Real-Time Virtual Machines*

Tommaso Cucinotta, Gaetano Anastasi
Scuola Superiore Sant'Anna
Pisa, Italy
{cucinotta,anastasi}@sssup.it

Luca Abeni
Università di Trento
Trento, Italy
luca.abeni@unitn.it

Abstract

This paper tackles the problem of guaranteeing appropriate timeliness guarantees to real-time applications running in a virtualised Operating System. Preliminary experimental results are presented, highlighting how the appropriate use of well-established real-time scheduling strategies may be effective in facing with this challenging issue.

1 Introduction and related work

Information and Communications Technologies (ICTs) are moving towards a new generation of business models for service provisioning, where connectivity will represent, for users, only the basic point of access to an infinity of on-line services. The widespread availability of high-speed network connections puts the foundations of new paradigms of ICT usage and software development, where more and more of the resources needed by users are provisioned remotely. In the (near) future Internet, distributed computing is likely to become much more widespread than today, not only for activities related to batch processing and storage, but also for interactive and (soft) real-time applications. Application scenarios that can benefit of remote real-time processing include, just to cite a few examples, virtual reality (that need to carry on heavyweight physics simulations or complex rendering activities), or interactive distributed editing of high definition video as needed in the film post-production industry.

A promising approach for building complex distributed applications is constituted by Service Oriented Architectures (SOAs), which are software infrastructures that allow for the composition of loosely coupled, distributed services in a location-independent manner.

Distributed applications, and particularly SOAs, are taking advantage of the recent rediscover of resource virtualisation [6], whose early works date back to 1967, with some IBM projects. Virtualisation[1] basically refers to the technology that allows a system to host one or more emulated other systems, called Virtual Machines (VMs), which may also be seamlessly migrated across physical hosts. Thanks to the encapsulated nature of VMs and their simple interface to the outside world, it is easy to use, in the context of a distributed network, techniques that are commonly applied to tasks, like migration, load balancing and redundancy. Software components may transparently benefit of these capabilities without any need of explicit code level support.

Virtualisation technologies are also useful in some networking testbeds, such as PlanetLab[2], a distributed testbed using VMs to increase scalability, resource confinement and protection. In all the cited application fields it is of paramount importance to control the impact of the virtualised environment on the applications' temporal behaviour: in the SOA environment, it affects the predictability of hosted service and thus the capability of providing service level guarantees; in testbeds it affects reliability of performance measurements, as the ratio between the speed of the native hardware and the speed of the emulated environment is usually not constant nor predictable.

Although the problem of controlling the temporal behaviour of VMs has been previously addressed in various works, none of them achieved the level of determinism needed to run real-time tasks inside a VM.

For example, Xen [2] uses a reservation-like scheduler (based on EDF) to enforce temporal isolation between the different VMs. However, the proposed S-EDF scheduler lacks a solid theoretical foundation, and some problems have been experienced with it, in particular when Xen is used for partitioning the available resources between different activities in GRID computing [7].

Similar problems have been investigated in PlanetLab, where the concurrent execution of multiple VMs, without an appropriate scheduling strategy, led to a non-deterministic view of time, thus to unreliable measurements, from inside a VM (called *slice* in this case). The PlanetLab architecture [3] tries to address this problem by combining a proportional share scheduler with a mecha-

*This work has been supported by the IRMOS FP7/2008/ICT/214777 European Project.

[1] http://en.wikipedia.org/wiki/Virtualization

[2] More information at the URL: http://www.planet-lab.org

nism that limits the maximum amount of time executed by each VM. However, additional experiments [4] show that the scheduler used in PlanetLab is not able to fully isolate the temporal behaviours of the various slices, and the authors propose to implement a hard reservation mechanism.

This paper presents preliminary results in this challenging research area, showing that a well-known reservation-based scheduler is capable of guaranteeing appropriate response times to individual activities running inside a VM.

2 Virtualisation and Real-Time

The term *virtualisation* refers (in this paper) to the capability, for a computing machine (referred to as the *host*), to emulate the behaviour of one or multiple computing machines (the *guests*), in such a way that any software capable of running on the raw hardware may also seamlessly run within the emulated machine.

In a virtualised environment, multiple activities may be hosted on the same physical hardware in different ways. They may run in different VMs that are multiplexed on the same bare hardware (*inter-VM scheduling*), or they may coexist within the same VM where a OS-level scheduler multiplexes them on the same virtualised hardware (*intra-VM scheduling*), and other VMs may possibly be running concurrently on the same physical node. In all cases, appropriate inter-VM and intra-VM scheduling mechanisms are needed to guarantee that the individual activities exhibit the expected Quality of Service (QoS) levels, whenever timeliness requirements are in place.

