Empirical Virtual Machine Models for Performance Guarantees

Andrew Turner
andrewtu@cmu.edu

Akkarit Sangpetch
asangpet@andrew.cmu.edu

Hyong S. Kim
kim@ece.cmu.edu

Department of Electrical and Computer Engineering
Carnegie Mellon University
Pittsburgh, PA, USA

ABSTRACT

Existing Virtual Machine (VM) management systems rely on host resource utilization metrics to allocate and schedule VMs. Many management systems only consolidate and migrate VMs based on hosts' CPU utilizations. However, the performance of delay-sensitive workloads, such as web services and online transaction processing, can be severely degraded by contention on numerous of the hosts' components. Current VM management systems typically use threshold based rules to decide when to migrate VMs, rather than using application-level performance. This means that they cannot easily provide application-level service level objective (SLO) guarantees. Providing SLO guarantees is even more difficult when considering that today's enterprise applications often consist of multiple VM tiers.

In this paper we show how the performance of a multi-tiered VM application can be empirically captured, modeled and scaled. This allows our management system to guarantee application-level performance, despite variable host utilization and VM workload levels. Additionally, it can predict the performance of an application at host utilization levels that have not been previously observed. This is achieved by performing regression analysis on the previously observed values and scaling the applications performance model. This allows the performance of a VM to be predicted before it is migrated to or from a new host. We have found that by dynamically, rather than statically, allocating resources, average response time can be improved by 30%. Additionally, we found that resource allocations can be reduced by 20%, with no degradation in response time.

1. INTRODUCTION

Modern data centers contain a large number of virtual machines (VMs). Additionally, internet Cloud services use VMs to run multiple applications across multiple physical servers, under the premise that the Cloud is a single resource pool. While hypervisor vendors such as VMware [1], Citrix [2] and Microsoft [3] tout the potential benefits of VMs, these benefits are not always fully realized. This is typically due to increased overheads and resource contention cause by other VMs. In this paper we show how application-level performance can be guaranteed for multi-tier VM applications. Additionally, we show how hardware utilization can be increased over current VM management systems by more densely packing VMs than threshold based systems. Finally, we show that the overall performance of the applications in a datacenter can be improved by dynamically setting resource allocation levels.

VMs were originally deployed as a way to increase resource utilization levels. This is achieved by consolidating multiple machines that have low resource utilization levels onto a single physical host, saving both hardware capital and energy costs. This is possible as VMs are isolated from each other by the hypervisor, allowing them to share the same physical resources. Additionally, modern VMs can be live-migrated [21] and will run on heterogeneous hardware. While consolidating under-utilized applications is easy, consolidating even moderately used applications can be difficult. This is because VMs are not entirely isolated from each other, and virtualization adds additional overhead. Thus, two VMs running on the same physical host can have an impact on each other's performance; as shown in [12] and [13].

To achieve the greatest amount of capital and energy savings, VMs must be placed to minimize the number of physical hosts required. However, a placement scheme must also ensure that the applications' performances remain at an acceptable level. To achieve this, VMs must be placed in such a way as to minimize the performance impact they have on each other. Current placement schemes

primarily focus on setting utilization threshold levels. However, resource utilization levels can be a poor indicator of application level performance**.** This suggests that VM placement schemes should not be solely based on the idea of bin-packing resource utilization levels.

Commercial VM placement technologies, such as VMware Distributed Resource Scheduler (DRS)[4], place VMs based on resource utilization levels. DRS uses the VMs' CPU utilization level and RAM usage commitment to automatically decide which VMs should be placed on which physical hosts. VMs are then migrated between hosts as resource utilization levels change. Placement schemes such as this rely on the assumption that resource utilization levels reflect application-level performance. However, as resource utilization levels do not always reflect application level performance, such a scheme cannot easily guarantee application-level SLO.

In this paper we show that a VM management system can model multi-tiered applications to guarantee application-level SLO. This would allow system administrators to choose performance guarantees, such as response time < 500ms, without having to manually configure resource allocations. We show how the applications' model can be scaled to unobserved utilization levels, to allow SLO guarantees despite varied host workloads. Additionally, scaling the applications' model can predict the performance impact on an application before migrating VMs to or from a host where one of the application's tiers resides. Lastly, we show that modeling applications can help to more effectively and flexibly place VMs over a threshold based approach. For example, an application's VMs tiers can be placed to minimize power usage, or to minimize the risk of a certain response time being exceeded.

The paper is organized as follows: In Section 2 we discuss related works. In Section 3 we describe our system. In Section 4 we describe our experimental setup. In Section 5 we evaluate our results, followed by our conclusion in Section 6.

2. Related Works

There are many works on maximizing resource utilization levels and increasing efficiency in the virtual environment [5], [6], [7]. Existing commercial products are also available to facilitate the task of managing and relocating VMs. For example, VMware DRS [4] monitors the CPU and memory usage of VMs and migrates them to balance utilization levels. Similarly, VMware Distributed Power Management [4] minimizes the power usage of a data center by migrating VMs from lowly-utilized hosts and powers them off. Both systems focus on maintaining CPU and memory usage. Our work focuses on service level performance.

Recent efforts such as [8], [14], [15] and [16] have attempted to further increase resource utilization levels by migrating VMs. Each VM's resource utilization level is monitored and VMs are migrated to new hosts such that host resource utilization is maximized, and no host is overloaded. Kochut et. al [14] consider both autocorrelation and a periodogram to decide which VMs are best candidates to be placed together. Ideally, colocated VMs should have a low probability of overloading the host. Hermenier et. al. [15] consider the order that the migrations occur in addition to which VMs to migrate to minimize the impact of the migrations on system performance.

Another method to maximize resource utilization levels is overbooking resources. Urgaonkar et. al. [9] shows that a 500% increase in utility can be achieved by overbooking hosts by 5% of their peak load values. This only causes a 4.6% decrease in overall throughput. However, the study focuses on a shared hosting environment, not a virtual one, and considers neither contention nor the overhead caused by a virtual environment.

