

# AutoParam: Automated Control of Application-Level Performance in Virtualized Server Environments

Zhikui Wang, Xue Liu, Alex Zhang, Christopher Stewart, Xiaoyun Zhu, Terence Kelly, Sharad Singhal  
Hewlett Packard Laboratories  
Palo Alto, CA

*Abstract*— Configuring virtual machine parameters in a virtualized server environment is a challenging task for IT operators. For instance, while current management products can dynamically size virtual machines in order to maintain resource utilization targets, default target values are rarely ideal for individual workloads and operators find it difficult to decide appropriate values for the targets. This paper presents *AutoParam*, a tool that provides application-level performance guarantees by automatically determining system-level parameters such as the utilization targets and the sizes of the virtual servers hosting individual tiers of multi-tier applications. *AutoParam* is based on synthesis of a feed-forward transaction-mix-based queueing model and feedback control loops. We describe the integration of *AutoParam* with the Xen virtualization system, and present empirical results showing that *AutoParam* effectively adjusts sizes of Xen virtual machine containers to maintain mean transaction response time at a desired level in spite of variable workloads. We compared *AutoParam* to four other designs, and show that it provides more reasonable tradeoffs between application performance and resource efficiency as well as more robust dynamic properties.

## I. INTRODUCTION

In contrast to the decentralized computing trends of the 1980s and 1990s, recent years have witnessed a resurgence of commercial interest in "re-centralized computing," e.g., server and data-center consolidation, shared hosting centers, and "utility computing." An important enabling technology underlying these trends is virtualization. As a result, many enterprises have become interested in virtualized server environments due to potential cost savings from consolidation. To be useful for shared-hosting and utility-computing, many virtualization technologies provide security, fault isolation, and performance isolation between co-hosted applications in different resource containers sharing the same physical server. In particular, performance isolation is central to the value proposition of virtualization and consolidation and is a rapidly evolving feature of various virtualization technologies [1][2].

The workloads of Internet servers and enterprise applications fluctuate considerably, and statically-partitioned resources suffer the same fate as dedicated resources: they are either over-provisioned or overloaded. Thus modern virtualization technologies expose interfaces to allow dynamic resizing of resource containers. But in practice it is difficult to configure and adjust the relevant parameters properly for a number of reasons. First, aggregate workload intensity (transaction arrival rate) may vary in regular patterns, e.g., diurnal cycles, and sometimes exhibits irregular variations, e.g., "flash crowds." Second, the mixture of transactions present in the workload may be highly variable at both long and short time

scales [3]. Therefore, it is challenging to properly tune the container size (referred to as "resource entitlement") parameters exposed by virtualization technologies in response to workload changes.

At the same time, existing workload management tools such as HP's Global Workload Manager [4] can dynamically size virtual containers in order to maintain user-specified resource utilization targets. While it is possible to benchmark applications to obtain default values for the utilization targets, these targets become sub-optimal in production environments as application workloads change or as new transaction types are added to the application. In addition, management goals are naturally stated in terms of application-level performance, e.g., transaction response time. The relationship between an application's system resource utilization and its application-level performance is complex even in the absence of virtualization. Virtualization makes performance modeling fundamentally harder.

This paper describes *AutoParam*, a generic resource allocation scheme for virtualized server environments. The design involves a synthesis of feed-forward and feedback control loops. A performance model, combining transaction mix models [5] with queueing models [6], is applied to determine the appropriate utilization levels of resource containers for a given workload in order to achieve desired application-level performance, stated in terms of transaction response times.

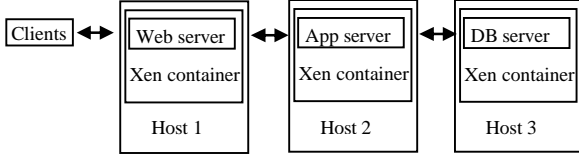
Feedback control theory has been applied in computing systems [7], e.g., in the context of web servers [8][9][10][11][12], caching services [13] and multi-tier applications [14]. Queueing models were applied to model or predict performance levels or resource demands in [14][15][16]. However, none of these specifically dealt with virtualized environments. In [17], we designed adaptive controllers to manage both resource utilization and service level objectives for a single-tier application on a system with resource partitions.

