vATM: vSphere Adaptive Task Management

Appears in the seventh International Workshop on Feedback Computing (FC'12)

Zhichao Li‡, Aalap Desai†, Chirag Bhatt†, Rajit Kambo†, Erez Zadok‡

‡Stony Brook University, †VMware. Inc

ABSTRACT

Virtualization provides several benefits to users in the data-center in terms of infrastructure cost savings (e.g., capital, power, space, cooling, labor). Examples include highly efficient and available resource, networking, and storage management. As many workloads have moved to virtualized environments, it is critical that vSphere handles scale and performs optimally. However, there are scenarios where the vSphere platform exhibits suboptimal performance if user-level operations are scheduled poorly.

In this paper we propose a feedback-based approach to maximize the platform performance of vSphere with a gradient based hill climbing algorithm. We have implemented the gradient based hill climbing approach. Our initial results show promising performance improvements, in terms of end-to-end latency, under the vSphere environment.

Categories and Subject Descriptors

H.3.4 [**Information Storage and Retrieval**]: Systems and Software—*Performance evaluation (efficiency and effectiveness)*; I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search—*Scheduling*

General Terms

Management, Performance

Keywords

Task Scheduling, Feedback, Virtualization

1. INTRODUCTION

Virtualization [18] is massively deployed nowadays in the private cloud environment. As the private data-center scales up, management operations in the virtualized data-centers become pervasive, which places more pressure on the virtualization platform. Therefore, it is important to manage the workloads in the virtualization platform effectively [13]. However, naïve scheduling of those user-level operations can overload the virtual center and slow down the platform's productivity and overall performance.

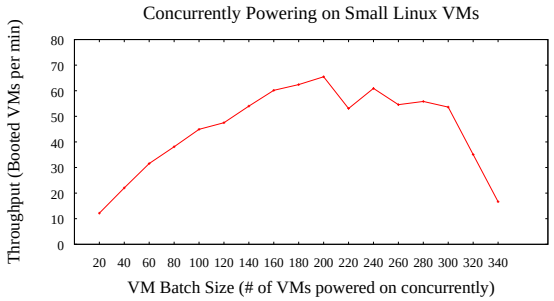


Figure 1: Tasks Throughput Experiment. *The throughput is measured by the batch size divided by the end-to-end batch completion time for each batch size.*

With the current VMware vSphere [19] infrastructure, we carried out a series of *boot-storm* experiments on the vSphere platform where different groups of virtual machines (VMs) were powered on to understand the impact of highly concurrent workloads on overall vSphere platform performance. We show the results in Figure 1. A boot storm occurs when many virtual machines start their operational workloads closer together, thus creating a larger load on the CPU, network, and storage. A boot storm scenario is likely to occur after systems have recovered from a power outage or when logging in for the first time.

Our results indicated that the throughput of the system hits a maxima after a certain batch size. Batch means a group of concurrent operations, and batch size is the size of the grouping. When increasing the batch size beyond a certain number, throughput for that batch size dropped, indicating that the vSphere platform encountered a bottleneck. The ideal batch size that maximized throughput in our experiments was powering on 200 VMs in this test scenario.

Currently, there is no way for the vSphere client to get any feedback from the vSphere platform if the system is overloaded or under loaded. vSphere client solutions like Site Recovery Manager [14], Virtual Desktop Infrastructure [17], and vCloud Director [16] could potentially overload the platform and slow down the performance. Static throttling schemes are often implemented to prevent such scenarios. However, this approach may observe limited performance when the platform is capable of supporting more tasks. What we are trying to solve could

.

potentially help increase the throughput of vSphere-related platforms. Furthermore, VMware customer feedback suggests that provisioning and deployment scenarios tend to be bursty. Issuing an optimal number of tasks on the platform, during such scenarios, would result in reduced recovery time, better end-to-end boot time, and a quicker provisioning process.

