On the Performance Isolation Across Virtual Network Adapters in Xen

Blazej Adamczyk, Andrzej Chydzinski
Institute of Computer Sciences
Silesian University of Technology
44-100 Gliwice, Poland
{*blazej.adamczyk,andrzej.chydzinski*}*@polsl.pl*

***Abstract*—Virtualization has recently become a very popular technique for utilizing hardware capabilities and lowering infrastructure and maintenance costs. However, making several virtual machines share the same resources can potentially introduce performance isolation problems. Depending on the application, proper quality of service and the performance isolation may present critical requirements for the system.**
In this paper, we focus on network performance isolation among virtual adapters in Xen. We present several experiments demonstrating how activity of one virtual machine can affect the network performance of any other. Additionally, we examine the network I/O scheduler in Xen to see if it is fair, predictable and configurable enough. Finally, we propose an idea on how to modify Xen back-end drivers to improve the network performance isolation.

***Keywords*-performance isolation; Xen; virtualization; network scheduler.**

I. INTRODUCTION

The increasing number of different IT services are making the virtualization idea a very important aspect of computer science. Virtual Machine Monitors (VMMs) bring about the dynamic resource allocation and enable full utilization even of the most powerful servers, while still maintaining good fault isolation between virtual machines (VMs). However, the services provided over the network may require a certain quality, which is not easy to ensure in a virtualized environment. Several VMs can share the same physical network interface as well as other hardware (processor, memory etc.) what likely makes one VM affect other VMs performance. Therefore, the *performance isolation* is crucial in case of some applications and has to be carefully verified.

In this paper, we focus on Xen VMM, [1], which is one of the most popular virtualization platforms and an Open Source project. Firstly, we present a study of the network performance isolation between Xen virtual machines. Different test scenarios allowed us to identify several problems. Secondly, we carefully analyze the Xen CPU scheduler and the network IO scheduler to find out their possible source and resolution method.

The remaining part of the paper is structured as follows. In Section III, Xen general architecture is overviewed. Then, a description of the Xen schedulers is presented in Section IV. Section V describes the testing environment and its parameterizations. The results and discussion on them are contained in Section VI. Finally, an idea of improving the network performance isolation in Xen is presented in Section VII. Conclusions are gathered in Section VIII.

II. STATE OF THE ART

This study verifies that there are problems related with performance and isolation of virtualized network resources. Several previous studies (see [9], [10], [11], [12], [13], [14], [15]) focus on analysis of the performance of IO operations and some of them present partial solutions. Unfortunately, these studies do not examine isolation and manageability in the field of resource sharing in considered virtualization platforms. In [7], however, the authors tried to approach the performance isolation problem focusing on all kinds of resources. Unfortunately, this study was performed on older version of Xen with an older CPU scheduler implementation. They assumed that the main source of the problem is connected with CPU assignment and scheduling. We think however, that to achieve good performance isolation across virtual network adapters the proposed CPU scheduler improvement could be used but is not sufficient. We present that even on a low CPU utilization the problem is still noticeable and is related with network scheduler itself. We have verified that applying a modified for virtualization Weighted Round Robin (WRR) network scheduler improves the performance isolation and provides better control over virtual network devices.

III. XEN VMM

Different virtualization environments have been developed throughout the years. Xen, due to its unique architecture (Fig. 1), is one of the leading solutions. 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. In contrast to other VMMs, the virtualization of all other resources is moved outside the hypervisor. Such original approach has the following advantages:

- Device drivers are not limited to the hypervisor operating system because they are installed on a virtual machine (any OS),

 ISBN: 978-1-61208-153-3

- Device drivers, as the most vulnerable software, are isolated from the hypervisor, significantly increasing the stability,
- Distributed virtualization of resources allows creation of several driver domains, eliminating the single point of failure,
- Small hypervisor operating system is much more reliable, efficient and stable.