In comparison, *AutoParam* is designed for complex multi-tier applications in a variety of virtualized server environments. In our case study, we consider a three-tier application running in a Xen-virtualized environment [1] subjected to realistic workloads. Our results demonstrate that *AutoParam* enables automated control of application-level performance and resource allocation in virtualized server environments, provides reasonable tradeoffs between performance and resource efficiency as well as robust dynamic properties.

## II. VIRTUALIZED SERVER ENVIRONMENTS

### A. Overall testbed setup

We consider a virtualized server environment for running common three-tier applications in a data center, including the Web server, the application server and the database server.



**Figure 1: A virtualized server testbed hosting a 3-tier application**

Figure 1 shows our virtualized server testbed. It consists of four HP Proliant servers, three of which are used to host a three-tier application. Each tier of the application is hosted in a virtual container on a separate server. We use a Xen-enabled 2.6 Linux kernel in a stock Fedora 4 distribution. The fourth host is used to generate client requests to the three-tier application.

### B. Application and workload generator

We use a *modified* version of the Rice University Bidding System (RUBiS) [18] as our test application. It is an online auction benchmark with 22 transaction types providing services such as browsing for items, placing a bid, and viewing user information. In the testbed, the application runs on top of an Apache 2.0.55 Web server, a JBoss 4.0.2 application server and a MySQL database server.

We developed a workload generator [3] to simulate the real-world workload other than using the default RUBiS generator. The workload generator bundled with RUBiS is unsuitable for our evaluation because it only produces a stationary workload, in the sense that the relative frequencies of the different transaction types remain constant over time, whereas real-world workloads are highly non-stationary in terms of transaction mix [3]. Our workload generator has two features that support much more realistic evaluations than are possible with typical benchmark-like generators. First, our generator allows us to replay renamed traces of transactions collected on real production systems; this means that our test workloads contain the same non-stationary behavior of transaction-mix found in production workloads. Second, our generator allows us to mimic periodicity in real workloads (e.g., diurnal cycles) by varying transaction arrival rates between specified maximum and minimum levels.

The workload that we use for our evaluations is derived from a trace of transactions in the "VDR" application, a globally-distributed business-critical enterprise application (see [5][6] for a detailed description). We renamed the VDR transactions to RUBiS transactions in the following way: First we separately ranked VDR and RUBiS transactions according to their popularity; for the latter, we used the popularity of transactions in the workload generated by the default RUBiS generator. Then we simply replaced each transaction in the

VDR trace with the RUBiS transaction with the same popularity rank to obtain our final renamed trace.

### C. Sensors and actuators

In the virtualized server environment, we define *resource utilization* as the ratio of the resource consumption to the resource entitlement. In this paper, CPU capacity is the only resource considered. Xen hypervisor exposes an interface to access the counters that accumulate the CPU time (or cycles) consumed by each domain. The counters are sampled at fixed intervals, effectively yielding a sensor of the CPU consumption. Information on the completed transactions such as URL and response time is logged on the client side

Xen also exposes interfaces in Domain 0 that allow runtime adjustment of scheduler parameters such as the CPU share for each domain. In our experiments, we use the Credit Scheduler as the actuator for CPU entitlement, operated in the capped mode, which means that a domain cannot use more than its share of the total CPU time, even if there are idle CPU cycles. This non-work-conserving mode provides a straightforward guarantee on the CPU time available to the domains and provides performance isolation among the applications hosted by the domains. We expect that this mode will be more common in hosting centers and utility data centers that charge users for resources.

