A dynamic bandwidth allocator for virtual machines in a cloud environment

Ahmed Amamou, Manel Bourguiba, Kamel Haddadou, Guy Pujolle

A Dynamic Bandwidth Allocator for Virtual Machines in a Cloud Environment

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 to customers to rent infrastructure, platforms and software as a service. With resource sharing and reuse through virtualization technology, cloud environments become even more effective and flexible. Nevertheless, networking within virtualized cloud still presents some challenges in performance and resource allocation. In this paper, we propose to integrate an SLA-based Dynamic Bandwidth Allocator (DBA) in a vitualized cloud environment. DBA manages bandwidth allocation efficiently through allocating bandwidth according to the application requirements and the established agreement. It also adjusts the allocated bandwidth dynamically upon change and reduces physical resources usage by dropping packets in the virtual machines rather than the driver domain. Through experimental evaluation we showed the efficacy of the proposed algorithm and the agreements respect.**

I. Introduction

Cloud computing is a new technology trend that is reshaping the information technology landscape and gaining much of the interest of industry as well as academia. Cloud computing enables providers to deliver software, platform and infrastructure as a service to remote customers over the network. The need behind the cloud computing is the deployment of large scale data centers at low costs. Henceforth, customers do not need to plan for provisioning anymore; they rent computing and networking resources on demand and increase those resources and pay for them on a short-term basis as needed [1]. Cloud computing offers cost effectiveness and high availability of resources. The provider owns a pool of resources that it configures, adjusts and offers to customers according to Service Level Agreements (SLAs).

When coupled with virtualization, the cloud computing model even enables higher utilization rates while reducing dedicated hardware costs. Virtualization is an old technology that gained renewed interest recently. It basically offers a partitioning technique to run multiple and isolated virtual machines on a single physical machine. Thus, virtualization optimizes hardware usage through resource reuse and multiplexing which decreases the cost of power, hardware and network bandwidth. Furthermore, through on demand virtual machine creation and migration and dynamic resource allocation it enables flexible, scalable and cost effective virtualized data centers deployment.

A Virtual Machine Monitor (VMM) is a software layer that manages resource sharing among the concurrent virtual machines (VMs) and ensures that diverse and different applications run in isolated environments. The driver domain is a special virtual machine that is in charge of managing the shared access to the devices, especially the network interface card (NIC). The driver domain handles networking by multiplexing outgoing and demultiplexing incoming traffic. This additional layer in the packets path obviously incurs an additional overhead. The I/O mechanism of the VMM consists in copying the packets to the shared memory between the driver domain and the virtual machine. This costly mechanism is behind this additional overhead.

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 Allocation algorithm that dynamically manages bandwidth allocation among virtual machines according to their priorities while minimizing physical resources consumption.

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

II. Overview and background

A. Cloud computing environment

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[9]. Services in the Cloud can be: Software (SaaS), Platform (PaaS), or Infrastructure (IaaS). The distinction is made based on the level of abstraction presented to the client and the level of management of resources. To better understand the performance limitations of a cloud computing infrastructure, we first need to understand how cloud platforms are designed. A cloud platform is basically composed of multiple data centers with a web portal, connected through a WAN. The data center is composed of multiple physical nodes connected through a LAN. Inside the data center, the infrastructure can be

