How to Tame Your VMs: an Automated Control System for Virtualized Services

Akkarit Sangpetch asangpet@andrew.cmu.edu
Andrew Turner andrewtu@cmu.edu
Hyong Kim kim@ece.cmu.edu
Department of Electrical and Computer Engineering
Carnegie Mellon University
Pittsburgh, PA, USA

Abstract

Modern datacenters contain a large number of virtualized applications and services with constantly changing demands for computing resources. Today's virtualization management tools allow administrators to monitor current resource utilization of virtual machines. However, it is quite challenging to manually translate user-oriented service level objectives (SLOs), such as response time or throughput, to suitable resource allocation levels. We presented an adaptive control system which automates the task of tuning resource allocations and maintains service level objectives. Our system focuses on maintaining the expected response time for multi-tier web applications. Our control system is capable of adjusting resource allocation for each VM so that the applications' response time matches the SLOs. Our approach uses individual tier's response time to model the end-to-end performance of the system. The system helps stabilize applications' response time. It can reduce the mean deviation of the response time from specified targets by up to 80%. Our system also allows the physical servers to double the number of VMs hosted while maintaining the target response time.

Tags: VMs, research, control, resource, allocation

1. Introduction

Modern datacenters contain a large number of virtualized applications and services; with constantly changing demands for computing resources. These virtual workloads are executed on multiple virtual machines (VMs) which can be consolidated onto a smaller number of physical hosts. Today's virtualization management tools allow administrators to monitor current resource utilization of virtual machines. Management capabilities such as adjustable resource allocation [9] are also provided as a way to configure the underlying resource to meet applications' demands.

However, it is quite challenging to manually translate user-oriented service level objectives (SLOs), such as response time or throughput, to suitable resource allocation levels. Such tasks demand experience administrators and significant amount of time. Moreover, virtualized applications are often distributed and dependent on each other. It is imperative that the administrators understand the complex behaviors of the applications before they are able to manually tune them effectively.

In this work, we developed an adaptive control system which automates the task of tuning resource allocations and maintains service level objectives. Our system initially focuses on the expected response time for multi-tier web applications as our primary SLOs. Our control system is capable of adjusting CPU share allocation for each VM so that the applications' response time matches the SLOs. Our approach uses individual tier's response time to model the end-to-end performance of the system. This allows our model to capture systems' dynamics without relying on just their resource utilization level.

Our system helps stabilize applications' response time. It can reduce the mean deviation of the response time from specified targets by up to 80%. The system also allocates only the required amount of resource to satisfy the SLOs for each VM. Without over-provisioning, our system can increase the number of hosted applications by up to 100%.

The capabilities provided by our system are useful for administrators. It provides a way to express levels of services in terms of actual application performance. Our system can be applied to a cloud-based service provider model as well as in smaller clusters where resources are limited and applications may have different priorities. Our controllers can allocate just enough resource to satisfy the level of service required, allowing individual host to process more workloads.

We deployed our system on Linux and Kernel Virtual Machine environment in a local cluster. Our results suggest that the system can maintain the service response time for different VMs running on the host. Our system can also adapt to the level of workload changes and adjust system parameters in order to match the service response time.

We will explain the overall design of our system in section 2. The detailed specification on each component could be found in section 3. We evaluate our system's performance in section 4. The related works are reviewed in section 5. The discussion and ongoing works are explained in section 6.

2. System Overview

Our control system consists of four components; sensor, actuator, modeler, and controller as shown in Figure 1. Our design resembles a closed-loop control system. During each control interval, the sensors collect application-related performance (such as response time) from VMs hosting the controlled application. The collected information is fed to the modeler and is used to update the application's performance model. The modeler creates a performance model for targeted applications by adjusting the model parameters based on sensor inputs. The model obtained can be later used to predict the applications' performance for possible system configurations. The controller then uses the model to find the optimized system configurations and send the result to actuators. The actuators then adjust the system parameters accordingly. The impact on the applications' performance can be measured during subsequent intervals by the sensors, forming a closed-loop control system.