## III. Design of AutoParam

### A. Virtualized Server Queueing Model

A transaction-mix-based queuing model was developed in [5][6] for a multi-tier enterprise application running in non-virtualized environments. The application is modeled as an open queuing network with multiple service stations. Each station represents one tier within the application, running on a physical server, modeled as an M/GI/1 Processor Sharing queue (M/GI/1/PS). Assume that there are  $N$  transaction types, the numbers of which are sampled in fixed-width time intervals. Then for each interval  $i$ , the mean response time is

$$MRT_i = \frac{\sum_j \alpha_j N_{ij}}{\sum_j N_{ij}} + \frac{T}{\sum_j N_{ij}} \left( \sum_t \frac{U_{it}^2}{1 - U_{it}} \right), \quad (1)$$

where  $N_{ij}$  is the number of transactions of type  $j$  in interval  $i$ .  $U_{it}$  is the CPU utilization of tier  $t$ ,  $T$  is the sampling interval length, and  $\alpha_j$  represents the sum of service times for transaction type  $j$  across all tiers plus fixed delays such as network switching and transmission delays. It was shown in [6] that the model (1) can achieve substantially higher accuracy in predicting response times for real-world workloads than the queuing models that only employ a scalar measurement of the workload intensity.

We extended the model to the virtualized environment, where each tier of the application is hosted in one single virtual container, and each container receives a capped resource entitlement. The utilization of each container is defined as the ratio of average CPU consumption and CPU entitlement. Experimental results show that this extension is reasonable.

## B. Design of *AutoParam*

Figure 2 shows the overall architecture of *AutoParam* for a typical 3-tier application. It consists of three modules:

- A *utilization controller* for each virtual container. As the resource demand for the application changes, the controller adjusts the resource entitlement ( $E_t$ ,  $t=1,2,3$ ) to the container and maintains the ratios of the consumption ( $C_t$ ,  $t=1,2,3$ ) to the entitlement of each tier at the utilization targets ( $U_t^{ref}$ ,  $t=1,2,3$ ).
- A *feed-forward controller* for the application. Using the transaction-mix-based queuing model, the controller takes as inputs the response time target ( $RT^{ref}$ ) and the transaction mix ( $N$ ) estimated from the transaction history, and computes the optimal utilization levels ( $U^{mdl}$ ) so that the mean response time can track its target.
- A *feedback controller* for the application. Working together with the feed-forward controller, the feedback controller tunes the utilization targets ( $U$ ) of the individual containers driven by the error between the mean response time target ( $RT^{ref}$ ) and the measured *MRT*.

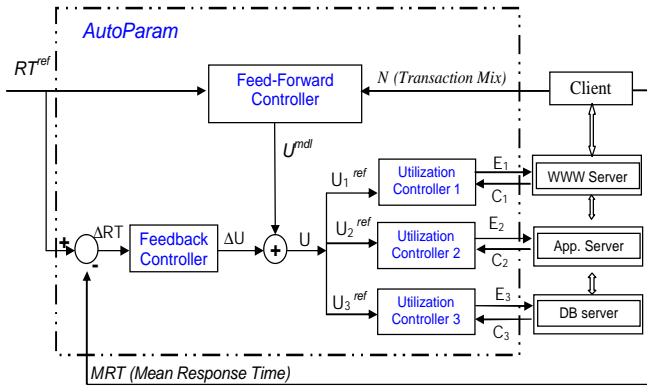


Figure 2 Architecture of *AutoParam*

Design of *AutoParam* is to integrate reactive and proactive control techniques to provide performance guarantee and high resource efficiency. More specially, four principles are used to design the overall controller.

1) *Resource utilization is under control in the innermost layer, but the targets need to be decided automatically.*

Existing workload management products, e.g., [4], implement utilization controllers for dynamic sizing of virtual containers. These controllers use local metrics to maintain system utilization at some utilization target by adjusting resource entitlement based on resource assumption. Most data center operators use the default utilization target, for instance, 75%, on their workload management systems. However, the default values are rarely ideal for individual operational environments when application-level performance guarantees are desirable, and it is usually challenging for the operators to find an appropriate value for a given application. This is the main motivation for *AutoParam*, a tool that can automatically determine the appropriate utilization targets for individual environments.