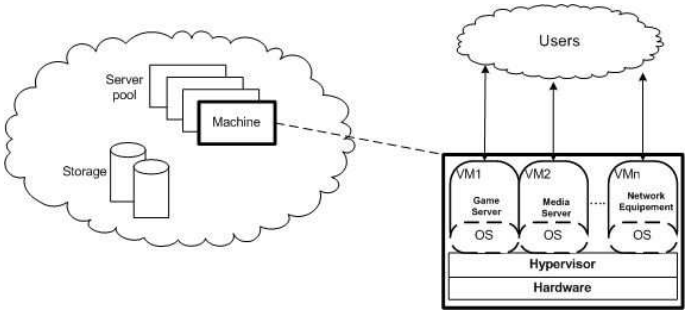

Fig. 1. Virtualized Cloud Platform

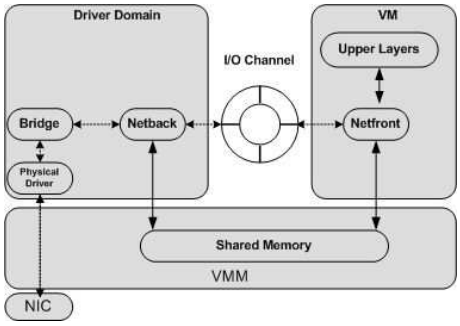

Fig. 2. Packet transmission in a Xen environment

virtualized (Figure 1) in which case each node supports multiple isolated virtual machines. 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. In native virtualization technologies, virtual machines also share the access to the network device, and the available bandwidth. This latter is equally allocated among the concurrent 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. The user can either have access to only the application, or the the development platform or even the whole stack.

B. Virtualization technology

Most deployed virtualization technologies include Xen, VMWare, OpenVZ, and Linux VServer. OpenVZ and LinuxVServer offer operating system (OS) level virtualization, where the OS supports multiple isolated user-space instances called containers. They share the same kernel. Xen and VMWare fall into the full and para-virtualization categories respectively. In both categories, the Virtual Machine Monitor (VMM) presents software interfaces to VMs. The main difference between these two categories is that the guest OS must be ported in para-virtualization and not in full-virtualization. A VMM enables multiple virtual machines to share the same physical machine. The VMM must provide shared access to the network interface and ensure isolation. Shared access is offered by a special virtual machine called I/O domain or driver domain (DD).
Xen [2] is a popular open source VMM for the x86 architecture that uses this networking model. Guests in a Xen environments are called Unprivileged domains (DomU). One special privileged domain called Domain0 (Dom0) is responsible for managing (creating, migrating, destroying...) the other guest machines. The driver domain is a dedicated domain (that can be either one DomU or Dom0 itself) responsible of the shared access to devices especially the network device. The driver domain is usually Dom0 itself. Xen offers a high level of isolation through its memory sharing secure mechanism. The driver domain is responsible for protecting I/O access and is trusted to transfer traffic to the appropriate virtual machine. Moreover, high flexibility is also offered since it is possible to customize data planes by modifying the network stack in the kernel, which is not possible in container-based virtualization since only the application level is virtualized and all virtual instances share the same kernel. In our work, we will use Xen as a virtualization layer and Dom0 as the driver domain. In the text Dom0 and driver domain are used interchangeably.

C. Adopted Xen network I/O architecture

In this section, we will detail networking with Xen. In addition to real device drivers, virtual drivers are implemented in both Dom0 and the guest domains. A virtual driver is split into the netback (in Dom0) and the netfront (in each DomU). In each virtual machine, the netfront corresponds to a virtual interface (vif) which is characterized by a transmission bandwidth. All the virtual interfaces are connected to the bridge through the netback. The bridge demultiplexes the incoming traffic to the different netbacks and multiplexes the outgoing traffic to the NIC. Inter-domain communication as well as communication between the hypervisor and the virtual machines is ensured by the event channel. It is a notification mechanism that is particularly used by the hypervisor to notify Dom0 of the arrival of a packet, or by Dom0 to notify the DomU destination that a packet was placed in its memory space. Shared memory pages are used to really transfer the packet between domains. Network transmissions and receptions are achieved as illustrated by Figure 2.
In this paper we are only interested in traffic transmission, we below detail packets transmission path. Whenever a virtual machine has a packet to transmit, it copies the packet to a memory page and issues a grant through the VMM in order to inform the Dom0 that he is allowed to access that page. Then it sends a notification to the Dom0 to inform him that the packet has been copied to the memory page. When scheduled, the Dom0 will se the notification and handle it by accessing the memory page in order to get the packet. The Dom0 notifies then the virtual machine and the packet is henceforth handled by the bridge, which will relay the packet to the NIC. As soon as the virtual machine gets the notification, it revokes the grant. Incoming packets will basically follow the opposite path.

