PERFORMANCE ENHANCEMENT OF CPU SCHEDULING USING IMPROVED ALGORITHMS IN XEN: AN OPEN-SOURCE VIRTUALIZATION TOOLS[1]

Rajesh Kumar Pathak, Dr Setu Kumar Chaturvedi

Department of Computer Science, TIT Bhopal

ABSTRACT

Cloud Computing and Virtualization is one of the fore most technology which has attracted many researchers recently, which is directly going to benefit the end users and data center service providers. When we look things from server side resource usage, things become quite challenging. Just to meet peak loads, high capacity servers are deployed, which is remains underutilized most of the times on an average. Several scheduling algorithms have been in place, proposed and implemented on one or other Operating Systems from time to time. But, since we have limited processors and things work in concurrent fashion, overload situation can occur hampering the overall objective, performance and throughput of the system. One of the key commercial players in the virtualization is Xen, which is open-source free software. Today, Credit scheduler is the default scheduler in the Xen hypervisor. There are many scopes for improvement in the credit scheduler to make it suitable for cloud computing kind of platforms. In this paper, we have identified few mathematical models which simulates real life situation and suggest algorithm for Enhancement in Performance of CPU Scheduling in Xen.

XEN: AN OPEN-SOURCE VIRTUALIZATION TOOL

Today’s hardware is quite powerful to run multiple applications meeting real-time demands on peak loads quite efficiently. But most of the times this sufficiently powerful hardware and computing facility goes underutilized. This has led to resurgence in the virtualization and hosting multiple virtual machines.

1 XEN ARCHITECTURE

Xen is an x86 virtual machine monitor which allows multiple operating systems to share physical hardware in a safe and resource managed fashion, without sacrificing either functionality or speed and accuracy performance. This is achieved by providing an idealized virtual machine abstraction to which operating systems such as Linux, BSD and Windows XP, can be ported with minimal effort[4].

[1] How to cite the article:
Pathak K.R., Chaturvedi K.S., Performance Enhancement of CPU Scheduling Using Improved Algorithms in Xen: An Open-Source Virtualization Tools, *International Journal of Advances in Engineering Research*, April 2013, Vol 3, Issue 4, 1-14

Xen hypervisor can run operating systems in two modes:

a) Complete-Virtualization and,
b) Para Virtualization.

The initial design was targeted to host up to 100 virtual machine instances simultaneously on a modern server. The virtualization approach taken by Xen is extremely efficient. It allows operating systems such as Linux and Windows XP to be hosted simultaneously for a negligible performance overhead at most a few percent compared with the unvirtualized case.

Its design considers many things quite effectively and efficiently. Few of them are enlisted as follows:

- It supports facility to host different operating systems to meet the heterogeneity of applications.

- It enables administrators to move active virtual machine or guests from one server to another independent of various CPU virtualization support.

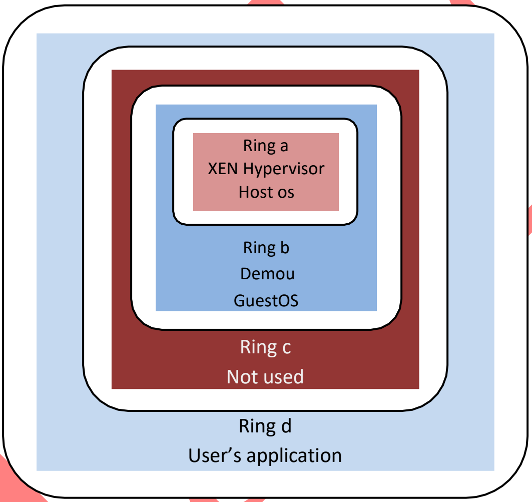

Figure 1: Xen Enabled Operating System Layers

- It takes advantage of the latest hardware support for power consumption monitoring and reduction by intelligently powering down components within an individual processor.

- It delivers new solution to better secure VM start-ups as well reduce possible hacking opportunities by moving critical management processing out of global space into separate virtual sessions.

- It supports new memory access algorithms to reduce system wait time during critical memory requests and new scanning technology to optimize frame buffer searches.
- The Xen hypervisor supports wide range of guests including Windows, Linux, Solaris & various

versions of the BSD Operating systems.