In this paper, the adaptive controller developed in [17] is applied that implements the following control logic, in the time interval  $k$  for tier  $t$ :

$$E_t(k) = E_t(k-1) + G_t(k)(U_t^{ref} - U_t(k-1)), \quad (2)$$

where

$$G_t(k) = -\beta C_t(k-1)/U_t^{ref}, \quad 0 < \beta < 2.$$

Note that  $U_t(k-1)$  and  $C_t(k-1)$  are the measured CPU utilization and consumption in the last interval.

The gain  $G_t$  is self-tuning based on the measured CPU usage in the last interval. This design was driven by the bimodal behavior of the system. The controller aggressively allocates more CPU when the system is suddenly overloaded so that the application performance does not suffer too much. On the other hand, the controller slowly decreases CPU allocation when the consumption is suddenly drops far below the entitlement so that there is more capacity buffer in case the demand increases again as observed for busy workloads. Experimental results in [17] also showed that this adaptive controller resulted in lower response time and higher throughput with less average CPU allocation compared to a fixed-gain integral controller.

2) *Target resource utilization is estimated based on the performance model to meet the application-level SLO.*

Given the time parameter  $\alpha_j$  of each transaction type  $j$ , transaction mix  $N_{ij}$  in the interval  $i$ , and mean response time target  $RT^{ref}$ , the expected utilization target of each tier,  $U_t^{ref}$ , can be computed based on the model (1) such that the following equation holds:

$$RT^{ref} = \frac{\sum_j \alpha_j N_{ij}}{\sum_j N_{ij}} + \frac{T}{\sum_j N_{ij}} \left( \sum_t \frac{U_t^{ref 2}}{1 - U_t^{ref}} \right) \quad (3).$$

In practice, application-level metrics including response times and transaction mix vectors can be extracted from application logs or a front-end proxy, and system-level metrics such as CPU consumption are readily available in almost all operating systems. Note that Equation (1) represents a linear regression equation in the unknown coefficients  $\alpha_j$ . As argued in [6], these parameters can be trained offline based on data extracted from transaction logs in real production environments.

In real-time control, the transaction mix in the current interval is unknown before control is actuated. In *AutoParam*, these values are estimated based on history data of the application. Different approaches are available. The simplest way, as used in our experiments, is to use the transaction mix in the last interval as an estimate for that of the current interval. This is reasonable if the transaction mix changes slowly compared to the time scale of the feed-forward controller, which is not unusual in real production sites.

Equation (3) is under-constrained, and does not have a unique solution for  $U_t^{ref}$ . Thus, it is possible to find many different utilization values that can meet the response time target for a

given transaction mix. Additional criteria are needed to identify an optimal solution. For instance, with certain utility functions, the resource utilization of the multiple tiers can be optimized such that the cost of the resources is minimized. In our experiments, we make the utilization targets for the three tiers to be the same to simplify the computation.

3) *Feedback control is applied to compensate the error in the model and respond to shorter time-scale changes.*

A standard integral controller is used in the feedback loop to compensate for prediction errors of the feed-forward controller due to inaccuracy in the model and possible sharp changes of the workload mix. For control interval  $m$ , the feedback controller determines an adjustment  $\Delta U$  for the utilization targets based on the observed error in the mean response time:

$$\Delta U(m) = \Delta U(m-1) + G^{FB} (RT^{ref} - RT(m-1)) / RT^{ref} \quad (4)$$

The gain  $G^{FB}$  needs to be chosen for this controller. Since the utilization target is in the range of  $[0, 1]$ , the error term is normalized by the response time targets so that the closed-loop system could be stabilized within a larger operation regions for a certain gain value.

4) *The controllers work in different time scales.*

Each controller works around its own sampling interval. The sampling intervals for the utilization, feed-forward and feedback controllers are denoted as  $T_{Util}$ ,  $T_{FB}$  and  $T_{FF}$  respectively. To ensure convergence, the rule of thumb is to set  $T_{Util} \ll T_{FB} < T_{FF}$ .

