Performance Isolation Framework for Virtualized Server Applications

Hlaing May Tin
University of Computer Studies, Yangon, Myanmar
hlaingmaytin1982@gmail.com

Abstract

Recently, most IT organizations are transforming their data centers to smaller virtualized ones with the help of server virtualization. In virtualized servers, multiple applications are consolidated into a physical server by sharing and multiplexing their physical resources. For such environment, performance isolation among consolidated applications is the desirable thing to meet Service Level Agreements (SLAs) of those applications. This paper describes the way to control the total amount of CPU resource consumption in privileged and driver domains of each virtual machine (VM). By limiting the CPU resource usage of each VM in both domains, performance isolation among co-hosted application can be achieved. To accomplish this purpose, state space representation of Multi-Input Multi-Output (MIMO) controller is designed. The proposed framework is implemented and tested on a testbed which used Xen virtualization environment as an ongoing work.

1. Introduction

Today's data centers host a variety of business-critical applications such as web hosting, e-commerce sites and enterprise systems. Such application owner pay for renting server resources, and in return, the data center provider pays guarantees on resources availability and performance by means of SLAs. To meet these SLAs, data center must provision sufficient resources to applications as their need. Such provisioning can be based either on a dedicated or a shared platform. In a dedicated environment, some numbers of cluster nodes are dedicated to each application and provisioning technique must determine how many nodes to allocate to the application. In a shared environment, an application can share resources of physical node or server with other applications and the provisioning technique needs to determine how to partition resources on each physical server among competing applications. Since physical resources are shared, providing guarantees and isolation to the performance of applications in the shared data center model is more complex.

Several issues need to be addressed for virtualized servers such as mapping of resource requirements from physical to virtual environment, placement policies for virtual machines, dynamic resource provisioning, workload monitoring, and migration among VMs. Performance isolation of co-hosted applications in virtual execution environment, is another important goal [10]. Performance isolation means ensuring the performance requirement of one application should not impact the performance of another applications running in the same host.

The key contribution of this paper is to effectively control the total CPU consumption of each VM in both privileged and driver domains. Firstly, the system relates the desired performance of each application request or workloads to the required amount of resource to handle that workload. Next, the system accurately measures the resource consumption, including work done on behalf of a particular VM in Xen's driver domains. Finally, by using aggregate VM resource consumption in

allocating CPU that is collected from the Credit Scheduler, the MIMO controller limits the total amount of resources consumed in both domains without violation SLAs.

The rest of the paper is structured as follow; Section 2 describes the related work of various researches on performance isolation of VM. In section 3, a brief overview about server virtualization technology and Xen that is used as our testbed architecture is stated. Knowledge of State Space Model is described in section 4. Section 5 explores the MIMO controller that is used to control the CPU consumption of running VMs in each host is presented. In section 6, block diagram representation of testbed architecture is shown. Section 7, concludes the paper and our future work is described.

2. Related Work

Within the last decade, data centers have started employing virtualization solutions to consolidate multiple server applications on the same platform [6]. There have been a few studies on measurement and characterization of consolidation effects. For example, Cherkasova and et.al [4] measure the CPU and I/O overheads of virtualization.

In [5], the design and evaluation of a set of primitives implemented in Xen to address performance isolation issue is presented. In their work, they implemented XenMon to accurately measure the per-VM resource consumption and used SEDF-DC scheduler. In our work, the credit scheduler is used.

Adamczyk and et.al [1] proposed an idea on how to modify Xen back-end drivers to improve the network performance isolation. They found that by taking the aggregate CPU consumption into consideration, the performance isolation would be increased.

In [7], design of a performance isolation benchmark that quantifies the degree to which a virtualization system limits the impact of a misbehaving virtual machine on other well-behaving virtual machines running on the same physical machine is presented. They showed that without resource control, there would no evidence of isolation.

3. Overview of Server Virtualization

Server Virtualization, also referred to as platform virtualization, is abstraction of server resources (i.e., physical servers). A physical server is divided into multiple virtual server environments. Each virtual server environment is known as a Virtual Machine and the software used to divide the physical server is known as Virtual Machine Monitor/Hypervisor. The Virtual Machine creates an impression to the user of owning a complete physical server.

3.1. Models for Server Virtualization

There are three main models of server virtualization [2]:

Full Virtualization: In full virtualization, the guest OS is fully abstracted (completely decoupled) from the underlying hardware by the virtualization layer. As a result, the guest OS is not aware that it is being virtualized and requires no modification. No support is sought from underlying hardware as well. The hypervisor translates all the privileged instructions issued by the operating system on they while unprivileged user level instructions run unmodified on the processor. VMware's Server and Microsoft's Virtual Server are examples of full virtualization.

Paravirtualization: In paravirtualization a virtual machine is provided with an interface similar but not identical to the underlying hardware. Paravirtualization involves modifying the guest OS kernel to replace nonvirtualizable instructions with hypercalls that communicate directly with the virtualization layer hypervisor. Paravirtualization provides better performance guarantees than full virtualization. Xen supports paravirtualization model of virtualization.