- The Xen hypervisor is a thin software layer which provides an abstraction of physical server to run one or more “virtual servers”.

- The hypervisor is just made of approximately 150,000 lines of code, which means low- overhead and near-native performance for guests. The performance over head is quite less as shown in the initial results by P. Barham et al [4].

- Xen reuses existing device drivers (both closed and open source) from Linux, making device management quite easy.

- The virtual machines are isolated and don’t adversely affect the performance of other.

Xen multiplexes physical resources at the granularity of an entire operating system[4]. Thus, Xen is able to provide performance isolation between guests. Whereas, In contrast to process- level multiplexing which also allows a range of guest operating systems to gracefully coexist rather than mandating a specific application binary interface. There is a price to pay for this flexibility of running a full OS is more heavyweight than running a process, both in terms of initialization (e.g. Booting or resuming versus fork and exec), and in terms of resource consumption.

Xen avoid the drawbacks of emulation by presenting a virtual machine abstraction that is similar but not identical to the underlying hardware an approach which has been dubbed paravirtualization[16]. This promises improved performance, al- though it does require modifications to the guest operating system. It is important to note, however, that we do not require changes to the application binary interface (ABI), and hence no modifications are required to guest applications.

According to P. Barham et al. [4] Xen design principles includes:

1. Support for unmodified application binaries is essential, or users will not transition to Xen.

2. Supporting full multi-application operating systems is important, as this al- lows complex server configurations to be virtualized within a single guest OS instance.

3. Paravirtualization is necessary to obtain high performance and strong resource isolation on uncooperative machine architectures such asx86.

4. Even on cooperative machine architectures, completely hiding the effects of resource virtualization from guest operating system risks both correctness and performance.

One important feature of Xen that is extremely important to our work is that, it supports features like live migration among guests between different physical servers.

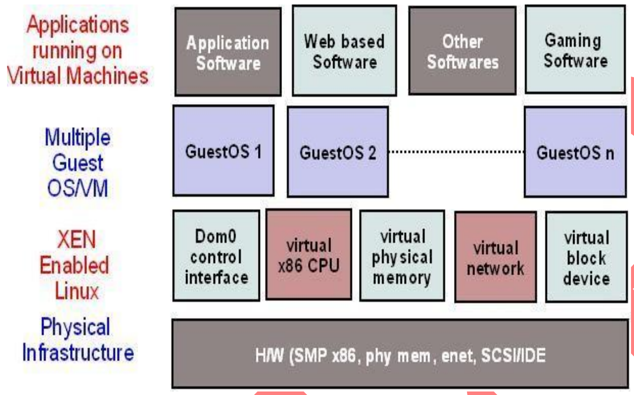

Figure 2: Xen Virtualization Stack

A running guest can be moved to a new virtual machine with downtime as low as 60 milliseconds [6]. This is usually accomplished by doing a pre-copy of RAM. Before the guest is migrated, RAM is copied from the source to the destination. Usually, the virtual machine will only be using a relatively small amount of its total memory, so even though some pages of memory will have to be sent again, many pages of memory only need to be sent once. This reduces the total time or downtime needed for the guest to be turned off and on.

2 RESOURCE MANAGEMENT

Resource management is a key component of all modern operating systems and hypervisors. In the context of virtualization, the ability to manage resources efficiently is critical for giving each virtual machine the illusion of being a dedicated physical machine that is fully protected and isolated from other virtual machines running on same server. Virtualization platforms should support flexible over commitment of processor, memory, and other resources in order to reap the benefits of statistical multiplexing, while still providing quality-of-service guarantees to VMs of varying importance. New challenges and opportunities arise as virtualization systems and the hardware they manage continue to evolve.

A challenge that comes with server virtualization is how to effectively manage capacities of virtual machines to increase resource utilization while ensuring that the service level objectives (SLOs) of applications running on them should be satisfied. Park et al. [14] tried to focus on the issues related to resource management in server virtualization. They have tried to use linear programming model to optimize the resources available for hosting virtual machines over multiple servers. The capacity management can be categorized into the capacity planning and dynamic resource allocation. The former is to allocate computing resources to virtual machines as an initial long-term planning action, while the latter is to respond to dynamic workload fluctuations as an instantaneous corrective or preventive action. Scheduling or Load balancing is the problem of finding a mapping between jobs to physical resources. In other words, load balancing is the problem of deciding which jobs should be allocated to which physical resources.