To maintain end-to-end service level performance, Stewart et.al [11] offers a response time prediction model. The model is based on an identified trait model for multi-tier applications. Their work focuses on predicting the service response time, based on pre-identified trait model relationships between processor properties and observed response time. Liu et. al. [18] use an autoregressive model to control CPU allocation. This allows VMs to be assigned a certain resource level as to normalize multiple applications' performance. Padala et. al. [19] and [20] have further used an autoregressive moving average to assign VM multiple resources.

3. Motivation

The motivation behind our work is to remove the need for administrators to perform resource allocation in the virtual environment. Our system aims to achieve SLOs by automatically allocating resources when they are required by a VM. Resources are then taken away and reallocated to other VMs as resource needs change. In a non-virtualized datacenter, applications avoid performance degradations by being isolated and run on dedicated hardware. However, this typically means low resource utilization levels, resulting in high hardware and energy costs. It is therefore attractive to place applications within VMs to reduce

these costs. However, once applications are placed in a virtual environment, they must contest for resources as they are no longer entirely isolated. This can cause applications to suffer from performance degradations.

To ensure applications perform satisfactorily, Virtual Machine Monitors can be set to allocate a certain amount of hardware resources to each VM. There are however, a number of problems with current VM management systems. Firstly, administrators typically need to set the resource allocation levels manually. This requires administrators to monitor their applications' performances, and set each VMs' resource allocation and priority in the VM management system. This task can be made more difficult if the VMs' resource requirements frequently change. Secondly, the resource allocation levels only guarantee that a VM will receive a certain share of a resource. They do not provide any application-level performance guarantee. This can lead to lower hardware utilization levels, as administrators will typically over-provision resource allocations to ensure satisfactory performance. Lastly, administrators must manually set the utilization levels at which VMs will be migrated to and from hosts. This can again lead to lower hardware utilization as migration thresholds must be set low enough to ensure application-level performance does not suffer due to high resource contention.

To address these problems, our system monitors application-level performance and automatically allocates VMs the minimum level of resources they need to meet an application-level SLO guarantee. Our system works by monitoring the applications' performances at various user, resource allocation, and resource contention levels. Resource contention occurs on a host when multiple VMs require the use of the same resource. Once our system has multiple readings at different values, it can interpolate the minimum resource allocations needed to achieve a certain response time.

Figure 1 shows the basic flow of information in the management system. The process starts by an application reporting its response time and the level of resource contention on each host where one of its VMs resides. The management system then chooses the model that best describes the application's response time based on the current resource contention levels. Initially, this model will be empty as the management system does not have any data about the application. The model is then stretched based on how far the readings in the model are from the current resource contention levels. The missing data points in the model are then interpolated from the data that is available. The minimum resource allocation levels that allow the application to meet its response time target are then found in the interpolated model. Finally, the resource allocations are set on the hosts, and the hosts wait to take a new reading to report to the management system.

The applications' performance models are created automatically by analyzing the performance achieved at the various resource allocation levels. Although such models could contain millions of potential data points, we have found that a model can be constructed with only 10's of data points. Although each application will have a unique model, in future work it may be possible to apply a generic model to different types of applications, and then quickly tailor them with even fewer data points.

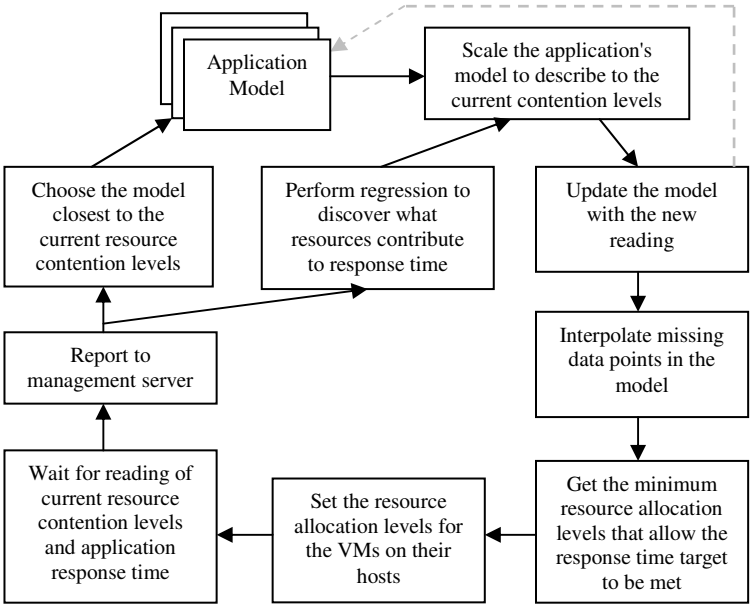

Figure 1: Management system flow

In addition to interpolating the minimum resource allocations needed to achieve a certain response time, our system can also interpolate a response time value for a given resource contention level. This can help predict the performance of a VM before it is migrated to or from a host. This allows migration decisions to be made more flexibly, as they can be based on VM performance, rather than occurring at a fixed threshold.

As many of today's datacenter applications rely on multiple tiers, our system allows for this. Our system sets the resource allocations at each tier, such that the total response time experienced by the end user is below the SLO target. This allows an administrator to configure a single SLO value for an

entire application stack. This is in contrast to current management systems, where the resource allocation must be configured individually for each tier.

3.1 Monitoring

To collect the data we need for our system we record the application's response time at its first tier; as shown in Figure 2. Throughput based applications can be monitored in a similar fashion, with throughput per time period recorded rather than response time. While monitoring response times at each individual tier could possibly provide a more accurate model, such monitoring would incur a significant overhead. Additionally, monitoring at intermediate tiers does not always reflect the overall performance characteristics experienced by the end-users.