Our work aims to identify the throughput maxima for any given vSphere environment and control the number of tasks issued to the system, while adapting to changes in the environment. To solve the above problem, we implemented a prototype, call vATM (vSphere Adaptive Task Management), combining feedback mechanism and adaptive hill-climbing algorithm. Our initial results show promising performance improvement with the the proposed approach being adaptive to the system background load. We achieved up to 54% improvement in terms of end-to-end latency in the case of multiple tasks experiment.

Note that we refer to vSphere as the target system through out the paper, because our prototype implementation is based on vSphere. However, our approach is generic enough to be applicable to other components of the system such as vCloud Director and the Cloud Infrastructure Suite (CIS) [2].

2. RELATED WORK

Feedback-based approaches have been widely applied in various field of computer systems [5] and networks [15] to provide QoS (e.g., performance and power) guarantees in various levels.

A standard feedback-based approach is using control theory, which offers various methodologies to solve such problems that demand dynamic throttling and task management. It has been applied to database systems [4], storage systems [7, 8], Web servers [3,10], and data-centers [9,21–23] to provide QoS (e.g., performance and power) guarantees. Abdelzaher et al. surveyed the application of feedback control to software systems [1] as well. A proportional-integral-derivative (PID) controller [6, 11, 20, 23] is one such approach.

In this paper, we picked a simple feedback based hill climbing algorithm instead of exploring a PID controller which involves specifying a fixed set point variable that the system tries to converge upon based on the controller feedback and requires additional expertise and experiments. The vSphere system is constantly under flux because of the wide variety of operations being executed by varying amount of users, which essentially means that the system does not have a specific set point where it performs optimally, rather the set point varies dynamically based on the type of workload and number of users issuing tasks on the system. We also expect to achieve much more performance improvement once the current prototype takes into account the token-based normalization as we discussed in Section 6.

3. DESIGN AND IMPLEMENTATION

3.1 Overall Design

vSphere has a broad spectrum of tasks ranging from VM operations like clone and power on to host operations like add and remove a host. Each operation has an associated cost attribution in terms of operation latency. We introduce the notion of a work unit called a *token* to represent one unit of operation cost. If we use t_p to represent the notion of the maximum

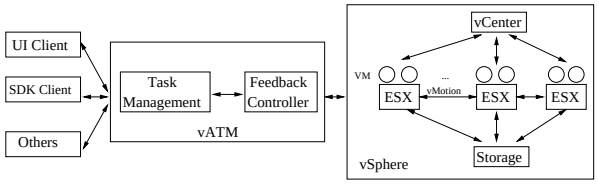


Figure 3: Architecture

token capacity of the vSphere platform installation, it means that the vSphere platform has t_p tokens to give. Currently, we assign each task one token to play around with. We will use this notation of work units and throughput when we describe the details of the feedback controller in Section 3.3.

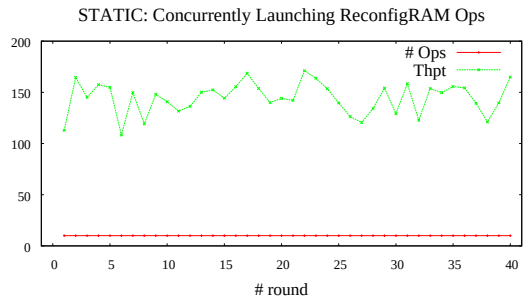
vATM (vSphere Adaptive Task Management) is our own invented new term to describe the prototype we built. It is resides between the clients and vSphere. We envision this to be either a stand-alone service or a component residing within the vSphere suite. The precondition is that all tasks are directed through this component. Our vATM comprises of two main components: Task Management and a Feedback Controller. The task management component is responsible for sending tasks to vSphere as well as collecting task information such as operation latency. The task-management part determines how many tasks to launch in each round. The feedback controller leverages this information in combination with system feedback to adaptively control the outstanding operations on the system. The architecture of the above discussion is shown in Figure 3.

We explain the two components mentioned above in more detail in subsection 3.2 and 3.3.

