DBA-VM: Dynamic bandwidth allocator for virtual machines

Ahmed Amamou, Manel Bourguiba, Kamel Haddadou, Guy Pujolle

DBA-VM: Dynamic Bandwidth Allocator for Virtual Machines

Ahmed Amamou, Manel Bourguiba, Kamel Haddadou and Guy Pujolle
LIP6, Pierre & Marie Curie University
4 Place Jussieu
75005 Paris, France
Email: {ahmed.amamou, manel.bourguiba, kamel.haddadou, guy.pujolle}@lip6.fr

***Abstract*—Cloud computing is an emergent paradigm that allows customers to rent infrastructure, platforms and software as a service. With resource sharing and reuse through virtualization technology, cloud environments become even more cost effective and flexible. Nevertheless, networking within virtualized cloud still presents some challenges in performance and resource allocation. In this paper, we propose DBA-VM, a Dynamic Bandwidth Allocator for Virtual Machines with regard to the established SLAs. The proposed scheme enforces the isolation between the virtual machines through the transmission bandwidth adjustment at the network I/O channel. The experimental performance evaluation shows that DBA-VM allows to the virtualized system to respect each virtual machine SLA while reducing the global physical resources (CPU and memory) consumption.**

I. Introduction

Cloud computing is a new technology trend that is expected to reshape the information technology landscape. It is a way to deliver software, infrastructure and platforms as a service to remote customers over the Internet. Cloud computing reduces hardware's management and software resources cost by shifting the location of the infrastructure to the network. It offers high availability, scalability and cost-effectiveness since it is particularly associated with the provision of computing resources on-demand and according to a pay-as-you-use model.

These resources are kept on the provider's servers which are located in various parts of the Internet. Their management is then shifted from the user to the provider. Cloud computing refers to both the applications delivered as services over the Internet and the hardware and systems software in the data centers that provide those services [1]. The cloud is defined in [2] as a large pool of easily usable and accessible virtualized resources (such as hardware, development platforms and/or services). These resources can be dynamically reconfigured to adjust to a variable load (scale), allowing also for an optimum resource utilization. This pool of resources is typically exploited by a pay-as-you-use model in which guarantees are offered by the infrastructure provider by means of customized Service Level Agreements (SLAs).

This definition introduces the virtualization as a key enabling technology for cloud computing. In fact, virtualization basically allows partitioning one physical machine to multiple virtual instances running concurrently and sharing the same physical resources. Advances in system virtualization make infrastructure-as-a-service a compelling paradigm since it offers cost effectiveness through resources sharing. It also offers flexibility through the ability of migrating virtual machines from one physical machine to another which helps reducing energy consumption. Furthermore, it enhances the cloud platform scalability and availability through the instantiation of new isolated virtual instances on demand.

Virtualization has been widely studied and deployed in recent years [3]. The Virtual Machine Monitor (VMM), also called hypervisor is a software layer that presents abstractions of the underlying physical resources to the guest machines. It allows the different virtual machines to share the physical resources including the network device. Network I/O virtualization is essential to provide connectivity to the virtual machines. However, the current implementations of VMMs do not provide high enough throughputs, especially when the applications running on different virtual machines within the same physical machine are I/O intensive (web services, video servers,...)[5][6][7]. Network intensive applications are among the applications dominating the cloud-based data centers today [9].

Although there are compelling advantages behind virtualizing the cloud computing infrastructure, there are still performance issues that need to be addressed before virtualizing the data centers could be fully advantageous. Indeed, concurrent applications share equally the available bandwidth. Current VMMs only offer a static allocation of the bandwidth. In this paper, we propose an SLA aware dynamic bandwidth allocator that dynamically manages bandwidth allocation among virtual machines according to the established SLAs. The proposed mechanism allocates the required bandwidth in terms of both bits per second and packets per second while minimizing global physical resources consumption.