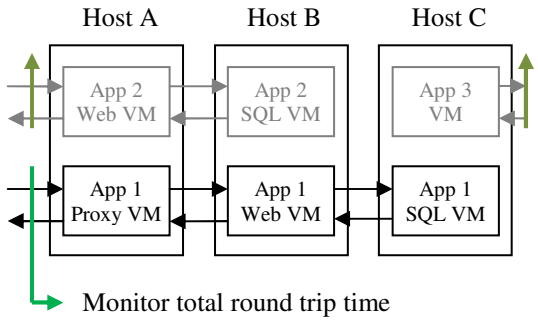

Figure 2: Response time monitoring

The data we capture are the applications' average total response time, CPU utilization, and storage and network throughput. All of the data are captured outside of the VMs, thereby not requiring a client to be inside the VMs. To allow our system to react quickly to changes, we take a reading every 10 seconds. This period could be increased or decreased as needed, depending on the system being controlled.

After the data is captured, it is passed to a server and added to our model. The model then interpolates the resource allocations that each VM should receive to meet a specified response time and chooses the minimum value. These resource allocations are then set on each host so that each VM receives the amount of resources calculated by the model, as shown in Figure 3.

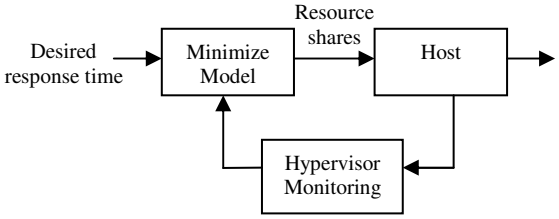

Figure 3: Control flow

3.2 Model Interpolation

Once the data is reported to our management system it is added to an application system model. An application's system model describes the previous response time values that we have observed for an application at various user, resource allocation, and hardware contention levels. We then use this model to predict the minimum resource allocations an application's VMs require to meet a certain response time.

To predict the required resource allocation levels we must first identify trends in the data. Figure 4 shows the effect of CPU contention on the host containing the web tier of TPC-W. The CPU contention is the total CPU utilization minus the amount used by the VM itself. As shown, the response time curve follows an exponential distribution. As the data closely fits an exponential distribution very few points are needed during run time to interpolate estimated resource allocation values.

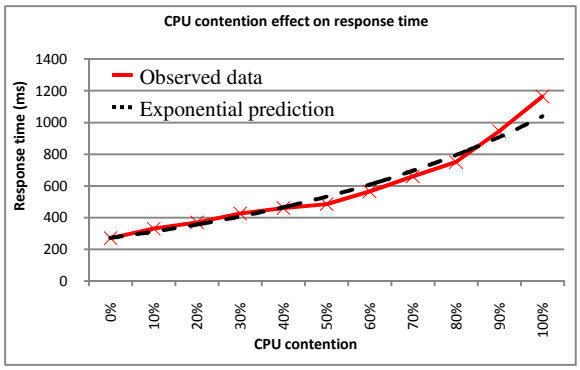

Figure 4: CPU contention and response time degradation

Figure 5 show the response time of TPC-W as the web tier has its CPU allocation changed from 10% to 100%. The resource allocation levels are currently capped to a minimum of 10% in our system as we have found that response times quickly approach infinity (the website crashes) for extremely low resource allocation values. Both the proxy and SQL tiers were set to 80% CPU allocation. As shown, for 45%-100% CPU allocation the response time for all four contention levels can be roughly predicted by the same linear function. For allocation values less than 45%, each contention level follows its own steeper linear function. This occurs as the web server tier is not the bottleneck of the application until it receives less than 45% CPU allocation.

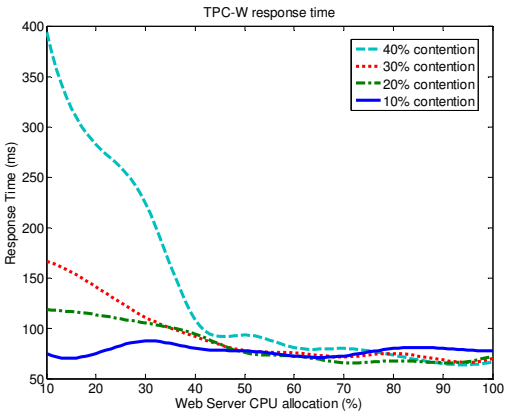

Figure 5: TPC-W response time with proxy and web server set at 80% CPU allocation

If the SLO guarantee we are trying to fulfill is 100ms, for example, Figure 5 would suggest that we allocate the web tier 45% of the CPU share if the CPU contention on the host is 20% or above. However, this only considers a single tier of the application. Figure 6 shows the response time curves when the web tier's CPU allocation is changed from 10% to 100%, but the proxy tier's allocation has been reduced to 30%. In this situation, there is no way to meet the 100ms response time goal if the contention on the host is more than 10%. This is because the proxy tier has become the application's bottleneck, so assigning more resources to the web tier will not significantly improve the response time. Because of this, it is clear that to minimize the resource allocations the model must include every tier of the application as a dimension.

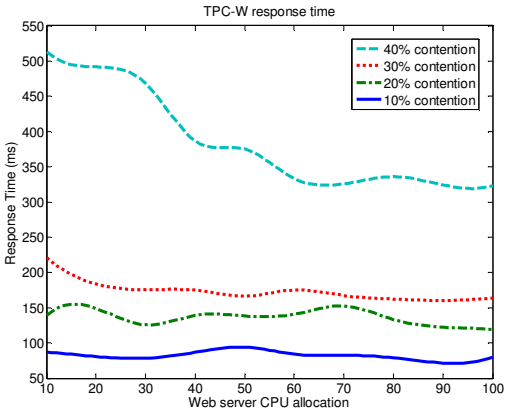

Figure 6: TPC-W response time with proxy set at 80% CPU allocation

Figure 7 shows the surface plot for the TPC-W proxy and web tiers with 300 active users and 40% CPU contention on each host. It should be noted that our system uses data from every application tier and from multiple hardware components. However, displaying graphs with more than three dimensions is difficult.

