Providing Performance Guarantees to Virtual Machines using Real-Time Scheduling*

Tommaso Cucinotta, Dhaval Giani, Dario Faggioli, and Fabio Checconi

Scuola Superiore Sant'Anna, Pisa, Italy

Abstract. In this paper we tackle the problem of providing Quality of Service guarantees to virtualized applications, focusing on computing and networking guarantees. We propose a mechanism for providing temporal isolation based on a CPU real time scheduling strategy. This allows not only to have control over the individual virtual machine throughput, but also on the activation latency and response-time by which virtualized software components react to external events. We show experimental results gathered on a real system validating the approach.

1 Introduction

Virtualization is being increasingly used for easing deployment and management of software components. Nowadays, it constitutes a core technology that can be leveraged by infrastructure providers in the Cloud Computing domain, for the purpose of allowing customers to access a "cloud" of shared physical resources such as computing elements and networking (according to the Infrastructure-as-a-Service paradigm — IaaS).

When deploying virtualized distributed applications over a set of physical resources, by means of machine and network virtualization, one of the hot problems that is receiving an increasing attention [17, 15] is the one of how to provide a *stable performance* of *individual* virtualized applications. This problem is due to a multitude of factors: on the networking side, multiple data flows need to be streamed over a pool of shared physical network links; on the computing side, multiple Virtual Machines (VMs) need to be concurrently scheduled over a set of shared processors and cores; for storage, multiple data flows need to be concurrently supported during access to shared storage devices.

Sharing of physical resources constitutes a great opportunity for IaaS and PaaS providers. It allows for a better utilization of the underlying physical infrastructure. This is especially true with the increasing need [3] to deploy complex, distributed, interactive real-time applications over virtualized infrastructures (as common in the Cloud Computing world), a scenario implying a potential under-utilization of resources. In fact, in interactive and collaborative applications, at each activation of the distributed computing pipeline, triggered for example by explicit user actions, the remote physical resources may not always be used at their full extent and saturated. An efficient utilization of resources in datacenters may also lead to the deployment of interactive

* The research leading to these results has received funding from the European Community's Seventh Framework Programme FP7 under grant agreements n. 214777 "IRMOS—Interactive Realtime Multimedia Applications on Service Oriented Infrastructures" and n. 248465 "S(o)OS – Service-oriented Operating Systems."

applications on the same physical hosts occupied by HPC applications, which typically are CPU intensive and may rely on communication channels to pass data and synchronize themselves. When this is the case, it is crucial to provide CPU *and* I/O isolation between the two classes of workloads to avoid decreasing the customer satisfaction.

However, without an appropriate support for temporal isolation, concurrently running VMs may interfere each other in a way that it becomes impossible to guarantee a stable performance level to each one of them. This problem has been previously addressed in the case of compute-intensive VMs [7, 8], however it still remains a hot topic in the case of I/O-intensive and mixed workloads.

Paper Structure. This paper is structured as follows. In Section 2, related work on the topic is surveyed. In Section 3, background concepts are recalled. In Section 4, the proposed approach is described. In Section 5, experimental results are shown that prove the effectiveness of the proposed technique. Finally, in Section 6 conclusions are drawn, and possible directions for future work on the topic are discussed.

2 Related Work

The interaction between CPU scheduling and I/O performance of virtualized environments was studied before, mainly in virtualization systems based on the Xen hypervisor. When first introduced, in [2], Xen was designed as a hypervisor containing the device drivers and exporting their functionality to the virtual machines; later, its design was changed, moving the device drivers to dedicated domains (see [11]). The papers cited below all consider the latter setup.

In [6], the authors proposed a monitoring infrastructure for Xen and estimated the CPU overhead induced by I/O virtualization using a set of HTTP-based benchmarks. In [4], the authors characterized the overheads of network virtualization in Xen using full system simulation; in this way, they were able to estimate the effects of the hardware architecture on the virtualization stack performance.