III. Related Work

Over the last few years, a fair number of research efforts has been dedicated to the enhancement of I/O virtualization technology in the context of virtualized cloud environments. In both [3] and [4], 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 showed how different resources scheduling and allocation strategies and workloads may impact the performance of a virtualized system. In [5] the authors proved that cache and memory architecture, network architecture and virtualization overheads can be scalability bottlenecks in a virtualized cloud, depending on whether the application is compute or memory or network I/O intensive respectively. None of these works proposed new techniques to improve I/O performance. [6] 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 [7], 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, showing that even after much research effort I/O virtualization bottlenecks still challenge the designers of modern systems. In[8], Kesavan and Al formalize the way in which hypervisors support proportional sharing for I/O requests, by developing and presenting the novel notion of Differential Virtual Time (DVT). A specific technical problem addressed by DVT and elaborated in this paper is that in the case of proportional sharing of network I/O, the presence of a conventional I/O scheduler introduces an additional delay into the network processing path. Moreover, the delay experienced by individual VMs changes with the number of other concurrently active VMs and with their traffic patterns. In [10], the authors used the SLA approach to propose an architecture for resources provisioning in the context of a virtualized cloud. The paper mainly addresses issues related to negotiation and brokering for virtual resources provision. However no practical performance evaluation has been conducted to show the effectiveness of the solution. Our contribution represents an enhancement of the networking performance of virtual machines in the cloud. This enhancement takes into consideration the agreements set up between the customer and the provider. Experiments show its feasibility with respect to the established agreement.

IV. 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 interfaces transmit can not thus exceed the physical NIC bandwidth. Some applications like vido 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. Virtual machines share the available bandwidth equally. Then, instantiating new virtual machines may compromise the QoS required by already running applications. The native system of Xen only offers a tool to statistically set a cap on the bandwidth a virtual machine can enjoy. The whole system need to be restarted after each reconfiguration. To encounter this problem, we propose to integrate an SLA-based Dynamic Bandwidth Allocator (BDA) that will be run in the driver domain to dynamically adjust the transmission bandwidth of each virtual machine according to the established service level agreements and the available bandwidth.
Furthermore, 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. In [12], the authors have shown that the memory access is the bottleneck preventing the transmission throughput from scaling up to line rates. Indeed, the I/O mechanism of Xen involves much operations including granting the memory page, revoking the grant, copying the packet and notifying the netback of the packet transfer. All of these operations are shown to require multiple memory transactions and CPU cycles. In order to minimize this resource consumption, we further propose to drop packets in the netfront (rather than the netback) whenever the packet is dedicated to be dropped due to bandwidth passing. Thus we eliminate unnecessary and costly packet copies and notifications between the netfront and the netback.

V. DBA: Dynamic Bandwidth Allocator

A. Algorithm

We consider a virtualized system with a driver domain and multiple virtual machines hosting different applications with different QoS requirements. Each virtual machine transmits traffic through its virtual interface (vif). Each vif is connected to the physical interface through the bridge. A virtual machine is instantiated with a set of characteristics defined in the Service Level Agreement (SLA) established between the customer and the provider. The SLA specifies the system physical resources allocated to the virtual machine and the networking parameters. Physical resources include CPU cycles and memory and networking parameters include bandwidth, packets delay and jitter,etc. In order to guarantee an acceptable bandwidth to virtual machines hosting applications requiring QoS (for example: video streaming servers), we propose a differentiation mechanism operating at the driver domain that dynamically readjusts transmission bandwidth according to the SLAs. This mechanism classifies the different virtual interfaces into classes that are characterized by a priority and

by a maximum and minimum bandwidth. The aim of the proposed mechanism is to guarantee to each virtual machine a minimum bandwidth at which it can transmit and to prevent it from exceeding a cap bandwidth not to compromise the rest of the machines QoS.
We denote by:

- N the number of virtual machines.
- vif_i virtual interface i, i=1..N
- B_p the physical interface maximum bandwidth of the physical interface P.
- B_i the bandwidth at which vif_i is transmitting, i=1..N
- 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.

For each physical interface P, the DBA browses each vif_i attached to P starting with the ones belonging to the highest priority class. The DBA measures B_i for each vif_i. In the case where multiple virtual interfaces belong to the same class, the DBA 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})
$B_i \leftarrow B_i^{max}$

Finally in the case where B_i went below B_i^{min} then the DBA 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.