While Figure 7, 8 and 9 contain hundreds of data points to show the complete resource allocation to response time model, the runtime model does not require this much data. If, for example, the administrator has set 150ms as the SLO target, each model will contain points around that response time, but only a few points for the rest of the model. For example, in Figures 8 and 9 the model will mostly need to record data points between the dotted lines. In addition to having to store less data points, being able to characterize the application with fewer data points helps the model converge and adapt to changes quickly.

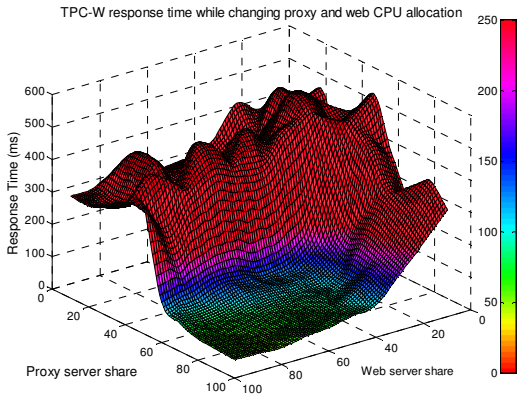

Figure 7: Proxy and Web tier CPU allocation response times for 40% CPU contention

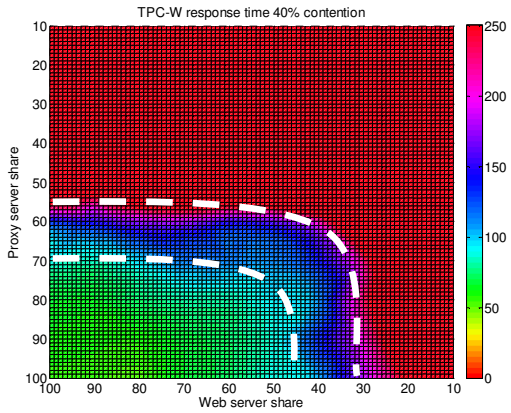

Figure 8: Proxy and Web tier CPU allocation response times for 40%CPU contention

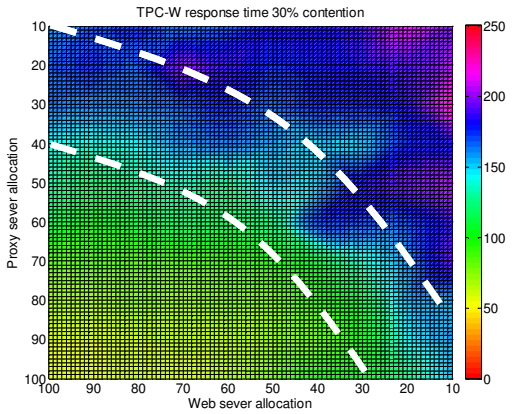

Figure 9: Proxy and Web tier CPU allocation response times for 30% CPU contention

3.3 Dimensional Reduction

As there are potentially thousands, or even millions, of resource contention combinations, it is infeasible to keep a model for every combination we encounter. Instead, we keep a subset of models, and scale the response time values to fit the current contention levels. To achieve this scaling we use the same data used in the resource allocation to response time models (Figures 7, 8 and 9), but instead interpolate contention to response time for a given resource allocation level. We use piecewise multiple linear regression to estimate the value that each point in the model should be scaled by. When we are estimating resource allocation values for a resource contention level that we do not have a model for, the management system needs to choose the model that most accurately represents the current resource contention levels. The model chosen to be scaled is the one with the smallest Euclidean distance from the current resource contention levels.

Figure 10 shows the actual response time for TPC-W and the estimated response time calculated using the regression coefficients. The data shown is a subset of data points where the CPU contention is between 10% and 40% for each tier. As can be seen in Figure 10, the estimated and actual response times are highly correlated, as would be expected given the fit of the data shown in Figure 4.

Figure 11 shows the response time increases as the number of users increase; in this case there is 10% CPU contention on each of the hosts where the TPC-W VMs are placed. As can be seen, the response time increases exponentially with the number of users. This can be accurately represented by linearly scaling three copies of an application's resource allocation model.

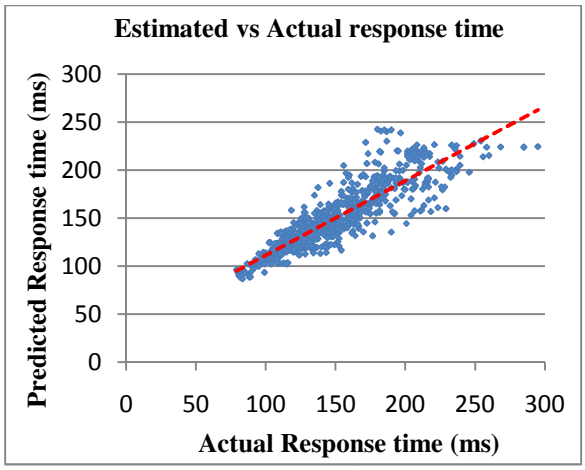

Figure 10: Estimated and Actual TPC-W response time

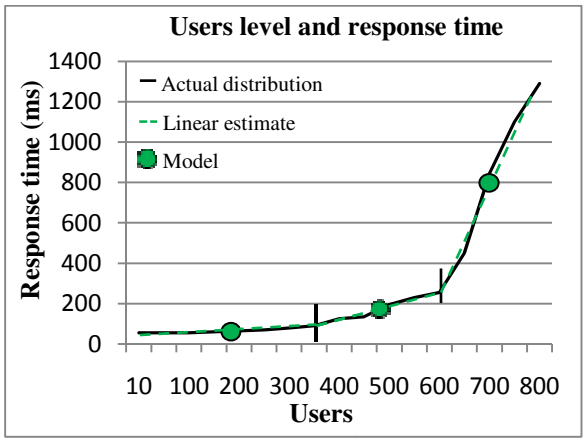

Figure 11: Response time increase vs. user level

Figure 12 shows the degredation in TPC-W's responce time at various CPU contention levels. The responce times shown are when the web tier is assigned 50% or 10% CPU allocation. At 50% CPU share allocation the CPU contention has little affect on the response time. This is because the web tier recieves CPU cycles very frequently, and is not the application's bottleneck. At 10% CPU share allocation the response time quickly degrades to almost a 50% increase in response time with a 10% increase in CPU contention. Even though the CPU had over 40% free cycles, the web tier does not receive its cycles promply enough, cauing it to become the bottleneck tier and causing degraded response time.