The basic purpose of a hypervisor is to schedule the use of single or multiple physical computing resources by single or multiple logical computing resources. The complexity of this operation is dependent on the capabilities of the hypervisor, including:

- Guaranteed resource service levels: The hypervisor should be able to schedule the logical resources fairly, such as CPU time slice, Network packets, I/O, etc.

- Dynamic addition/removal of logical resources: The hypervisor should provide facilities for dynamic updating of the logical resources.

- Moving of logical resources without interruption of service to the logical resource. One can move the whole OS or even a logical disk to other physical servers without disrupting the normal services.

- Grouping of logical resources for prioritization: The hypervisor should be enough smart to handle prioritized virtual systems and handle it accordingly.

- De-duplication of resources: It should provide mechanism to backup entire system without any inconsistencies in data without disrupting the normal services.

CPU virtualization abstracts physical CPUs into logical CPUs. For example, 8 logical CPUs could be created on a physical machine where only 2 CPUs exist. When a logical CPU is idle, the physical CPU can be assigned to other logical CPUs that have work to do.

3 MANAGING COMPUTATIONAL RESOURCE(CPU)

Computational resource (CPU) scheduling algorithms play an important role in assigning CPU time slice to each process for achieving fairness. Coming to virtual machine managers, there are compelling reasons to assign CPU slice to each guest OS proportionally. Proportional share scheduling algorithms allocate CPU slice in proportion to the number of shares (weights) that guests have been assigned.

Proportional share schedulers are evaluated based on the fairness level achieved on the given time interval. An important distinction between fair-share schedulers and proportional share schedulers is the time granularity at which they operate. Proportional share schedulers aim to provide an instantaneous form of sharing among the active clients according to their weights. In contrast, fair-share schedulers at- tempt to provide a time averaged form of proportional sharing based on the actual use measured over long time periods.

CPU schedulers can be further distinguished as work conserving and non-work conserving modes. Definition: In Work Conserving mode (WC), the shares are merely guarantees, and the CPU is idle if and only if there is no runnable client. It means that in a case of two clients with equal weights and a situation when one of these clients is blocked, the other client can consume the entire CPU. Definition: In Non-Work Conserving mode (NWC), the shares are caps, i.e., each client owns its fraction of the CPU. It means that in a case of two clients with equal weights, each client will get up to 50% of CPU, but the client will not be able to get more than 50% even if the rest of the CPU is idle.

Generally, there are two broad categories of scheduling algorithms i.e. Preemptive and Non-Preemptive.

Definition: Preemptive schedulers make scheduling decision every time when- ever a new client becomes ready. If the new client has "priority" over the running client, the CPU preempts the running client and executes the new client.

Definition: Non-preemptive scheduler only makes scheduling decisions when the running client voluntarily gives up CPU.

Having a preemptive scheduler is important for achieving good performance of I/O intensive workloads in shared environment. These workloads are often blocked waiting for I/O events, and their performance could suffer in presence of CPU intensive jobs if the CPU scheduler is non-preemptive.

However, choosing a right time slice size may alleviate this problem. To choose right quantum needs a good prediction algorithm for dynamic load balancing. Predicting load can help in balancing the load of the guests and allocate CPU time slice based on it. There are three scheduling algorithm which are already implemented in Xen over the period since its inception in 2003.

a. CPU SCHEDULARS INXEN

A. BORROWED VIRTUAL TIME(BVT)

BVT [5, 7] is a fair-share scheduler based on the concept of virtual time. It dispatches the runnable virtual machine with the smallest virtual time first. BVT also provides low-latency support for real-time and interactive applications by allowing latency sensitive client to "warp" back in virtual time to gain scheduling priority. The client effectively "borrowed" virtual time from its future CPU allocation.

The scheduler is configured with a context switch allowance C , which is the real time by which the current virtual machine is allowed to advance beyond another runnable virtual machine with equal claim on the CPU (it is the basic time slice or time quantum of the algorithm). Each runnable domain receives a share of CPU in proportion to its weight i . To achieve this, the virtual time of the currently running Domi is incremented by its running time divided by weight i.