We currently use Linux and KVM as our hypervisor. However, the system can be extended to support other environments.

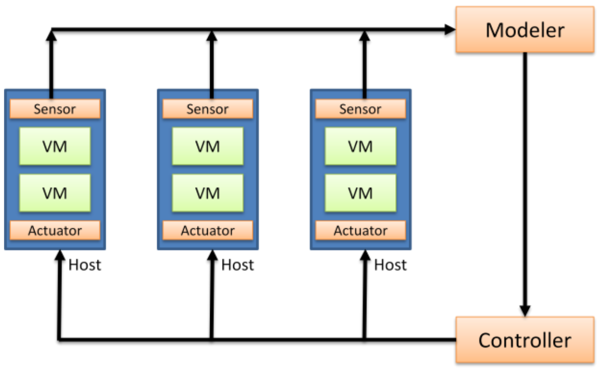

Figure 1: Control System Overview

Our initial system design primarily focuses on the response time as our controlled objective. The system tries to control the CPU share allocation for each VM in order to match the specified response time objectives. The system can automate the task of finding suitable CPU allocation for each VM tier. By controlling the number of shares allowed for each VM, we are able to increase the number of VMs running on a host without impacting the response time of the controlled applications. This allows the overall cluster to be more efficient and able to accept more workloads while maintaining existing SLOs.

The detailed description of our system is discussed in the next section.

3. System components

The components of our system could be described as followed.

3.1 Sensors

Sensors utilize packet filtering and capturing tools to analyze packets intended for the controlled VM. Our sensors can extract response time from the target applications' components. The response time is the time from the moment the last packet of the request is sent to the moment the last packet of the response arrives.

We collect the application performance metrics from different application tiers. For our initial system design, our sensor try to determine the application performance based on network packets going through the VMs. We are currently using the response time collected from each application tier. However, the sensor can also be used to collect other performance-oriented metrics such as the application's throughput, or number of concurrent requests.

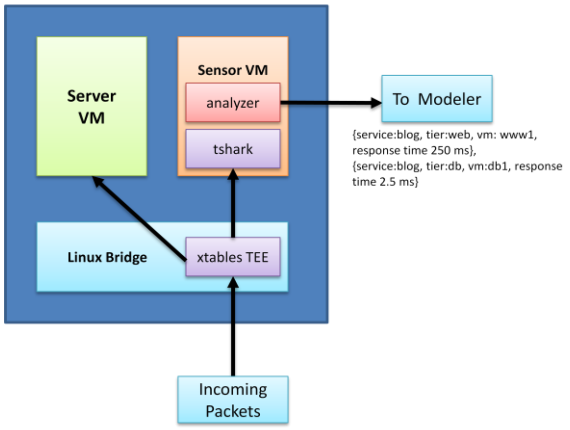
Server VM
Sensor VM
analyzer
tshark
To Modeler
{service:blog, tier:web, vm: www1, response time 250 ms}, {service:blog, tier:db, vm:db1, response time 2.5 ms}
Linux Bridge
xtables TEE
Incoming Packets

Figure 2: Sensor implementation

Our current implementation of the sensor is a combination of packet filtering and capturing tools which capture packets intended for the concerned server (as shown in Figure 2). The sensor is a guest VM running on the controlled hosts. This allows us to deploy and modify the sensor without too much modification on the physical host. The sensor utilizes *tshark* (packet analyzer) and *pcap* (packet capture driver) to extract the response time of the controlled applications.

The applications' response time is determined by recording the timestamp of packets (belonging to the same connection) with matching request parameters on the specified port number. Currently, the administrators have to supply URIs' pattern for HTTP requests/responses, or MySQL command for database queries as the request parameters.