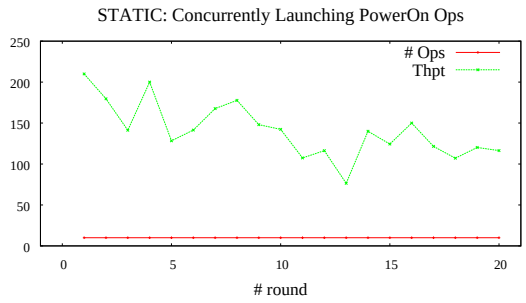
3.2 Task Management

All task requests reach vSphere through the task management component. If the task management component detects that the vSphere platform is overloaded, it buffers the request in an internal queue and reissue it when the platform is ready. In our prototype, the queue is large enough to hold all the requests. If, however, no overload condition is detected, then the task request passes through. This component relies on the feedback controller to obtain the overload condition.

Task management can be made more effective by targeting tasks towards a granular domain such as a data-center, cluster, or other grouping based-on the affinity of tasks. The advantage of going with a granular approach is that we prevent hot spots (e.g., a busy cluster) from unnecessarily throttling other parts of the inventory. This is especially a concern in the multi-user environment where the use cases and performance characteristics may vary across parts of the virtual center server inventory. For example, cluster A could be overloaded with several boot storms, and hence additional tasks to cluster A may need to be queued by the vATM to avoid cluster busy time. At the same time, there could be another idle cluster B where tasks do not need to be queued and therefore the performance feedback metrics from cluster A do not need to be applied to cluster B. In order to adopt this approach, vATM requires a mechanism to map each task to an inventory component. Such a mechanism of queuing tasks separately based on tasks ownership is one of the extensions of this paper that needs further study with additional experiments in future work.



(a) Throughput fluctuates due to background activities



(b) Optimal throughput varies as system is more loaded

Figure 2: Fluctuation and variation. *Observations with static throttling. A "round" is completed when batched concurrent tasks are finished. Throughput is measured for a single round.*

3.3 Feedback Controller

Once tasks start arriving, the task management component normalizes the workload via the token-based management system and asks the feedback controller for guidance as to whether to queue up the tasks or pass them through.

If the current queued task number t_{curr} is less than the concurrency target number t_{tar}, then the controller does not kick in and tasks are passed to vSphere. However, in the case that t_{curr} is greater than t_{tar}, the controller kicks in.

Initially, t_{tar} is set to t_{min}, the initial number of concurrent tasks that the system will handle to start processing tasks. t_{min} is a configurable value. In case that t_{curr} is greater than t_{tar}, t_{tar} is increased to be twice as much as t_{min}, in order to collect at least two data points about the task throughput to help identify the trend of the throughput hill. To avoid overshooting the system, t_{min} should not be too large. Further increase of t_{tar} is determined by the feedback controller according to the hill-climbing mechanism. t_{tar} is gradually increased till the system achieves an experimental maxima with capacity t_p.

During each round, the controller issues t_{tar} tasks concurrently and waits for them to complete. As tasks are completed in each round, the throughput is measured after all the tokens in the current round were received. The next batch size, represented by t_{tar+1}, is increased or decreased based on the following formula:

$$gradient = \frac{thpt_{tar} - thpt_{tar-1}}{t_{tar} - t_{tar-1}} \quad (3.1)$$

$$gradient_{norm} = arctan(gradient)/90 \quad (3.2)$$

$$t_{tar+1} = t_{tar} * (1 + gradient_{norm}) \quad (3.3)$$

where $thpt_{tar}$ is the overall throughput as of the current round, $thpt_{tar-1}$ is the overall throughput as of the previous round, t_{tar} is the number of concurrent tokens issued in the current round, t_{tar-1} is the concurrent tokens issued for the previous round. We divide by 90 in the formula to ensure the normalized gradient $gradient_{norm}$ is between -1 (inclusive) and 1 (exclusive).