The remainder of this paper is organized as follows: Section 2 introduces some related works. We state the problem through the native system evaluation in section 3. In section 4 we detail the proposed solution and its experimental evaluation in section 5. Finally, section 6 concludes the paper and introduces our future work.

II. Related Work

Over the last few years, a fair number of research efforts has been dedicated to the enhancement of the I/O virtualization

technology. In both [5] and [6], the authors conducted extensive measurements to evaluate the performance interference among virtual machines running network I/O workloads that are either CPU or network bound. They show how different resources scheduling and allocation strategies and workloads may impact the performance of a virtualized system. In [10] the authors show that cache and memory architecture, network architecture and virtualization overheads can be scalability bottlenecks in a virtualized cloud platform, depending on whether the application is compute or memory or network I/O intensive respectively. Network performance evaluation of virtual machines was the objective of multiple others works [11] [12]. The transmission, reception and emission throughputs of virtual machines are shown to be very low compared to the Dom0 (the privileged domain) performance. The multiple context switches and the costly I/O communication between the driver domain and the virtual machines through the event channel are behind this drastic performance degradation. A deep analysis of the network I/O operations within Xen in [8] shows that the grant mechanism incurs significant overhead when performing network I/O operations. This overhead is mostly due to the overheads of grant hypercalls and of the high cost of page mapping/unmapping. For this purpose, the authors proposed several optimizations to the memory sharing mechanism implemented in Xen. They improved the cache locality by moving the grant copy operation from the driver domain to the guest. Besides, they proposed to relax the memory isolation property to reduce the number of grant operations performed. In this case, performance would come at the cost of isolation, one of the most attractive benefits of the Xen architecture. In [13], the authors proposed a new design for the memory sharing mechanism with Xen which completes the mechanism presented in [8]. The basic idea of the new mechanism is to enable the guest domains to unilaterally issue and revoke a grant. This allows the guest domains to protect their memory from incorrect Direct Memory Access (DMA) operations. Beyond the memory sharing mechanism, the authors of [14] proposed to optimize the interrupt deliver route and shorten the network I/O path. In [10], the author shows that the out-of-the-box network bandwidth to another host is only 71% and 45% of non-virtualized performance for transmit and receive workloads, respectively. These bottlenecks are present even on a test system massively over-provisioned in both memory and computation resources. Similar restrictions are also evident in commercial clouds provided by Amazon [19], showing that even after much research effort I/O virtualization bottlenecks still challenge the designers of modern systems [20].

III. Background and problem statement

A. Virtualized cloud environment

A cloud platform basically consists in multiple data centers connected through a WAN and a web portal. The data center is composed of multiple physical nodes connected through a LAN. Inside the data center, the infrastructure can be virtualized, in which case each physical machine supports multiple isolated virtual machines. Different applications (game server, media server..) run over these virtual machines and users have direct access to those applications through the web portal. These virtual machines share the same hardware and storage, and can be migrated from one physical machine to another in the same data center or even in a remote data center. The VMM ensures physical resources sharing (CPU, memory, etc.) and provides isolated shared access to the devices through a special virtual machine called driver domain (Figure 1). The driver domain hosts the devices physical drivers and is responsible for protecting the I/O access as well as transferring the traffic to the appropriate virtual machine. With the driver domain I/O model, all the virtual machines share the same network interface and the driver domain demultiplexes incoming and multiplexes outgoing traffics. A great level of transparency is hence reached since the guest machines do not have to implement the eventually buggy device drivers. Besides, since all the traffic goes through the driver domain, this latter enjoys more traffic monitoring abilities like admission control or priorities establishment between the flows with regard to their types. However, this model performance experiences limitations due to the overhead incurred by the communication between the driver domain and the guests. We will further analyze this limitation in the next section.