To increase the control over I/O virtualization, various solutions were proposed; most of them used CPU scheduling to isolate the VMs from the performance perspective. In [12], the authors proposed to augment the Xen hypervisor with a set of mechanisms to account for and to control the CPU time spent on behalf of VMs doing I/O. In [18], the authors proposed an extension to the Xen credit-based scheduler improving its behavior in presence of multiple different applications with heavy I/O workloads, prioritizing the I/O bound ones. Also, in [14], the authors proposed to modify the Xen CPU scheduler (making it cache-aware) and networking architecture to improve the performance of virtualized I/O on 10 Gbps Ethernet.

None of the approaches described above deal with service guarantees, most of them aim at improving fairness and/or throughput; we advocate the need for providing explicit guarantees in order to obtain predictable performance.

The existing solutions that support QoS, like for example Open vSwitch [19], or VMWare vNetwork [1], tend to be confined to the networking domain, and enforce QoS

[1] `http://www.vmware.com/products/vnetwork-distributed-switch`

policing and shaping network traffic according to user-defined policies. A widely used virtual networking tool, VDE [10] uses the Linux bridging capabilities to achieve similar results. Our work differs from these latter approaches in that it tries to take into account isolation and CPU scheduling effects on I/O performance.

Finally, in previous work [8] it was proposed to use real-time scheduling techniques for supporting proper timeliness guarantees to virtualized applications concurrently running on different VMs deployed on the same CPU. Also, experimental results were shown [9] proving that the approach is effective in keeping under control the response times of virtualized web servers, independently of what other load is running on the system. However, in these works, the investigation was limited to CPU-bound workloads, while in this paper we consider also the effect of I/O-intensive workloads.

3 Background on KVM

The focus of this paper is on the KVM virtualization layer, in which VMs are basically processes that run inside the Linux OS. KVM is an hypervisor designed to be part of the Linux kernel. It is composed of a userspace component (a modified `qemu` instance) providing device emulation, and a kernel module providing guarded transition to the "guest mode" supported by the virtualization extensions of modern CPUs. User and kernel space communicate using a character device interface. When possible, the VMs execute in guest mode, returning control to the host whenever the CPU virtualization extensions are not able to emulate the hardware behavior, which mostly happens when the VMs are doing I/O; when in host mode, control is returned to the userspace code, that handles all the effective I/O: emulates reads/writes to I/O ports, generates interrupts for the VMs and interacts with the host devices using the standard Linux interfaces. Note that in this paper we do not consider hardware accelerated I/O.

KVM uses one thread per virtual CPU, plus a thread handling the interaction with host devices. We used *virtio*, a paravirtualization extension available in KVM, that reduces the number of host–guest mode switches using in-memory interaction between the virtual CPUs threads and the thread handling I/O to and from the host devices.

4 Proposed Approach

In this paper, we propose to provide stable computing and networking performance guarantees to VMs concurrently running on the same CPU(s) using an EDF-based soft real-time scheduling strategy for the CPU, which we developed in the context of the IRMOS project[2]. The proposed approach is particularly useful when mixing VMs with workloads that are heterogeneous with respect to the time granularity over which the temporal requirements of the hosted applications need to be fulfilled. For example, on the same CPU one might be running a VM carrying on a file-transfer activity, and a VM running a game server, two workloads with very different responsiveness levels.

First, for the sake of completeness, we recall the basic characteristics of the IRMOS scheduler, reminding to [5] for a more complete description. Then, we describe how

[2] More information is available at: `http://www.irmosproject.eu`.

it has been used and configured for isolating performance of compute-intensive and network-intensive virtualized workloads.