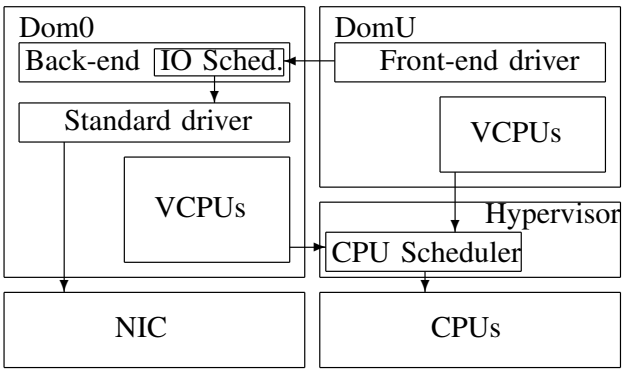

Figure 1. Xen architecture (Dom0 - Xen primary virtual machine, DomU - other Xen virtual machine, Hypervisor - main Xen operating system running directly on hardware, NIC - Network Interface Card, VCPU - virtual CPU)

There are two main virtualization methods. The first one allows to run any kind of OS and emulates all the necessary hardware to create an impression that the guest system is running on a physical machine. Second approach is to run a modified guest operating system, which is ”aware” of being virtualized. The latter, called *paravirtualization*, is much more efficient, but limited to some operating systems only. Xen provides both methods, but performs much better in the paravirtualization mode, which will be the only method used further in this paper.

To make the IO operations as fast as possible, Xen introduced also paravirtualized device drivers. Each guest domain (Xen VMs are also called ”domains”) has the front-end drivers installed. Such drivers, provided with Xen, are communicating with the back-end drivers running on a special driver domain (Dom0 in Fig. 1). All requests addressed to a certain hardware are first scheduled and processed by the back-end driver, then are sent to the standard device driver inside the driver domain and finally reach the hardware. Thanks to Xen internal page-flipping mechanism called XenBus, (see [2], [3]), such solution is much more efficient than the standard emulation technique.

IV. XEN SCHEDULERS

The main goal of this study is to examine the network performance isolation across Xen guest domains. It means to check, if activity of one virtual machine influences the network performance of any other. The resulting knowledge is of great importance from the perspective of many network-related applications.

There are two elements in Xen, which may influence such isolation, namely the CPU scheduler and the network IO scheduler [6]. In the following two sections a description of these two schedulers is given.

A. CPU Scheduler

The fundamental part of each multitasking operating system is the CPU scheduler. Its aim is to create an impression that all running processes are executed in parallel. Typically, there are much more processes than available physical CPUs and the processes have to share CPU time. The scheduler is responsible for this division.

Inside Xen VMM, the hypervisor is the main operating system running on the physical machine. It is responsible for scheduling physical CPU time among virtual machines. To make the process easier the term *virtual CPU (VCPU)* is introduced. Every VM in Xen can have multiple virtual processors. Also, every domain is running operating system with another scheduler, which divides a VCPU time among processes running inside the guest operating system. The hypervisor on the other hand, schedules the physical CPU time among VCPUs.

The newest version of Xen uses the *credit scheduler* [4] [5]. It assigns two parameters for each domain - *weight* and *cap*. The weight defines how much CPU time a domain gets comparing to other virtual machines. The cap parameter is optional and describes the maximum amount of CPU a domain can consume. Using this two parameters the number of credits can be calculated. As a VCPU runs, it consumes credits. While VCPU has existing credits, its priority is called *under* and it gets CPU time normally. When there are no credits left, the priority changes to *over*. Each physical CPU maintains its own local VCPU queue. In the first place, the VCPU tasks with priority *under* from the local queue are executed. Then, if there are no VCPUs with priority *under*, the scheduler looks for such tasks in other CPU queues. If there are no tasks with priority *under*, the tasks with priority *over* from the local queue are executed. The credit scheduler in Xen can by summarized in the following algorithm and diagram (Fig. 2):

1) Process preemption - the scheduler takes control over CPU.
2) Last taken VCPU inserted back into the local queue according to its credits number.
3) Have the highest priority VCPU from the local queue used all its credits?
   - No: Highest priority VCPU taken from the local queue.
   - Yes: SMP Load Balancing - highest priority VCPU taken from other CPU queues.
4) Switching context to the currently taken VCPU - the VCPU takes control over CPU.