$B_P^{ex} \leftarrow B_P^{ex}$ - (B_i^{min} - B_i)
$B_i \leftarrow B_i^{min}$

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..N belonging to the same class to B_j^{min} so that B_i could reach B_i^{min}.

for (j in 1..N){
$B_j \leftarrow B_j^{min}$
$B_P^{ex} \leftarrow B_P^{ex}$ +(B_j - B_j^{min}) }
if ($B_P^{ex} > B_i^{min}$){
$B_P^{ex} \leftarrow B_P^{ex}$ - (B_i^{min} - B_i)
$B_i \leftarrow B_i^{min}$
} else{ $B_i \leftarrow B_P^{ex}$
$B_P^{ex} \leftarrow 0$ }

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.

for (x in 1..N){
for (j in 1..N_x){ $B_j \leftarrow B_j^{min}$
$B_P^{ex} \leftarrow B_P^{ex}$ +(B_j - B_j^{min})}
if ($B_P^{ex} > B_i^{min}$){
$B_P^{ex} \leftarrow B_P^{ex}$ - (B_i^{min} - B_i)
$B_i \leftarrow B_i^{min}$
} else{ $B_i \leftarrow B_P^{ex}$
$B_P^{ex} \leftarrow 0$ }

Finally if the remaining available bandwidth $B^{ex} > 0$ then it will be reallocated to the different virtual interfaces based on their priorities.

B. Performance Evaluation

We have developed the proposed DBA as module that we integrated to the driver domain. It consists of a daemon that periodically executes the algorithm we described, checks the rates at which the different virtual interfaces are transmitting and configures them accordingly. Below we present the experimental evaluation of our system.

1) Experimental setup: For our experiments, we used a Dell PowerEdge 2950 server, with two Intel Quad-core CPUs with a frequency of 2490Mhz for each core. Pairs of cores share the same L2 cache of 8MB, and all 8 cores share the same main DDR2 667Mhz memory. Networking is handled by one gigabit card using a PCI x4 channel. The e1000 driver was used with NAPI enabled. Xen 3.4.0 is used as a hypervisor. We developed the proposed mechanism with language C, as a module that we integrated to the driver domain. The sink of the traffic is a Nec PC, with a 2400 Mhz core 2 duo processor and a DDR2 667Mhz memory also equipped with one Gigabit card. For traffic emission and reception we used Click [11]. We instantiated 3 virtual machines for traffic transmission, allocated one core and 1GB of memory each. The Dom0 is allocated the rest of the cores and memory and is playing the role of driver domain. One UDP flow is generated within each virtual machine. Packets size is set to 1500 bytes.
Scenario:
We first evaluated the throughput achieved by the 3 virtual machines with the native system with the following scenario: Each virtual machine belongs to a different class. MV_i belongs to class C_i, i=1..3. We set the minimum bandwidth to 300 Mb/s, 100 Mb/s and 0 Mb/s and the maximum bandwidth to 600 Mb/s, 300 Mb/s and 100 Mb/s for C_1, C_2 and C_3 respectively. The traffic generation lasts for 300s and the transmission rate is distributed as follows: MV1 transmits at 800 Mb/s for the first 100 s. Then at 200 Mb/s for the next 40 s. The input rate is increased to 600 Mb/s for the next 30 s and decreased to 200 Mb/s during the next 20 s. After that, MV1 transmits at 500 Mb/s during 60 s and finally at 800 Mb/s during 50 s. However, both of MV2 and MV3 transmit at a constant bit rate of 800 Mb/s during the whole test duration.