The IRMOS Real-Time Scheduler. The IRMOS real-time scheduler allows to reserve a "slice" of the processing capability of a system to a group of threads and/or processes (shortly, tasks). This is done by specifying two scheduling parameters for each group: a budget Q and a period P, with the meaning that the tasks in the group are entitled to run on each of the CPUs (processor, or cores when present) available to the OS, for Q time units every period of P time units. This constitutes a scheduling guarantee and a limitation at the same time, that can only be approximated in simplified conditions, but in general cannot be reproduced, using the default priority-based Linux real-time scheduling extensions. This is achieved by a hard-reservation variant of the EDF-based Constant Bandwidth Server (CBS) scheduler [1], implemented as a partitioned scheduling strategy, where each CPU has its own private task queue, and it is scheduled independently. However, when a group is entitled to run on each CPU, the IRMOS scheduler employs a POSIX priority-based real-time scheduling strategy [13] among its tasks, in such a way that, if there are m CPUs, (at most) the m tasks with the highest priority are the ones which actually run. The system performs admission control over admitted reserved groups, so that the overall system capacity may be properly partitioned among concurrently running activities in the system, without overloading it. Also, the scheduler has a hierarchical configuration capability, by which it is possible to define groups and nested subgroups of real-time tasks with given scheduling parameters.

Temporal Isolation of Virtual Machines. With KVM, there is little control on how multiple VMs compete in accessing the available CPUs on Linux. In fact, the default Linux scheduling strategy (`SCHED_OTHER`) implements the Completely Fair Scheduler (CFS) policy, which tries to be as fair as possible across competing processes. Therefore, we used the scheduler described above to isolate the temporal behavior of concurrently running VMs, and at the same time provide them with their specifically required scheduling guarantees.

When dealing with compute-intensive VMs only, most of the time dedicated to a VM is spent by the host by running the corresponding KVM process. Therefore, providing proper CPU scheduling guarantees to the process allows for the achievement of a sufficient isolation degree between that VM and other VMs. This is done in a straightforward way by using our real-time IRMOS scheduler. This requires the set-up of proper scheduling parameters for a VM. This can be done as follows. The scheduling period controls the activation latency of the VM, and it is closely related to the degree of interactivity required by it. In fact, the period can be set equal to the minimum expected interarrival period of external requests triggering the VM services. For a VM running an interactive application, this may easily happen to be in the order of 10–$200ms$. The ratio budget over period controls how much computing capability of the host is reserved for the VM, thus the budget may be tuned by performing a preliminary benchmarking phase on the performance of the VM as arising from multiple budget assignments. Note that, thanks to the hard reservation nature of a real-time scheduler, the performance obtained when the VM is running in isolation on the host, with given scheduling parameters, is only marginally affected by the workload imposed on the host by other VMs.

However, the situation becomes more complex when dealing with I/O-intensive workloads. In fact, in such case, the host may spend a significant part of the CPU time related to a VM outside the context of the KVM threads. The lowest level of the networking code executes in interrupt context, preempting the execution of VMs potentially unrelated to the I/O traffic that is being handled, thus "stealing" part of the budget reserved to the interrupted VMs, even under the use of our real-time scheduler.

In our approach, we suggest to overprovision the budget assigned to each VM, as compared to the minimum one detected when benchmarking the VM in isolation (on the same hardware). Specifically, not only the budget should be increased of the amount necessary to deal with the interferences of multiple VMs at the cache level (this is unavoidable in modern systems), but also of a quantity that is strictly dependent on the overall networking traffic performed by the VMs hosted on the same system. Such aggregate figure is usually available to the infrastructure that handles the deployment of the VMs on the physical host.

Also, higher-level in-kernel networking code often executes in *softirq* context [16]. Furthermore, when using the PREEMPT-RT kernel [20], part of the low-level networking driver code runs in dedicated kernel threads, where it may be at risk of not getting a proper chance to run, compromising networking performance.

In our approach, we put all the threads relative to the same VM into the same reservation, comprising both KVM threads and kernel threads necessary for dealing with the (para-)virtualized networking of that VM. Our real-time scheduler allows for the provisioning of overall scheduling guarantees to the entire group of threads, even if not belonging to the same process. Fig. 1a depicts the overall architecture, showing how the temporal capsule extends also to the interrupt threads, which act as interconnecting channels between the kernel and the virtual machines.