 ISBN: 978-1-61208-153-3

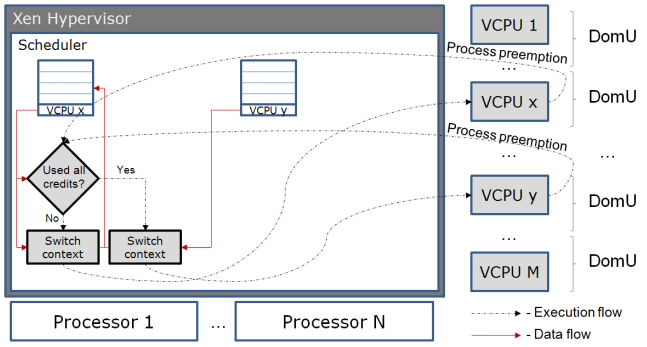

Figure 2. Xen CPU scheduler

Considering this CPU scheduler in the context of the network performance isolation, it is worth noticing that the scheduler operates on virtual CPUs only, so it should not have a strong impact on IO performance. However, it may happen that one misbehaving VM will slow down the total responsiveness and performance of other domains. Also, as it was presented in [7], the Xen CPU scheduler does not take into account the amount of CPU consumed by the driver domain on behalf of other VM. This may also have an impact on the network performance isolation, as some domains may use more CPU time than they are allowed. Furthermore, a different type of IO request (e.g., more demanding, like disk driver requests) can potentially slow down the driver domain and affect the network performance of other VMs.

B. Network IO scheduler

Looking at Xen architecture and analyzing its source code from the network performance isolation point of view, one can easily note that the most interesting part is the back-end network driver, called Netback. It contains another scheduler, responsible for gathering all IO requests sent to a certain physical network adapter. This network scheduler is not a complex mechanism and probably can be improved. Its only configuration parameter is the maximum rate (parameter *rate*) - in fact it can be perceived as the credits number in the scheduler. The administrator can specify only the maximal throughput achieved by a certain virtual network adapter. Unfortunately, there is no way to prioritize and control the quality of service in more details.

The scheduler itself counts the amount of data sent/received in given periods. If *rate* has been reached, it sets a callback to process the request in next periods. Such solution is efficient, but does not guarantee any fair share or quality. In fact, a misbehaving VM can theoretically flood driver domain with requests.

V. Experimental setup

To perform the tests, we installed Linux Gentoo with Xen 4.0.0 on Intel Quad Core 2 (2.83GHz), 2GB RAM, with hardware virtualization support. Two guest domains, each having 1 VCPU and 1GB of RAM, were created. Although there were separate physical CPU available for each VM, both VCPUs were pinned to the same physical CPU. Such configuration was used in order to check the influence of the CPU scheduler on the network performance. All network measurements were taken using *iperf* application. The UDP protocol transferring datagrams of 1500B to an external host over 100Mb link was used. We used the 100Mb link instead of 1Gb to present that the isolation problems are still present without a heavy CPU utilization. Only outgoing traffic was measured, as this was our main point of interest. The testing environment is presented in Fig. 3.

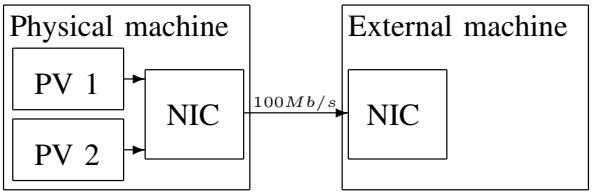

Figure 3. Testbed configuration. (PV1, PV2 - Xen paravirtualized machines, NIC - Network Card Interface)

VI. Results

In the first experiment, we observed how activity of one VM can affect the performance of another, when both VMs are configured with the same *rate* parameter. Four values of *rate* were used in different test runs: $25Mb/s$, $30Mb/s$, $35Mb/s$ and $40Mb/s$. In every run one machine started its transfer at the very beginning and the other started after 5s of delay. For every *rate* value, the experiment was repeated 10 times and the 0.95 confidence intervals were derived. The results are presented in Fig. 4.

Firstly, we can see that the actual rate is always a little smaller than *rate* parameter. As for the performance isolation, it is not too bad for low values of *rate*. However, with growing *rate*, the confidence intervals are getting larger and larger - in sample runs we can observe stronger variations of the throughput achieved by each VM. For the value of *rate* equal to 35Mb/s, the performance isolation becomes rather weak (although only about 60 percent of the total bandwidth is consumed).