Since all VMs in the host share a single virtual network bridge, we can filter only packets destined to controlled VMs (with *iptables*) and forward copies of the packets (with *xtables-TEE* target) to the sensor VM. This reduces the overall number of packets that our sensor has to process and analyze. We also avoid placing pcap driver directly on the host because it can only capture packets that actually pass through the machine's network interface. By placing pcap driver in the guest VM, we can intercept packets from dependent VMs communicating within the same host.

Our sensors periodically generate a response time summary for each VM. The summary consists of the name of service being monitored, its application tier, VM server, and its response time.

As the sensor is located on the host, the response time is measured starting at the moment when a packet has arrived on the host and stopping when a response packet has been observed by the sensor. In our test environment, the network propagation time is negligible since all hosts are located on the same local area network.

3.2 Actuators

Actuators are small agents installed on the host. They adjust the hypervisor parameter as specified by the controller. Currently our actuators can control the number of CPU shares allocated for VMs on physical hosts. It is possible to extend the actuator to control other system parameters.

In our test environment we adjust the scheduler level of CPU share for each VM using Linux Control Groups subsystem (cgroups.) Cgroups allows us to set the CPU share for each process running on the host. By default, KVM utilizes the Linux kernel's Completely Fair Scheduler (CFS.) The scheduler's behavior is configurable via cgroups cpu share (*cpu.shares*).

Cgroups allows us to set the CPU share for each process running on the host. We use the default Linux Completely Fair Scheduler (CFS) configurable via cgroups CPU share (*cpu.shares*). In the CFS scheduler, each process (or a virtual CPU) is given 1024 shares, unless configured otherwise. The portion of time scheduled for the process is calculated as a ratio between the number of the shares given to the process and the sum of all shares given to every runnable process (on the same physical CPU).

Moreover, the CFS scheduler exhibits work-conserving behavior. This means that if a process happens to be the only one running on a CPU, it gets all available CPU time regardless of the number of shares allocated. Such behavior also indicates that the share configured for CFS does not constitute CPU limits for the process.

In a system with multiple CPUs, the scheduler also utilizes a load-balancer which tries to balance the amount of workload equally amongst each CPU. However, the load-balancer can move a virtual CPU of a VM after it is assigned a preferred number of shares. Such behavior can lead to inaccurate measure between the number of share allocated and the observed applications' performance. In order to effectively control the scheduling parameter, we also have to pin CPUs of all controlled VMs on the same physical core. This makes the relationship between the number of share allocated and the measured response time to be more stable. Our actuators then only have to set the share to match a number specified by the controller. It is the controller's task to find the best possible share for the current workload.

3.3 Modeler

The modeler creates a performance model for controlled applications by resolving its internal parameters based on sensor inputs. The obtained model can later be used to predict the application's performance for specified system configurations.

Our modeler updates a prediction model for the application performance based on the sensor inputs. The model is based on observations between the measured response times from different application tiers. The model uses control signals (CPU share) and measured input (individual tier response time from the sensors.) Although building an accurate model may be a time-consuming process and could be applied only to a specific application, we found that an intuitive model based on application tier relationship could be used to derive a practical performance prediction model.

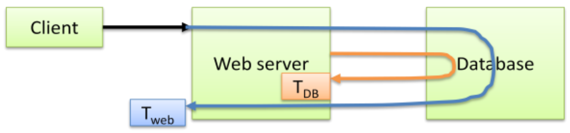

Figure 3: Two-tier web application model

Consider a generic two-tier web application shown in Figure 3; we could build an empirical model for the end-to-end system response time as a linear combination of the time spent in the database and web tier. When a client requests a (dynamic) web page, the web server will make additional requests to the database. The web server then processes the responses before returning the value to the client. Assuming that our concerned requests exhibits similar behavior, the relationship between the web response time (T_{web}) and the database response time (T_{web}) could be represented by $T_{web} = A \cdot T_{DB} + B$.

For example, in Figure 4, the response time used to access a Wordpress home page (a popular blogging web application) exhibits a linear relationship with its database server response time. When the time takes to process database requests increases (due to additional load from another VMs residing on the same host with the database server), the overall web response time also increases.