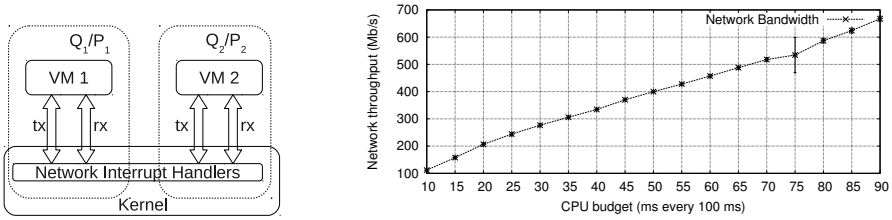
In the preliminary results reported below, we show how it is possible to achieve a proper degree of isolation in presence of I/O-intensive VMs, deferring to future work the adoption of more sophisticated techniques we are investigating (see Section 6).

5 Experimental Results

For the purpose of validating the proposed approach, in this section we report results gathered from an experimental set-up involving a real Linux system running KVM as hypervisor.

All the described experiments have been conducted by using two physical systems equipped with an Intel Quad Core Q6600 CPU running at 2.4 Ghz, 4 GB of RAM, and a Gigabit Ethernet card. One of the two systems played the role of *server*, and was running a Fedora 11 Linux distribution with a modified version of the kernel including our real-time scheduler. VMs were started with KVM in bridge mode and with 1 GB of guest memory. The networking was setup to use the virtio interface. The other system was used as *client* during the experiments described below. On multi-core systems, the problem addressed in this paper appears when deploying VMs with an overall number of virtualized CPUs greater than the number of available physical CPUs. In order to keep a simple experimental set-up, the tests were run with only one core brought online, pinning VMs on the same core.

In what follows, first, resource-level experiments are shown, demonstrating how the proposed technique improves isolation of I/O intensive traffic across VMs while they are concurrently running, gathered running a synthetic network-benchmarking tool. Then, application-level results are shown, from an experiment involving a real world web server.



a) Architecture. b) Network bandwidth with varying CPU reservation for a single VM.

Fig. 1: System architecture and network bandwidth for a VM with various CPU reservations.

Resource-level isolation. As mentioned in Section 4, in order to set-up VMs with proper scheduling parameters, a preliminary benchmarking phase is mandatory. In the following experiment, we wanted to investigate on the impact of different CPU share allocations over the networking throughput achievable by the VM. To this purpose, a VM was run alone on the server, with all of its threads attached to a unique real-time reservation for the VM, with a period of $100ms$ and different budgets varying from $10ms$ to $90ms$. We used `iperf` [3] to measure the network throughput between an `iperf` client running on the client machine, and an `iperf` server running inside the VM. The test was repeated 100 times for each budget value. As shown in Fig. 1b, there is a nearly linear relationship between the network throughput achieved by the VM and its CPU share. Each point corresponds to the average throughput over the 100 repetitions, and is flanked by a small vertical segment highlighting the standard deviation, which is barely noticeable except for a budget of $75ms$, where some unidentified kernel-level activity disturbed the experiment.

Now, in order to measure the degree of temporal isolation enforced by the real-time scheduler, we started two VMs on the same physical host and core, each one isolated in a different resource reservation. Each VM was running an `iperf` server. We launched two `iperf` clients on the client machine against the two VMs, and we measured the achieved throughput for one VM at varying scheduling parameters both for itself, and for the other VM.

The obtained results are shown in Fig. 2 and 3, from different perspectives. Fig. 2 shows (on the Y axis) the throughput obtained by a VM as a function of its own reservation share (on the X axis), at varying reservation share for the other VM (different

[3] More information at: `http://sourceforge.net/projects/iperf/`.

