Billing system CPU time on individual VM

Boris Teabe, Alain Tchana, Daniel Hagimont
Institut de Recherche en Informatique de Toulouse (IRIT)
Toulouse, France
e-mail: first.last@enseeiht.fr

***Abstract*—In virtualized cloud hosting centers, a virtual machine (VM) is generally allocated a fixed computing capacity. The virtualization system schedules the VMs and guarantees that each VM capacity is provided and respected. However, a significant amount of CPU time is consumed by the underlying virtualization system, which generally includes device drivers (mainly network and disk drivers). In today's virtualization systems, this CPU time consumed is difficult to monitor and it is not charged to VMs. Such a situation can have important consequences for both clients and provider: performance isolation and predictability for the former and resource management (and especially consolidation) for the latter. In this paper, we propose a virtualization system mechanism which allows estimating the CPU time used by the virtualization system on behalf of VMs. Subsequently, this CPU time is charged to VMs, thus removing the two previous side effects. This mechanism has been implemented in Xen. Its benefits have been evaluated using reference benchmarks.**

***Index Terms*—Cloud computing; Virtual machines; isolation; predictability; billing**

I. Introduction

A majority of cloud platforms implement the Infrastructure as a Service (IaaS) model where customers buy virtual machines (VM) with a set of reserved resources. This set of resources corresponds to a Service Level Agreement (SLA) which should be fully provided to customers [1], [2]. On their side, providers are interested in saving resources [21], [22] while guaranteeing customers SLA requirements. In this paper, we focus on CPU allocation to VMs. In the IaaS model, a VM is generally allocated a fixed CPU computing capacity. The underlying hosting system (hypervisor) schedules VMs and ensures that the allocated CPU capacity is provided and respected. The respect of the allocated capacity has two main motivations: (1) For the customer, performance isolation and predictability [8], [9], i.e. a VM performance should not be influenced by other VMs running on the same physical machine. (2) For the provider, resource management and cost reduction, i.e. a VM should not be allowed to consume resources that are not charged to the customer. However, we observe that in today's virtualization systems, the respect of the allocated CPU capacities is approximative [13], [14]. A significant amount of CPU is consumed by some components of the underlying hosting system (hereafter called system components). These components include device drivers (mainly network and disk drivers). Surprisingly, The CPU time consumed by the system components is not charged to VMs. This situation has an important impact on both VM performance isolation and resource management in the IaaS. In this paper, we propose a system extension which determines the CPU time consumed by the system components on behalf of individual VM in order to charge this time to each VM. This extension relies on three main mechanisms.

1) a counting of device drivers I/O requests. Each VM has a counter representing its number of I/O request.
2) a calibration of each type of I/O request. This calibration computes for each I/O request type, the average CPU time needed by the system components to handle the request.
3) an integration of I/O CPU times in the VM scheduler. The scheduler of the virtualization system integrates the CPU time consumed by I/O requests in the computation of the CPU time used by VMs.

Thus, the CPU time used by the system component is charged to VMs, i.e. a VM is not given a blank cheque regarding its CPU consumption through system components. Consequently, we significantly reduces performance disturbance arising from the competition from collocated VMs. We implemented a prototype in the Xen virtualization system. We evaluated this prototype with various workloads. The evaluations show that performance isolation and resource management can be significantly impacted with the native Xen system. Furthermore, our extension ensures performance isolation for the customer, and prevents resource leeks for the provider, while having a negligible overhead. In summary, the contributions of this paper are:

- precise measurement of CPU time used by system components (e.g. driver domain) on behalf of VMs;
- smart charging of this CPU time to VMs vCPU;
- implementation of a multi-core prototype within Xen system;
- intensive experimentation with real workloads.

The rest of the article is structured as follows: section II presents the motivations of this work. Section III presents our contributions. An evaluation is reported in Section IV. The latter is followed by a review of related work in Section V, we present our conclusions in Section VI.