C. Experimental Results

1) System throughput: Figures 3 and 4 show the throughput of the native system and DBA-enhanced system respectively

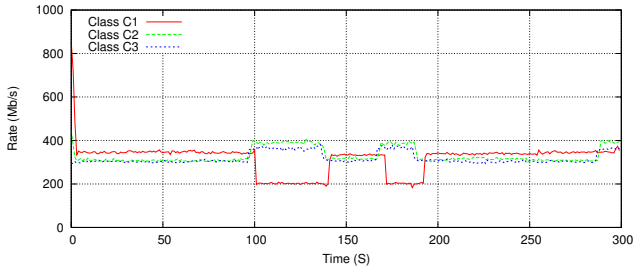

Fig. 3. Native System throughput

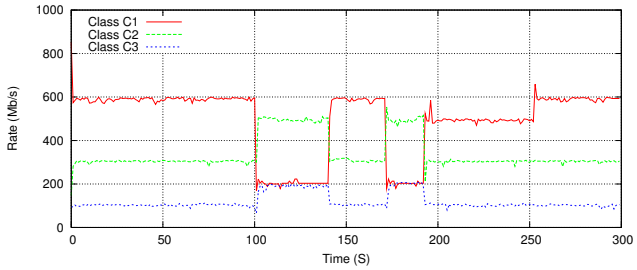

Fig. 4. System with DBA throughput

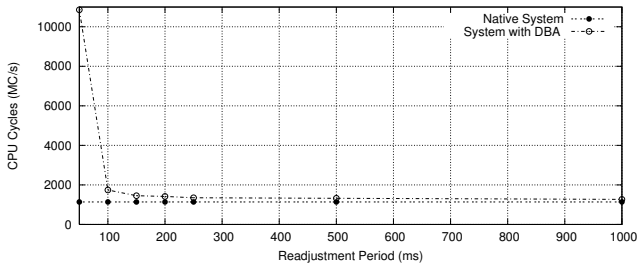

Fig. 5. CPU consumption

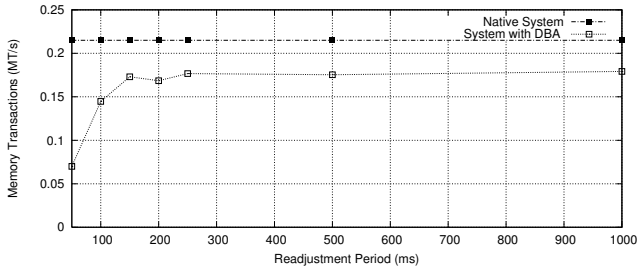

Fig. 6. Memory transactions gain

for a readjustment period of 100ms . Note that with the native system the bandwidth is globally equally shared between the three virtual machines. However, when three of the virtual machines transmit at the same rate, we can notice that MV1 achieves a slightly better throughput (350 Mb/s against 300 Mb/s for MV2 and MV1). This is due to the fact that MV1 was the first machine to be started which allows it to be scheduled first to execute jobs on the CPU. Furthermore, we note from figure 3 that after diminution in the transmission rate of one virtual machine (MV1 after 100s), the remaining bandwidth is shared between the other two virtual machines. Globally, the packet loss for the C1 class traffic is about 55.87% with the basic system.

From figure 4, with the DBA-system we can notice first that the most prioritized virtual machine is allowed to transmit at only 600 Mb/s although its generated traffic at 800 Mb/s. The DBA adjusted its transmission rate to B_1^{max} (during the first 100s). A decrease in the MV1 from 600 Mb/s to 200 Mb/s input rate incurs an increase in MV2 and MV3 throughput to 500 and 200 Mb/s respectively. Notice that although MV1 transmits at 200 Mb/s, MV2 and MV3 are prevented from enjoying the totality of the remaining bandwidth (800 Mb/s). In fact, since the DBA adjusted the B_1 to B_1^{min} (300 Mb/s), MV2 and MV3 can only share the remaining 700 Mb/s. We can conclude then that globally our algorithm respects the agreements on the minimum guaranteed bandwidth and the maximum allowed bandwidth. Furthermore, the C1 class loss rate has dropped to 16.56% with the DBA system (against 55.87% with native system).