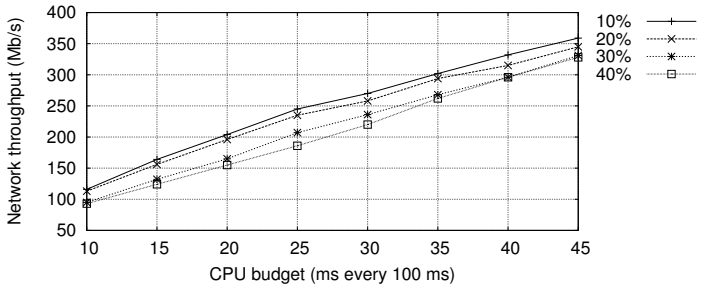


Fig. 2: Network throughput (Y axis) for a VM as a function of its own CPU share (X axis), at varying CPU shares for the other VM (different curves).

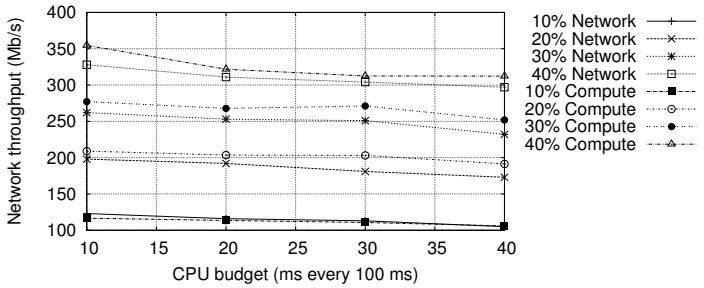


Fig. 3: Network throughput (Y axis) for a VM as a function of the CPU share of the other VM (X axis), at varying CPU shares for itself (different curves), in case of CPU- and I/O-intensive loads.

curves). Ideally, if the temporal isolation were perfect, we should see perfectly superimposed curves. However, as expected, a performance drop is experienced by the VM under observation, quantified in a 20%–30% drop when the reservation share of the other VM is increased from 10% (similarly to the single-VM case in Fig. 1b) to 40%.

Fig. 3 shows the throughput (Y axis) obtained by the VM under observation as a function of the CPU share assigned to the other VM (X axis), and at varying CPU shares for itself (different curves labeled with "Network"). Again, if the isolation were perfect, we should see horizontal lines. Instead, we see again the performance drop that is achieved. Finally, we made a third experiment with a computation-intensive workload on the other VM (it was running Octave[4] inverting a 1000x1000 matrix). The results are shown on the same graph, in the set of curves labeled as "Compute". As expected, the performance drop in this case is smaller, being due exclusively to cache interference. The difference between the two sets of curves may be basically attributed to the increased interrupt activity experienced in the former experiment, which was "stealing" CPU from the first VM despite the reservation at the scheduling level.

However, we would like to point out that the adoption of our real-time scheduling strategy is capable of providing a controllable bound on the maximum interference that

[4] More information is available at: `http://www.octave.org`.

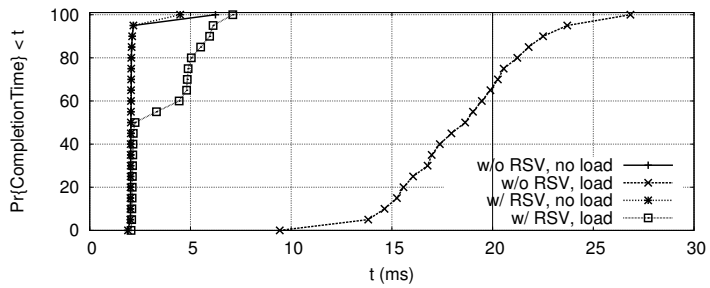


Fig. 4: CDF of the completion time of the requests to the first VM in various conditions

a VM may undergo due to intensive networking activities of other VMs. For example, in order to counteract the expected interferences from other VMs, Fig. 2 may be used in place of Fig. 1n for "looking-up" the (correctly overprovisioned) budget for sustaining a given VM throughput, depending on the expected networking load of other VMs.

