Study on performance management and application behavior in virtualized environment

Pengcheng Xiong, Zhikui Wang, Gueyoung Jung and Calton Pu

Abstract— **Control theory has been utilized in recent years to manage the resources in virtualized environment for applications with time-varying resource demand. The systems under control, including the servers and the applications, are taken as black-boxes, and the controllers are generally expected to be adaptive to the underline systems. However, little attention has been paid to the behaviors of the applications themselves, and most of time, single performance target such as the mean response time threshold has been tracked. In this paper, we experimentally show that more than one performance metrics have to be considered to characterize the quality of service that the end users receive when the performance is managed through dynamic resource allocation. Moreover, the behavior of the applications, especially that of the workload generators has significant effect on the quality of the service. Our study provides insights and guidance for feedback control design in virtualized environment.**

I. INTRODUCTION

VIRTUALIZATION enables sharing of physical information technology (IT) resources in cloud computing environments, allowing multiple services to run in different virtual machines (VMs) in a single physical server. Workloads for the services and the enterprise applications can fluctuate considerably, and thus statically-configured virtual resources are often either over-provisioned or over-loaded [1]. To deal with the time-varying demand of the workloads, at least three resource management strategies have been actively studied recently for virtualized environments: capacity planning, virtual machine migration, and dynamic resource allocation. These techniques are complementary to one another because they typically operate at different time scales and different scopes of a data center. Multiple techniques were applied for sharing of the resources, e.g., statistical multiplexing, optimization, and in recent years control theory [1-6].

Control theoretic approaches are natural choices to tackle the time-varying workloads through especially dynamic allocation of virtualized resources. A few representative works follow. Zhu *et al.* [1] and Padala [2-3] use an adaptive controller to control the CPU shares to each virtual machine where the application is held in order to meet the performance requirement. Wang *et al.* [4] compares the CPU utilization and also application performance when using feed-back and feed-forward based controller. Hellerstein *et al.* [5] and Diao *et al.* [6] propose a MIMO controller to control the CPU utilization and memory by tuning Apache's KeepAlive and MaxClient parameters. Although much progress has been made in this area, there are at least two aspects in previous work that can raise issues. First, a single metric like mean response time is usually considered for individual applications, and the performance management problem is formulated as a tracking problem in which the performance of the application is maintained at the target level by pushing the resource allocation to its limit. Second, the systems under control including the workloads and the servers are usually taken as "black-boxes" for controller design and configuration, evaluation of the controllers is done on single benchmark application.

In this paper, we study the performance management problem in virtualized environment through experiments on a test bed with multi-tier applications hosted on multiple virtual machines. We first show that, the end-to-end performance of the applications that can be perceived by the users has to be characterized by multiple metrics, for instance, resource utilization, mean response time (MRT), percentile response time (PRT) and throughput. The performance management problem may not be formulated as "tracking" problem. Second, the behavior of the applications, including that of the workload generators has to be considered for controller design and configuration. Evaluation of the controllers has to be done on multiple applications that can represent different practical scenarios.

II. EXPERIMENTAL SETUP AND APPLICATION MODELS

To evaluate the quality of service provided by applications hosted in virtualized environment, we set up test bed based on Xen technology, and implemented feedback utilization controllers and response time controllers that allocate the CPU resources dynamically to the multi-tier applications.

A. Test bed, utilization control and performance control architecture for a multi-tier application

Figure 1 shows a 3-tier web application. We used RUBiS as the benchmark application [7]. It is an on-line auction benchmark comprised of a front-end Apache Web server, a Tomcat application server, and back-end MySQL database server. Each tier of the application is hosted in one Xen virtual machine. Our test bed consists of three machines as shown in

P. Xiong, G. Jung and C. Pu are with College of Computing, Georgia Institute of Technology, Atlanta, GA 30332 USA. e-mail: xiong@gatech.edu, gueyoung.jung@cc.gatech.edu, calton@cc.gatech.edu)

Z. Wang is with Hewlett Packard Laboratories, Palo Alto, CA 94304 USA. e-mail: zhikui.wang@hp.com

Fig. 1, one for hosting the three VMs, one for client emulator and the last one for performance controller.