2) Resources consumption: Our mechanism drops packets emitted beyond the allocated bandwidth in the netfront before being transferred to the netback. We expect then the system to consume less physical resources (CPU cycles and memory transactions). We profiled the resources (CPU cycles and memory transactions) usage using Xenoprof [13] with both the native system and DBA system and determined the impact of the readjustment period on the system throughput and physical resources usage. Figure 5 illustrates total CPU usage as a function of the readjustment period size. However, for memory transactions we only presented memory transactions achieved by the I/O part as a function of the readjustment period size too. We notice that for a period of less than 100 ms, our algorithm consumes much CPU, which will impact the system throughput, note the high packet loss rate for this period in Figure 7. For periods lasting more than 100 ms, our system CPU consumes as much CPU as the native system. Nevertheless, our system reduces memory transactions achieved by the I/O part involving transferring the packets from the netfront to the netback. Note that for a period of less than 100 ms, the system suffers from high packet loss rate. Packets are then dropped in the netfront which eliminates memory transactions that would have been necessary to transfer packets to the netback. Globally, and even for periods longer than 100 ms, the DBA allows reducing necessary memory transactions. Finally, we can conclude from figure 7 that our DBA-system notably reduces packet loss rate for periods beyond 100ms. Note also that the packet loss rate of 16,6% for MV1 traffic is relative to the scenario we considered, where MV1 transmits at rates higher than its maximum allowed bandwidth set in the agreement. Our system then totally respects this agreement while reducing achieved memory transactions. This leads to an optimal resource allocation among virtual machines and then higher system scalability [12].

VI. Conclusion

Virtualized clouds are definitely a compelling technology for both users and providers. They offer flexible service to the customer according to the pay-as-you-use model. Virtualization allows providers to optimize hardware through resource sharing and reuse and thus reduce hardware and power costs.

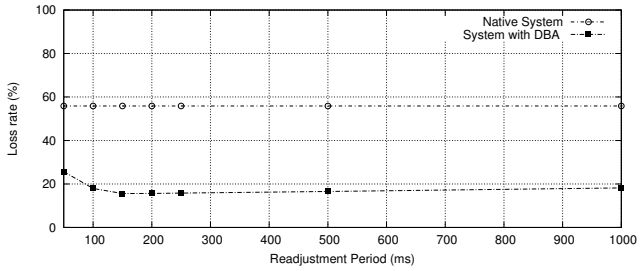

Fig. 7. Packet loss rate

In a virtualized data-center, several virtual machines share the same network device. Access to the NIC is handled by the driver domain and bandwidth is statically allocated to the different virtual machines. In this paper we have proposed and developed DBA, a dynamic bandwidth allocator for virtual machines. Virtual machines are classified into classes that are differentiated by the required bandwidth of transmitted flows. DBA guarantees to each virtual machine to transmit at the required bandwidth as agreed in the SLA. Remaining bandwidth is also shared between the concurrent virtual machines according to their priorities. DBA updates the allocation periodically in order to react as fast as possible to traffic changes. Furthermore, it optimizes physical resources usage (CPU and memory) through dropping packets beyond the allowed transmission bandwidth at the virtual machine instead of the driver domain. Thus it prevents transferring packets destined to be dropped through the I/O channel and then extra memory transactions and CPU cycles. Experimental evaluation of our module shows that DBA indeed respects the service level agreements and considerably reduces the packet loss. We intend next to extend our algorithm to establish SLA 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] 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.

[2] 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.

[3] 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.

[4] 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.

[5] M. Hasan Jamal, A. Qadeer, W. Mahmood, A. Waheed, J.J. Ding, Virtual Machine Scalability on Multi-Core Processors Based Servers for Cloud Computing Workloads, International Conference on Networking, Architecture and Storage, 2009.

[6] 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.

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

[8] M. Kesavan, A. Gavrilovska and K. Schwan, Differential virtual time (DVT): rethinking I/O service differentiation for virtual machines, Proceedings of the 1st ACM symposium on Cloud computing, 2010

[9] 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.

[10] A. Kertesz, G. Kecskermeti, and I. Brandic "An SLA-based Resource Virtualization Approach for On-demand Service Provision", International Workshop on Virtualization Technologies in Distributed Computing, Jue 2009, Barcelona, Spain.

[11] E. Kohler, R. Morris, B. Chen, J. Jahnotti, and M. F. Kasshoek, The click modular router, ACM Transactions on Computer Systems, vol. 18, no. 3, pp. 263-297, 2000.

[12] M. Bourguiba, K. Haddadou, and G. Pujolle, A Container-based Fast Bridge for Virtual Routers on Commodity Hardware, IEEE GlobeCom, 2010, Miami, USA.

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