If the $gradient$ calculated across t_{tar} and t_{tar-1} is positive, it suggests that there is additional headroom within the system and t_{tar+1} is increased in accordance. The larger the $gradient$ value is, the larger the batch size will be for the next round, and the quicker the local maxima will be found. If the gradient is negative, it means that the token throughput has dropped: vATM then decreases the tokens based on the gradient, to obtain a high overall throughput across rounds. We set a floor value for the system available tokens as well so that the vATM is able to launch at least one operation in the worst case.

The increase in batch size is reflective of the throughput trend of the system (e.g., the batch size will be increased more aggressively if the vATM detects a steep rise in throughput). Once the throughput curve starts flattening out, the batch size will be increased more conservatively. At any given instant, the vATM maintains a constant in-flow of t_{tar} tokens on the system until the feedback controller updates t_{tar}.

3.4 Observations

Two observations influenced our design and implementation. In the first experiment, we have in total 400 VM reconfiguration RAM operations. We launch 10 tasks per round statically and observe the throughput of each single round. In the second experiment, we have in total 200 VM power on operations. We issue 10 tasks per round in a static way and observe the throughput of each single round.

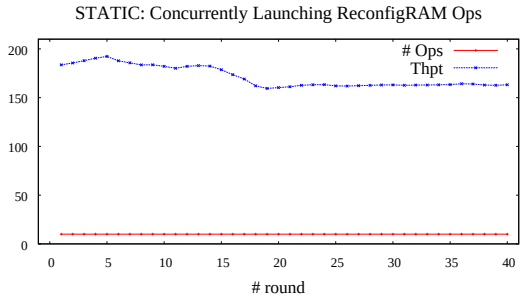
The first one is shown in Figure 2(a). We can see that even when the number of operations is fixed, the throughput of each round fluctuates. This is especially true for shorter rounds. That challenges the hill-climbing based approach a lot since it is sensitive to fluctuations. Therefore, our prototype should be able to minimize these fluctuations (i.e., smoothing) by utilizing the overall throughput across multiple rounds instead of the throughput obtained from a single round, as we will see in Section 4.

The second observation is shown in Figure 2(b). We can see that when the number of operations is fixed, the throughput for a single round decreases as more and more virtual machines are powered on, increasing the background load. It suggests that the maximum throughput for a single round is likely to vary depending on what the background load is. Therefore, our prototype should be able to launch the optimal number of tasks adapting to varying background loads.

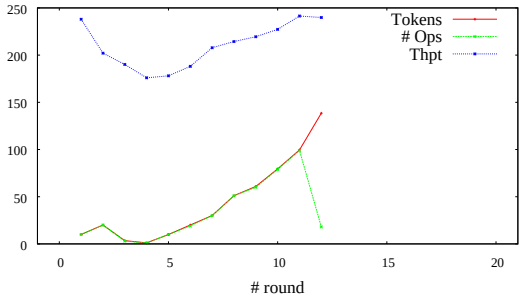
4. EVALUATION

4.1 System Setup

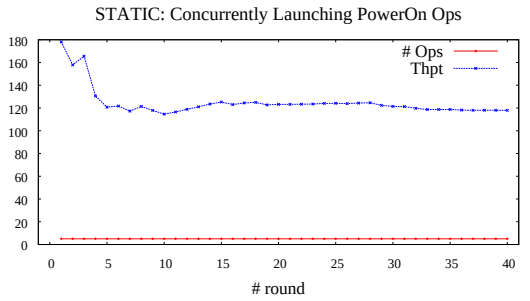
For experiments, we used a cluster consisting of three ESX hosts. Each host is a Dell PowerEdge R610 machine with dual quad-core Intel Xeon 3.0GHz processor and 16–49GB of RAM with 540GB local disk. We installed hundreds of VMs



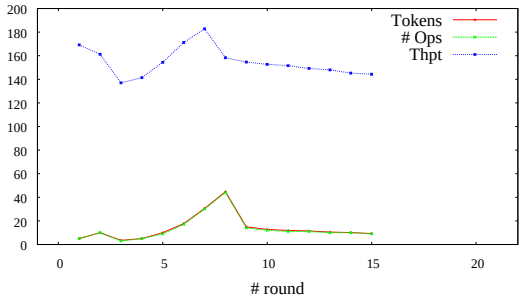
(a) Static throttling of reconfigRAM operations