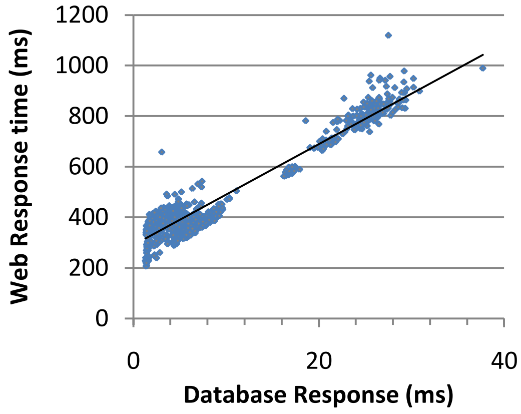

Figure 4: Linear relationship between web and database server response time

We can use this linear model to predict possible performance values for the next sensor interval. Given previous measurement values for the web (T'_{web}) and database response time(T'_{DB}), we can represent the current measurement from our sensor as $T_{web} = T'_{web} + A \cdot (T_{DB} - T'_{DB})$. By performing an ordinary least-square regression on multiple data points (obtained from the sensors), we can estimate the common coefficient A and use the same equation to predict the web response time for the next sensor interval.

If we want to be able to adjust the number of shares for the database VM, we also have to find a relationship between the database server CPU share allocation (S_{CPU}) and the database response time(T_{DB}). On a physical host with very high CPU utilization, which represents a worst-case scenario for consolidation, we found that the relationship between the database response time and its CPU share could be represented by a power law curve $(T_{DB} = a \cdot S_{CPU}^{b})$. Figure 5 shows the observed relationship between the response time of the database server and the number of CPU shares allocated for the database VM. We can also obtain the relationship coefficient by fitting a least square on the log-scale of T_{DB} and S_{CPU} , i.e. $\ln T_{DB} = a + b \ln S_{CPU}$

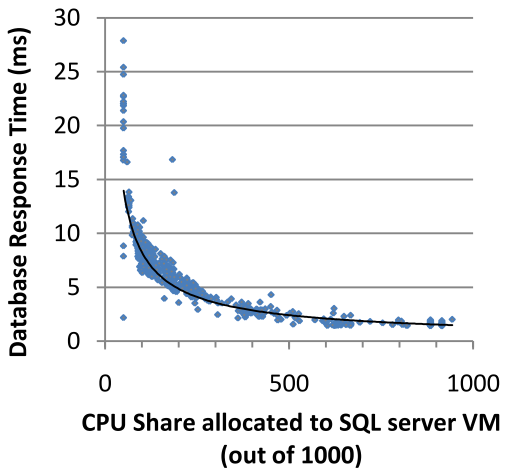

Figure 5: Power law relationship between the database server's response time and its allocated shares

Note that the model given in this section might initially seem to be very specific to our scenario. However, such scenarios are quite common in actual deployments. For example, the linear relationship can be directly applied to many existing web applications. The relationship between allocated CPU shares and the database server response time can also be used to approximate other scenarios where a controlled VM is placed on a very busy host.

Additionally, our model parameter can be obtained on-line by periodically updating the regression parameters with recent measurements. This also allows our system to dynamically adapt its model based on the current level of workloads. However, since our current model relies on many past sensor readings, its ability to adjust the models for sudden change of workload levels will be limited.

3.4 Controller

The controller is the final component which glues all the pieces of our system together. Our controller takes the updated model obtained by the modeler and sensor inputs from the current interval. It then tries to find the minimal virtual CPU allocation that yields the response time closest to the one defined in the SLOs.

Our controller also utilizes both long-term and short-term prediction. The long-term prediction uses the moving average value generated from previous sensor readings as the input for the model. The short-term prediction uses the most recent sensor reading as the model input. The controller primarily determines the number of shares based on the long-term prediction to maintain system stability. However, the short-term prediction is utilized when the sensors' reading shows SLO violations. This allows the system to avoid immediate SLO violations while still maintaining stability.