II. Motivations

The contributions presented in this paper have been implemented in the Xen [24] system, which is the most popular open source virtualization system, used by Amazon EC2. In the Xen para-virtualized system, the real driver of each I/O device resides within a particular system component called "*driver domain*" (DD). The DD conducts I/O operations on

behalf of VMs which run a fake driver called *frontend*. The frontend communicates with the real driver via a *backend* module located in the DD. The *backend* allows multiplexing the real devices. This I/O virtualization architecture, generally called the "*split-driver model*", is used by the majority of virtualization systems. In this model, the hardware driver is not modified and can be run in a separate VM. From the above description, we can deduce that I/O requests are handled by the DD which consumes a significant CPU time. Current virtualization system schedulers do not integrate this CPU time when computing VM's processing time. This situation can be problematic for both cloud clients, and cloud provider namely:

- If the CPU capacity of the DD is limited, performance isolation can be compromised because a VM performance is influenced by other VMs which shared the DD resources.
- Otherwise, if the CPU capacity of the DD is unlimited, the resource management system (especially consolidation) can be affected.

III. Contributions

In this paper, we propose a solution which overcomes the problem identified in the previous section. Although the solution is relatively easy to label, its implementation should face the following challenges:

- Accuracy. How to accurately count the CPU time used by the DD for each VM knowing that the DD is shared among several tenants?
- Overhead. The processing time needed to run the solution must be negligible.
- Intrusion. The solution should require as few modifications as possible within the guest OS.

A. General approach

We propose an implementation which takes into account all the challenges listed above. This implementation mainly relies on calibration. It is summarized as follows. First of all, the provider measures the CPU time needed by the DD to handle each I/O request type (See section III-B). This is done once. These measurements are made available to the scheduler. The DD is modified in order to count the number of I/O request handled per type for each VM. This modification is located in the backend, which is the ideal place to track all I/O requests. The DD periodically sends the collected information to the scheduler. Based on the received information and the calibration results, the scheduler computes the CPU time used by the DD on behalf of each VM. This computed CPU time is then charged to the VMs. This is done by balancing the computed CPU time among all VM's vCPUs in order to avoid the penalization of a single vCPU. The next section gives more details about the calibration.

B. Calibration

Calibration consists in estimating the CPU time needed by the DD to process each I/O request type. To this end, we place sensors both at the entry and the exit of each component involved. We implemented a set of micro-benchmark[1] applications to provide an accurate calibration. For each I/O device type, we consider all factors that could impact the CPU time needed by the DD for their processing. The most important factors are:

- the configurations in the DD. For each I/O device, Xen provides several ways to configure how the device driver interacts with the backend.
- the type of the I/O request. The path followed by I/O requests within the DD depends on their type.
- the size of the request. I/O requests are from different sizes.

Network calibration: We implement in C a sender/receiver application based on UDP to calibrate the cost of handling a network operation in the DD. The sender always sends the same frame to the receiver. Both the sender and the receiver are within the same LAN. Xen provides three possible network configurations in the DD :bridging, routing and NATing. Bridging is the most used configuration. The path, between the device driver and the backend, taken by frames through the Linux network stack differs for each configuration. Routing and NATing use a very similar path while it is much different for bridging. Bridging is used in the rest of the article, unless otherwise specified. Fig. 1(a) presents the calibration measurements for our experimental environment. We can observe that the cost of handling a network packet by the DD varies with the size and the direction (receiving or sending).

Disk calibration: We use the Linux *dd* command to calibrate disk operations. As for the network, Xen provides different ways to configure how a VM disk is managed in the DD. These are: *tap*, *qdisk*, and *phy*. The latter is the most used. The configuration mode influences the processing time used by the DD. We use *phy* in the rest of the article. Disk operations are read and write. Unlike the network, they are always initiated by the VM. Fig. 1(b) shows the calibration results for our experimental environment.