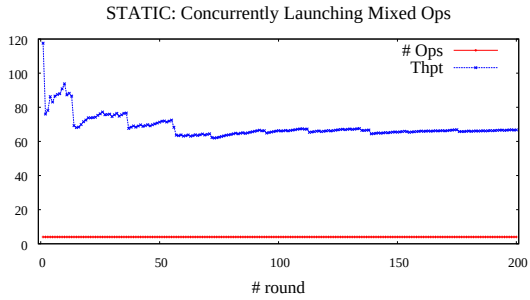
(b) vATM throttling of reconfigRAM operations



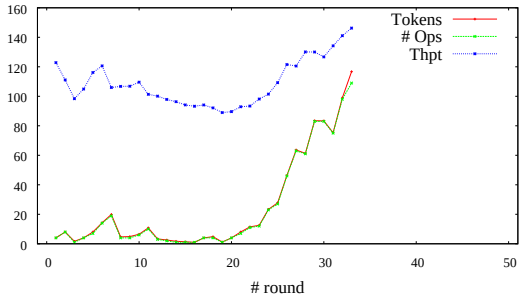
(c) Static throttling of powerOn operations



(d) vATM throttling of powerOn operations



(e) Static throttling of mixed operations



(f) vATM throttling of mixed operations

Figure 4: Two different approaches on operations throttling. *Throughput here is the overall throughput defined by total number of finished tasks divided by total latency. The unit is number of tasks per min. Tokens in the figures are the system available tokens. One operation represents one task. Since we assign one token to each operation, x tokens should trigger launching x operations as long as there are enough tasks left in one specific round.*

on each of the host. Each VM has one virtual CPU, 256MB RAM, and one 20GB virtual disk.

We implemented the vATM prototype with the gradient-based approach in Java using vSphere SDK [12] using about 4K LoC. We evaluated the system with different workloads such as boot-storm workload, reconfiguration workload, and workload with mixed operations. We carried out experiments via both the vATM and static throttling approaches.

4.2 Results

We now present the results we got with different approaches. We show how the vATM approach adapts to the overall throughput variations, while the static throttling approach does not.

As shown in Figure 4, we performed three groups of experiments. Figure 4(a) and Figure 4(b) represent the case of VM RAM reconfiguration operations, where there are in total 400 concurrent tasks. Figure 4(c) and Figure 4(d) represent the case of VM power-on operations, where there are in total 200 concurrent tasks. Figure 4(e) and Figure 4(f) represent the case of mixed VM operations with equal distribution among four operations: VM RAM reconfiguration, VM power on, VM creation, and VM snapshot creation operations. There are in total 800 concurrent tasks. The figures on left side describes the static throttling approach, whereas the figures on right side show the vATM approach.

In the case of VM RAM reconfiguration operations, the static throttling approach, as seen in Figure 4(a), launches 10 tasks in each round. For the vATM approach, as seen in Figure 4(b), starts launching 10 tasks in the first round, and 20 tasks in the second round as required by the gradient calculation, and then adapts to the overall throughput variations. In the static case, the overall throughput is relatively stable. However, it is not improving further, even when there is still a large gap for performance improvement. The vATM approach, on the

other hand, adapts to the overall throughput variations by first decreasing the batch size—as it observes overall throughput drops in the beginning several rounds—and then increasing the batch size as it observes overall throughput improvement in the last several rounds. In the last round, there are not enough tasks left; therefore, the number of operations is not the same as the value of tokens.

In the case of VM power on operations, the static throttling approach, as seen in Figure 4(c), launches five tasks in each round. For the vATM approach, as seen in Figure 4(d), starts launching five tasks in the first round, and ten tasks in the second round as required by the gradient calculation, and then reacts in an adaptive way. For the static case, the overall throughput drops in the first several rounds because of the fluctuations as we discussed in Section 3.4, and then becomes relatively stable. The vATM approach, on the other hand, decreases the batch size in the third round, and then increases the batch size in the following rounds until it hits a local maxima. Then, the system becomes overloaded since there are now more powered-on VMs, and the overall throughput drops as we discussed in Section 3.4. At this point, the vATM approach adapts to the changing background load by dynamically dropping the token value, and therefore, maintaining an overall high throughput.