Thus the only way to achieve a good isolation is to limit virtual adapters by far, which is not a satisfactory solution. Also, it is worth mentioning that having only the upper limit parameter is not enough in many cases. It would be much better to have any means to prioritize certain virtual adapter or even to have a minimum rate parameter and a scheduler satisfying these requirements.

In the second experiment, different *rate* values per each VM were used. Fig. 5 shows results for $rate = 30Mb/s$ in one VM, and $rate = 40Mb/s$ in another. The isolation

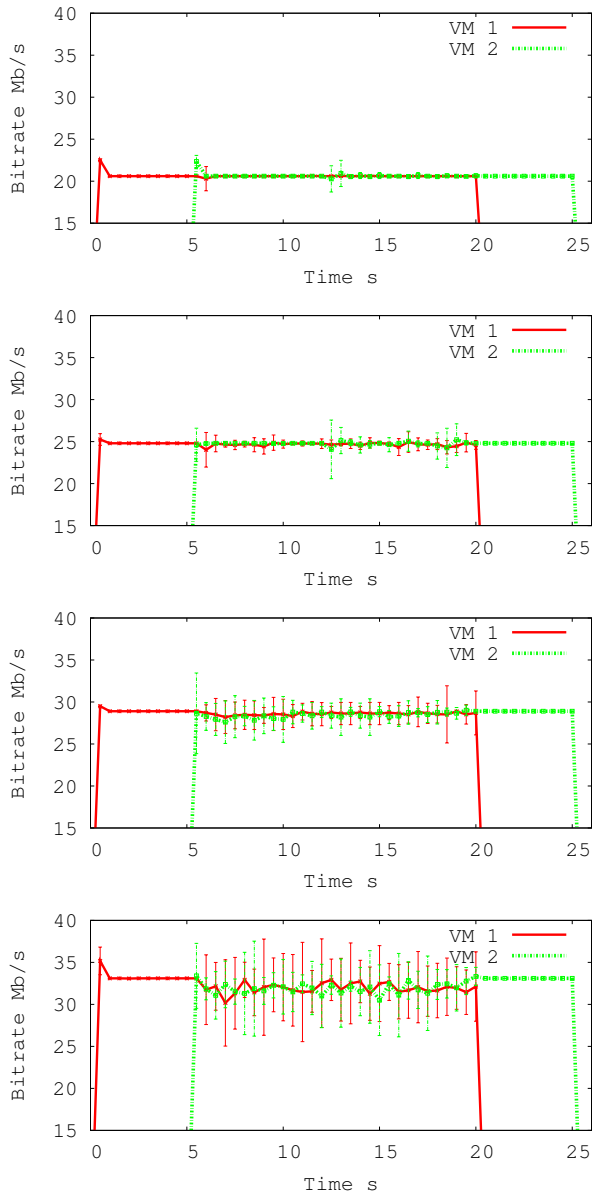

Figure 4. The throughput per VM for different values of *rate* parameter, namely for $25Mb/s$, $30Mb/s$, $35Mb/s$ and $40Mb/s$, counting from the top.

problem still remains but, what is worth noticing, both VMs affects each other similarly.

In the presented two experiments the performance isolation problem was either mild or moderate, depending on the configuration. In the following two experiments, we will demonstrate more severe performance isolation issues.

In the third experiment, we verified how Xen divides available bandwidth among two VMs when the maximal rate is not set. A sample path of the throughput achieved by each VM in time is presented in Fig. 6. Surprisingly, sometimes one virtual machine gets the total throughput and the other's