Macro Benchmark: Apache Bench. Now, in order to demonstrate the achievable level of temporal isolation on a real-world application, a ramdisk with the Apache web server [5] was setup inside each VM[6].

The two VMs were started, and the download of a file of 100 kBytes was requested (via the HTTP protocol) to the first VM every $20ms$ using the `ab` tool[7], from the client machine. Also, 1000 concurrent requests of a file of the same size were being continuously sent to the second VM, serving as both networking and computation "load".

The experiment was performed with and without the load imposed by the second VM, and we measured the completion time of the requests, i.e., the time at which the download of the file finished, for 500 consecutive requests. Moreover, both the unloaded and loaded experiments were repeated while the two VMs were co-scheduled by the standard Linux kernel mechanisms, and with the first VM in a reservation with parameters $RSV(Q, P) = (4ms, 20ms)$. The Cumulative Distribution Function (CDF) of the completion time in all the four evaluated scenarios is shown in Fig. 4. The vertical line at 20 ms represents the time by which a request must complete, since another one should start (i.e., the deadline). It is easy to see how, in the unloaded case (curves labeled with "no load"), both with and without the reservation (curves labeled with "w/RSV" and "w/o RSV", respectively), the performance is good enough, since the completion time is almost constant at about $2.1ms$ among the various requests and it is always far from the deadline (with peaks of nearly $6.2ms$). However, in the loaded case, the original VM performance is completely subverted, and more than 30% of the requests for the first VM cannot complete within the $20ms$ deadline/period, with peaks of download-time of nearly $27ms$. When encapsulating the VM in a properly tuned real-time reservation,

[5] `http://httpd.apache.org/`

[6] By setting it up in a ramdisk, additional interference in the form of disk I/O was avoided.

[7] We modified `ab` to behave as described.

instead, the download times returned to be well below the deadline. This can also be verified on the statistics reported in Table 1.

Table 1: Statistics of the download times (in ms) from the first VM in various conditions.

Scenario	Mean	Std Dev	Max
w/o RSV, no load	2.114	0.451	6.242
w/o RSV, load	18.528	3.289	26.844
w/ RSV, no load	2.0859	0.278	4.488
w/ RSV, load	3.5402	1.6162	7.103

6 Future Work and Conclusions

In this paper, the problem of provisioning QoS guarantees to VMUs concurrently running on the same CPU was tackled. The focus was on VMs with I/O intensive workloads, where even if the guest OSes are for most of the time suspended for performing I/O, actually the host needs to execute the para-virtualized and native networking drivers necessary to deliver the packets, what is a major cause of interference between the VMs. Therefore, I/O-intensive and compute-intensive VMs may strongly interfere with each other, leading to a performance that is completely subverted as compared to the case in which they were running or benchmarked in isolation.

We showed that, by recurring to soft real-time scheduling strategies at the virtualization layer, it is possible to provide a good level of isolation between the concurrently running VMs. Furthermore, it is possible to achieve both a good throughput of the VMs and to keep the individual guarantees at the latency level, something that is not possible with the standard Linux scheduling strategies. However, the proposed solution is all but conclusive in this regard. In fact, as highlighted in the experimental section, still there is a degree of interference which is due to the resources that are implicitly shared among the VMs inside the host OS, namely network interface drivers and bridging logic that runs on the host OS. We plan to enhance the isolation with this regard by slightly reworking the networking driver infrastructure in Linux for such purpose, exploit some recent kernel features that allows for putting the networking code in a per-VM thread-/context. Also, we plan to experiment with the PREEMPT-RT branch of the kernel, in which part of the drivers logic is moved to dedicated kernel threads, thus it is possible to control when they execute with our real-time scheduler, and for which a variation of our real-time scheduler is already being ported.

Finally, we plan to investigate on the use of adaptation for fine-tuning the resource reservation parameters so as to better suit the needs of virtualized applications.