For our experimental evaluation purpose, we implemented the nested controllers as shown in Fig. 1. In the inner loop, there is one utilization controller for each VM. In the outer loop, a performance controller is used for end-to-end performance guarantee of the whole application. We use "entitlement" (u) and "consumption/usage" (v) to refer to the CPU shares (in percentage of total CPU capacity) allocated to a virtual machine and the CPU share actually used by the virtual machine respectively. We use "utilization" (r) to refer to the ratio between consumption and entitlement, i.e, $r=v/u$.

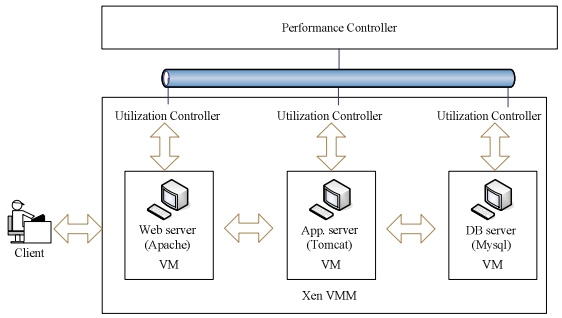


Figure 1 Nested controller for a multi-tier web application

A simple utilization controller is used to keep utilization towards its target r_{ref}: $u(k)= v(k\text{-}1)/r_{ref}$. In our experiments, the CPU consumption is collected using "*xentop*" command. The CPU shares are entitled using "*xm sched-credit-c*" command. The new CPU shares were changed every 10 seconds.

In the experiments, we use an end-to-end mean response time controller for performance control. It sets the utilization targets for the inner loop every 30 seconds through an integral controller $r(j)= r(j\text{-}1)+ G(MRT_{ref} - mrt(j\text{-}1))/ MRT_{ref}$, to track the MRT target MRT_{ref}. The parameter G is the integrator gain, set to 0.1 in our experiments.

Many controllers have been developed for dynamic resource allocation in virtualized servers The simplicity of this architecture makes it easier to analyze and understand the effect of factors such as application behavior on the performance of the closed-loop system. In our experiments, the total CPU requests from the utilization controllers are always less than the CPU capacity to avoid resource contention, for which arbitration is needed [5].

B. *System models for the application: open or closed*

Although many factors can affect the close-loop performance, we focus on workload generators in this paper which little attention has been paid to in previous work but actually has significant effect on the close-loop system.

As denoted in [8], a workload generator is called closed if new user requests are triggered only after previous requests have been completed or timeout. The default RUBiS client emulator, called ORC in this paper, is closed. Each session represents a virtual user. Each virtual user connects to the front-end Apache server using a persistent HTTP connection. Within each session, the client generates new request after previous request is replied, and waits for an exponentially distributed "think time". We denote the total number of sessions as multiprogramming level (MPL), a parameter that represents the intensity of the workload. There are 26 transaction types in RUBiS. The types of the new request generated by the virtual users are defined by a transition table. In our experiments, we used "Browsing mix" that has 10 transaction types, e.g., Home, Browse, ViewItem.

There are also many open workload generators, which generates new requests independently of completion of previous requests. We modified the original RUBiS client emulator so that it can work as an open system, and generate new requests with exponentially distributed thinking time. We call it MRC.

III. PERFORMANCE THROUGH UTILIZATION CONTROL

The relationship between resource utilization and end-to-end performance has been studied and modeled extensively. However, no much work has been done from the view point of dynamic resource allocation. We evaluate this relationship for the 3-tier application when the CPU shares of the VMs are dynamically tuned through the utilization controllers.

A. *Performance of the application under control*

In our experiments, the applications were driven by three types of workloads: open, closed with MPL=10, and closed with MPL=100. We set the mean "think times" in the three cases to 35ms, 350ms, 3500ms respectively so that their average sending rates are all approximately 28 req/sec when the response times of the requests are much less than the think times. Utilization targets of the VMs varied among experiments, but in each experiment, they were kept unchanged and maintained by the utilization controllers.