For developers and designers of time-sensitive software components, virtualisation adds a set of new challenging issues that need to be addressed by research.

First, new methodologies are needed to correctly account for the impact of the virtualisation overhead on the execution time of real-time tasks, especially in presence of virtualised peripherals, that turn I/O intensive activities into CPU intensive ones (e.g., networking). The capability to migrate VMs on different types of hardware adds complexity to the problem. For example, in the context of SOAs, it is necessary to foresee how a software component would perform if deployed on various physical nodes, in order to choose the optimum deployment that provides the performance promised in the Service Level Agreement (SLA). Common approaches based on direct measurement of the execution time distribution on the target hardware, is not sufficient. A hardware-independent characterisation of the execution times, plus a hardware-specific model of the variability of execution times, may be needed.

Second, when the OS is being hosted along with other OSes concurrently running, the time measured in a VM may be discontinuous, or have a strange granularity. Timer devices are emulated by the virtualisation engine, and their resolution may be dramatically affected by the inter-VM scheduler and timer virtualisation mechanism (affecting the precision and granularity of timers and clocks). The progress rate of a virtualised OS is also not as uniform as expected, due to inter-VM scheduling, and this may have high impact on the response time of virtualised software components.

Finally, multi-processor and multi-core platforms add a new dimension to the problem of real-time virtualised computing. Software components written for high performance parallel machines may not run as expected when executing within a virtualised environment, especially if multiple multi-core VMs are running concurrently. For example, spin-lock synchronisation primitives (that usually rely on the assumption that the lock owner running on a different processor will release the lock in a short time) may cause problems if the virtualisation layer schedules away the VM owning the lock. Suitable mechanisms are needed to mitigate such issues. For example, the VMWare ESX Server[3] embeds mechanisms to address such issues, but more investigations are needed to understand what solutions are most suitable for meeting real-time application requirements.

3 The proposed approach

As outlined in the previous sections, the problem of providing temporal isolation among multiple VMs running in the same host is still open. Simple solutions based on proportional share schedulers may fail to guarantee a sufficient degree of isolation, and do not generally provide enough control over the granularity of the CPU allocation to the various VMs.

The approach envisaged in this paper is to use well-established real-time scheduling techniques, and in particular resource reservations, for scheduling the VMs. Such techniques allow to attach a software component (a VM, in this case) to a *reservation* $RSV = (Q, P)$, meaning that the component is reserved the processor for Q time units every P. A reservation-based scheduler is said to implement a *hard* reservation behaviour if it does not allow the served component to execute for more than Q time units every P. The scheduler used in this paper is a variation of the Constant Bandwidth Server (CBS) [1] implementing hard reservations. For the sake of simplicity, this work focuses on uniprocessor systems[4].

The proposed approach suits the needs of concurrently running VMs, because it allows both to control the "progress rate" that each VM experiences over time and to provide deadline guarantees for the tasks running inside the VM. In fact, once a VM runs within a reservation (Q, P),

[3] See "Co-scheduling SMP VMs in VMware ESX Server, version 3" at http://communities.vmware.com/docs/DOC-4960

[4] Investigation on the opportunity to realize a partitioned SMP scheduler for VMs by using multiple CBS schedulers is work in progress.

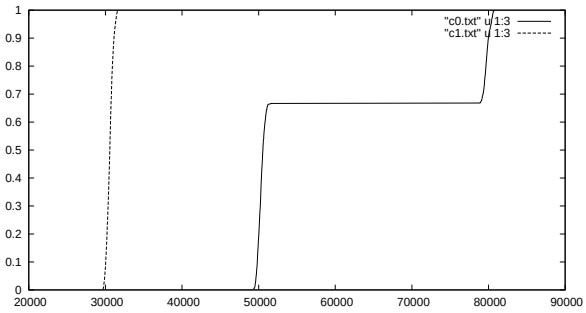

Figure 1. CDF of the response times on the host machine.

