MB of RAM and limited to 50% of the CPU capacity. Second, Elastic VM with CPU/Memory controllers, it is a VM controlled with the CPU and Memory controllers seen in equations 1 to 5, the CPU limits of this machine is 80% of CPU capacity, and the Memory is 512MB of RAM. Third, Elastic VM with Apache, it has the same setup of first VM except that it is equipped with our Apache controller in addition to CPU and Memory controllers. In all our experiments, SLO is to keep response time threshold (RT threshold) less than 20 milliseconds.

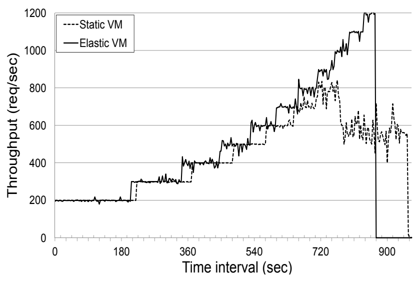
5.1 Experimental Setup 1

In this experiment, we would like to study our Elastic VM ability to cope with traffic change to maintain the specified SLO. To express the improvements, we ran the same experiment onto a Static VM with similar but static resources. As a basis of our experiments; we used dynamic web pages requests, in each request, the web server executes a public key encryption operation to consume a certain amount of CPU time. The step traffic initiated with the help of *autobensh* tool [14], it started with 20 sessions, each session contains 10 connections. The number of sessions increases by 10 with each load step. The total number of connections for each step is 5000, and the timeout for the request is 5 seconds. Throughput result from the generated web traffic is seen in figure 4(b).

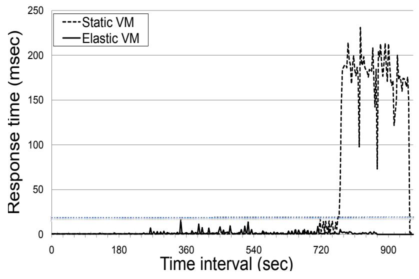
Each step of the graphs in figure 4(b) represents the throughput of a specific traffic rate, for example, in period between 0 to 210 seconds; both VMs respond to 200 req/sec successfully without any requests loss or time-out, in this period of time, both VMs were able to consume the required CPU capacity that copes with coming requests. In first period, we notice in figure 4(a) how the Elastic VM started a slow release of over-allocation CPU from the highest starting allocation (i.e. 80%) to the predicted suitable value. This behavior of Elastic VM, allocating resources aggressively then converging slowly to the optimum allocation, enabled it to respond to the whole traffic rates successfully. In the other hand, the static allocation of CPU, enabled the Static VM to respond successfully until second 780, afterwards, the Static VM's CPU is saturated, which caused requests to wait longer in the TCP accept queue, and consequently increased response time, this results in a continues period of SLO violation as seen in figure 4(c). Furthermore, some of the queued requests timed out before being served, the percentage of timed-out requests with the corresponding traffic rate is illustrated in table 1. The table started at 900 req/sec because there was no significant timed-out



(a) CPU consumption



(b) Throughput



(c) Response time

Fig. 4: Static VM vs. Elastic VM response to step traffic

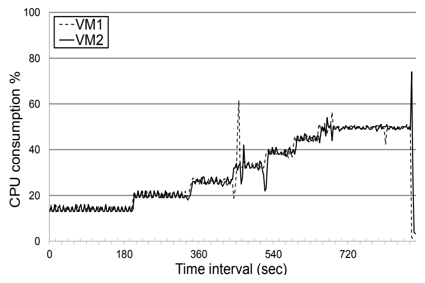
Table 1: The timeout started after the Static VM received 900 req/sec.

Requests rate(req/sec)	Static VM (timeout %)
900	7.232
1000	15.328
1100	18.258
1200	27.772

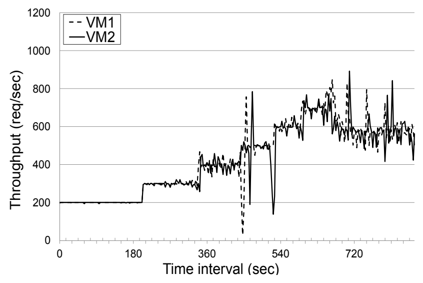
traffic before this rate. If compared to the Elastic VM for the same high traffic rate (i.e. 800 to 1200 req/sec), figures 4(a) to 4(c) show how the Elastic VM was able to borrow more resources dynamically, serve more requests, maintain a low response time, and prevent SLO violation.