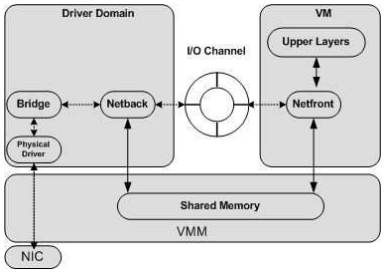


Fig. 1. Driver domain based I/O Virtualization model

B. Xen network I/O architecture

Xen [2] is a popular open source VMM for the x86 architecture. Xen relies on the driver domain to host device drivers and to ensure shared access to the network device among the guest machines [15]. In a Xen environment, the driver domain hosts the physical device drivers. Each guest machine is associated one or more virtual interfaces (vif) that are connected via a bridge to the physical interface. A vif is split into the netback (in the driver domain) and the netfront (in each guest machine). Shared memory pages are used to transfer the packets between the driver domain and the guests. Network transmissions and receptions are achieved as illustrated by Figure 1.

As soon as a packet is sent by upper layer, it is relayed to netfront, this latter notifies the netback of the arrival of the packet and copies the packet to its address space.When the driver domain is scheduled, the netback see the notification, look for the packet in shared memory page and relays it to the Bridge. The Bridge relays the packet to device driver that transmit it to the network device. Incoming packets will follow

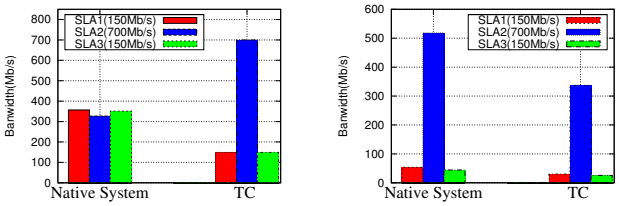
the opposite path.

C. Problem statement

In a virtualized cloud, multiple virtual machines are dedicated to different types of applications while sharing the same physical machine and network device. The sum of rates at which the virtual machines transmit cannot thus exceed the physical Network Interface Card (NIC) bandwidth. Some applications like video streaming servers are required to sustain an acceptable throughput so that the contract with the customer could be respected. The video server thus requires a bandwidth that may not be guaranteed in the presence of concurrent flows. In a native virtualized system, the virtual machines share the available bandwidth equally. Then, instantiating a new virtual machine may compromise the QoS required by already running applications. Native Xen only offers a tool to statistically set a cap on the bandwidth that a virtual machine can enjoy and the whole system needs to be restarted after each reconfiguration. First, we show through experimental evaluation how the Xen native system is unable to respect the bandwidth allocation specified in the SLAs.

1) Experimental Setup: The system that we are using is a Dell PowerEdge 2950 server, with two 2940 Mhz Intel Quad-core CPUs. Pairs of cores share the same L2 cache, and all 8 cores share the same main DDR2 667Mhz memory. Networking is handled by one quad-gigabit card using a PCI X4 channel. As a hypervisor, we use Xen 3.4.0 in para-virtualization mode. We instantiate a driver domain and three guest machines: VM1, VM2 and VM3 for traffic transmission. The driver domain is allocated four cores and each guest virtual machine is allocated only one core. As traffic sink, we used one NEC machine with a 2400 Mhz core 2 duo processor, 1GB DDR2 667Mhz and 1 Gb NIC. We used Iperf for the traffic transmission. VM1, VM2 and VM3 are characterized by the SLAs SLA1, SLA2 and SLA3 respectively, as follows: VM1 and VM3 require a bandwidth of only 150Mb/s while VM2 requires 700Mb/s. Each virtual machine is connected to one virtual interface. The three virtual machines send traffic at the rate of 1Gb/s. We consider the following two scenarios: In the first scenario, the three virtual machines send packets of 1500 bytes. In the second scenario, VM1 and VM3 send packets of 64 bytes while VM2 sends packets of 1500 bytes. In both scenarios, the three virtual machines transmit packets at the rate of 1Gb/s.