In the case of mixed VM operations, the static throttling approach, as shown in Figure 4(e), launches four tasks in each round. For the vATM approach, as shown in Figure 4(f), starts launching four tasks in the first round, and eight tasks in the second round as required by the gradient calculation, and then launches in an adaptive way. With static throttling, the overall throughput fluctuates and degrades in the first several rounds because of the issues we discussed in Section 3.4, and then turns to be relatively stable. The vATM approach, instead, adapts to the local overall throughput changes, and hits two local maxima in the beginning several rounds, and then gradually increases the batch size to further improve the performance throughput in the last several rounds. In the last round, there are not enough tasks left; therefore, the number of operations is not the same as the value of tokens.

We also collected the end-to-end workload completion time for different approaches with the above three groups of experiments. We considered the optimal static throttling case, i.e., the best static threshold we could derive based on offline analysis of workload, and naïve static case, i.e., the threshold used for the static experiments described above. These comparisons are shown in Table 1.

Operation Type	Naive Static	Optimal Static	vATM
reconfigRAM	2.45	1.56	1.67
powerOn	1.70	1.35	1.39
mixed	11.99	4.67	5.47

Table 1: End-to-end latency comparison (minutes)

We can see from the table that the vATM approach, compared with the naïve static approach, improves the end-to-end latency of the VM RAM reconfiguration operations, VM power on operations, and VM mixed operations, by 32%, 18%, and 54%, respectively. It shows that the vATM approach is effective in adjusting to the vSphere overload and therefore improves overall throughput.

The vATM approach shows great performance improvement compared with the naïve static-throttling approach. When compared with the optimal static approach which usually requires manual analysis on the history raw data, vATM is close. The optimal static approach outperforms the vATM approach by 4% for VM RAM reconfiguration operations, 3% for VM power on operations, and 7% for VM mixed operations, respectively. The reason is that the vATM approach takes some time to find the local maxima as the it starts launching tasks conservatively. We believe that with a larger enough operation set, the time period the vATM takes to find the local maxima can be ignored, and the vATM approach will perform even better.

5. CONCLUSION

In this paper, we proposed one feedback based approach aiming to maximize the vSphere platform performance with gradient-based hill-climbing approach. We have implemented the gradient-based hill-climbing approach, and the results showed promising performance improvement under the vSphere environment. Compared with the naïve static approach, vATM improves the end-to-end latency of the VM RAM reconfiguration operations, VM power on operations, and VM mixed operations, by 32%, 18%, and 54%, respectively. Compared with the optimal static approach, vATM is close with 4% less for VM RAM reconfiguration operations, 3% less for VM power on operations, and 7% less for VM mixed operations, respectively.

6. FUTURE WORK

We plan to assign tasks with dynamic tokens based on what the background load is at specific time. We plan to integrate the token normalization approach, that normalizes each task token adaptively, into the vATM and collect some results with various task loads. We also plan to have a periodic intervention thread to update the system token in a more timely way in case there are any system metrics (e.g., system resource usage, individual operation latency, etc) that vSphere has to abide to. This will prevent the hill-climbing approach from grossly overshooting the throughput hill. Besides, to further improve system throughput, we plan to enable launching tasks in the middle of each round once vATM observes task completeness and has enough system tokens to assign to additional tasks. Moreover, we plan to scale the experimental setup to perform larger tests as well to understand the vATM behavior in the long run. Lastly, the current implementation considers task management at the entire vSphere level. However, going forward, we plan to use a more granular approach to manage tasks across both data-centers and clusters.

7. REFERENCES

[1] T. F. Abdelzaher, J. A. Stankovic, C. Lu, R. Zhang, and Y. Lu. Feedback performance control in software services. *IEEE Control Systems Magazine*, 23(3):74–90, 2003.