5.2 Experimental Setup 2

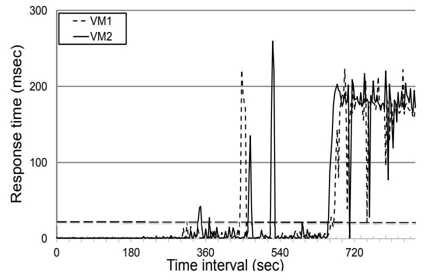
In the previous experiment, we studied the ideal case where the host was able to satisfy the Elastic VM's need for more resources to cope with the increase of incoming requests. In this experiment, we study the competition on the CPU between two Elastic VMs. Unlike experiments that have been done by [6], where each VM's virtual CPU has been pinned into a different physical core, we pinned the virtual CPUs of two Elastic VMs into same physical core to raise the competition level. For the following experiment, the step-traffic has been run two times simultaneously onto both Elastic VMs, one time without Apache controller and another time with Apache controller, to clarify the benefits of Apache controller usage. The first part of the experiment, illustrated in figures 5(a) to 5(c). Fig-



(a) CPU consumption



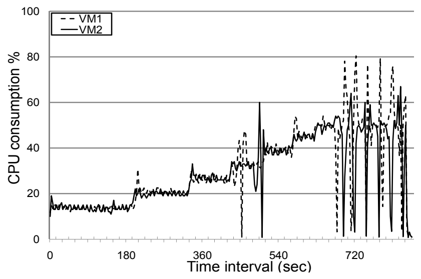
(b) Throughput



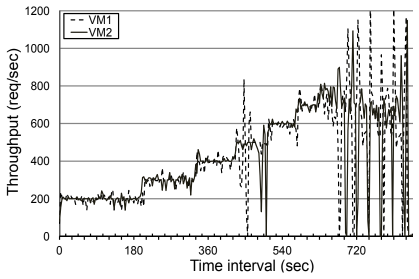
(c) Response time

Fig. 5: Two Elastic VMs (without) Apache controller responding to step traffic

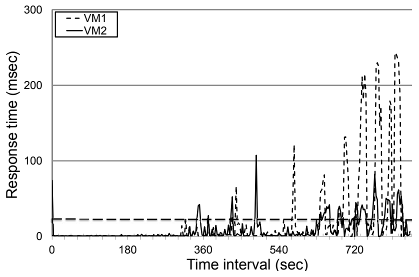
ure 5(b), shows that Elastic VMs were not able to cope with the traffic rate higher than 800 req/sec while the host committed only 50% of the CPU power for each VM starting from second #660 as seen in figure 5(a). The reason behind this fair sharing is Xen credit scheduler, during this experiment, we setup the scheduler with the same share for running VMs. According to competition on CPU, many requests are queued for a long time causing high response time and continues violation of SLO, as seen in figure 5(c), moreover, many other requests are timed-out before being served as seen in second and third columns of table 2. From the above experiments, we can conclude that Elastic VM can improve the performance if the host has more resource to redistribute, but in case of competition on resources, under the fair scheduling, Elastic VM (without) Apache controller merely behaves as a Static VM. The previous experiment is repeated on two Elastic VMs (with) Apache controller, figure 6(a) shows that in spite of the limited CPU capacity (50%) available to each VM, starting from second #660, the Apache controller do two improvements, first, the moment of the Apache reload is a good chance for the other Apache server to have more processing power and serve more requests as seen in figure 6(a), second, after



(a) CPU consumption



(b) Throughput



(c) Response time

Fig. 6: Two Elastic VMs (with) Apache controller responding to step web traffic

the reload, the Apache servers are tuned with a new *MaxClients* value, if this value achieved better performance, the Apache controller will keep it, otherwise it will continue looking for more optimum value.