2) Experimental Results: Figure 2 shows how the native system is unable to guarantee the required bandwidth to each virtual machine. Indeed, the three virtual machines share equally the link bandwidth and transmit at 330 Mb/s each. Then, one can imagine that using a traffic shaping method as traffic controller (TC),deployed in driver domain, would resolve the problem. This is indeed true with the first scenario when virtual machines transmit large packets of 1500 bytes. However, with regard to the second scenario, we notice that neither the native system nor the native system with TC respect the established SLA. In fact, no virtual machine is able to transmit at the required throughput: VM1 and VM3 require



(a) scenario1 (b) scenario2

Fig. 2. Transmission throughput

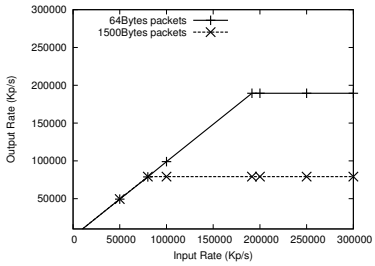


Fig. 3. Transmission Throughput in packets per second

150 Mb/s each, but they are able to achieve 50 Mb/s. VM2 sees its throughput limited to 520 Mb/s while it requires 700 Mb/s. This is due to the fact that the transmission is limited to 190 Kp/s as shown by figure 3 with 64 bytes sized packets. The virtual machine is then unable to transmit 64 bytes sized packets at more than 50 Mb/s.

The transmission capacity is even worse with traffic control TC since this latter rejects packets in the driver domain after they are transferred from the virtual machine through the shared memory. This leads us to evaluate the system's consumption in terms of CPU and memory transactions in order to determine the system bottleneck. This bottleneck is behind the transmission throughput limitation of 190 Kp/s. Then, we consider the two main system physical components: the CPU and the memory. We aim to determine the component which has reached its maximum capacity when the virtual machine transmits at the maximum throughput of 190 Kp/s. For each component, we profile its usage using Xenoprofile [21] in order to determine the effectively used capacity of the component. Then, we compare, we compare the upper-bound capacity per transmitted packet with the effectively consumed capacity per packet. Figure 4 shows that at a throughput of 190 Kp/s, the system has consumed all the available memory transactions while there still are available CPU cycles. We conclude then that the memory is the physical bottleneck of the system.

In current VMM implementations, when one virtual machine transmits at a rate exceeding the available bandwidth, the driver domain drops the packets (in the netback). Packets are then dropped after they have been transferred through the memory from the netfront to the netback. All of these operations are shown to require multiple memory transactions.

To encounter this problem, we propose to integrate an

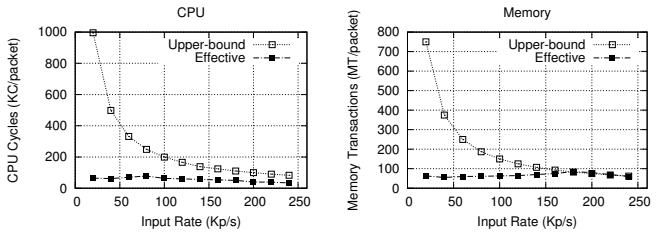


Fig. 4. Physical resources consumption

SLA-based Dynamic Bandwidth Allocator for the virtual machines called DBA-VM that will run in the driver domain to dynamically adjust the transmission bandwidth of each virtual machine according to the established SLA and the available bandwidth in terms of bits per second as well as packets per second.

Furthermore, in order to minimize the memory consumption, we propose that the DBA-VM drops packets in the netfront (rather than in the netback) whenever the packet is dedicated to be dropped due to bandwidth exceed. Thus we eliminate unnecessary and costly packet copies and notifications between the netfront and the netback.