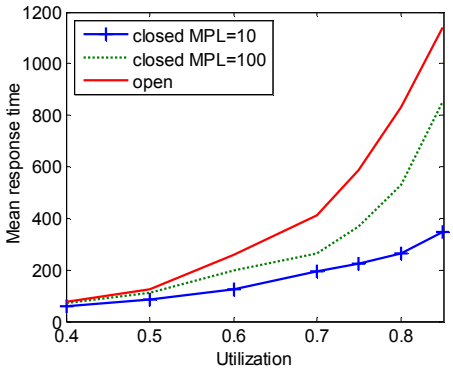
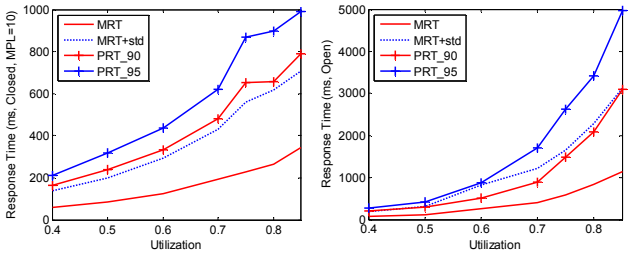


Figure 2: Relationship between utilization and MRT

Figure 2 shows the mean response times and the utilization levels when the utilization targets varied from 0.4 to 0.85, and the application was driven by the three workloads. Note that the utilization varies in our experiments due to changes of CPU allocation, instead of workload intensity as in tradition work. However, the mean response time in general still increases along with the utilization, due to reduced CPU capacity available and larger service times. Further study can be found in [4]. Between closed and open cases, the gains from utilization to MRT are much lower in closed cases. This difference is mainly due to the "self-tuning" capability of the closed workload generator, which tunes the sending rates of the requests

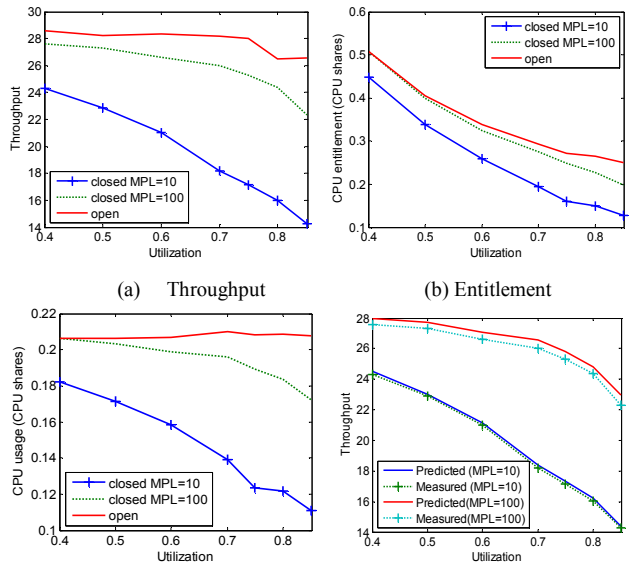
corresponding to the congestion conditions in the servers along the path. It slows down when the end-to-end response times increase and vice versa. Between the two closed cases, the case with MPL=100 is closer to the open case. It is because when MPL goes to infinity, the workload generator tends to behave as an open system.



(a) Closed (MPL=10) (b) Open

Figure 3: MRT and percentile response time (PRT)

Figure 3 provides more insights on the distribution of the end-to-end response times. It shows not only the MRT, MRT+std, but also the 90-percentile and 95-percentile response times as functions of the utilization levels for the closed (MPL=10) and open systems. As we can see, for both systems, there exists significant difference between the MRT and percentile response times. Further study on the per-request and per-session response times shows that, the difference can be partially due to that of the resource demands among the requests, a general property of Internet applications.