5.3 Experimental Setup 3

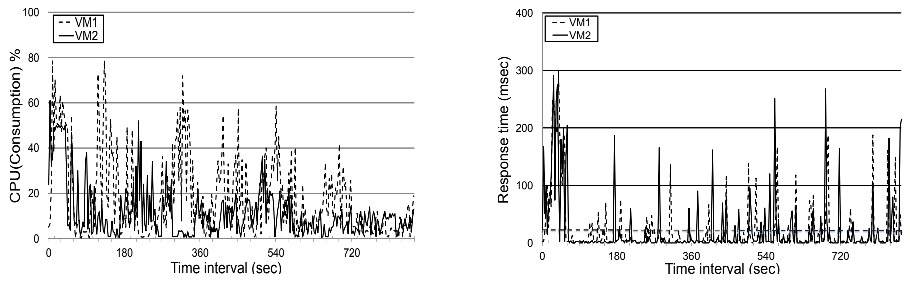
In the following experiment, we test our system against more real world demand traces traffic. For this purpose, we generate the same traffic described in [8]. The parameters of the generated workload are described in table 3 according to "WAGON" [9] benchmark, however, the session rate is selected to have uniform distribution, this enabled us to run the same traffic one time (without) Apache controller, and another time (with) Apache controller, to investigate Apache controller behavior under real workload. For both parts of the experiment, we used the same Elastic VMs described in section 3. First part of this experiment has been started by directing simultaneous instances of the generated traffic to the co-located Elastic VMs. Both Elastic VMs in this part of the experiment are running (without) Apache controller for 15 minutes. As seen in figure 7(a), there is a competition on the CPU power from the first run of the experiment until the 60th second, as a result, the percentage of timed-out requests for VM1 and VM2 were 12.7% and 15.5%, while the percentage of SLO violations are 18.6% and 17.5% as seen in first and second columns of table 3. Along the remaining run of the experiment, there was no competition on the CPU, and Elastic VM1

Table 2: Two Elastic VMs (without) Apache controller vs. two Elastic VMs (with) Apache controller responding to step traffic

	VM1	VM2	VM1	VM2
(req/sec)	Timeout requests(without)		Timeout requests(with)	
800	4.0%	0%	0%	0.2%
900	13.3%	23.8%	8.8%	8.2%
1000	20.5%	23.2%	16.52%	17.0%
1100	25.0%	35.0%	21.0%	22.0%
1200	31.0%	37.0%	26.2%	27.8%
	SLO violation(without)		SLO violation(with)	
	23.9%	26.4%	14.7%	16.8%

Table 3: Workload parameters

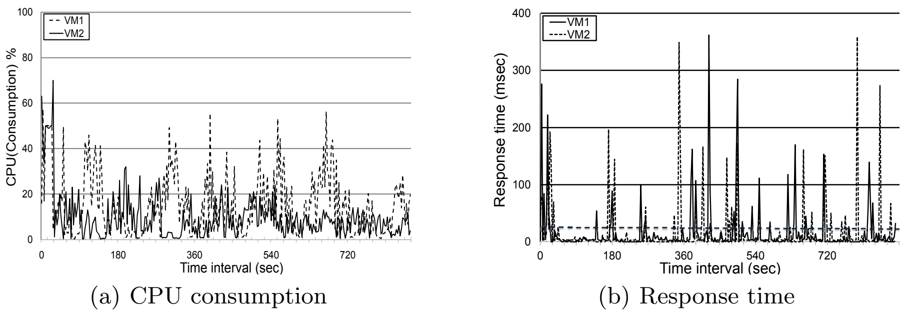
Parameter name	Distribution	Parameters
SessionLength	LogNormal	Mean=8, sigma =3
BurstLength	Gaussian	Mean=7, sigma=3
ThinkTime	LogNormal	Mean=30, sigma=30



(a) CPU consumption (b) Response time

Fig. 7: Two Elastic VMs (without) Apache controller responding to more realistic traffic

was able to consume more than 50% of the CPU power in periods from 120 to 180, and from 300 to 360 to keep the response time within the determined value. In the second part of this experiment, Apache controller has been run in parallel to CPU and Memory controllers. As seen in figure 8(a), the competition on CPU at the beginning of the experiment triggered Apache server tuning in both machines, as a result, Apache server at VM1 is reloaded one time at second #5 with *MaxClients*=160, and another time at second #30 with *MaxClients*=170, while Apache server at VM2 is reloaded at second #30 with *MaxClients*=160.



(a) CPU consumption (b) Response time

Fig. 8: Two Elastic VMs (with) Apache controller responding to more realistic traffic

The benefit of application tuning is illustrated in figure 8(b), instead of continues violation of SLO seen in figure 7(b) starting from the beginning of the experiment until second #60, SLO violation is limited to second #30 with the help of Apache controller. The timeout traffic and SLO violation of the complete run of the second part of the experiment is illustrated in third and fourth columns of table 4. First and second columns of table 4 show a small reduction in the percentage of the timed-out requests, but a significant reduction in percentage of SLO violation in case of Apache controller usage.

