Improving Responsiveness for Virtualized Networking Under Intensive Computing Workloads

Tommaso Cucinotta
Scuola Superiore Sant'Anna
Pisa, Italy
cucinotta@sssup.it

Fabio Checconi
IBM Research, T.J. Watson
Yorktown Heights, NY, USA
fchecconi@gmail.com

Dhaval Giani
Scuola Superiore Sant'Anna
Pisa, Italy
dhaval.giani@gmail.com

October 5, 2011

Abstract

In this paper the problem of providing network response guarantees to multiple Virtual Machines (VMs) co-scheduled on the same set of CPUs is tackled, where the VMs may have to host both responsive real-time applications and batch compute-intensive workloads. When trying to use a real-time reservation-based CPU scheduler for providing stable performance guarantees to such a VM, the compute-intensive workload would be scheduled better with high time granularities, to increase performance and reduce system overheads, whilst the real-time workload would need lower time granularities in order to keep the response-time under acceptable levels. The mechanism that is proposed in this paper mixes both concepts, allowing the scheduler to dynamically switch between fine-grain and coarse-grain scheduling intervals depending on whether the VM is performing network operations or not. A prototype implementation of the proposed mechanism has been realized for the KVM hypervisor when running on Linux, modifying a deadline-based real-time scheduling strategy for the Linux kernel developed previously. The gathered experimental results show that the proposed technique is effective in controlling the response-times of the real-time workload inside a VM while at the same time it allows for an efficient execution of the batch compute-intensive workload.

Acknowledgements

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."

1 Introduction and Related Work

Virtualization is increasingly gaining momentum as the enabling technology for the management of physical resources in data centers and Infrastructure-as-a-Service (IaaS) providers in the domain of Cloud Computing. Indeed, virtualization enhances the flexibility in managing physical resources, thanks to its capability to virtualize the hardware so as to host multiple Virtual Machines (VMs) executing potentially different Operating Systems, and the capability to live-migrate them as needed without interrupting the provided service, except for a very low downtime. Virtualized systems are also capable of exhibiting a performance nearly equal to the one experienced on the bare metal, due to the hardware virtualization extensions provided by modern processors.

As a consequence of virtualization, multiple under-utilized servers can easily be consolidated onto the same physical host. This allows a reduction in the number of required physical hosts to support a number of virtualized OSes, leading to advantages in terms of costs for running the infrastructure and of energy impact.

However, once multiple VMs are deployed on the same physical resources, their individual performance is at risk of becoming greatly unstable, unless proper mechanisms are utilized. A VM which temporararily saturates either the processing, networking, or storage access capacity of the underlying physical resources immediately impacts the performance of the other VMs which share the same resources. This is a potentially critical issue for IaaS providers where proper QoS specifications are included in the Service-Level Agreements (SLAs) with the customers.

The problem of providing a stable performance to individual VMs has been studied in the past. For example, Gupta et al. [8] introduce in the Xen hypervisor[1] a proper CPU scheduling strategy accounting for the consumption of device driver domain(s) as due to the individual VMs operations. In [11], an extension to the Xen credit-based scheduler is proposed, to improve its behavior in presence of multiple different applications with I/O bound workloads. Also, Liao et al. [9] propose to modify the Xen CPU scheduler, by making it cache aware, and the networking infrastructure to improve the performance of virtualized I/O on 10Gbps Ethernet.

For the KVM hypervisor[2], Cucinotta et al. [3, 4, 5] investigated on the use of hierarchical deadline-based real-time CPU scheduling [2] for the Linux kernel in order to stabilize the performance of individual compute-intensive VMs, tackling the problem of network-intensive VMs later [6].

The latter works rely on the use of a reservation-based scheduler [2] for the CPU (a hard-reservation variant of the Constant Bandwidth Server [1]) that allows for configuring the scheduling guarantees for a given VM in terms of a budget (Q) and a period (P). The scheduler will guarantee that each VM will be scheduled for Q time units every period of P time units, under the usual assumption of non-saturation for EDF ($\sum_i \frac{Q_i}{P_i} \leq 1$, see [10] for details). The reservation period can be specified independently for each VM, and it constitutes the time granularity over which the CPU allocation is granted to the VM.