(a) Throughput (b) Entitlement

(c) Consumption (d) Predicted vs. Measured

Figure 4: Throughput and Utilization

Response time is the main performance that the end users can experience directly for the services. As the owners of applications or services, more metrics have to be considered, for instance, throughput, and resource consumption. Figure 4 compares the three systems in terms of those metrics, again, as functions of utilization levels. As shown in Fig. 4 (a-c), in the open case, the throughputs are always around 28req/sec in all the experiments since the sending rates are the same. In the closed cases, the throughput decreases along with increase of utilization. It drops much faster in the case with MPL=10, and much less CPU is entitled or consumed. This is again due to the "self-tuning" capability of the workload generators. Actually, from the principle of the closed systems, the relation between the throughput and MRT can be approximated as follows,

$$Throughput = \frac{MPL}{MRT + ThinkTime}. \quad (1)$$

Figure 4(d) compares the measurement of throughput with those predicted based on MRT using the model. As we can see, the model can predict the relation with trivial errors.

B. *Implication for performance control*

The observation of the application performance as function of utilization levels provides us insights for definition of performance management problems.

First, application performance cannot be guaranteed at all if only utilization is under control. As we can see from Fig. 2, for the same utilization settings, e.g., 80%, the mean response times perceived by the users can be very different, depending on the models followed by the application/workload generator.

Second, the mean response time may not be controllable for the open system when the utilization can be pushed too high. As we can see from Fig. 2 again, the gain from utilization to mean response time increases exponentially along with the utilization. For high response time target, the actual response time can be very sensitive w.r.t. changes of the utilization, or the resource allocations. On the other hand, the mean response time for the closed systems can be much more controllable due to the smooth relationship between the metric and resource allocation.

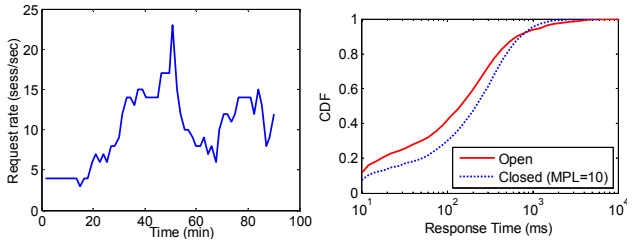
Third, the same MRT settings can result in very different response time distribution with different workload generators. As in Fig. 3, for the open one, the variance of the response times can be very different with different MRT settings. When the utilization is pushed too high, the variance can be much larger than the MRT. Large variance also exists with the closed systems, although in general it is much smaller than that in the open case. Between the two types of workload generator, we can see that, the performance that can be perceived by the end users can be very different even with the same MRT settings.

Fourth, the same mean response time settings can result in very different throughputs, depending again on the models the applications follow, as seen in Fig. 4. In the closed cases, higher response time settings can result in lower throughputs. For example, when MPL=10, for an MRT setting of 340ms, the throughput is around 14 req/sec, compared with 25 req/sec for a response time setting of 60ms.

IV. Case Study: Performance Guarantee through Dynamic Resource Allocation

For validation purpose, we run experiments with the MRT actually under control of the integral controller as shown in Fig. 1. Results are found to be consistent with what we argued in

previous sections. Due to space limit, Figure 5(a) shows the trace used in one of our many experiments, which mimics the demand of the 1998 World Cup workload [9]. The MRT was controlled through the nested controller, while the application was driven by the closed (MPL=10) or the open workloads. The target was set to MRT_{ref}=350ms. Each experiment runs 90 minutes, and statistics were collected after the first 30 minutes. Figure 5(b) and Table 1 compare the distribution of the response times of individual requests, their mean and percentiles, throughput in sess/sec, and also mean CPU resource utilization, entitlement and consumption between the closed and open systems. Even though the MRT was maintained at the same target level in both cases, the percentile distributions were significantly different. Less amount of CPU resource was utilized in the closed case. However, the throughput was 30% less than that in the open case.



(a) Traces (b) CDF of the response times

Figure 5 Dynamic performance control.