#### IV. EXPERIMENTAL EVALUATION

Due to space limitation, we describe partial empirical results on evaluation of *AutoParam* and provide simple analysis.

##### A. Workload and configuration

We use the customized workload generator, driven by the VDR-based trace, as described in Section II. Client transactions are submitted to the RUBiS application as a Poisson process. Figure 3 shows the rates of the top 3 transactions in each 60-second interval as a function of time for a workload intensity of 25 transactions per second. Together these three types of transactions account for 62% of all the transactions. Each type of transaction has a different resource demand. Even for constant workload intensity the changes we observe in the transaction mix could lead to a varying demand on system resources and performance.

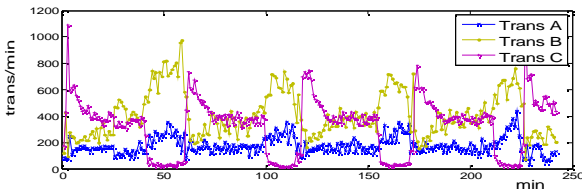


Figure 3: Mix of the top 3 transactions as a function of time

In our implementation, the utilization controller is located in Dom-0 for each of the three tiers, as it usually is in real products. The feed-forward and feedback controllers are both located at the client side, where the transaction mix and

response times are sampled from the logs of the clients. The utilization targets are sent to the three containers through SSH calls. The sampling intervals for the utilization, feed-forward and feedback controllers,  $T_{Util}$ ,  $T_{FB}$ ,  $T_{FF}$ , are set to 10, 30 and 90 seconds respectively. The parameter  $\beta$  of the utilization controller is set to 1.0. Unless specified otherwise, the integrator gain  $G^{FB}$  of the feedback controller is set to 0.05.

##### B. Model validation and performance prediction

The transaction-mix-based queuing model (1) was validated in [6] in the physical server environment through extensive experiments using a standard “training and testing” process. We performed similar experiments in our virtualized server environment. Our results show that the model can still describe the relationship among the transaction mix, the mean response time (MRT) and the resource utilization.

In the control experiments, the model was used to estimate the appropriate utilization targets as described in equation (3). Figure 4 shows both the measured MRT and the model-predicted MRT as functions of time. Fixed capacity was allocated and requests were sent out at a constant rate with a varying transaction mix. The MRT is estimated based on the predicted transaction mix. Define the normalized aggregate error as  $\sum_i |\hat{y}_i - y_i| / \sum_i |\hat{y}_i|$ , where  $\hat{y}_i$  and  $y_i$  are the estimated and measured MRTs respectively in interval  $i$ . Then the model has a 4.25% of error, as compared with 2.57% in the case with perfect knowledge of the transaction mix and the utilization. This reflects conditions in our control experiments.

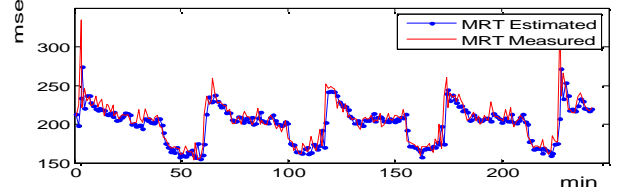


Figure 4 . Prediction results using estimated transaction mix

##### C. Performance evaluation

Extensive experiments were done to evaluate the performance of *AutoParam*. For comparison, we also ran experiments with four other ways of controlling container size:

- Fixed entitlement, or abbreviated as ‘Fixed Ent’. Each virtual container was entitled 0.9 CPU in this case.
- Fixed utilization control, or ‘Fixed Util’. This option is similar to the functionality offered by some commercial workload management tools. We implemented an adaptive utilization controller using Equation (2) for this design.
- Feed-forward plus utilization control, or ‘FF+Util’. No feedback compensation was applied for the response time control. The feed-forward control is expected to provide proactive and fast actions.
- Nested control, i.e., feedback performance control plus utilization control, or ‘FB + Util’. Compared with *AutoParam*, this design does not include the feed-forward part. For simplicity, we used the standard integrator as described in equation (4) for the feedback controller, except that the output of the controller is the utilization target itself instead of  $\Delta U$ .