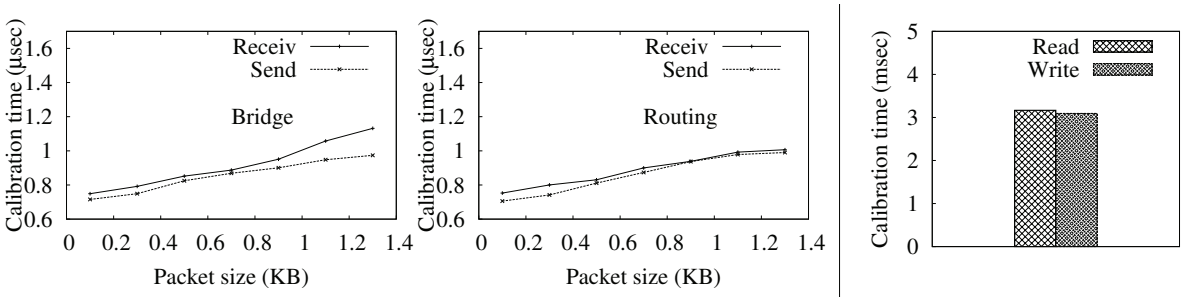
IV. Evaluations

We have implemented our solution in both the DD's kernel (version 3.13.11.7) and the Xen hypervisor (version 4.2.0) precisely in the Xen credit Scheduler. This section presents the evaluation results of this implementation. The experiments were realized with realistic applications. We evaluate the following aspects: (1) the overhead of the solution, (2) its efficiency regarding performance predictability.

A. Experimental environment

The experiments were performed in our private cluster. Its consists of HP machines with Intel Core i7-3770 CPUs and 8 Gbytes of memory. PMs are linked to each other with a gigabyte switch. The dom0 is used as the DD. Its CPU computing capacity is configure to 30% of the processor. VMs

[1]Note that, once both the hypervisor and the DD are patched, a calibration round does not take a lot of time (about 5 minutes).



(a) Network calibration (b) Disk calibration

Fig. 1: Network calibration (a) and Disk calibration (b)

are configured with a single vCPU (pinned to a dedicated processor, different from the one used by the DD).

B. Overhead and Accuracy

Our solution introduces a very negligible overhead (near zero) because we use existing mechanisms to implement the solution. This allows us to avoid any additional cost. We experimented our solution with two micro-benchmarks: (1) a web application based on wordpress [26] for the network evaluation, and (2) Linux *dd* command for the disk evaluation. The computing capacity of the VM is set to 30% of the processor when running the network benchmark and 15% for the disk benchmark. The experiments were realized in two contexts: with our solution and with the native Xen system. We show the ability of our approach to ensure that the aggregated CPU consumed by a client VM remains within its booked CPU capacity, which is not the case with the native Xen system. This allows us to guarantee performance predictability. The leftmost curve in Fig. 2 presents the results of these experiments. We can see that using our solution, the aggregated CPU load of the client VM is about the value of its booked CPU capacity (30 or 15). The margin of error is very negligible. The two rightmost curves in Fig. 2 focus on the case of the network evaluation. They present performance predictability results. The second curve highlights the issue of performance unpredictability in the Xen system when two VMs share the DD. The throughput of the VM goes from 1200req/sec when it is alone to 800req/sec when it is colocated. The third curve shows the results of the same experiment when our solution is used. We can see that the VM keeps the same performance, about 800req/sec. The latter represents the throughput the VM should provide for this booked credit. Indeed, our implementation avoids the saturation of the DD since its allocated credit was enough for handling VMs traffics when their aggregated CPU load stay within their booked credit.

V. Related works

I/O virtualization has received a fair amount of attention recently. [3], [4], [5], [6], [7] have focused on boosting I/O intensive VMs. Despite all this, several studies [8], [9] have pointed performance unpredictability in the cloud because of VMs competition on shared resources. Studies in this domain can be classified into two categories. The first category includes research [18] at a micro-architectural level (e.g. cache contention effects). The main approach they advocate consists in placing VMs intelligently so that compete workloads are avoided atop of the same machine. Researches [14], [25], [11] of the second category have addressed the problem at the software level. They propose to use fair-share bandwidth allocation where a minimum bandwidth is guaranteed to each tenant [12]. All these works do not accurately guarantee performance predictability, even less charged DD CPU time to clients VMs as we do in this paper. In a virtualized system, shared resources are not only hardware. As we have shown, some software components such as the DD are shared and should be considered. Thus focusing on micro-architectural aspects as done by studies in the first category is not sufficient. By studying shared software components, our work is complementary to those in the first category. Concerning the second category, the approach could be efficient if a given bandwidth always leads to the same system time in the DD but this is not true.