Bibliography

[1] Abeni, L., Buttazzo, G.: Integrating multimedia applications in hard real-time systems. In: Proc. IEEE Real-Time Systems Symposium. Madrid, Spain (December 1998)

[2] Barham, P., et al.: Xen and the art of virtualization. In: SOSP '03: Proc. nineteenth ACM symposium on Operating systems principles. New York, NY, USA (2003)

[3] Boniface, M., et al.: PaaS architecture for real-time quality of service management in clouds. International Conference on Internet and Web Applications and Services (2010)

[4] Chadha, V., et al.: I/o processing in a virtualized platform: a simulation-driven approach. In: VEE '07: Proc. 3rd international conference on Virtual execution environments. pp. 116–125. ACM, New York, NY, USA (2007)

[5] Checconi, F., et al.: Hierarchical multiprocessor CPU reservations for the linux kernel. In: Proc. OSPERT 2009. Dublin, Ireland (June 2009)

[6] Cherkasova, L., Gardner, R.: Measuring cpu overhead for i/o processing in the xen virtual machine monitor. In: ATEC '05: Proc. annual conference on USENIX Annual Technical Conference. pp. 24–24. Berkeley, CA, USA (2005)

[7] Cherkasova, L., Gupta, D., Vahdat, A.: Comparison of the three cpu schedulers in xen. SIGMETRICS Perform. Eval. Rev. 35(2), 42–51 (2007)

[8] Cucinotta, T., Anastasi, G., Abeni, L.: Real-time virtual machines. In: Proceedings of the 29^{th} IEEE Real-Time System Symposium (RTSS 2008) – WiP Session. Barcelona (December 2008)

[9] Cucinotta, T., Anastasi, G., Abeni, L.: Respecting temporal constraints in virtualised services. In: Proc. IEEE RTSOAA 2009. Seattle, Washington (July 2009)

[10] Davoli, R.: Vde: Virtual distributed ethernet. In: Proc. First International Conference on Testbeds and Research Infrastructures for the DEvelopment of NeTworks and COMmunities (TRIDENTCOM'05). pp. 213–220 (2005)

[11] Fraser, K., Hand, S., Neugebauer, R., Pratt, I., Warfield, A., Warfield, A., Williamson, M., Williamson, M.: Reconstructing i/o (2004)

[12] Gupta, D., Cherkasova, L., Gardner, R., Vahdat, A.: Enforcing performance isolation across virtual machines in xen. In: Proc. ACM/IFIP/USENIX 2006 International Conference on Middleware. New York, NY, USA (2006)

[13] IEEE: Information Technology - Portable Operating System Interface - Part 1: System Application Program Interface Amendment: Additional Realtime Extensions. (2004)

[14] Liao, G., et al.: Software techniques to improve virtualized i/o performance on multi-core systems. In: Proc. ACM/IEEE ANCS 2008. New York, NY, USA (2008)

[15] Lin, B., Sundararaj, A., Dinda, P.: Time-sharing parallel applications with performance isolation and control. In: Proc. 4^{th} International Conference on Autonomic Computing (ICAC 2007). pp. 28–28. Jacksonville, FL (June 2007)

[16] Love, R.: Linux Kernel Development (2nd Edition) (Novell Press). Novell Press (2005)

[17] Nathuji, R., Kansal, A., Ghaffarkhah, A.: Q-clouds: managing performance interference effects for qos-aware clouds. In: EuroSys '10: Proc. 5th European conference on Computer systems. pp. 237–250. ACM, New York, NY, USA (2010)

[18] Ongaro, D., Cox, A.L., Rixner, S.: Scheduling i/o in virtual machine monitors. In: Proc. ACM SIGPLAN/SIGOPS VEE '08. ACM, New York, NY, USA (2008)

[19] Pfaff, e.a.: Extending networking into the virtualization layer. In: 8th ACM Workshop on Hot Topics in Networks (HotNets-VIII). New York City, NY (October 2009)

[20] Rostedt, S., Hart, D.V.: Internals of the rt patch. In: Proc. Ottawa Linux Symposium (OLS 2007). pp. 161–172 (June 2007)