Once the control decision has been made, the controller forwards the result to the actuator which actually adjusts the system based on the control signal.

In more complex scenarios, we may have to optimize for a large number of potential parameters. We could view such scenario as a state-space search problem and additional heuristic will be needed. Alternatively, we are currently exploring methods used in classical control theory which could be applied to our linear models.

4. Evaluation

4.1 Experimental Setup

Although the framework of our system design is generic, we deployed a proof-of-concept system on an example system as shown below. The targeted application is a blogging web application (Wordpress) which consists of a web server (Apache and PHP) and a database server (MySQL).

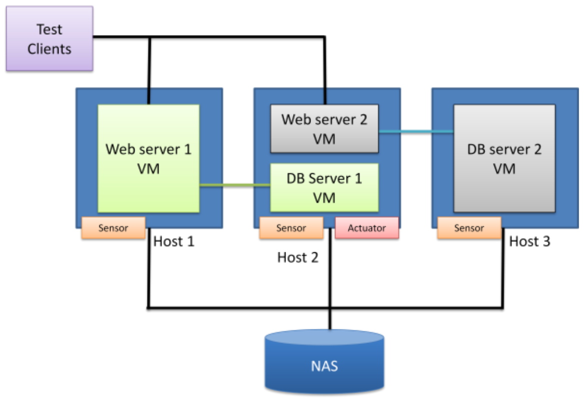

Figure 6: Test setup for evaluation

Our test system setup is shown in Figure 6. We run two instances of the described web application. Each instance may have different service level requirement. This represents differentiate levels of service demanded in actual infrastructure deployment and will be described in the evaluation.

We use Linux KVM as our hypervisor in the experiment. The deployed operating system on all systems is Fedora 12 with Linux 2.6.33 kernel. The host systems contain Intel Quad Core Q6600 with 4GB of memory. Each VM is allocated 1 GB memory with one virtual CPU. The VM image is storing on a dedicate NFS server on the local network. As our test workload fits in the system memory, the storage system does not cause a bottleneck in our test scenario. The web server is more CPU-intensive, compared to the database. Client loads are generated from other machines located on the same local area network. Each client is associated with a single web server VM. Every 1,000ms, the client generates a request to the home page on its associated web server.

We placed sensor on all participating hosts. For the purpose of evaluation, we only concern with the actuator on Host 2 where potential contention could occur. The actuator needs to arbitrate the amount of CPU shares allocated between the web and the database server for two different services with different service response time objectives. Both sensor and control interval are set to 5,000ms.

4.2 Evaluation Result

Our evaluations suggest that our control system can be used to maintain the service level objectives for the hosted applications. It could also react to change in workload level.

4.2.1 Multiple SLOs

Our control system allows multiple service level objectives to be achieved. In this experiment, we set the demand so that the expected end-to-end response time for the blogging web application instance 1 should be 800ms while the response time for instance 2 should be 4,000ms. Note that this represents differentiate level of services, and the objective is described in term of the end-user experience. We expect that our control system will try to adjust the CPU allocation on host 2 so that both SLOs could be met. As for the driving workload, instance 1 was serving 2 concurrent clients. Instance 2 was serving 15 concurrent clients.

The system response time and its target for each instances is shown in Figure 7 and 8. Figure 7 shows the result when we do not use our control system and each VM is allocated the default number of CPU shares. Figure 8 shows the result when we enable our control system. The absolute mean deviation from the target response time for each instance is shown in Table 1.