Table 2 shows the performance, i.e., the average of the per-interval MRT across all the intervals in each experiment running for 2 hours. The average CPU entitlement across the three containers is also shown in number of CPUs.

Table 2: Tradeoff of the control designs

Control designs (targets)	Mean of MRT (ms)	Total entitlement (# of CPUs)
Fixed Ent	204	2.7
Fixed Util (75%)	578	0.53
FF + Util (300ms)	329	0.66
FB + Util (300ms)	288	0.86
AutoParam (300ms)	320	0.70

What we care most is the tradeoff between performance and resource efficiency. For these five designs,

- It is not surprising that the least response time is achieved under control of ‘Fixed Ent’ since the maximum capacity is provisioned for the containers. However, it also requires the largest amount of resource.
- Under control of ‘Fixed Util’, 80% less CPU is entitled to the containers compared with the ‘Fixed Ent’ case, however, with the cost of the largest response time. Given the expected 300ms of mean response time target, the default utilization level, 75%, cannot meet the requirement.
- Compared with the first two, the other three achieve better tradeoffs between response time and resource entitlement, i.e., less error in performance tracking, and higher resource utilization. *AutoParam* is in the middle, and demonstrates a reasonable capacity allocation and achieves a reasonable mean response time.

To highlight the differences among the latter three designs, we show the time series of performance and control metrics from the three in Figure 5, Figure 6 and Figure 7, including the mean response times in each control interval in subfigures (a), the utilization targets in subfigures (b) provided by the feed-forward and/or feedback controllers for the innermost loop, the CPU entitlement and the actual consumption of the application server in subfigures (c).

- The ‘FB+Util’ controller results in the maximum CPU entitlement due to delay in the control reactions. As we can see from Figure 6, it takes a long time for the utilization target to converge to the equilibrium once it is pushed down. It may be due to the slow convergence rate of the integrator. Although larger gain could be applied, it can potentially result in bad performance and instability.

Table 3 shows the performance and CPU allocation in the case of ‘FB+Util’ when the gain is changed. With the same target but a larger gain of 0.1, the performance is maintained with smaller error, and less CPU is allocated. However, this gain may not work always. When the target is changed to 500ms, the controller could be too aggressive, resulting a response time of 469ms.

Compared with ‘FB+Util’ design, the ‘FF’ scheme in the other two designs helps the controllers to converge much faster by setting the operating point closer to its equilibrium,

as we can see in Figure 7(b). This means that the integral controller works around the equilibrium most of the time, resulting in a much larger stable operation region with a similar gain as in the ‘FB+Util’ case.