A shorter period improves the responsiveness of the VMs at the cost of higher scheduling overheads, thus being beneficial for time-sensitive workloads. On the other hand, a longer period leads to lower scheduling overheads, thus it is beneficial for batch and high-performance workloads, at the cost of potentially longer time intervals during which the VM is unresponsive (in the worst-case, a VM might have to wait as much as $2(P - Q)$ before being scheduled again). However, for VMs embedding both batch computing activities (including both main VM functionality or typical bookkeeping OS activities, such as updating indexes) and time-sensitive tasks (e.g., reporting on the progress of batch tasks, or realizing independent features), both configurations do not fit very well, as highlighted by Dunlap in the discussion about future work on the new upcoming Xen Credit Scheduler [7].

In this paper we propose a novel mechanism for scheduling VMs with both compute-intensive and network-responsive workloads. In absence of external requests the VM progresses with its (long) period configuration (e.g., hundreds of ms) and can perform batch computing activities reducing scheduling overheads to the minimum. However the occurrence of external requests allows the VM to be woken up by the scheduler within a much shorter interval (e.g., ms or tens of ms), to perform relatively short activities configured at a higher priority inside the VM, so as to respond very quickly to external events.

[1] More information at: `http://www.xen.org/`.

[2] More information at: `http://www.linux-kvm.org/`.

2 Approach

The mechanism proposed in this paper applies to virtual machines scheduled under a reservation-based real-time scheduler like the one presented in [2]. For the sake of simplicity the focus is on single-core VMs scheduled according to a partitioned EDF policy (so one or more VMs are pinned on each physical core and scheduled on it).

Each VM can be configured with a set of scheduling parameters denoted by (Q, P), with the meaning that Q time units are granted to the VM for each P time units. The interest in having $Q/P < 1$, thus the possibility to have multiple VMs co-scheduled on the same processor and core, comes from the fact that the infrastructure provider may have an interest in "partitioning" the big computing power available on a single powerful core into multiple VMs with lower computing capabilities and rent them separately, or merely from the fact that the hosted VMs have an expected workload (e.g., as due to requests coming from the network) that cannot saturate the computing power on the underlying physical core, thus enabling the provider to perform server consolidation. The Q value constitutes both a guarantee and a limitation (i.e., we are using hard reservations). This ensures that the performance of each VM is not affected (too much) from how much intensively other VMs are computing [5, 4].

Roughly speaking, at equal Q over P ratios, the chosen value for P regulates the responsiveness of the associated VM. It is easy to see that, if the VM is running alone, then its schedule comes out as shown in Figure 1, and the non-responsiveness time interval for the VM may be as long as $P - Q$. However, the worst-case condition when the VM is co-scheduled with other VMs is the one shown in Figure 2, with the budget granted to the VM at the beginning of a P time window (for example, because at that time all other VMs were idle), and at the end of the time window immediately following (for example, as due to the wake-up of a VM at the beginning of this second time window, with a deadline slightly shorter than the first VM, under theoretical saturation for the EDF scheduler).

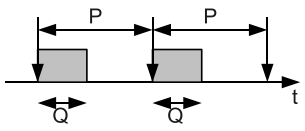

FIGURE 1: *Example schedule of a VM with generic scheduling parameters of (Q, P), when running alone, exhibiting a non-responsiveness time interval of $P - Q$.*

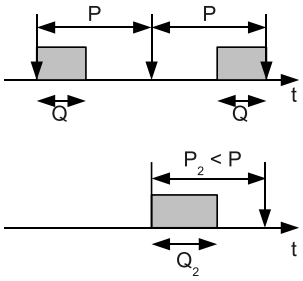

FIGURE 2: *Example schedule of a VM with generic scheduling parameters of (Q, P), when co-scheduled, exhibiting a non-responsiveness time interval of $2(P - Q)$.*

Also, P controls the scheduling overheads imposed on the system. In fact, the scheduler forces a context switch at least every interval as long as the minimum P value across all the reservations configured on the core.