[2] VMware Cloud Infrastructure Suite. *http://www.vmware.com/cloud-infrastructure/business-critical-applications.html*.

[3] 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. In *Proceedings of the Network*

Operations and Management Symposium, pages 219–234, 2002.

[4] Y. Diao, J. L. Hellerstein, A. J. Storm, M. Surendra, S. Lightstone, S. Parekh, and C. Garcia-Arellano. Using MIMO linear control for load balancing in computing systems. In *2004 American Control Conferences*, 2004.

[5] P. C. Diniz and M. C. Rinard. Dynamic feedback: an effective technique for adaptive computing. *ACM SIGPLAN Notices*, pages 71 – 84, 1997.

[6] J. L. Hellerstein, Y. Diao, S. Parekh, and D. M. Tibury. *Feedback Control of Computing Systems*. Wiley-IEEE Press, 2004.

[7] M. Karlsson, C. Karamanolis, and X. Zhu. Triage: Performance differentiation for storage systems using adaptive control. *ACM Trans. Storage*, 1(4), 2005.

[8] H. D. Lee, Y. J. Nam, K. J. Jung, S. G. Jung, and C. Park. Regulating I/O performance of shared storage with a control theoretical approach. In *NASA/IEEE Conference on Mass Storage Systems and Technologies (MSST)*. IEEE Society Press, 2004.

[9] H. C. Lim, S. Babu, and J. S. Chase. Automated control for elastic storage. In *Proceeding of the 7th International Conference on Autonomic computing*, ICAC '10, pages 1–10. ACM, 2010.

[10] Y. Lu, A. Saxena, and T. F. Abdelzaher. Differentiated caching services; a control-theoretical approach. In *21st International Conference on Distributed Computing Systems*, pages 615–624, 2001.

[11] P. Padala, K. G. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, A. Merchant, and K. Salem. Adaptive control of virtualized resources in utility computing environments. In *Proceedings of the European Conference on Computer Systems*, pages 289–302, 2007.

[12] VMware vSphere Web Services SDK. *https://www.vmware.com/support/developer/vc-sdk/*.

[13] V. Soundararajan and J. M. Anderson. The impact of management operations on the virtualized datacenter. *Proceedings of the 37th Annual International Symposium on Computer Architecture (ISCA'10)*, pages 326 – 337, 2010.

[14] VMware vCenter Site Recovery Manager. *http://www.vmware.com/products/site-recovery-manager/overview.html*.

[15] A. S. Tanenbaum. *Computer Networks, 4th Edition*. Prentice Hall, 2002.

[16] VMware vCloud Director. *http://www.vmware.com/products/vcloud-director/overview.html*.

[17] VMware Virtual Desktop Infrastructure. *http://www.vmware.com/pdf/virtual_desktop_infrastructure_wp.pdf*.

[18] VMware, Inc. *Understanding Full Virtualization, Paravirtualization, and Hardware Assist*, 2007. *www.vmware.com/files/pdf/VMware_paravirtualization.pdf*.

[19] VMware vSphere. *http://en.wikipedia.org/wiki/VMware_vSphere*.

[20] Q.-G. Wang, Z. Ye, W.-J. Cai, and C.-C. Hang. *PID Control For Multivariable Processes*. Lecture Notes in Control and Information Sciences, Springer, March 2008.

[21] X. Wang, M. Chen, and X. Fu. MIMO power control for high-density servers in an enclosure. *IEEE Trans. Parallel Distrib. Syst.*, 21:1412–1426, 2010.

[22] X. Wang and Y. Wang. Coordinating power control and performance management for virtualized server clusters. *IEEE Trans. Parallel Distrib. Syst.*, 22:245–259, February 2011.

[23] Y. Wang, X. Wang, M. Chen, and X. Zhu. Power-efficient response time guarantees for virtualized enterprise servers. In *Proceedings of the 2008 Real-Time Systems Symposium*, pages 303–312. IEEE Computer Society, 2008.