IV. DBA-VM: Dynamic Bandwidth Allocator for Virtual Machines

We consider a virtualized system with a driver domain and several virtual machines with different QoS requirements depending on applications that each one hosts. We use an SLA that, in addition to system requirements (CPU, memory), specifies bandwidth usage in terms of bits per second and a maximum packets per second rate for each virtual machine as network requirements. Such an SLA definition takes also into consideration the physical machine packets per second rate limit. The proposed DBA-VM is built in a with regard to such an SLA definition. In order to guarantee an acceptable bandwidth to virtual machines hosting applications requiring QoS, DBA-VM proposes a differentiation mechanism operating at the driver domain level that dynamically readjusts transmission bandwidth according to the SLAs. This mechanism classifies the different virtual interfaces into classes that are characterized by a priority, by a maximum and minimum bandwidth and by maximum allowed packet per second rate.

Since the memory bottleneck is due to multiple useless packets copies from the netfront to the netback, DBA-VM will be deployed between these two components to avoid such useless memory usage.
Indeed, with the DBA-VM, the packets dedicated to be dropped whenever the maximum allowed bandwidth is exceeded, will be dropped at the netfront, before their transfer to the netback

In our algorithm we use the following notations:

- N the number of virtual machines.
- VM_j the virtual Machine j, j= 1..N
- vif_i virtual interface i, i=1..M
- B_p maximum bandwidth of the physical interface p.
- B_i the bandwidth at which vif_i is transmitting, i=1..M
- B_i^{max} the maximum bandwidth at which vif_i is allowed to emit, set in the SLA.
- B_i^{min} the minimum guaranteed bandwidth of vif_i, set in the SLA.
- B_p^{ex} is the available physical interface bandwidth.
- C_i the class of vif_i.
- pps^{VM_j} the maximum rate in packets per second that VM_j can send, set in the SLA
- pps^{vif_i} the maximum rate in packets per second that vif_i can send
- BT^{VM_j} total bandwidth emitted by VM_j

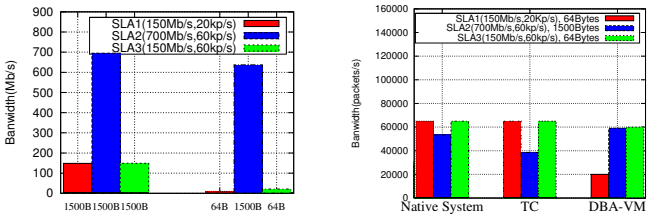
The DBA-VM is run in two steps: first it computes the maximum bandwidth in bits per second and second it computes the maximum packets per second rate.

a) Step 1: Maximum bandwidth computation in bits per second

For each physical interface P, the DBA-VM browses each vif_i attached to P starting with the ones belonging to the highest priority class. The DBA-VM measures B_i for each vif_i. In the case where multiple virtual interfaces belong to the same class, the DBA-VM will start with the first created one.
For each vif_i, if B_i is between B_i^{max} and B_i^{min} ($B_i^{max} < B_i < B_i^{min}$), then no change is made.
In the case where B_i exceeds B_i^{max} ($B_i > B_i^{max}$) then B_i will be readjusted to B_i^{max} and the available bandwidth B_P^{ex} will be augmented by the resulting difference of B_i-B_i^{max}. ($B_P^{ex} \leftarrow B_P^{ex}$ +(B_i - B_i^{max})
Finally in the case where B_i went below B_i^{min} then the DBA-VM checks whether there still is available bandwidth (B^{ex}) on the physical interface and whether (B_i - B_i^{min}) < B^{ex} or not.
If so, B_i^{min} is readjusted to B_i and B_P^{ex} is diminished by the difference B_i^{min} - B_i.
If not, in the case where the current virtual interface belongs to the least important class, it readjusts the bandwidth of all the other virtual interfaces vif_j, j=1..K belonging to the same class to B_j^{min} so that B_i could reach B_i^{min}. In the case where there are other less prioritized classes C_x, x=1..N, then the bandwidth of each virtual interface belonging to the class C_x is also readjusted to the minimum bandwidth of the class C_x: B_x^{min} starting with the least prioritized class.If there is an remaining available bandwidth $B^{ex} > 0$ then it will be reallocated to the different virtual interfaces based on their priorities.