This kind of scheduler allows heterogeneous virtualized workloads to safely coexist as far as they belong to different VMs. One can easily configure a short P value for a VM with a real-time workload that needs to be responsive, and a long P for a VM that performs mainly batch computations. However, mixing such types of workloads in the same VM may lead to problems. One can configure the responsive activities in the VM to run at a higher priority as compared to the batch computing ones (i.e., by exploiting priority-based scheduling as available on every OS). However, still the non-responsive periods of the VM will largely dominate the response time of the real-time task(s). So, in order to keep such response times low, the normal option would be the one to use small P values, obtaining high scheduling overheads also while the VM is doing its batch computing activities without any request from the outside triggering the real-time functionality.

In order to resolve this problem, in this paper we propose the following mechanism (see Figure 3). The VM is normally attached to a reservation configured with scheduling parameters (Q, P), with a period P tuned for the batch computing case, i.e., it is relatively large, for example in the range of hundreds of milliseconds. In addition, a second "spare" reservation is configured in the system with parameters (Q_s, P_s) tuned for the operation of the real-time activity, i.e., P_s is relatively small, for example in the range of tens of milliseconds or shorter, and Q_s sufficient to complete an activation of the real-time activity. Now, whenever the VM receives a network packet and its current budget is exhausted (i.e., it is in the non-responsiveness time frame), the VM

is temporarily attached to the "spare" reservation. Having a much shorter deadline, the spare reservation forces the VM to be scheduled and receive Q_s execution time units on the processor within the P_s deadline from the packet receive time; this will cause the VM to run, receive the packet and possibly activate the real-time activity that will perform some fast computation (and possibly provide a response packet). If the real-time activity cannot complete within the first activation of the spare reservation, it will be resumed during the subsequent activations, so it will receive additional Q_s time units during the following P_s time window, and so on, till the time of replenishment of the original reservation budget, at which time the VM relinquishes the spare reservation. With a proper tuning of the Q_s and P_s parameters a VM configured for batch computing activities should exhibit a tremendously improved response-time to sporadic requests coming from the network, at the cost of keeping some extra-capacity unused in the system.

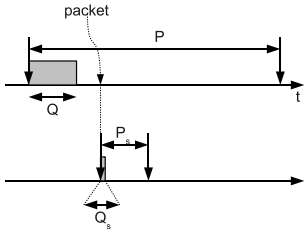

FIGURE 3: *Example schedule of a VM with generic scheduling parameters of (Q, P), and a spare reservation of (Q_s, P_s) which is dynamically activated and attached to the same VM on a new packet arrival. Despite the budget for the VM at packet arrival time was exhausted, the VM can complete a short real-time activity of duration Q_s within the spare reservation period P_s.*

The requirements of the real-time workload are assumed to be relatively small, and in any case the additional reservation to be attached dynamically to a VM cannot bee too large in terms of utilization (budget over period), because it needs to remain unused for all the time in which the VM does not access the network. For example, it might require a 10% or a lower CPU utilization to complete. This should allow the real-time activity triggered by the received network packet to complete, assuming it is configured in the VM for running at higher priority than other activities. For example, the VM may perform kernel-level activities inside the networking driver and stack, and relatively short userspace activities, which may be running in a task that was waiting for the packet arrival.

Finally, in order to avoid keeping a spare reservation for each and every VM hosted onto the same physical host, we propose to use a pool of spare reservations which can be used for the purpose illustrated above. The idea is that, exploiting statistical multiplexing of the networking traffic patterns among independent VMs, one can assume that the probability of having all the VMs requiring a spare reservation attached dynamically at the same time be very small. This way, the additional utilization to keep for spare reservations may be kept limited.

Therefore, a pool of a few reservations with short periods will be ready to be used for boosting reservations (with longer periods) of VMs when they receive packets from the external world but their normal budget is exhausted due to compute-intensive activities. This allows for a very quick reaction-time of the VMs.

3 Implementation Details

In order to validate the proposed approach we implemented a proof of concept in the Linux kernel, using the KVM hypervisor to execute the VMs. We started from the IRMOS scheduler [2], modifying it to include support for reservations providing "spare" bandwidth, and introducing the glue code needed to use this new feature.