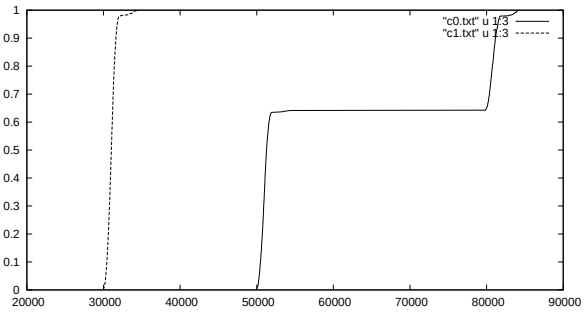

Figure 2. CDF of the response times on the guest (KVM), when the host is unloaded.

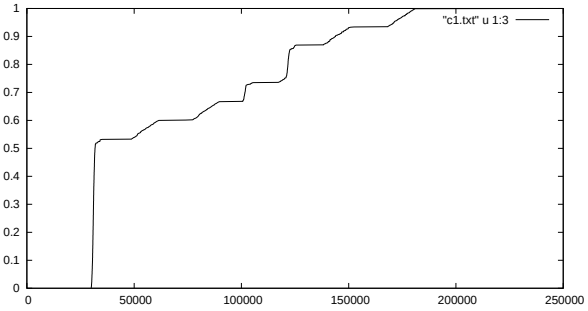

Figure 3. CDF of the response times on the guest (KVM), when the host is loaded.

the ratio Q/P allows to control the speed of the virtual CPU, and the reservation period P allows to control the granularity of the allocation so that the temporal constraints can be respected.

Although this paper focuses on statically assigned reservations, feedback scheduling techniques can be used to adapt the time reserved to each VM to compensate for time-varying workloads experienced by software components running in the VMs.

4 Experimental results

The effectiveness of resource reservations in increasing predictability of virtualised software components has been tested by running KVM[5] in AQuoSA [5], a framework for the Linux kernel that embeds a hard CBS scheduling policy.

4.1 Micro Benchmarks

A first set of experiments has been run to verify that scheduling a VM through a hard CBS allows for the achievement of predictable scheduling inside the host system. This goal has been achieved by running a task set $\mathcal{S} = \{\tau_1, \tau_2\}$ composed of 2 periodic real-time tasks that have been tuned so as to exhibit, at each activation, an execution time as constant as possible: $\tau_1 = (30ms, 150ms)$ (using the notation $(ExecutionTime, Period)$) and $\tau_2 = (50ms, 200ms)$. Then, the experimental Cumulative Distribution Functions (CDF) $C_i(x) = P\{\rho_i < x\}$ of the response times ρ_i have been measured.

Figure 1 plots the response times CDFs for the two tasks when they are scheduled on the host machine using the POSIX `SCHED_FIFO` scheduling policy and priorities assigned according to Rate Monotonic: τ_1's worst case response time is a little bit larger than $30ms$ (the difference with respect to the ideal time of $30ms$ is due to a small variation in the execution times), whereas τ_2's worst case response time is a little bit larger than $80ms$, as expected from response time analysis.

[5] More information at the URL: `http://kvm.qumranet.com`

The response times of the two tasks are only marginally affected when the task set $\mathcal{S}$ is executed inside a KVM instance and the host system is not overloaded (see Figure 2), but when the load on the host system increases the response times are not predictable. For example, Figure 3 plots the response times CDF for task τ_1 when the two tasks are executed in a KVM instance and the host is overloaded by two periodic real time tasks $\tau_3 = (50ms, 120ms)$ and $\tau_4 = (20ms, 80ms)$. Note that τ_1 and τ_2 are still executed at real-time priorities inside the guest, but KVM is executed as a regular Linux task inside the host, while τ_3 and τ_4 are scheduled with real-time priorities in the host. As a result, the worst case response time for τ_1 is about $180ms$ and τ_1 misses many deadlines; moreover, there is no upper bound for the response times of task τ_2.

This problem can be addressed by reserving a sufficient amount of time to the KVM instance: Figure 4 shows the results obtained when running $\mathcal{S}$ inside a KVM instance attached to a $(28ms, 50ms)$ hard reservation. Thanks to the temporal isolation provided by the CBS, increasing the load in the host does not affect the response times: when the host is loaded by the same periodic real-time tasks used in Figure 3, the CDF of the response times is still identical to Figure 4.