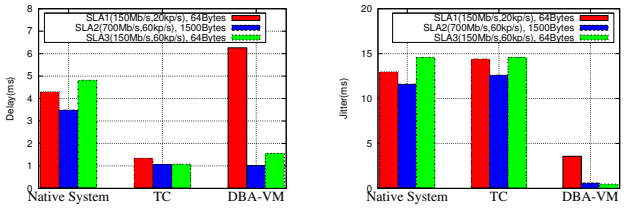
b) Step 2: Maximum bandwidth computation in packets per second

For each virtual machine j VM_j the DBA-VM browses each vif_i attached to VM_j. The DBA-VM measures B_i for each vif_i and then sums all B_i this sum is BT^{VM_j} which is the total bandwidth consumed by VM_j



(a) (b)

Fig. 5. (a) Transmission throughput with DBA-VM in bits per second (b)Transmission throughput in packets per second with different configurations



(a) Delay (b) Jitter

Fig. 6. Packets delay and jitter

For each virtual interface the bandwidth in packet per second is the total virtual machine packet per second multiplied by interface usage coefficient which is interface bandwidth divided by virtual machine total bandwidth (B_i /BT^{VM_j}).

V. Performance Evaluation

We have developed the proposed DBA-VM as a module that we integrated to the driver domain kernel. It consists of a daemon that periodically executes the described algorithm, checks the rate at which each virtual machine is transmitting, and reconfigures all the virtual machines rates according to these results and SLA definition.

In order to evaluate our algorithm performance, we will compare it to Native Xen System and also to traffic shaping mechanism using TC deployed in the driver domain. We use the same experimental setup and scenarios as in section III.c.

We also modify the three SLA by introducing the SLA packets per second parameters as follows: SLA1 fixes the maximum rate to 20 kilos packets per second (Kp/s), while SLA2 and SLA3 fix it to 60Kp/s.

We present DBA-VM evaluation of bandwidth , QoS parameters and System resources consumption.

A. System throughput

For homogenous traffic of large packets (1500 bytes) figure 5(a), DBA-VM allows the system to respect the SLA. In fact, as total packet rate per second is well below VM maximum achievable packets per second rate, all the SLAs throughput in terms of packets per second and bits per seconds are respected. In the second scenario, the virtual machines transmit a mixture of large and small packets. We notice a decrease in the transmission of the three virtual machines especially for VM1 and VM3 which transmission throughput dropped from 150 Mb/s to 9.1 Mb/s and 22 Mb/s respectively. In such case VM2 throughput decreases slightly while we have a more important decrease in VM1 and VM3 throughput. However M2 throughput for the DBA-VM case in scenario 2 is clearly better than for native System and native system with TC.VM1 is more affected than VM3 by this decrease since its SLA allows less packets per second rate. We also notice that even if the VM1 and VM3 throughput is decreased, the packets per second rate specified in the SLA is respected as shown by

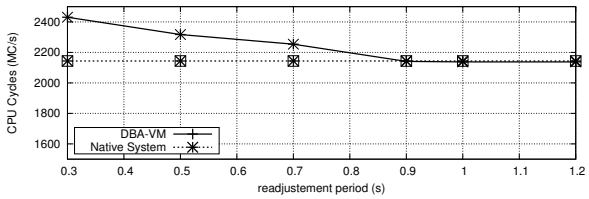


Fig. 7. System Resource consumption: CPU

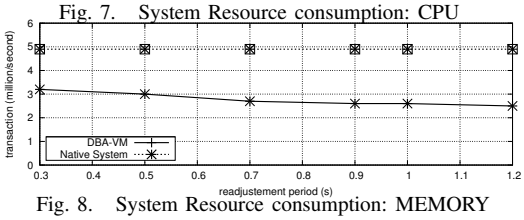


Fig. 8. System Resource consumption: MEMORY

figure 5(b). Unlike native system and native system with TC, DBA-VM imposes a strict packet per second allowed rate. This constraint allows a better network bandwidth sharing between the different virtual machines.