To the best of our knowledge, [20] is the only work close to what we propose. They propose to use Xenmon [23][2] for limiting bandwidth per VM (ShareGuard) and also to realize a scheduler which takes into account CPU time used by the DD on behalf of VMs (SEDF-DC). In comparison to our solution, [20] presents the following weaknesses. (1) [20] only studies network devices, whereas we have shown that disk operations can generate a significant load in the DD. (2) SEDF-DC scheduler is limited to mono-processor machines, while today's machines are mostly multi-processors. (3) ShareGuard is very intrusive since it drops VM's network packets which CPU load within the DD is above the configured capacity. In addition, dropping packets does not remove DD's activity since it needs to analyze packets before dropping them. Our solution is very smart in that it just imposes a CPU time consumption to the VM. (4) SEDF-DC and ShareGuard use XenMon which needs to be constantly activated, thus consuming a non negligible system time. Finally, (5) [20] does not consider all the factors which impacts the CPU time used by the DD (e.g. packet size).

[2] a monitoring tool for determining the CPU load generated by a VM within the DD

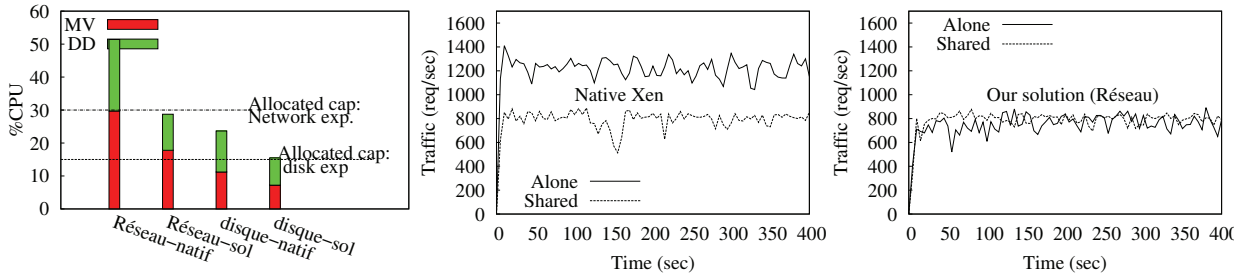


Fig. 2: Accuracy of our solution using micro-benchmark

VI. Conclusion

In todays virtualization systems, the CPU time consumed by device drivers in the underlying hosting system is not charged on VM CPU capacities. This situation can have significant impact on VM performance isolation and on resource management (consolidation) in a IaaS. Therefore, we proposed a system extension which allows to estimate CPU time consumed by the virtualization system components on behalf of individual VMs and to charge it to VMs. This extension was implemented in the Xen system and evaluated in our private cluster. The results show that this CPU time consumed by virtualization system components is significant, and that according to the provider strategy, it can lead to unpredictable performance for the client. Our solution allows to precisely charge system time on VMs capacities, following the *pay as you go* philosophy.

Acknowledgements

The authors sincerely thank the anonymous reviewers for their feedback on earlier versions of this manuscript. This work benefited from the support of the French "Fonds national pour la Société Numérique" (FSN) through the OpenCloudware project.

References

[1] Lionel Eyraud-Dubois and Hubert Larcheveque, "Optimizing Resource allocation while handling SLA violations in Cloud Computing platforms", IPDPS 2013.

[2] Douglas B. Terry, Vijayan Prabhakaran, Ramakrishna Kotla, Mahesh Balakrishnan, Marcos K. Aguilera, and Hussam Abu-Libdeh, "Consistency-Based Service Level Agreements for Cloud Storage", SOSP 2013.

[3] Xiao Ling, Hai Jin, Shadi Ibrahim, Wenzhi Cao, and Song Wu, 'Efficient Disk I/O Scheduling with QoS Guarantee for Xen-based Hosting Platforms", CCGRID 2012.

[4] Dimitris Aragiorgis, Anastassios Nanos, and Nectarios Koziris, 'Coexisting scheduling policies boosting I/O Virtual Machines", Euro-Par 2011.

[5] Yaozu Dong, Zhao Yu, and Greg Rose, 'SR-IOV networking in Xen: architecture, design and implementation", Usenix WIOV 2008.