Finally, note that a simple hierarchical scheduling anal-

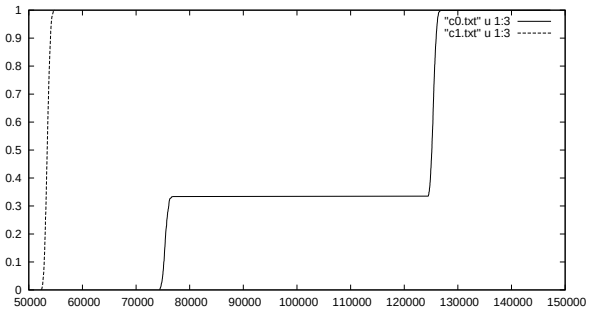

Figure 4. CDF of the response times on the guest served by a $(28ms, 50ms)$ reservation.

ysis shows that a $(27ms, 50ms)$ reservation should be theoretically able to properly serve the real-time tasks hosted in the virtual machine. However, due to some overhead introduced by the emulator, a $(28ms, 50ms)$ reservation is needed in practice.

4.2 Macro Benchmarks

After verifying the schedule predictability through a synthetic workload, the proposed approach has been evaluated by using a real application. In particular, the Apache 2 web server has been selected as a representative of a typical SOA workload. In this experiment, a set of 10 clients requested a dynamic web page, generated by using a CPU-bound CGI process which rotates of an image of 2000x2000 pixels by an angle $\alpha = 20°$. Each one of the 10 clients generates 10 requests, for a total of 100 requests per simulation, and each simulation has been repeated 20 times.

The first column pair of table 1 reports statistics on the response times of the service concerning two cases: (a) the service is provided by the web server running on the host machine; (b) the service is provided by the web server running inside a KVM instance. The table reports the minimum, maximum, and average response time per request, plus the standard deviation of such a value. The 90% confidence interval is 0.2% of the average value for the web server running on the host, and 1.3% of the average value for the web server running on the guest. It can be seen that the overhead due to virtualisation is 7% for average times and 6.4% for maximum times. As this overhead does not affect service response times in a tangible way, it makes sense to exploit all the benefits of virtualisation for this type of tasks.

However, the situation changes drastically when the host system is overloaded. In fact, the third column of table 1 shows how response times obtained in the guest (the 90% confidence interval is 0.4% of the value) increase when the host is put through a synthetic load that tends to saturate CPU bandwidth. Note that, respect to the case of the KVM instance running in an unloaded host, service response times increase by a factor of 10. Moreover, the standard deviation value is quite large, to indicate that fluctuations from average values often occur. This issue is particularly critical in SOA environments, where it is often necessary to provide guarantees in service provisioning: such fluctuations do not allow for precise estimations of service response times, what precludes the possibility for a provider to share the same physical node for multiple VMs that need to exhibit precise QoS levels.

	Host (unloaded)	**Guest** (unloaded)	**Guest** (loaded)	**Guest-rsv** (loaded)
min	0.22	0.26	1.153	0.790
avg	1.14	1.22	11.367	2.044
max	7.91	8.42	89.880	10.832
std.dev	1.26	1.07	15.449	1.275

Table 1. Service response times (in seconds)

As already shown by the micro benchmarks, these problems can be addressed by reserving a proper amount of execution time to the KVM instance. The fourth column of table 1 reports service response times obtained by running the web server inside a KVM instance attached to a $(3ms, 5ms)$ hard CBS. In this case, the 90% confidence interval is 0.9% of the value. The results show how the response times scale to values much closer to that of the first column pair of table 1, even when the host is overloaded. This fact, due to the temporal isolation property provided by the CBS, is particularly remarkable because it could allow service providers to offer services with QoS guarantees.

In the nearest future work, there is a plan to consider a more complex set-up, with multiple VMs, each hosting multiple time-sensitive services, and with an overall network traffic generated by the VMs that needs to be appropriately accounted for.

References

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

[2] P. Barham, et al. R. Neugebar, I. Pratt, and A. Warfield. Xen and the art of virtualization. In *SOSP*, 2003.

[3] A. Bavier, others L. Peterson, T. Roscoe, T. Spalink, and M. Wawrzoniak. Operating system support for planetary-scale services. In *NSDI Design and Implementation (NSDI)*, March 2004.

[4] A. Bavier, others In VINI veritas: Realistic and controlled network experimentation. In *SIGCOMM*, 2006.

[5] T. Cucinotta et al. AQuoSA – adaptive quality of service architecture. *Software – Practice and Experience*, 2008.

[6] P. A. Dinda et al. Resource virtualization renaissance. *Computer*, 38(5):28–31, May 2005.

[7] T. Freeman, I. T. Foster, et al. Division of labor: Tools for growing and scaling grids. In *ICSOC*, pages 40–51, 2006.