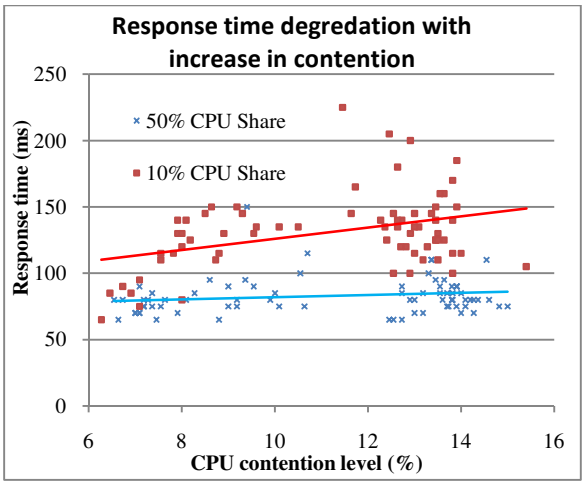

Figure 12: Regression values used to stretch a model

To scale an application's performance model, we multiply the model from the current contention level to the new one for each resource allocation level. For example, if we wanted to know the response time at 20% CPU contention and 50% CPU share allocation we would estimate 73 + 1.1 * 20 = 95ms. If our SLO target is 100ms, we would know that we could place the web tier on a host with 20% CPU contention if it could receive 50% of the CPU share allocation. However, if the host only had 40% CPU share allocation remaining, the estimated response time would be 80 + 1.15 * 20 = 103ms. Therefore, we would not expect that we could palce the web tier on that host.

4. Experimental Setup

4.1 Infrastructure

Our experiments are setup on a flat local area network using commodity hardware. The host operating system is Fedora 12 with Linux kernel 2.6.31. We use KVM as our hypervisor. The VM hosts consist of three nodes with tri-core 2.1 GHz CPU, 4GB RAM. The test clients consist of two nodes with quad-core 2.66 GHz CPU, 4GB RAM. The storage node contains a dual-core 2.8 GHz CPU, 4GB RAM.

The network topology we use is two flat-networks each with one switch: the user data network and the management network. Each physical host has two network interface cards (NICs). One NIC is connected to a user data network using a 24-port Gigabit switch. The user network carries all of the user workload and benchmark traffic. The other NIC is connected to a management network using a separate Gigabit switch as shown in Figure 13. The management network carries management-related commands and network attached storage traffic for the VMs' virtual disk images.

The storage system is hosted on two-spindle RAID-0, 2TB, 7200rpm hard disks. The storage server exports an NFS share. All virtual machine images are served from this location. To ensure network storage was not the bottleneck in our system, we benchmarked the network storage and found it more than capable of handling all of the VMs' disk traffic.

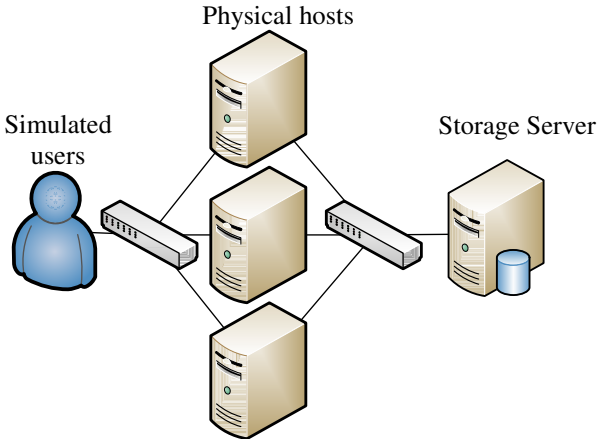

Figure 13: Test bed setup

4.2 Workloads

To test our system we use the TPC-W benchmark suit [22]. We use TPC-W as a test of a real-world delay-sensitive application. TPC-W mimics an online-bookstore application. It consists of an Apache web proxy front-end, a Tomcat application server, and a MySQL database back-end. There are 15 types of page requests. The benchmark client is a closed-loop client which simulates multiple users concurrently accessing the server. TPC-W's performance is measured based on response time for each action performed.

5. Results

In this section we discuss the results from our system. We test our system by running the TPC-W benchmark with each of the application's tiers on a separate host. Each host also contains another VM running an Apache web server hosting computationally intensive web pages. The additional VMs are used to create resource contention on the hosts. They represent other applications that would undoubtedly also be running in a shared virtual environment. The number of requests per second to each Apache server was varied throughout the experiments to change the resource contention levels.

5.1 Meeting SLO target

Figure 14 shows the resulting response times of TPC-W when the resource allocation levels are set manually and when they are controlled by our system. When the resource allocations are set manually, each tier receives the same resource allocation on each host. For example, in the 50% resource allocation experiment, each tier has a fixed 50% resource allocation throughout the experiment.

As can be seen in Figure 14, when using our system TPC-W's response time closely follows the SLO target that is set. It is expected that the response time will oscillate above and below the SLO target as our system attempts to make the median response time equal to the SLO target. It is also evident from Figure 14 that the response time when using our system is usually faster, rather than slower, than the SLO target, and therefore averages to faster than the required SLO value. This is due to the resource allocation optimizer being cautious in its estimates. This is a conscious design decision, as a system that constantly over performs is more useful than a system that constantly under performs.