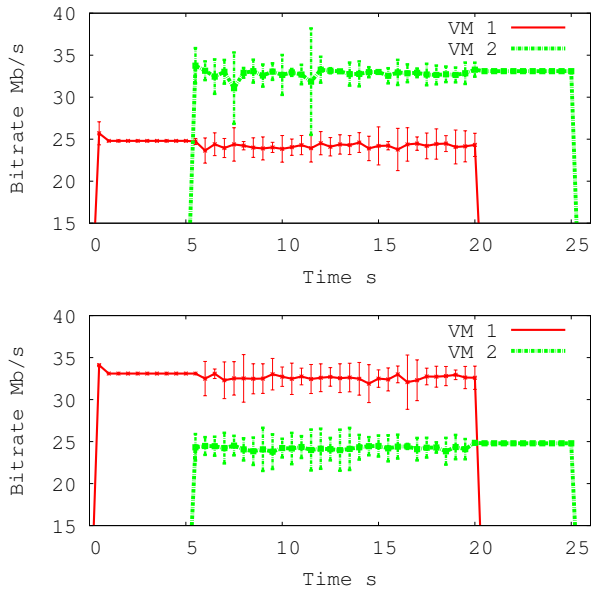

Figure 5. Total throughput per VM for different values of *rate* parameter ($30Mb/s$ and $40Mb/s$.

throughput decreases to 0. Moreover, there are long periods when one VM dominates the other by far. Therefore, we have in fact no performance isolation at all in this case.

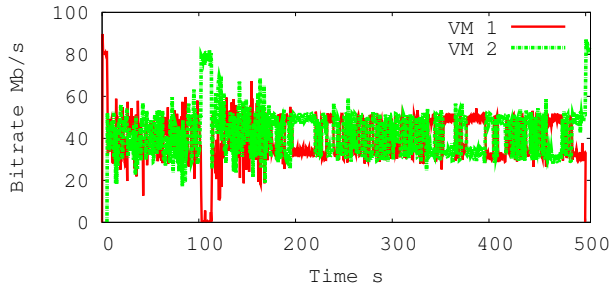

Figure 6. Sample throughput processes in time for two separate VMs without limits

In the fourth set of tests, we wanted to verify if a very abusive virtual machine can take more bandwidth then others. This time we wanted to check the performance isolation of the network IO scheduler only, therefore we pinned one physical CPU to each VM.

In the first test, one domain was trying to transfer data over one connection using full available speed, while the second domain was using two connections, both of them trying to achieve full available speed. In the next test, the second domain was using three connections at full available speed.

The results are presented in Fig. 7. As it can be observed, the more abusive domain is, the better throughput it achieves. Naturally, if the rate parameter had been set, the overactive

 ISBN: 978-1-61208-153-3

domain would never have crossed the maximum rate. In the lower ranges however, the problem remains.

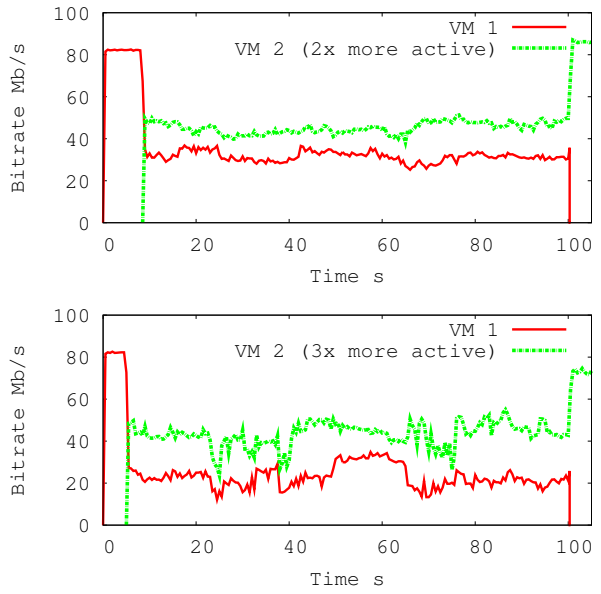

Figure 7. Bandwidth division with one overactive VM.

In the last experiment, we wanted to check if non-network IO requests can influence the network performance isolation of another domain. During the experiment one VM was constantly sending datagrams at full speed, while the second VM was performing some extensive disk operations (*fio* tool was used for this purpose). The results are presented in Fig. 8; t_0 and t_1 are points in time when the extensive disk operations were initiated and finished, respectively.

We can see that other IO request can also have a strong impact on the network performance. This is probably caused by driver domain not being able to process all the IO requests. Block device access is being handled by separate block device back-end drivers. Disk operations are much more demanding in the driver domain than the Netback drivers.