Hardware-assisted virtualization: This approach requires support for virtualization from the underlying hardware. Guest OSes run unmodified in this model. Intel VT and AMD-V are the architectures supporting virtualization. In these architectures, some new instructions and a new privilege level, “Ring_1", is provided. The

hypervisor can run in this new privilege level while guest OSes run unmodified in Ring 0.

3.2. Xen

Xen is a virtualization system supporting both paravirtualization and hardware-assistant full virtualization. Its name comes from neXt gENeration virtualization. It is open source and initially created by University of Cambridge Computer Laboratory.

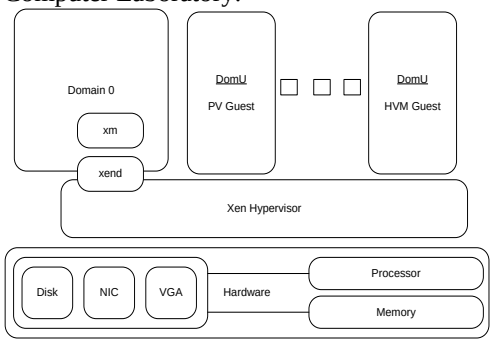

Figure 1. The architecture of xen virtualization

The above figure [11] shows the internal architecture of Xen. The core of Xen, which is responsible for control over all virtual machines, is a tiny operating system called Xen Hypervisor. Its main tasks are CPU scheduling, memory assignment and interrupt forwarding. Domain 0 is responsible for the creation and management of guest VMs via xm. It also interacts with Xen hypervisor by using xend (Xen daemon). There are two elements in Xen which may influence performance isolation, namely the CPU scheduler and the network IO scheduler.

3.2.1. CPU Schedulers in Xen

Xen is unique among VM platforms because it allows users to choose among different CPU schedulers. According to [3], three different CPU schedulers were introduced, all allowing users to specify CPU allocation via CPU shares (weights). They are

- **Borrowed Virtual Time (BVT)**: It is a fair-share scheduler based on the concept of virtual time, dispatching the runnable VM with the smallest virtual time first.
- **Simple Earliest Deadline First (SEDF)**: It provides weighted CPU sharing in an intuitive way and uses realtime-algorithms to ensure time guarantees.
- **Credit Scheduler**: It is a proportional fair share CPU scheduler built from the ground up to be work conserving on SMP hosts.

Among these three schedulers, the credit scheduler is chosen in our work. In credit scheduler, each domain (including Host OS) is assigned a **weight** and a **cap.** The **weight** defines how much CPU time a domain gets comparing to other virtual machines. A domain with a weight of 512 will get twice as much CPU as a domain with a weight of 256 on a contended host. Legal weights range from 1 to 65535 and the default is 256 [12]. The **cap** parameter is optional and describes the maximum amount of CPU a domain can consume. The cap optionally fixes the maximum amount of CPU a domain will be able to consume, even if the host system has idle CPU cycles. The cap is expressed in percentage of one physical CPU: 100 is 1 physical CPU, 50 is half a CPU, 400 is 4 CPUs, etc. The default, 0, means there is no upper cap. Using these two parameters the **number of credits** for each VM can be calculated**.**

4. State Space Model

In control engineering, a **state space representation** is a mathematical model of a physical system as a set of input, output and state variables related by first-order differential equations. State-space models use state variables in two ways [9]. The first is to describe dynamics by showing how x (k + 1) evolves from x (k). The second is to obtain the measured output y (k) from the state x (k). They provide a scalable approach to modeling MIMO systems, those with a multiple inputs and outputs. Specifically, if there are m_n inputs and m_O outputs, then there

are $m_n \times m_O$ transfer functions but only two state-space equations.

$$x(k+1) = Ax(k) + Bu(k)$$
$$y(k) = Cx(k) + Du(k) \quad (1)$$

In equation (1), the first-order differential equation is known as the *state equation* of the system and ***x (k)*** is the **state vector** and ***u (k)*** is the **input vector**. The second equation is referred to as the *output equation*. *A* is called the state matrix, *B* is the input matrix, *C* is the output matrix, and *D* the direct transition matrix. In our work, the state vector *x (k)* is the usage of CPU in each VM. The input vector is the performance requirement of each application workload. The output values are actual resource need to handle that workload.

5. MIMO Controller

Modern control theory utilizes the time-domain state space representation, a mathematical model of a physical system as a set of input, output and state variables related by first-order differential equations [9]. A control system must always have some robustness property and must adapt changes according to dynamical behavior of input. The process of determining the equations that govern the model's dynamics is called system identification. There are two main designs in the control system: SISO (Single-Input, Single-Output) and MIMO (Multiple-input, Multiple-Output).