It can also be seen in Figure 14 that setting the resource allocation levels manually does not always produce a consistent response time. This is because resource contentions may increase over time, but the resource allocations do not. When the resource allocation is set to 50%, TPC-W's response time is faster for a longer period of time than when the SLO target is set to 150ms. However, at time period 480, a 50% resource allocation is no longer sufficient to continue providing that fast response time. However, with a dynamically set resource allocation our system can keep providing the same response time despite the CPU contention increase.

Test	RT average	Resource allocation average	Apache VM average
SLO = 100ms	89ms	48%	125ms
SLO = 150ms	127ms	35%	107ms
50% resource allocation	150ms	50%	120ms
10% resource allocation	355ms	10%	83ms

Table 1: Response time for TPC-W and contention workload

As can be seen in Table 1, despite the 50% resource allocation test having a faster response time for a longer period of time than the SLO 150ms test, its final average response time is greater. Additionally, the SLO 150ms test uses on average 15% less resources to achieve this faster average response time. As TPC-W uses less resources in the SLO 150ms test, the Apache workload on the host receives a greater share of resources; thus reducing its average response time from 120ms to 107ms. This is because the optimizer does not needlessly overprovision TPC-W, allowing the host scheduler to allocate remaining resources as needed. This shows that dynamically setting the resource allocation levels can not only guarantee a specified response time, but is also a more efficient use of resources. In this case, both applications have benefited from faster response times, despite our system only guaranteeing one of them.

Comparing the two tests with the closest resource allocation levels, we find that dynamic resource allocation helps achieve a faster average response time while using overall fewer hardware resources. Even excluding the final 120 readings, where the 50% allocation test performed poorly, dynamic allocation still performs faster, with an average response time of 89ms vs. the static allocation average of 106ms.

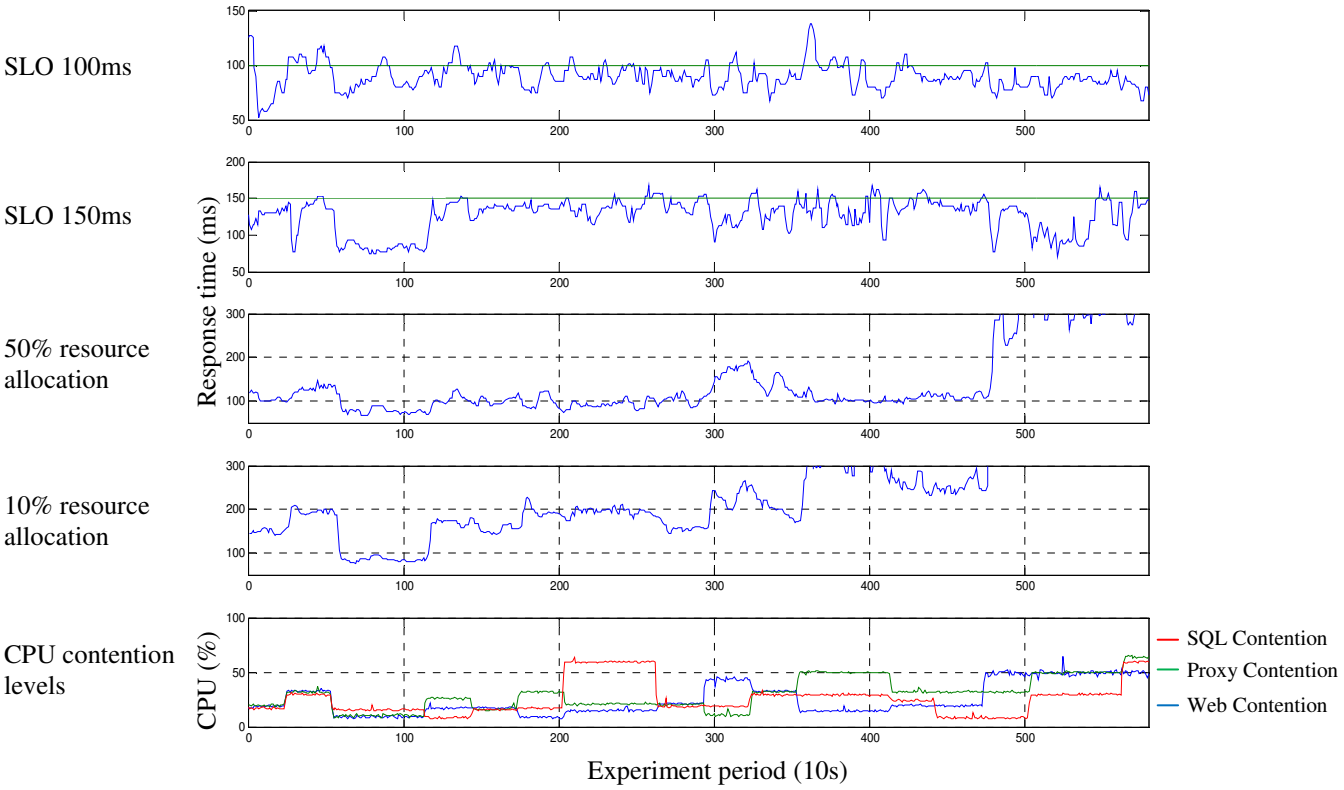

Figure 14: Response time results for dynamic and static resource allocations, changing CPU contention

5.2 Resource Allocation

Figure 15 shows the resource allocation levels that TPC-W received for the SLO 100ms and 150ms tests. The other two tests remain at 50% and 10% allocation throughout and are not shown.

At time period 200 it can be seen that the CPU contention on the SQL VM's host jumps 40%; however, the resource allocation only increases roughly 10%. This shows an advantage of modeling and predicting the application's performance over a more simple resource control scheme, such as increasing the resource allocation by a fixed factor of CPU contention. The regression analysis identifies that the CPU contention on the SQL VM's host does not cause large increases in response time. Therefore, when a model is used to predict the resource allocations for the new contention level, the scaling factor is low. This is in contrast to time period 110, when the CPU contention on the web server VM's host increases by 10%. In this case, the resource allocation increases by 20% in the SLO 150ms test and by 30% in the SLO 100ms test. This is because the model has correctly predicted that increased CPU contention on the web server VM's host will cause an increase in response time and has scaled the resource allocation model accordingly. We can see that the system predicted the correct resource allocation increases in both cases, as the response times for the SLO tests in Figure 14 both change to the configured SLO level at time period 110.