B. Performance analysis for QoS parameters

We first notice that the proposed mechanism considerably reduces the packets delay for the flows transmitted by VM2 and VM3 from respectively 3 and 4ms to less than 1ms and 1.6ms.However, we notice an increase in the delay for packets transmitted by VM1 from 4ms up to 6ms. As VM1 has low packets per second rate, it is scheduled for a smaller period comparing to VM2 and VM3 so this lead to a bigger packets transmission delay. As VM1 has a relatively low packets per second rate compared to VM2 and VM3, it is expected that the packets transmitted by VM1 experience a higher delay. The jitter value in DBA-VM is around 0.5ms for high priority flows. This represents a relatively good result for QoS compared with a TC based system.

It is interesting to point out that with DBA-VM, when we are under the SLA limits, the loss rate is lower than with the native system since we control the packets transmission rate for each Virtual machine so we can limit packets drop. However, as soon as we reach the SLA limitations, the loss rate grows rapidly. This is an intended mechanism to avoid affecting the other machines performances.

C. System Resources consumption

We also evaluated the system CPU and memory resources consumption for different readjustment periods. The readjustment period is the period after which bandwidth is recomputed in terms of bits per second and packets per second for each Virtual Machine.
The DBA-VM avoids transferring packets emitted from virtual machines beyond their SLA. It also avoids packets loss in driver domain. This leads to less system resources consumption. However the algorithm introduces a CPU and memory overload. Using a long readjustment period leads to less computation in the daemon, so this will lead to less memory and CPU usage for a non I/O operation, in return this leads to a biggest adaptation time. A compromise should be found between adaptation time and system resources consumption.

In order to evaluate the DBA-VM impact on system resource consumption we profiled the system resources usage (memory transaction and CPU cycles) using Xenoprofile [21] with both native and DBA-VM system. Figures 8 and 9 present CPU and memory usage for different readjustment periods.

First we observe that for a period lasting more than 0.3 second, the DBA-VM consumes less memory transactions than the native system. This is due to the fact that we avoid useless memory copies between the netfront and the netback.

With regard to the CPU consumption, the DBA-VM consumes as much CPU cycles as the native system form a readjustment period equal to 0.9 second. This value represents a good check period since with such a value, the DBA-VM also reduces the achieved memory transactions compared to the native system.

VI. Conclusion

In this paper we proposed DBA-VM, a new mechanism for the dynamic bandwidth allocation to the virtual machines, in a virtualized cloud environment. We first showed the native system and the TC based system shortcomings in guaranteeing the required transmission bandwidth to the virtual machines. We have also evaluated the system's capacity in terms of transmitted packets per second and show that the memory severely limits this transmission rate. These findings led us to propose a novel scheme that enables the system to adjust the transmission rate of the virtual interfaces in the I/O channel according to the virtual machine SLA, the transmission bandwidth being defined in terms of both bits per second and packets per second in the SLA. The experimental evaluation first shows that the proposed mechanism allows the respect of the virtual machines SLA by enforcing the isolation between the different flows as well as an improvement in the QoS parameters. Furthermore, this gain is achieved while reducing the total cost in terms of physical resources (CPU and memory) usage. We intend next to extend our algorithm to establish the SLAs based on flows classes rather than virtual machines classes. Furthermore, our proposal could be extended to define classes according to multiple QoS parameters like packet delay and jitter in order to enable virtualized cloud totally respond to customers expectations.

References

[1] M. Armbrust, A. Fox, R. Griffith, A.D Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, A.Zaharia, "Above the Clouds: A Berkeley View of Cloud Computing", Technical Report No. UCB/EECS-2009-28, February 10, 2009.