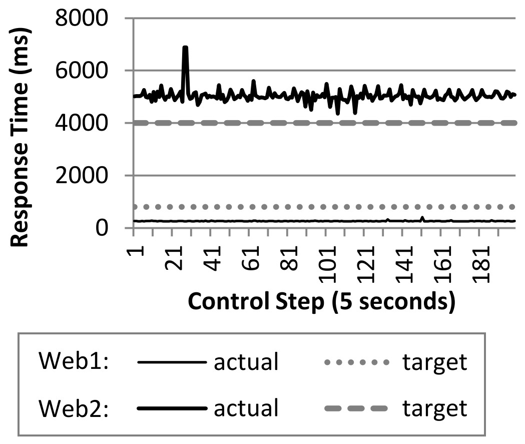

Figure 7: Response time without the control system (static workload)

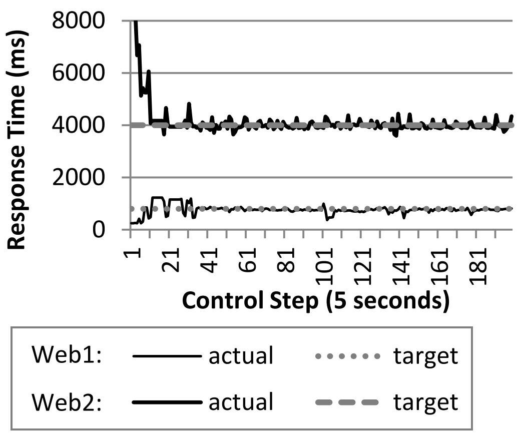

Figure 8: Response time with the control system (static workload)

Application Instances	Mean Deviation from SLOs	
	Without Control	With Control
Instance 1	540 ms	109 ms
Instance 2	1043 ms	282 ms

Table 1: Mean deviations for static workloads

Without the control system, the default number of shares allocated for the database VM instance 1 is too high. Therefore, its response time is much lower than the expected value. However, the response time for instance 2 is also much higher than the SLO specifies as too many resources are given to instance 1. Such system fails to meet the given service demands for instance 2.

With our control system, both instances can be satisfied as the controller adjusts the share to track the expected response time. As a result, both instances can operate within the demanded response time. The share allocated for the database VM for instance 1 is shown in Figure 9. Initially, the adjusted allocation will have high variance as it attempts to find the stable operating points.

Once the operating points have been found, it is also possible that the controller will react to unanticipated system event (such as disk paging, or periodic system maintenance tasks.) These events are indicated by occasional spikes in the response time and the share graph. However, the control system will finally try to revert back to its normal operating points in order to maintain the SLO. The current controller takes about one minute to readjust after such event occurs.

Our current implementation of the controller only attempts to match the actual and the expected response time. As a result, some of the requests may go over the given requested time. In actual deployment, it is possible to specify a lower expected response time than the wanted limits to account for the variances.

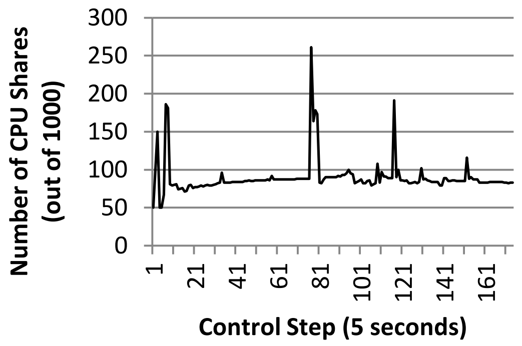

Figure 9: Number of CPU shares allocated by the controller for instance 1's database server

Note that it is also possible for system administrators to manually analyze the workload characteristics and preset the allocation accordingly. However, such task is time-consuming and the administrator may not be able to react as quickly to changes in workload or other system events.

4.2.2 Dynamic Workload

Another benefit of having the control system is it can adapt the allocation for dynamically changing workload. The result in this section shows the effectiveness of the control system while workload level changes for instance 1. The workload level for the instance is shown in Figure 12. The system setup is the same as in previous section. The differences are the number of concurrent clients for instance 1. Also, due to higher overall load, we set the response time required by instance 1 to 1,000ms. For non-controlled system, the number of CPU shares for DB server 1 has been set to satisfy the average load over the evaluation period.