In summary, BVT has the following features:

- Preemptive (if warp is used), Work Conserving (WC) – mode only;
- Optimally-fair: the error between fair share and actual allocation is never greater than context switch allowance C;
- Low-overhead implementation on multiprocessors as well as uni-processors.

The lack of Non-Work Conserving (NWC)- mode in BVT severely limited its usage, and led to the introduction of the next scheduler in Xen.

B. SIMPLE EARLIEST DEADLINE FIRST(SEDF)

SEDF [5, 11] uses real time-algorithms to ensure time guarantees. Each domain Domi specifies its CPU requirements by a tuple (si , pi , xi), where the slice si and the period pi together represent the CPU share that Domi requests: Domi will receive at least si units of time in each period of length pi . The Boolean flag xi indicates whether Domi is eligible to receive extra CPU time (WC-mode). This slack time is distributed in a fair manner after all the runnable domains received their CPU share. One can allocate 30% CPU to a domain by assigning (3 ms, 10 ms, 0) or (30 ms, 100 ms, 0). The time granularity in the definition of the period impacts the scheduler fairness[5].

For each domain Domi, the scheduler keeps track of two additional values (di , ri):

- d_i - time at which Dom$_i$'s current period ends, also called the deadline. The runnable field with earliest deadline is picked to be scheduled next;
- r_i - remaining CPU time of Dom$_i$ in the current period.

In summary, SEDF has the following features:

- Preemptive, WC and NWC modes;
- Fairness depends on a value of the period.
- Implements per CPU queue: this implementation lacks global load balancing on multiprocessors.

C. CREDIT SCHEDULAR

Credit scheduler [1, 5] is the recent proportional share scheduler in Xen. The features include automatic load balancing of virtual CPUs across physical CPUs on an SMP host. Before a CPU goes idle, it considers other CPUs to find any runnable VCPU. This approach guarantees that no CPU idles when there is runnable work in the system.

Each domain is assigned a certain number of credits by the system administrator. The credits are consumed during the course of execution. Appropriate credits are deducted after specified intervals by the scheduling mechanism. It can be changed at any time by the administrator.

Properties	BVT [5, 7]	SEDF [5, 11]	CS [1, 5]
Scheduling	Fair Share	Proportional	Proportional
Scheduler Type	Preemptive	Preemptive	Preemptive
Working Mode	WC	WC and NWC	WC and NWC
Fairness	Optimally Fair	depends on the value of thepe-	Depends on the credits
Overhead	Low overhead on uniprocessor, multi-	lacks global load balancing on	Scales well on multi-processors

Table 1: Comparison of different schedulers in Xen

The overall objective of the credit scheduler is to allocate the processor resources fairly, weighted by the number of credits each domain is allocated.

Each guest is assigned a weight and a cap. If the cap is 0 then virtual machine can receive extra CPU (Work Conserving mode). A non-zerocap (expressed as a percentage) limits the amount of CPU a virtual machine receives (Non-Work Conserving mode). The credit scheduler uses 30 ms time slices for CPU allocation. A virtual machine (VCPU) receives 30 ms before being preempted to run another virtual machine. Once every 30 ms, the priorities (credits) of all runnable virtual machines are recalculated. The scheduler monitors resource usage every 10 ms. To some degree, credits computation of credits resembles virtual time computation in BVT. However, BVT has a context switch allowance C for defining a different size of the basic time slice(time quantum), and an addition allow-latency support(via warp) for real-time applications.

In Practice, the credit scheduler [13] works as follows:

Domains can be in one of the two states: OVER and UNDER. If they are in the UNDER state, then they have credits remaining. If they are in the OVER state, then they have gone over their credit allocation. Credits are debited periodically after scheduler interrupt, which occurs in 10ms. At each scheduler interrupt, the currently running domain is debited 00 credits. When the sum of all the credits for all the domains in the system goes negative, all domains are given new credits.