Table 1. Statistics of the experiments

	Response Times (ms)				Thr	CPU Resource		
	mean	50-p	90p	95-p		Util	Ent	Con
open	349	167	714	1337	25	0.66	0.33	0.21
closed	343	244	750	1000	17	0.78	0.17	0.13

V. Discussions and Conclusion

Based on the observation from the modeling and controlled experiments, we may claim a couple of principles on feedback controllers design and evaluation for dynamic resource allocation and performance guarantee.

First, the end-to-end performance management problem may not be always formulated as a tracking problem, even for time-varying workloads. In the closed case, there is a tradeoff between throughput and response time as approximated by the model (1). Pushing the MRT to a (high) threshold can compromise the throughput significantly. One possibility is to define the problem as a utility maximization problem to address the tradeoff between the cost and profit, where the utility may be represented as a function of both response time and throughput. However, as shown in Fig. 6, the model from MRT to throughput can be dependent on many factors such as MPL, and also the systems themselves.

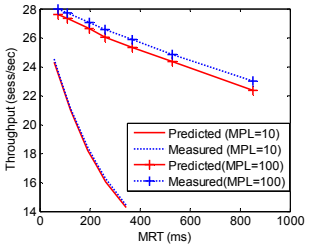


Figure 6: Throughput and MRT for closed system

It would be interesting to solve the optimization problem through online identification of the models or their parameters behind.

Second, the metrics and the target of the metrics for performance management have to be carefully chosen. Mean response time may not be good enough. More metrics, for instance, throughput, and percentile response time may be used that can better characterize the performance the user will experience.

Third, the systems under control may not be totally taken as "black-boxes". The behavior of the system under control can be very different along with factors like the workload generators and the operation region. Careful consideration has to be taken to configure the controller parameters such as the gains. When to evaluate the performance of controllers (not the application under control), different benchmarks, e.g., open, closed, or semi-open ones, have to be applied before conclusion can be made on the performance of the controllers.

In summary, the experimental evaluation and analysis provide us insights and guidance on problem formulation controller design, configuration and evaluation for end-to-end performance management through dynamic resource allocation. As on-going work, we are studying systems with larger scale, and pursuing alternative solutions to address the issues proposed in this paper.

References

[1] X. Zhu, Z. Wang and S. Singhal, "Utility-driven workload management using nested control design". *Proc. of American Control Conference (ACC)*, June 2006.

[2] P. Padala, K. Hou, X. Zhu, M. Uysal, Z. Wang, S. Singhal, A. Merchant, K. Shin, "Automated Control of Multiple Virtualized Resources", *Proc. of the Euro. Conf. on Comp. Sys.*, Mar 2009.

[3] P. Padala, K. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, A. Merchant and K. Salem, "Adaptive control of virtualized resources in utility computing environments", *Proc. of the Euro. Conf. on Comp. Sys.*, 2007, Lisbon, Portugal, pp. 289-302.

[4] Z. Wang, Y. Chen, D. Gmach, S. Singhal, B. Watson, W. Rivera, X. Zhu, and C. Hyser, "AppRAISE: Application-level Performance Management in Virtualized Server Environment", *IEEE Trans. on Networking and Service Management*(to appear)

[5] J. Hellerstein, Y. Diao, S. Parekh, and D. Tilbury, *Feedback Control of Computing Systems*, Wiley-IEEE Press, 2004.

[6] Y. Diao, N. Gandhi, J. L. Hellerstein, S. Parekh, and D. M. Tilbury, "Using MIMO feedback control to enforce policies for interrelated metrics with application to the Apache Web server," *Proc. of Net. Oper. and Mgmt. Symp.*, Florence, Italy, April 2002.

[7] http://rubis.ow2.org/

[8] B. Schroeder, A. Wierman, and M. Harchol-Balter, "Open versus closed: a cautionary tale", *Proc. of 3rd Conf. on Networked Systems Design & Implementation*, San Jose, CA, 2006.

[9] M. Arlitt and T. Jin, "Workload Characterization of the 1998 World Cup. Web Site", *HP Tech. Rep.*, HPL-99-35. (1999)