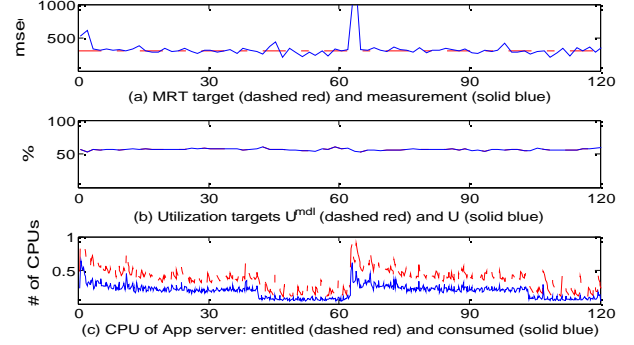


Figure 5. Tracking performance of FF + Util algorithm

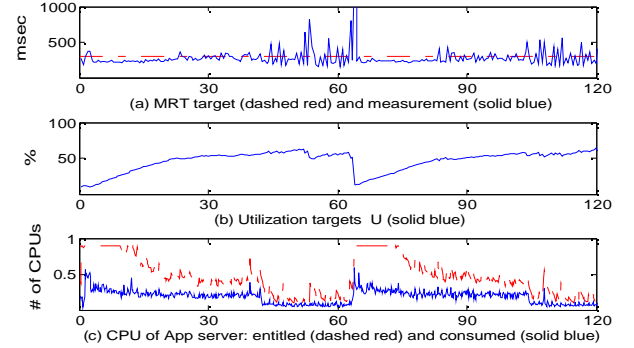


Figure 6. Tracking performance of FB+Util algorithm

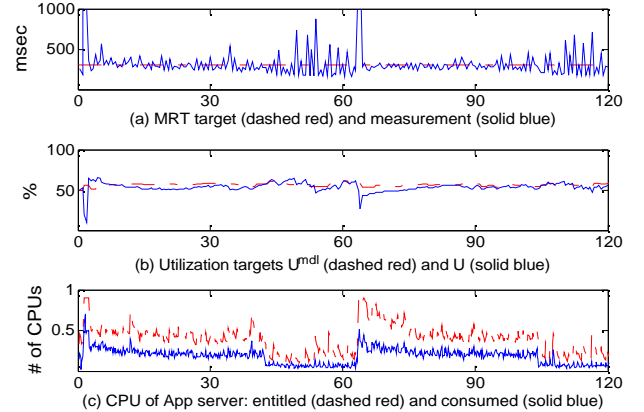


Figure 7. Tracking performance of FB + FF+Util algorithm

Table 3: Sensitivity of ‘FB+Util’ design to the value of the integrator gain parameter

Control designs (targets, gain)	Mean of MRT (ms)	Total entitlement (# of CPUs)
FB + Util (300ms, 0.05)	288	0.86
FB +Util (300ms, 0.1)	299	0.81
FB+Util (500ms, 0.1)	469	0.68

- Compared with ‘FF+Util’, *AutoParam* is more robust to modeling errors. Table 2 shows little difference between the

two designs. This is mainly because under that scenario, i.e., for the given workload, testbed setup, and performance target, the model can predict the utilization pretty well, as we can see from the traces in Figure 5 and Figure 7.

The benefit of feedback is obvious when the model is not accurate and when the feed-forward controller over- or under-estimate the utilization targets. Table 4 shows the comparison of the two cases, with a good or a biased model, under the same experiment setup. As we can see, the model bias results in larger error in performance tracking. Adding feedback controller instead can compensate the error in the model, and achieve comparable performance in either case.

We have done additional experiments with different workloads and parameter configurations and have obtained similar results.

**Table 4. Comparison of *AutoParam* and ‘FF+Util’ with accurate or inaccurate models**

Control designs (target, model)	Mean of MRT (ms)	Total entitlement (# of CPUs)
‘FF + Util’ (300ms, good model)	329	0.66
‘FF + Util’ (300ms, bad model)	357	0.67
<i>AutoParam</i> (300ms, good model)	320	0.70
<i>AutoParam</i> (300ms, bad model)	314	0.72

## V. CONCLUSIONS

Empirical results demonstrate that *AutoParam* can provide application-level performance guarantees by automatically determining system level parameters for the containers in a virtualized server environment. It integrates the benefits of both feedback and feed-forward control techniques, provides the managed application robust performance and reasonable tradeoff between performance and resource efficiency.

*AutoParam* is the first tool to introduce a transaction-mix-based queuing model for the determination of system-level resource utilizations in order to meet application-level performance objectives. We are continuing to study performance models for virtualized servers in further details and for different virtualization technologies. More evaluation is required for the performance of *AutoParam* in different scenarios including one with multiple applications. More analysis is required to choose appropriate values for the controller parameters. In addition, we would like to investigate online identification of the performance model, and automated configuration of the critical controller parameters such as the gains. Beyond control of only the CPU resource, we are also working on control of multiple resources including CPU, network I/O bandwidth and disk I/O bandwidth.