While making scheduling decisions, domains in the UNDER state are always run before domains in the OVER state. A domain that is over its credit allocation is only executed if there are no domains in the UNDER state that are ready to run. This allows domains to use more than their fair share of the processor resources only if the processors would otherwise have been idle. The absolute number of credits that a domain has remaining is irrelevant. Rather, domains in the same state are simply run in a first-in, first-out manner. Domains are always inserted into the run queue that are in the same state, and the scheduler always selects the domain at the head of the run queue, it is allowed to run for three scheduling intervals (for a total of 30ms) as long as it has sufficient credits to do so. In summary, Credit has the following features:

- Non-Preemptive Work Conserving and Non-Work Conserving modes;
- Global load balancing on multiprocessors.

Consumption and re-evaluation of credits

Credit Scheduler schedules the guest domains based on the credits assigned by the system administrator with weight and cap values. If the default values are set then, it assigns 300 credits which allocates 30 ms time slice for each domain. It is a manual process and some compute intensive application can suffer in terms of performance. The credits are deducted at a regular interval of 10 ms which is a fixed value. Even this can be changed to test the performance. Which means after 30 ms the domain will have no more credits and will go in OVER state and will have to wait for fresh credits to be allotted and become in UNDER state, which will only happen once all the guest domains have finished their credits. A better technique form an aging credits can help in achieving better results. This is the place where the compute intensive domain will suffer and needs better mechanism to deal the situation effectively and reduce the wait time.

OUR APPROACH

We would like to apply the "Linear Models with Feedback Mechanism" to predict the load of the virtual machines dynamically and earn credits accordingly. By this approach we can give more CPU time to the needy OS and guarantee QoS of virtual machines in terms of real CPU time.
This approach needs to be implemented in the hypervisor with the existing credit scheduling

algorithm adding merely negligible cost for the overall performance.

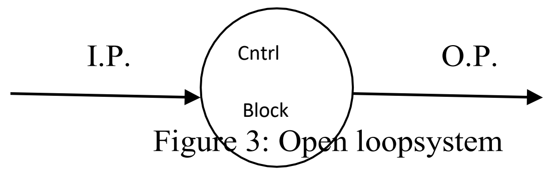

Figure 3: Open loopsystem

We expect that this kind of scheduling mechanism can deliver better performance to the virtual machine owner over the cloud and meet the laid down SLAs.

1 Concepts Related to Control Systems

Definition: According to Joseph [10], A Control system can be defined as "An arrangement of physical components connected or related in such a manner as to command, direct or regulate itself or another system". The purpose of the control systems usually identifies or defines the output and input. If the output and input are given, it is possible to identify, delineate or define the nature of the system components.

Open Loop and Closed Loop Control Systems

The distinction is determined by the control action, that quantity responsible for activating the system to produce the output.

Definition: An Open-Loop control system is one in which the control action is independent of the output.

Definition: A Closed-Loop control system is one in which the control action is somehow dependent on the output. Closed-loop control systems are more commonly called feedback control systems.

Feedback is that property of a closed-loop systems which permits the output to be compared with the input to the system so that the appropriate control action may be formed as some function of the output and input. Few papers [2, 15, 17] have used control theoretic approaches to predict and improve performance of web servers, network load and other servers.

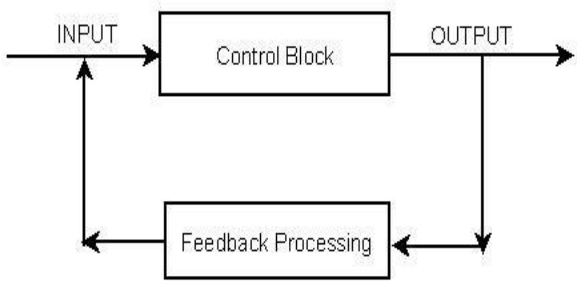

Figure 4: Closed loop with feedback mechanism

The fundamental reason feedback control theory is applicable in the computing domain is the fact that performance of computing services is tightly related to the status of various systems queues, such as the CPU scheduling queue and socket queues. At high load, these queues act as integrators of (request) flows, and hence can be described by difference equation models amenable to a control theoretical analysis [2]. To increase throughput, the fundamental thing in the computing system is the allocation of resources effectively and efficiently, such as CPU and network bandwidth, to computing tasks. Fair resource allocation determines how fast requests are served at different parts of the system, which is equivalent to manipulating and managing flows effectively.