5.3 Change in user levels

Figure 16 shows the TPC-W response time when the number of users is varied during the experiment. We again configure our system to meet either a 100ms or 150ms response time SLO. We also experiment with the VMs resource allocations set statically to either 50% or 10%.

It can be seen from Figure 16 that our system can dynamically adjust resource allocations to meet an SLO despite a varying user level. Our system keeps the response time near the SLO target, whereas the static resource allocation causes response time to vary from 100ms-400ms.

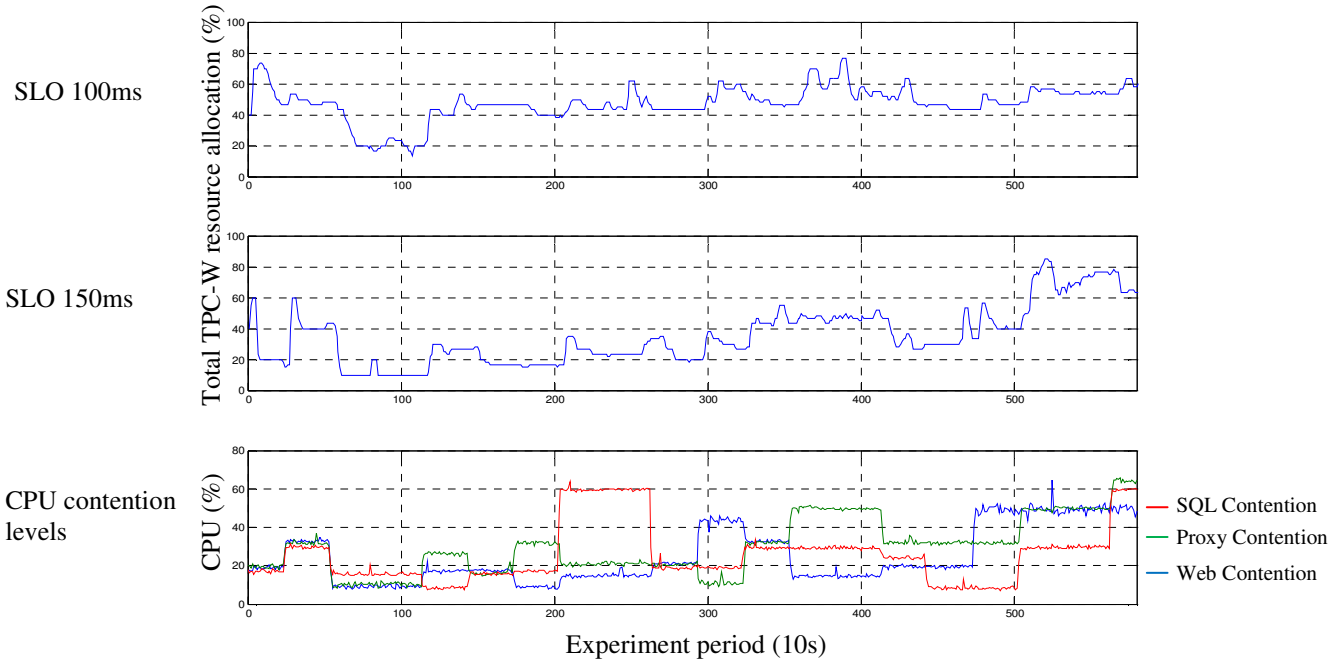

Figure 15: Allocated resource levels for dynamic resource allocation test

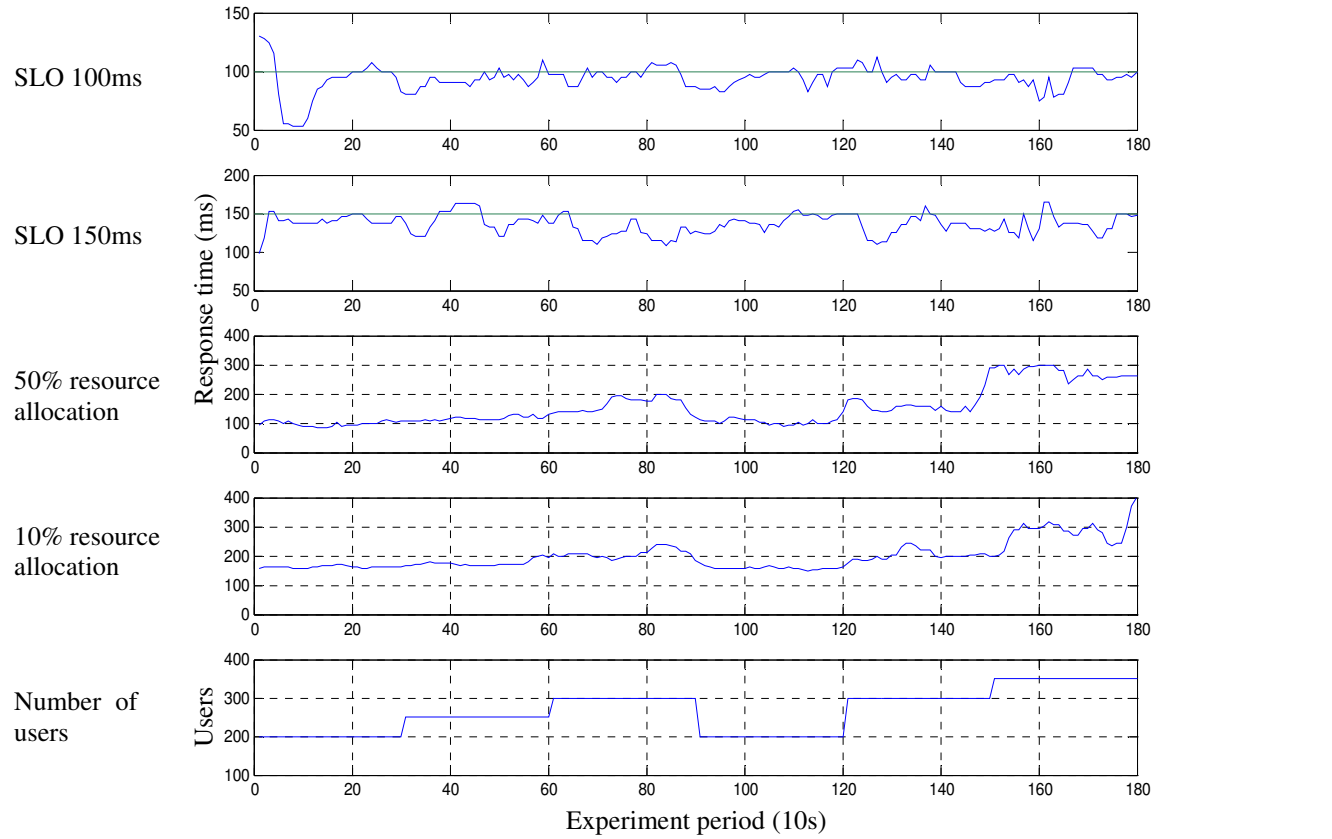

Figure 16: Response time results for dynamic and static resource allocations, changing user level

6. CONCLUSION AND FUTURE WORK

In this work we have shown that applications comprised of multiple VM tiers can meet SLOs by dynamically allocating host resources. We show that by capturing an application's previous performance, we can model and predict the minimum amount of resources it needs to meet an SLO. Additionally, we show that these models can be stretched to changes in host utilization levels. This allows the resource allocation to be quickly altered when resource utilization levels change.

We evaluate our system using TPC-W and setting response time SLO targets. The host utilization is then varied throughout the experiments. Our system adapts to the changes in host utilization levels, and helps maintain TPC-W's response time within the SLO target. Our system also assigns the minimum amount of resources required to meet the SLO, allowing the other application running on the same hosts to improve its performance.

Although our system allows applications to meet SLOs, minimizing the total amount of resources used by each application may not be the most desirable goal in a data center. As VM migration causes both performance degradation and increased utilization, assigning resources in such a way as to lower the number of migrations may achieve lower global resource utilization than attempting to minimize resource allocation alone. Additional study would be needed to analyze the application specific performance degradation caused by migration.

While our current control scheme ensures that VMs receive the correct amount of resources to meet an SLO, it does not actually provide a hard guarantee about the number of violations. In future work we will bound the number and severity of SLO violations to provide administrators with hard guarantees about application level performance.