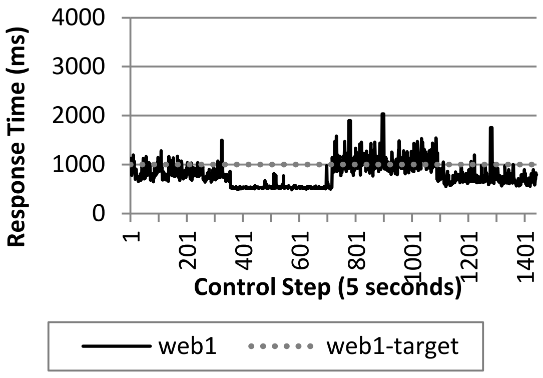

Figure 10: Response time without the control system (dynamic workload)

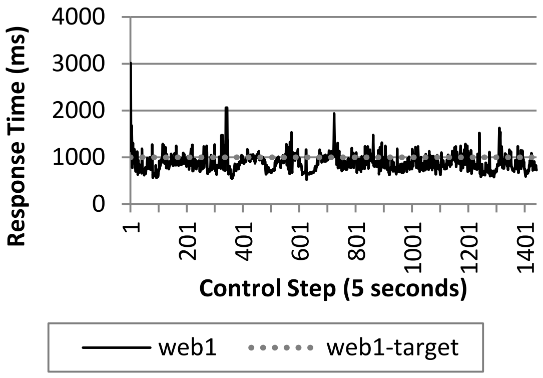

Figure 11: Response time with the control system (dynamic workload)

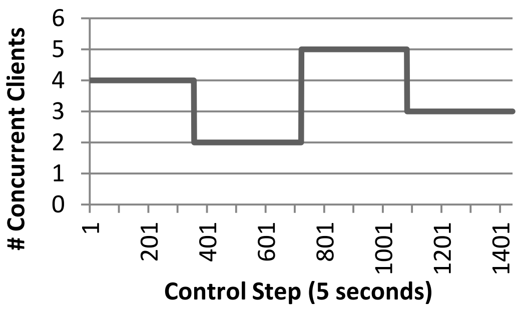

Figure 12: Number of concurrent clients over the test period

Application Instances	Mean Deviation from SLOs	
	Without Control	With Control
Instance 1	276 ms	182 ms

Table 2: **Mean deviations for dynamic workloads**

Figure 10 shows the response time of the web server when the control system is not enabled. Figure 11 shows the response time of the web server when the control system is enabled. The absolute mean deviation from the target response time for instance 1 is also shown in Table 2.

Without the control system, it is possible that, for particular level of workload, the SLO could be easily met because the workload level is well below the average. However, when the level of work load is higher than the average, the instance also fails to meet the SLOs provided.

With the control system, the response time tracks more closely to the expected response. However, the controller may not react as quickly as in the static workload cases. This behavior happens because our model relies on past sensor readings to build up the system models. After the workload level has been changed, the system has to readjust itself and settle to a new model. However, the system can still maintain the target workload level, although it shows higher degree of variances.

5. Related works

Existing commercial solutions remain focused on the resource utilization aspect of VMs, not the applications' performance. Current management tools are capable of reacting to high levels of resource utilization by performing live migrations [7] to reduce the hardware usage.

Existing tools could assist in capacity planning by profiling the hardware utilization level and forecasting future resource demands in datacenters [8]. However, such tools do not directly address the problem of managing the applications' performance.

Previous works have been done to investigate the behavioral model of multi-tier application using profiling-based methods [2] [6]. Such model only predicts long-term statistical behavior and is applicable for static workloads.

Control systems have been used in tuning computer applications' parameters [4] [5]. Researchers have applied control-theoretical approach to address VMs' resource allocation [1] [3]. Such system adjusts its model by observing only the clients' response time whereas our system also responds to performance changes in related application tiers. Feedback-controlled systems have also been investigated in order to improve the system utilization for CPU throughput-based applications [11]. Our work could compliment such method as we are focusing on achieving target response time for delay-based applications.