[6] Hwanju Kim, Hyeontaek Lim, Jinkyu Jeong, Heeseung Jo, and Joonwon Lee, 'Task-aware virtual machine scheduling for I/O performance", VEE 2009.

[7] Denghui Liu, Jinli Cao, and Jie Cao, 'FEAS: a full-time event aware scheduler for improving responsiveness of virtual machines", ACSC 2012.

[8] Has Amazon EC2 become over subscribed?http://alan.blog-city.com/has_amazon_ec2_become_over_subscribed.htm.

[9] Xing Pu, Ling Liu, Yiduo Mei, Sankaran Sivathanu, Younggyun Koh, Calton Pu, and Yuanda Cao, 'Who Is Your Neighbor: Net I/O Performance Interference in Virtualized Clouds", TRANSACTIONS ON SERVICES COMPUTING 2013.

[10] George Kousiouris, Tommaso Cucinotta, and Theodora Varvarigou, "The effects of scheduling, workload type and consolidation scenarios on virtual machine performance and their prediction through optimized artificial neural networks," Journal of Systems and Software 84(8) 2011.

[11] Hitesh Ballani, Keon Jang, Thomas Karagiannis, Changhoon Kim, Dinan Gunawardena, and Greg O'Shea, 'Chatty Tenants and the Cloud Network Sharing Problem," NSDI 2013.

[12] Amit Kumar, Rajeev Rastogi, Avi Silberschatz, and Bulent Yenerc, "Algorithms for provisioning virtual private networks in the hose model', SIGCOMM 2001.

[13] Daniel Hagimont, Christine Mayap, Laurent Broto, Alain Tchana, and Noel De Palma, "DVFS aware CPU credit enforcement in a virtualized," Middleware 2013.

[14] Jorg Schad, Jens Dittrich, and Jorge-Arnulfo Quiane-Ruiz, 'Runtime Measurements in the Cloud: Observing, Analyzing, and Reducing Variance", VLDB 2010.

[15] Qun Huang and Patrick P.C. Lee, 'An Experimental Study of Cascading Performance Interference in a Virtualized Environment," SIGMETRICS Perform. Eval. Rev. 2013.

[16] 5 Lessons We've Learned Using AWS: http://techblog.netflix.com/2010/12/5-lessons-weve-learned-using-aws.html

[17] http://www.datadoghq.com/wp-content/uploads/2013/07/top_5_aws_ec2_performance_problems_ebook.pdf

[18] George Kousiouris, Tommaso Cucinotta, and Theodora Varvarigou, "The effects of scheduling, workload type and consolidation scenarios on virtual machine performance and their prediction through optimized artificial neural networks," Journal of Systems and Software 84(8) 2011.

[19] Joydeep Mukherjee, Diwakar Krishnamurthy, Jerry Rolia, and Chris Hyser, "Resource contention detection and management for consolidated workloads," IM 2013.

[20] Diwaker Gupta, Ludmila Cherkasova, Rob Gardner, and Amin Vahdat, "Enforcing performance isolation across virtual machines in Xen', Middleware 2006.

[21] Nedeljko Vasic, Dejan Novakovic, Svetozar Miucin, Dejan Kostic, and Ricardo Bianchini, "DejaVu: Accelerating Resource Allocation in Virtualized Environments", ASPLOS 2012.

[22] Hui Lv, Yaozu Dong, Jiangang Duan, and Kevin Tian, "Virtualization challenges: a view from server consolidation perspective," VEE 2012.

[23] Diwaker Gupta and Ludmila Cherkasova, "XenMon: QoS Monitoring and Performance Profiling Tool', HPL-2005-187 2005.

[24] Xen: http://www.xenproject.org/. bitemxenmon Diwaker Gupta and Ludmila Cherkasova, "XenMon: QoS Monitoring and Performance Profiling Tool', HPL-2005-187 2005.

[25] Lucian Popa, Arvind Krishnamurthy, Sylvia Ratnasamy, and Ion Stoica, 'FairCloud: Sharing the Network in Cloud Computing", HotNets-X 2011.

[26] "Wordpress", https://fr.wordpress.org/, visited on febuary 2016