Table 4: Two Elastic VMs (without) Apache controller vs. two Elastic VMs (with) Apache controller responding to more realistic generated traffic

VM1	VM2	VM1	VM2
Timeout requests(without)		Timeout requests(with)	
12.7%	15.5%	11.5%	13.8%
SLO violations(without)		SLO violations(with)	
18.6%	17.5%	13.3%	13.1%

The above results prove that running our Apache controller, in parallel to CPU/Memory controllers, reduces SLO violation and improves application performance for both synthesized and more real generated traffic.

6 Conclusions & Future work

In this paper, we have presented an implementation for elastic system architecture for optimizing resources consumption in consolidated environments. Our system includes three controllers CPU, Memory, and Application running in parallel to preserve the intended SLO. We have evaluated our system in a real Xen based virtualized environment; the experiments show that using Application controller maintains the performance and mitigates SLO violation and the timeout requests.

Our immediate future work will include analyzing more applications such as database and their optimization feasibility in such dynamic resources allocation environment. The analysis will consider analytical models such as queuing analysis. We will also extend our work to be integrated with other resource management schemes like "VM migration" and "running multiple instances" while considering both performance and security as priorities.

References

1. Apache: The Apache Software Foundation, http://www.apache.org/
2. Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., Warfield, A.: Xen and the art of virtualization, vol. 37. ACM Press, New York, New York, USA (Oct 2003)
3. Chess, Y.D., Hellerstein, J.L., Parekh, S., Bigus, J.P.: Managing Web server performance with AutoTune agents. IBM Systems Journal 42(1), 136–149 (Jan 2003)
4. Gandhi, N., Tilbury, D., Diao, Y., Hellerstein, J., Parekh, S.: MIMO control of an Apache web server: modeling and controller design. In: Proceedings of the 2002 American Control Conference (IEEE Cat. No.CH37301). pp. 4922–4927. American Automatic Control Council (2002)
5. Hellerstein, J.L., Diao, Y., Parekh, S., Tilbury, D.M.: Feedback Control of Computing Systems. John Wiley & Sons (2004)
6. Heo, J., Zhu, X., Padala, P., Wang, Z.: Memory Overbooking and Dynamic Control of Xen Virtual Machines in Consolidated Environments. In: Proceedings of IFIP-IEEE Symposium on Integrated Management IM09 miniconference. pp. 630–637. IEEE (2009)
7. Khanna, G., Beaty, K., Kar, G., Kochut, A.: Application Performance Management in Virtualized Server Environments. In: 2006 IEEE/IFIP Network Operations and Management Symposium NOMS 2006. pp. 373–381. IEEE (2006)
8. Liu, X., Sha, L., Diao, Y., Froehlich, S., Hellerstein, J.L., Parekh, S.: Online Response Time Optimization of Apache Web Server (2003)
9. Liu, Z.: Traffic model and performance evaluation of Web servers. Performance Evaluation 46(2-3), 77–100 (Oct 2001)
10. Mosberger, D., Jin, T.: httperf - A Tool for Measuring Web Server Performance. In: In First Workshop on Internet Server Performance. pp. 59–67 (1998)
11. Oberheide, J., Cooke, E., Jahanian, F.: Empirical exploitation of live virtual machine migration. In Proc. of BlackHat DC convention (2008)
12. P. Bovet, D., Cesati, M.: Understanding the Linux Kernel, Third Edition. O'Reilly Media (2005)
13. Padala, P., Hou, K.Y., Shin, K.G., Zhu, X., Uysal, M., Wang, Z., Singhal, S., Merchant, A.: Automated control of multiple virtualized resources. European Conference on Computer Systems pp. 13–26 (2009)
14. T J Midgley, J.: Autobench (2008), http://www.xenoclast.org/autobench/
15. Urgaonkar, B., Shenoy, P., Chandra, A., Goyal, P., Wood, T.: Agile dynamic provisioning of multi-tier Internet applications. ACM Transactions on Autonomous and Adaptive Systems (TAAS) 3(1) (2008)

16. VMWare: http://www.vmware.com/
17. Wang, Z., Zhu, X., Singhal, S., Packard, H.: Utilization and slo-based control for dynamic sizing of resource partitions (2005)
18. Wood, T., Shenoy, P., Venkataramani, A., Yousif, M.: Abstract Black-box and Gray-box Strategies for Virtual Machine Migration (2007)
19. Zhu, X., Uysal, M., Wang, Z., Singhal, S., Merchant, A., Padala, P., Shin, K.: What does control theory bring to systems research? SIGOPS Oper. Syst. Rev. 43(1), 62–69 (Jan 2009)
20. Zhu, X., Wang, Z., Singhal, S.: Utility-Driven Workload Management using Nested Control Design, pp. 6033–6038. American Control Conference (2006)