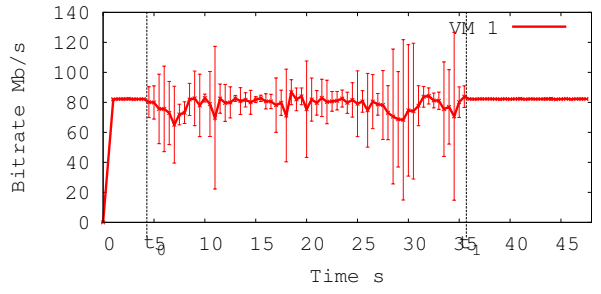

Figure 8. Disk IO influence on network performance. (t_0 - disk IO start, t_1 - disk IO finish)

VII. Improvement idea

After detailed analysis of the problem, we have gathered some ideas on how to modify Xen to improve the network performance isolation. Currently, in the driver domain several Netback kernel threads can be running, depending on the number of VCPUs. Furthermore, several virtual network adapters are mapped with one Netback kernel thread dynamically and this single Netback thread schedules the work using a simple round-robin algorithm, additionally taking into account *rate* parameter (omitting adapters, which used up all their bandwidth in the current period). Our idea is to introduce two additional parameters for every virtual adapter, namely *priority* and *min rate*. To implement the former, it would be necessary to change the round-robin mechanism to a more advanced priority based queue. Of course, we have to remember that the algorithm should not increase significantly the time complexity. The *min rate* parameter could use the same prioritization mechanism, assigning higher priorities to interfaces, which have not yet achieved the minimum rate. Depending on the results, it may be also necessary to introduce a user level application for maintaining the niceness level of each Netback thread inside the driver domain, according to actual needs.

A. Prioritization

The very first step to solve all the aforementioned problems is to introduce a prioritization mechanism into Xen's *Netback* driver. To achieve such functionality we implemented the simple *Weighted Round Robin* algorithm (see [8]). In virtualized environment where a packet passes several virtual adapters before it reaches the actual real interface and each interface has its own input buffer the WRR scheduler has to be modified to guarantee that the scheduled packets will not be dropped before they reach the wire. Dynamic and real-time priority assignment in this scheduler was created by additional Linux kernel *sysctl* parameters, i.e., *prioritize, priorities* and *delay*. The first parameter defines whether to use the *WRR* scheduler or not. Second parameter is an array of the actual priority values for each virtual adapter and the *delay* is used to define the inactivity period (i.e. a period of time after, which the *vif* is treated as inactive).

Each *vif* has a separate queue of data to transfer and a *priority*. The latter corresponds to the weight in the implemented *WRR* algorithm. Total bandwidth available at the physical link is shared proportionally between all active virtual interfaces according to their weights.

To test the prioritization we performed simple experiment where two VMs transmit data to an external host. In the meantime the priorities were changed every second. At the begging VM 1 had much bigger priority, in the end VM 2 was favored in the same proportion (i.e., $30/1$). The results are presented in Figure 9.

 ISBN: 978-1-61208-153-3

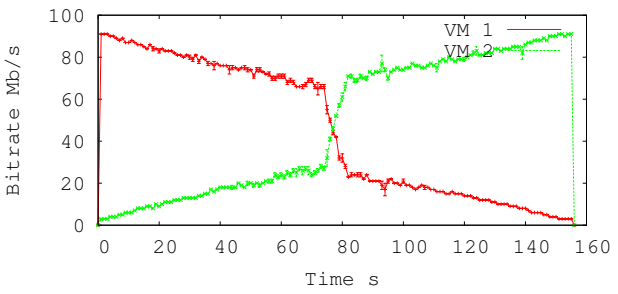

Figure 9. Results of the improved scheduler for changing priorities of each VM.

B. Further improvements

Prioritization brings a lot of new possibilities and improves the performance isolation by far. Nevertheless, in high CPU utilization scenarios it is not sufficient. Much more complicated mechanisms have to be created. Virtualization makes the problem very complex, as three different schedulers may affect the isolation: *CPU Scheduler, Domain 0 VCPU Scheduler* and *Netback IO Scheduler*. To achieve best results it might be necessary to *synchronize* all schedulers. Thus, partial solutions providing the *minimal rate* parameter for given virtual interface may prove very valuable. Finally, a modification proposed in [7] may also help to increase the performance isolation taking the aggregate CPU consumption into consideration. All these are subjects of our future study.