There are few well-understood prediction techniques which have been applied in several areas. Such as predicting financial, predicting market needs or other industries. Time series prediction is one of them. An AR(Auto Regression) model also has ability to predict the next value or demand based on recent history and current value.

Predictive control is a technique which determines the value of the next interval based on prediction of certain attributes. The applied models are determined to depict the behavior of complex dynamic systems. Hence, these models can be used to predict the behavior of dependent variables (i.e. output) of the modeled dynamic system with respect to changes in the process independent variables (i.e. inputs). Feedback control theory has been applied to solve a number of performance and QoS issues previously [3, 17] and references therein. In these applications, we can see two major challenges for appropriate system modeling, controller designs, and the time varying needs placed on these systems by stochastic and sometimes bursty workloads

2 Proposed Algorithm

Load Aware Credit Scheduling Algorithm

OBJECTIVE: To propose “Load-Aware CPU Scheduling Algorithm for Guest OS” in context to virtual environment to meet SLAs better in dynamic situation over the cloud computing kind of scenario.

INPUT: Old credit value allotted for the domain, n number of previous samples to be considered

OUTPUT: New credit value for domain

METHOD:

1. Assign credits to all guest domains

2. Apply prediction algorithm to change the credits at some fixed regular intervals. Prediction Algorithm will run on the hypervisor i.eXEN

3. Prediction algorithm will predict the load of the guest domains and re- turn newly calculated credit value which can be used to assign a new set of probabilities and allocate future CPU time slice.

4. Intervals can be decided after intensive experimentation.

We can start with the default behavior of the Credit scheduler after implementing these enhancements.

The queuing theory based predictors can better model the queuing behavior and anticipate the impact of the workload changes on the controlled target such as the response time. However, it does require additional implementation of schemes to model and estimate the arrival process and service demand. We are looking at the possibilities of predicting the future resource requirements through resource utilization metrics such as CPU time consumption. Workload forecasting, capacity planning, fault detection, network loads etc. are few of the areas which also needs these kind of prediction algorithm.

a. Auto Regressive Model for Prediction

Statisticians [8] painstakingly observe and record processes which evolve in time, not merely for the benefit of historians but also in the belief that it is an advantage to know the past when attempting to predict the future. Most scientific schemes (and many non-scientific schemes) for prediction are ‘model’ based, in that they make some specific assumptions about the process, and then use past data to extrapolate into the future.

For example, in the statistical theory of ‘Time Series’, one often assumes that the process is some combination of general trend, periodic fluctuations, and random noise, and it is common to suppose that the noise component is a stationary process having an auto covariance function of a certain form.

Let Y (a) be the time series, where Y (a) represents the measured value of Y during time interval a. At the beginning of every interval a, a standard autoregressive model predicts an attribute’s value for the current interval using a linear combination of its measured value in the past several intervals,

$$Y'_a = \beta_0 + \sum_{i=1}^{n} \beta_i y_{(a-i)}$$

Where β0, β1... βn are constants.

Alternatively, we can consider ‘autoregressive schemes Y, being sequence which satisfy

$$Y'_a = \beta_0 + \sum_{i=1}^{n} \beta_i y_{(a-i)} + Z_a$$

Where,

$Y'_a$is the predicted value forY_a,β_i’s are

the predictor coefficients,

n is the order of the model that indicates the number of past samples used for the prediction, and Z_a is a sequence of uncorrelated variables with zero means and constant finite variance.

This model is useful for systems with some amount of memory as it's attribute value is strongly correlated to its recent past. The predictor coefficient can be estimated using the recursive least-squares algorithm. This approach allows the AR model to be updated periodically, adapting to possible changes in the system.

CONCLUSION

We have seen three different scheduling algorithms that evolved within the short span of time since its inception in 2003 at University of Cambridge Computer Laboratory. Today, Credit scheduler is the default scheduler and the first two i.e. BVT and SEDF are being removed from the Xen hypervisor. There are many scope for improvement in the credit scheduler to make it suitable for cloud computing kind of platforms and meet the SLAs quite effectively. We have identified few mathematical models which simulates real life situation but this is just a theoretical model and needs to be implemented on the hypervisor to test its correctness. The standard AR model is not sufficient to represent long term repeatable patterns in an attribute. For example, if the attribute Y demonstrates certain behavior at the fixed time every day, while the sampling interval for Y is one minute then such periodic pattern will not be captured in the above model. Some modification or other models can be used to model the exact behavior of the load on guest operating system or virtual machine.