From the interface point of view, each reservation may have the property of *providing* spare bandwidth to the reservations needing it, and/or the property of *using* spare bandwidth from reservations providing it. The system administrator controls the parameters of the reservations and the dependencies between users and providers of spare bandwidth using the CGROUP filesystem interface.

To recognize the events that are related to VM I/O, and consequently activate the spare bandwidth mechanism we modified the networking code. In our modified kernel, when a packet arrives we check its destination and if is headed towards a Virtual Machine we retrieve its server using a simplified hash table. If the server has run out of bandwidth we set a flag to mark that it needs to access its spare reservation. Setting the flag may also imply requeueing the running tasks belonging to the same VM, as they may need to access the spare bandwidth too.

When a task is activated, along as performing a regular activation, the scheduler checks if the task belongs to a virtual machine, and if the VM's server needs spare bandwidth; if this is the case, the task is not only enqueued in its own server, as would be

done anyway, but it is also enqueued in the server providing the spare reservation.

The flag set on the VM's server needs to be reset, and this may happen on two conditions. The first possibility is when the emergency bandwidth has been set for a certain duration, empirically determined not as a function of time, but rather of the chances the server has had to execute its tasks. The other possibility is when the original server has its bandwidth restored.

4 Experimental Results

The approach presented in the previous section was validated through an experiment conducted on a prototype implementation of the mechanism, evaluated on a Linux 2.6.35 kernel patched with the IRMOS real-time scheduler [2], running on an Intel Core 2 Duo P9600 CPU configured for running at a fixed 2.66 GHz frequency. The VM was configured with the CPU thread running at real-time priority lower than the one used for all its other threads. We were unable to use the full implementation described in Section 3, and we used only a subset of it, handling part of the transition to the spare reservation from userspace; however in the experiments we made sure that the mapping of the VMs to the reservation was compatible with the described approach.

In order to show the advantages of the technique, the ping times for reaching the VM have been measured under various conditions (so, the ping time is representative of the responsiveness of the VM), while a fake compute-intensive workload was used inside the VM, using a `throughput` utility that has the capability to measure how many repetitions of a basic for loop with a few arithmetic operations have been realized over a time horizon. Note that a ping packet only reaches the kernel-level network driver of the target VM (which runs at higher priority as compared to user-space computing applications). The evaluation of the technique with real user-space applications (e.g., a webserver that needs to remain responsive) is deferred as future work on the topic.

In the experiment, the potential of the mechanism is highlighted by measuring the worst-case responsiveness of the system, under the assumption of sporadically interspaced, non-enqueuing ping requests, while the VM is under heavy compute-intensive workload. This has been achieved running the `throughput` utility inside the VM, attaching it to a reservation with scheduling parameters $(Q, P) = (40ms, 100ms)$, and by using a spare reservation configuration of $(Q_s, P_s) = (4ms, 10ms)$. Also, in order to evaluate the worst-case latency experienced by ping, the VM was pinned on the first physical core of the host, while a user-space tool, pinned on the other core, was used to spin-wait for budget exhaustion of the associated reservation, and issue a ping request at that time. As highlighted in Section 2, the minimum observed ping time is theoretically $P - Q = 60ms$ in this case (but far higher values were observed, actually). However, the mechanism introduced in this paper foresees the attachment of the spare reservation to the VM at the ping packet receive time, thus the VM has a chance to run for $Q_s = 1ms$ within the deadline of $P_s = 10ms$ (and for an additional $1ms$ for each subsequent $10ms$ time window, till the replenishment of the original reservation budget), thus responding to the request much more quickly.

The obtained ping times with the VM running under the real-time scheduler are shown in Figure 4. As it can be seen when using the spare reservation (bottom curve) mechanism, the experienced ping times are highly reduced as compared to when not using it (top curve).