VIII. Conclusion

Xen is a powerful and stable virtualization platform, what accompanied with its Open Source formula makes it one of the most interesting VMMs, especially for research purposes. However, when the network virtualization is considered, the weak point of Xen is its lack of proper performance isolation. We demonstrated this using five sets of tests. The problems with isolation are caused by several factors mostly connected with CPU and IO schedulers. We proposed the *Netback* driver modification using *WRR* algorithm to provide prioritization. We have also briefly presented an idea for future improvements.

IX. Acknowledgment

This work is partially funded by the European Union, European Funds 2007-2013, under contract number POIG.01.01.02-00-045/09-00 ”Future Internet Engineering”.

References

[1] 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. In: Proc.of the 19th ACM SOSP, New York, 2003, Vol. 37, pp. 164–177.

[2] Y. Xia, Y. Niu, Y. Zheng, N. Jia, C. Yang, and X. Cheng: Analysis and Enhancement for Interactive-Oriented Virtual Machine Scheduling, IEEE/IFIP International Conference on Embedded and Ubiquitous Computing, 2008, Vol. 2, pp. 393–398.

[3] Xen Wiki, http://wiki.xensource.com/xenwiki/XenBus, 29-06-2011.

[4] L. Cherkasova, D. Gupta, and A. Vahdat: Comparison of the three CPU schedulers in Xen, SIGMETRICS Performance Evaluation Review; September 2007, Vol. 35, No. 2., pp. 42-51.

[5] G. W. Dunlap: Scheduler development update, Xen Summit North America 2010, http://www.xen.org/files/xensummit_intel09/George_Dunlap.pdf, 29-06-2011.

[6] J. Matthews, E.M. Dow, T. Deshane, W. Hu, J. Bongio, P.F. Wilbur, and B. Johnson: Running Xen: A Hands-on Guide to the Art of Virtualization; Prentice Hall; April 2008.

[7] D. Gupta, L. Cherkasova, R. Gardner, and A. Vahdat: Enforcing Performance Isolation Across Virtual Machines in Xen; In Proceedings of the 7th ACM/IFIP/USENIX Middleware Conference, 2006, pp. 342-362.

[8] A. K. Parekh and R. G. Gallager: A generalized processor sharing approach to flow control in integrated services networks: The single-node case; IEEE/ACM Transactions on Networking; 1993, Vol. 1, pp. 344-357.

[9] P. Padala et al.: Adaptive control of virtualized resources in utility computing environments, ACM SIGOPS Operating Systems Review, Vol. 41, No. 3, 2007, pp. 289-302.

[10] Y. Song, Y. Sun, H. Wang, and X. Song: An adaptive resource flowing scheme amongst VMs in a VM-based utility computing, in Computer and Information Technology, 2007. CIT 2007. 7th IEEE International Conference on, 2007, pp. 10531058.

[11] J. Liu, W. Huang, B. Abali, and D. K. Panda: High performance VMM-bypass I/O in virtual machines, in Proceedings of the annual conference on USENIX, 2006, Vol. 6, pp. 3-3.

[12] V. Chadha, R. Illiikkal, R. Iyer, J. Moses, D. Newell, and R. J. Figueiredo: I/O processing in a virtualized platform: a simulation-driven approach, in Proceedings of the 3rd international conference on Virtual execution environments, 2007, pp. 116-125.

[13] D. Ongaro, A. L. Cox, and S. Rixner: Scheduling I/O in virtual machine monitors, in Proceedings of the fourth ACM SIGPLAN/SIGOPS international conference on Virtual execution environments, 2008, pp. 1-10.

[14] 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, San Jose, California, 2008, pp. 161-170.

[15] S. R. Seelam and P. J. Teller: Virtual I/O scheduler: a scheduler of schedulers for performance virtualization, in Proceedings of the 3rd international conference on Virtual execution environments, 2007, pp. 105-115.

 ISBN: 978-1-61208-153-3