6. Discussions & Future Works

Our results suggest that it is possible to use a control system to maintain target SLOs on virtualized system and also able to react to changes in workload levels. With the control system, administrators will be able to deploy virtualized workload without concerning about low level system-configuration such as CPU shares.

6.1 Workload modeling

Actual enterprise applications could be much more complicated than the current model given in this paper. The linearity assumptions may not be held for complex chain of dependent VMs. We are interesting to explore possible composition models (such as Markov Chains) that can be used to approximate the response time performance of such distributed applications. It should be possible to derive a more accurate performance model for complex application based on a composition of simpler models such as those described in this paper.

Our initial model only captures direct dependency between application tier (e.g. a web server directly makes request to a database server.) We also investigate the performance behavior and its relation between particular types of behaviors. These include partitioned requests, load-balance, or aggregate behavior of the application tiers. Such model could give us more insight into the relationship between the performance and application's composition which allows us to generate a model for complex applications.

6.2 Control Parameters

In this paper, we have been only experimented with controlling the CPU share allocation. In actual system, more control parameters could be used to change the behavior of the VMs. For example, it is possible to associate traffic for different VMs with multiple network traffic classes. This allows the system to have more control over the queuing and priority for behavior of the VMs' network traffic. Similarly, for local storage control, it is possible to use I/O-controller [10] to control the share for disk I/O access.

7. Conclusions

We presented an automated control system for virtualized services. Our system suggests that it is possible to use intuitive models based on observable response time incurred by multiple application tiers as a model for the overall performance. The models are also used in conjunction with a control system to determine the optimal share allocation for the controlled VMs. Our system helps maintain the expected level of service response time while adjusting the allocation to meets the demand for different level of workloads. Such behavior allows administrator to simply specify the required end-to-end service-level response time for each application, without the need of constant monitoring or understanding complex behavior of the applications. Our system helps simplifying the task of managing the performance of many VMs already exists in today's datacenter.

8. References

[1] P. Padala, K.Hou, K. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, and A. Merchant. Automated control of multiple virtualized resources. EuroSys 2009.

[2] T. Wood, L. Cherkasova, K. Ozonat, and P. Shenoy. Profiling and Modeling Resource Usage of Virtualized Applications. Middleware 2008.

[3] X. Liu, X. Zhu, P. Padala, Z. Wang, and S. Singhal. Optimal Multivariate Control for Differentiated Services on a Shared Hosting Platform. IEEE CDC 2007.

[4] Y. Diao, J. Hellerstein, S. Parekh, R. Griffith, G. Kaiser, and D. Phung, A control theory foundation for self-managing computing systems, IEEE journal on selected areas in communications, Dec. 2005

[5] Y. Diao, N. Gandhi, J. Hellerstein, S. Parekh, and D. Tilbury. Using MIMO feedback control to enforce policies for interrelated metrics with application to the Apache Web server. Network Operations and Management Symposium, 2002.

[6] B. Urgaonkar, P. Shenoy, and T. Roscoe. Resource overbooking and application profiling in shared hosting platforms. OSDI 2002.

[7] VMware,Inc. VMware Infrastructure: Resource Management with VMware DRS. http://www.vmware.com/pdf/vmware_drs_wp.pdf

[8] VMware Inc. VMware vCenter CapacityIQ. http://www.vmware.com/products/vcenter-capacityiq/

[9] VMware, Inc. vSphere Resource Management Guide. http://www.vmware.com/pdf/vsphere4/r40/vsp_40_resource_mgmt.pdf

[10] A. Gulati, I. Ahmad, and C. A. Waldspurger. Parda: Proportional allocation of resources for distributed storage access. FAST, February 2009.

[11] Nathuji, R., Kansal, A., and Ghaffarkhah, A. 2010. Q-clouds: managing performance interference effects for QoS-aware clouds. EuroSys, April 2010.