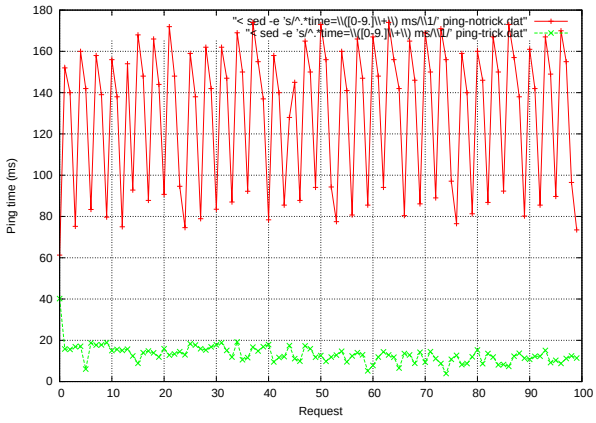

FIGURE 4: *Obtained ping times without the additional spare reservation (top curve) and with the spare reservation (bottom curve).*

Also, looking at the throughput that can be achieved by the batch computing activities inside the VM with various equivalent reservation configurations (in terms of occupied CPU share), we can observe that with a reservation of $(40ms, 100ms)$ our program was reporting 1.11 cycles per microsecond, while with a reservation of $(4ms, 10ms)$ it was reporting 0.56 cycles. The big difference is due to the additional scheduling overheads due to the ten times more context switches. Therefore, it is highly beneficial to keep the VM configured with the longer period, in this case, while our mechanism allows to greatly improve its responsivenes.

5 Conclusions and Future Work

In this paper we present a novel scheduling mechanism to provide efficiently a tight responsiveness to virtual machines hosting mixed compute-intensive and real-time workloads. It is possible to schedule such VMs with a reservation-based scheduler by using a large period for minimum overheads during compute-intensive periods, but at the same time ensure that the VM responds within a much shorter deadline when receiving input from the outside, as in the case of receiving a network packet.

The presented mechanism can still be improved, and various directions for future extension are possible. Firstly, the mechanism may be improved to shorten the response-time of the VM also during the periods in which its own reservation has still budget, but the deadline is still quite far away. This may be seen in workloads where the other VMs in the system have deadlines shorter than the one of the VM receiving the packet, but still quite far away as compared to the desired tightness of the real-time activity response. Second, the current implementation is only a proof-of-concept and needs to be better engineered to reach production quality levels. Third, the idea of the pool of spare reservations has only been sketched out, but it needs to be refined, implemented and experimented on a real system. Finally, the presented mechanism needs to be applied to some real-life workload in order to highlight its full potential for real application scenarios.

References

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

[2] F. Checconi, T. Cucinotta, D. Faggioli, and G. Lipari. Hierarchical multiprocessor CPU reservations for the linux kernel. In *Proceedings of the* 5^{th} *International Workshop on Operating Systems Platforms for Embedded Real-Time Applications (OSPERT 2009)*, Dublin, Ireland, June 2009.

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

[4] T. Cucinotta, G. Anastasi, and L. Abeni. Respecting temporal constraints in virtualised services. In *Proceedings of the* 2^{nd} *IEEE International Workshop on Real-Time Service-Oriented Architecture and Applications (RTSOAA 2009)*, Seattle, Washington, July 2009.

[5] T. Cucinotta, G. Anastasi, F. Checconi, D. Faggioli, K. Kostanteli, A. Cuevas, D. Lamp, S. Berger, M. Stein, T. Voith, L. Fuerst, D. Golbourn, and M. Muggeridge. Irmos deliverable: D6.4.2 final version of realtime architecture of execution environment. Available on-line at: `http://www.irmosproject.eu/Deliverables/Default.aspx`., 1 2010.

[6] T. Cucinotta, D. Giani, D. Faggioli, and F. Checconi. Providing performance guarantees to virtual machines using real-time scheduling. In *Proceedings of the* 5^{th} *Workshop on Virtualization and High-Performance Cloud Computing (VHPC 2010)*, Ischia (Naples), Italy, August 2010.

[7] G. Dunlap. Scheduler development update. Xen Summit Asia 2009, Shanghai, 11 2009.

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

[9] G. Liao 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.

[10] C. L. Liu and J. Layland. Scheduling alghorithms for multiprogramming in a hard real-time environment. *Journal of the ACM*, 20(1), 1973.

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