Additionally, rather than starting with a blank slate for each application, we hope to identify common traits between applications. This will allow performance models to be created and adapted more quickly, and could allow for different modes of control for different application types. This could potentially make the task of bounding the number of SLO violations easier.

7. REFERENCES

[1] VMware vSphere. www.vmware.com/products/vsphere/

[2] Citrix XenServer. http://www.xensource.com/

[3] Microsoft Hyper-V Server. http://www.microsoft.com/hyper-v-server/

[4] VMware Infrastructure: Resource Management with VMware DRS.

[5] Carrera, D. et. al. Utility-based placement of dynamic Web applications with fairness goals, NOMS 2008.

[6] Karve, A., et. al. A. Dynamic placement for clustered web applications. WWW 2006.

[7] Madhukar Korupolu, Aameek Singh, Bhuvan Bamba, Coupled placement in modern data centers, SPDP 2009.

[8] Choi, et.al. Autonomous learning for efficient resource utilization of dynamic VM migration. ICS 2008.

[9] Urgaonkar, B., Shenoy, P., and Roscoe, T. Resource overbooking and application profiling in shared hosting platforms. SIGOPS Oper. Syst. Rev. 36, SI Dec. 2002

[10] Karve, A., et. al. A. Dynamic placement for clustered web applications. WWW 2006.

[11] Stewart, C. et. al. A dollar from 15 cents: cross-platform management for internet services. USENIX 2008.

[12] Cherkasova, L, et. al. Comparison of the Three CPU Schedulers in Xen

[13] Somani, G. Chaudhary, S., Application Performance Isolation in Virtualization, CLOUD `09, IEEE International Conference on Cloud Computing, 2009

[14] Kochut, A., Beaty, K., On Strategies for Dynamic Resource Management in Virtualized Server Environments, IEEE MASCOTS, 2007

[15] Hermenier, F., et. al., Entropy: a consolidation manager for clusters, ACM/Usenix International Conference On Virtual Execution Environments, ACM SIGPLAN/SIGOPS, 2009

[16] Verma, A., Ahuja, P., Neogi, A., pMapper: Power and Migration Cost Aware Application Placement in Virtualized Systems, Middleware 2008, 2008

[17] Bobroff, N.; Kochut, A.; Beaty, K., Dynamic Placement of Virtual Machines for Managing SLA Violations, Integrated Network Management, 2007. IM '07

[18] Lui, X., et. al., Optimal Multivariate Control for Differentiated Services on a Shared Hosting Platform, Decision and Control, 2007

[19] Padala, P., et. al., Adaptive control of virtualized resources in utility computing environments, ACM SIGOPS Operating Systems Review, Volume 41 , Issue 3, 2007

[20] Padala, P., et. al., Automated control of multiple virtualized resources, ACM European conference on Computer systems, Cloud Computing, 2009

[21] Clark, C., et. al., Live migration of virtual machines, USENIX Networked Systems Design & Implementation - Volume 2, 2005

[22] TPC-W, http://www.tpc.org/tpcw/default.asp