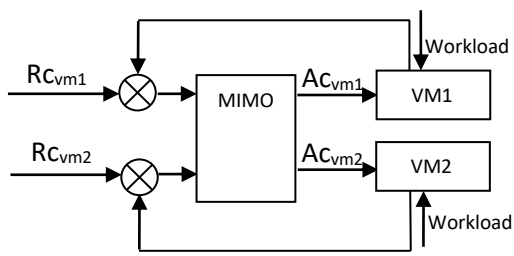

Figure 2. Architecture for MIMO controller

In our work, MIMO design is chosen to control CPU resource allocation to virtual machine in performance isolation way. Figure (2) shows example of controlling two VMs running in a host using MIMO controller. The credit scheduler is used to calculate CPU resource requested for each VM to meet performance requirement of dynamic workloads. According to information provided by scheduler, the requested CPU credit of each VM is fed into the controller in terms of Rc_{vm1} and Rc_{vm2}. Then, the controller gives the actual allocated amount of CPU resource to each VM in terms of Ac_{vm1} and Ac_{vm2}.

6. Testbed Architecture

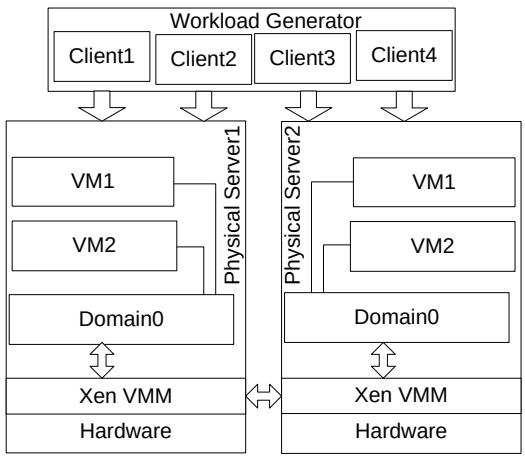

Figure 3. Block diagram of testbed

In our testbed, there are two physical server machine and one workload generator machine. In workload generator machine, four VMs running httperf tool [8] is created. These clients generate dynamic web service workload to two physical machines. In each physical machine or host, xen enabled hypervisor is installed. In Xen, domain0 is the most privileged domain to control the other VMs running in each host. Initially, our proposed system starts with two VMs running in each host. In each VM, CPU resource estimator in meeting application performance is included. A MIMO controller is presented in each host. It controls the aggregate CPU usage of virtual machines in that host. It takes CPU credits of each VM running as input matrix. According to workload variation of each application, controller tunes the

aggregate CPU limits for each VM in performance isolation fashion. It controls the total CPU usage of each VM in their driver domains and privileged domain to handle various workloads without violation SLAs.

7. Conclusion and Future Work

In this paper, performance isolation framework for the virtualized data center application is described with its related theoretical background and required technologies. The overall testbed environment of our work has also been described. The detailed implementation of our CPU resource controller will be described in our future work. The proposed framework will be implemented and evaluated with various workload benchmark scenarios in future as our ongoing work.

References

[1] B. Adamczyk and A. Chydzinski, "On the performance isolation across virtual network adapters in Xen", *CLOUD COMPUTING 2011: The Second International Conference on Cloud Computing, GRIDs, and Virtualization*.

[2] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauery, I. Pratt, and A. Warfield, "Xen and the art of virtualization", In Proceedings of *the 19th ACM Symposium on Operating Systems Principles*, October 2003.

[3] L. Cherkasova, D. Gupta, and A. Vahdat, "Comparison of Three CPU Schedulers in Xen", *ACM SIGMETRICS Performance Evaluation Review (2007), Volume: 35, Issue: 2*, Publisher: Citeseer, Pages: 42-51.

[4] R. Gardner and L. Cherkasova, "Measuring CPU overhead for I/O processing in the Xen virtual machine monitor", in Proceedings of *the USENIX Annual Technical Conference*, April 2005.

[5] D. Gupta, L. Cherkasova, R. Gardner, A. Vahdat, "Enforcing Performance Isolation Across Virtual Machines in Xen", in Proceedings of *the 7th ACM/IFIP/USENIX Middleware Conference* (November 2006).

[6] R. Iyer, R. Illikkal, O. Tickoo, L. Zhao, P. Apparao, and D. Newell, "VM3: Measuring, modeling and managing VM shared resources". *Computer Networks*, 53(17):2873 {2887, 2009}.

[7] J. N. Matthews, W. Hu, M. Hapuarachchi, T. Deshane, D. Dimatos, G. Hamilton, M. McCabe, and J. Owens, "Quantifying the Performance Isolation Properties of Virtualization Systems", in Proceeding of *the 2007 Workshop on Experimental Computer Science*, San Diego, CA, Jun. 13 - 14, 2007.

[8] D. Mosberger and T. Jin, "httperf---a tool for measuring web server performance", *ACM SIGMETRICS Performance Evaluation Review (1998)*, Volume: 26, Issue: 3, Publisher: ACM, Pages: 31-37.

[9] S. Parekh, D. M. Tilbury, J. L. Hellerstein, and Y. Diao, *Feedback control of computin g systems*, John Wiley and Sons, Inc, 2004.

[10] G. Somani and S. Chaudhary, "Application Performance Isolation in Virtualization", in Proceedings of *2009 IEEE International Conference on Cloud Computing*.

[11] Z. Shepherd, W. Hu, "Introduction to the Open Source Xen Hypervisor".

[12]http://wiki.xensource.com/xenwiki/CreditScheduler.html