## REFERENCES

- [1] B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, I. Pratt, A. Warfield, P. Barham, and R. Neugebauer, "Xen and the art of virtualization," in *Proceedings of the 19<sup>th</sup> ACM Symposium on Operating Systems Principles (SOSP)*, pp. 164-177, 2003.
- [2] VMware, <http://www.vmware.com>
- [3] C. Stewart, T. Kelly, and A. Zhang, "Exploiting Nonstationarity for Performance Prediction," in *Eurosys 2007 (to appear)*, 2007.
- [4] HP Global Workload Manager (gWLM), <http://docs.hp.com/en/vsemgmt/gwlm.html>.
- [5] T. Kelly, "Detecting Performance Anomalies in Global Applications," in *Proceedings of the 2<sup>nd</sup> USENIX Workshop on Real, Large Distributed Systems (WORLDS)*, 2005.
- [6] T. Kelly and A. Zhang, "Predicting Performance in Distributed Enterprise Applications," *Technical Report HPL-2006-76*, HP Labs, 2006.
- [7] J. L. Hellerstein, Y. Diao, S. Parekh, and D. M. Tilbury, *Feedback Control of Computing Systems*, Wiley-IEEE Press, 2004.
- [8] T. Abdelzaher, K. G. Shin, and N. Bhatti, "Performance Guarantees for Web Server End-Systems: A Control-Theoretical Approach," in *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, pp. 80-96, 2002.
- [9] C. Lu, T. Abdelzaher, J. Stankovic, and S. Son, "A Feedback Control Approach for Guaranteeing Relative Delays in Web Servers," in *Proceeding of the IEEE Real-Time Technology and Applications Symposium*, 2001.
- [10] Y. Diao, N. Gandhi, J.L. Hellerstein, S. Parekh, and D.M. Tilbury, "MIMO control of an Apache Web server: Modeling and controller design," in *Proceedings of the American Control Conference (ACC)*, 2002.
- [11] Y. Lu, T. F. Abdelzaher, C. Lu, L. Sha, and X. Liu, "Feedback control with queueing-theoretic prediction for relative delay guarantees in Web servers," in *Proceedings of the 9<sup>th</sup> IEEE Real-Time/Embedded Technology and Applications Symposium*, May, 2003.
- [12] D. Henriksson, Y. Lu, T. Abdelzaher, "Improved prediction for Web server delay control," in *Proceedings of the 16<sup>th</sup> Euromicro Conference on Real-Time Systems*, Catania, Italy, July, 2004.
- [13] Y. Lu, C. Lu, T. Abdelzaher, and G. Tao, "An adaptive control framework for QoS guarantees and its application to differentiated caching services," in *Proceedings of the 10<sup>th</sup> IEEE International Workshop on Quality of Service (IWQoS)*, 2002.
- [14] B. Urgaonkar, G. Pacifici, P. J. Shenoy, M. Spreitzer, and A. N. Tantawi, "An analytical model for multi-tier internet services and its applications," in *Proceedings of the ACM SIGMETRICS Conference*, pp. 291-302, 2005.
- [15] X. Liu, J. Heo, and L. Sha, "Modeling 3-tiered Web Applications," in *Proceedings of the 13<sup>th</sup> IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, 2005.
- [16] L. Sha, X. Liu, Y. Lu and T. F. Abdelzaher, "Queueing model based network server performance control", in *Proceedings of the 23<sup>rd</sup> IEEE International Real-Time Systems Symposium*, 2002.
- [17] Z. Wang, X. Zhu, and S. Singhal, "Utilization and SLO-based control for dynamic sizing of resource partitions," in *Proceedings of the 16<sup>th</sup> IFIP/IEEE Distributed Systems: Operations and Management (DSOM)*, October, 2005.
- [18] C. Amza, E. Cecchet, A. Chanda, A. Cox, S. Elnikety, R. Gil, J. Marguerite, K. Rajamani, and W. Zwaenepoel, "Specification and Implementation of Dynamic Web Site Benchmarks," in *WWC-5 Austin, TX, USA*, 2002.