REFERENCES

[1] Credit scheduler. http://wiki.-xensource.com/xenwiki/CreditScheduler.

[2] T. Abdelzaher, Ying Lu, Ronghua Zhang, and D. Henriksson. (2 July 2004), Practical application of control theory to web services. *In American Control Conference*, volume 3, pages 1992–1997 vol.3, June.

[3] Tarek F. Abdelzaher, Kang G. Shin, and Nina Bhatti. (2002); Performance guarantees for web server end-systems: A control-theoretical approach. *IEEE Trans. Parallel Distrib. Syst*., 13(1):80–96.

[4] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. (2003); Xen and the art of virtualization. *In SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 164–177, New York, NY, USA, ACM.

[5] Ludmila Cherkasova, Diwaker Gupta, and Amin Vahdat, (2007); Comparison of the three CPU schedulers in xen. *SIGMETRICS Perform. Eval. Rev.*, 35(2):42–51.

[6] Christopher Clark , Keir Fraser, Steven Hand, Jacob Gorm Hansen, Eric Jul, Christian Limpach, Ian Pratt, and Andrew Warfield. (2005); Live migration of virtual machines. *In NSDI'05: Proceedings of the 2ndconference on Symposium on Networked Systems Design & Implementation*, pages 273–286, Berkeley, CA, USA, USENIX Association.

[7] Kenneth J. Duda and David R. Cheriton. (1999); Borrowed-virtual-time (BVT) scheduling: supporting latency-sensitive threads in a general-purpose scheduler. *SIGOPS Oper. Syst. Rev.*, 33(5):261–276.

[8] Geoffrey Grimmett and David Stirzaker. (2005); *Probability and Random Processes*. OXFORD New York, third edition.

[9] A Hac and T Johnson. (1986); A study of dynamic load balancing in a distributed system. *SIGCOMM Comput. Commun. Rev.*, 16(3):348–356.

[10] Ivan J. Williams Joseph J. Distefano III, Allen R. Stubberud. (2003); *Feedback and Control Systems*. Tata McGraw-Hill, second edition.

[11] Ian M. Leslie, Derek Mcauley, Richard Black, Timothy Roscoe, Paul T. Barham, David Evers, Robin Fairbairns, and Eoin Hyden. (1996); The design and implementation of an operating system to support distributed multimedia applications. *IEEE Journal of Selected Areas in Communications*,14(7):1280–1297.

[12] Daniel A. Menasce, Lawrence W. Dowdy, and Virgilio A. F. Almeida. (2004); *Performance by Design: Computer Capacity Planning By Example*. Prentice Hall PTR, Upper Saddle River, NJ, USA.

[13] Diego Ongaro, Alan L. Cox, and Scott Rixner. (2008); Scheduling I/O in virtual machine monitors. *In VEE '08: Proceedings of the fourth ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, pages 1–10, New York, NY, USA, ACM.

[14] Jong-Geun Park, Jin-Mee Kim, Hoon Choi, and Young-Choon Woo. (Feb 2009); Virtual machine migration in self-managing virtualized server environments. *In Advanced Communication Technology*, 2009. ICACT 2009. 11th International Conference on, volume 03, pages 2077–2083.

[15] Dongxu Shen and Joseph L. Hellerstein. (2000); Predictive models for proactive network management: Application to a production webserver. *In Server, Proc. Network Operations & Management Symp*, pages 833–846.

[16] A. Whitaker, M. Shaw, and S. Gribble. (June 2002); Denali: Lightweight virtual machines for distributed and networked applications. *Proc. of the USENIX Annual Technical Conference*.

[17] Wei Xu, Xiaoyun Zhu, S. Singhal, and Zhikui Wang. (2006), Predictive control for dynamic resource allocation in enterprise data centers. *In Network Operations and Management Symposium*, 2006. NOMS 2006. 10th IEEE/IFIP, pages 115–126.