[2] L. M.Vaquero, L. Rodero-Merino, J.Caceres, M.Lindner, A Break in the Clouds: Towards a Cloud Definition, ACM SIGCOMM Communication Review, vol 39, no. 1, Jan. 2009, pp. 50-55.

[3] N. Feamster, L. Gao, and J. Rexford, How to lease the Internet in your spare time, in the Editorial Zone of ACM SIGCOMM Computer Communications Review, p. 61-64, January 2007

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

[5] P. Xing, L. Ling, M. Yiduo, A. Menon, S. Rixner, A.L Cox, W. Zwaenepoel, Performance Measurements and Analysis of Network I/O applications in Virtualized Cloud, International Conference on Cloud Computing, 2010.

[6] P. Xing, L. Ling, M. Yiduo, S. Sivathanu, K. Younggynm, P. Calton, Understanding Performance Interference of I/O Workload in Virtualized Cloud Environments. International Conference on Cloud Computing, 2010.

[7] P. Apparao, S. Makineni, and D. Newell, Characterization of network processing overheads in xen, in Proceedings of the 2nd International Workshop on Virtualization Technology in Distributed Computing VTDC 2006, Washington, DC, USA, 2006.

[8] JR. Santos, Y. Turner, G. Janakiraman, I. Pratt, Bridging the gap between software and hardware techniques for I/O virtualization, USENIX Annual Technical Conference, 2008.

[9] A. Li, X. Yang, S. Kandula, M. Zhang, CloudCmp: comparing public cloud providers, in Proceedings of the 10th annual conference on Internet measurement (IMC '10), 2010

[10] J. Shafer, I/O Virtualization Bottlenecks in Cloud Computing Today, Workshop on I/O Virtualization (WIOV 2010), Pittsburgh, 2010.

[11] F. Anhalt, and P. Vicat-Blan Primet, Analysis and experimental evaluation of data plane virtualization with Xen, in Proceeings of the fifth International Conference on Networking and Services 2009

[12] X. Xu, F. Zhou, J. Wan, and Y. Jiang, Quantifying performance properties of virtual machine, in the International Symposium on Information Science and engineering, 2008

[13] K.K Ram, Y. Turner and J.R, Santos, Redesigning Xen's memory sharing mechanism for safe and efficient I/O virtualization , In the second workshop on I/O virtualization, Pittsburgh, PA, USA, 2010.

[14] J. Zhang, X. Li, and H. Guan, The optimization of Xen network virtualization, in the proceedings of the International Conference on Computer Science and Software Engineering, 2008.

[15] K. Fraser, S. Hand, R. Neugebauer, I. Pratt, A. Warfield, and M. Williams, Safe hadrware Access with the Xen virtual machine monitor, In Proceedings of the first workshop on Operating System and Architectural Support for the on demand IT Infrastructure, OASIS 2004.

[16] X. Zhang, and Y. Dong, Optimizing Xen VMM based on Intel Virtualization technology, In the proceedings of the International Conference on Computer Science and Software Engineering, 2008.

[17] D. Guo, G. Liao, and L.N Bhuyan, Performance characterization and cache-aware core scheduling in a virtualized multi-core server under 10GbE, in the Proceedings of the 2009 IEEE International Symposium on Workload Characterization (IISWC) 2009

[18] G. Liao, D. Guo, L. Bhuyan, and S.R King, Software techniques to improve Virtualized I/O performance on multi-core systems, in Proceedings of the 4th ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS) 2008.

[19] http://aws.amazon.com/ec2

[20] S.K. Barker, P. Shenoy, Empirical Evaluation of Latency-sensitive Application Performance in the Cloud, in the Proceedings of the first annual ACM SIGMM conference on Multimedia systems (MMSys '10) 2010

[21] A. Menon, G. Janakiraman, JR. Santos, and W. Zwaenepoel, Diagnosing performance overheads in the Xen virtual machine environment,VEE 2005.