Respecting temporal constraints in virtualised services*

Tommaso Cucinotta, **Gaetano Anastasi**
Scuola Superiore Sant'Anna
Pisa, Italy
{t.cucinotta,g.anastasi}@sssup.it

Luca Abeni
DISI - University of Trento
Trento, Italy
luca.abeni@unitn.it

Abstract

This paper reports some experiences in providing service guarantees to real-time (RT) applications running in a Virtual Machine (VM), showing how proper scheduling is a necessary condition for a predictable execution. In particular, resource reservation techniques allow to cope with some of the overhead and unpredictabilities experienced when executing multiple VMs on the same host.

1 Introduction

The high availability of high-speed Internet connections at affordable rates and the widespread usage of mobile devices are driving the world of Information and Communications Technologies (ICTs) towards a new era of distributed computing, where more and more of the resources needed by a user are provided remotely. In order to account for this shift in resource provisioning, new paradigms of software design and development are needed. New business models are emerging, where resource providers may give on-line access to not only storage, but also computation and communication resources, while service providers may use them to offer sophisticated composable high-level services or even ready-to-use distributed applications to end users.

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. Recently, SOAs are taking advantage of the rediscover of virtualisation [9], a technology that allows resource providers to sell virtual resources, that may be allocated to the available physical resources, so as to scale down costs due to their management over a large customers base. Also, particularly interesting in this context is the possibility to share the same physical resources (computing nodes and links) across multiple Virtual Machine (VMs) concurrently running, in a seamless manner for the applications running within.

Unfortunately, whenever multiple VMs are hosted on the same node, the temporal interferences among them become hardly controllable, and the QoS experienced by the hosted applications may exhibit uncontrollable fluctuations.

This is very undesirable in the context of SOAs, where there is an increasing interest in enriching the SLA established between resource providers and consumers with such attributes as QoS constraints, that are essential for deployment of professional services with guaranteed levels of service and/or high interactivity levels, and penalties for the provider if such constraints are not respected.

Contributions of the paper This paper shows how mechanisms designed according to the theory of hierarchical real-time systems may be used in order to greatly enhance predictability of the temporal behaviour of virtualised software components. The paper provides a minor improvement over existing schedulability test for hierarchical real-time systems. Then, experimental results are presented in which resource reservation scheduling techniques are applied for the purpose of guaranteeing temporal isolation of multiple virtual machines (KVM[1]) within a Linux host OS.

2 Related Work

The need for real-time support within SOAs is witnessed by the RTSOA paradigm recently appeared [22, 15], and by the increasing interest in real-time service provisioning within the Grid community [8], just to mention a few. Unfortunately, most of the works in these directions the issue of how to do not consider time-shared nor virtualised nodes. Dinda et al. [9] proposed the use of time-shared systems, but their work did not address the issues concerned with

*The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/ICT/2008) under grant agreement n.214777, in the context of the IRMOS Project.

[1] Kernel-based Virtual Machine: http://kvm.qumranet.com.

low-level real-time scheduling algorithms. Steps in this direction have been moved by Almeida et al. [3], who applied real-time scheduling theory to the problem of guaranteeing temporal guarantees to distributed applications built as a network of composable services. However, their work addressed the distribution issue, while this paper focuses on node-level mechanisms that guarantee correct scheduling of concurrent RT services within the same physical host.

The latter problem has been attacked in some previous work, but the level of determinism needed to run real-time applications inside a VM has not been reached yet.

For example, Xen [4] uses an EDF-based reservation mechanism (called S-EDF) to enforce temporal isolation between the different VMs. However, the S-EDF scheduler lacks a solid theoretical foundation, and is not guaranteed to work correctly in presence of dynamic activations and deactivations. As a result, it seems to have problems in controlling the amount of CPU allocated to the various domains. In fact, in [11], it is shown that the Xen scheduler is not able to properly control CPU allocations for I/O intensive operations.

Other problems related to VM scheduling have been investigated in PlanetLab [17], a distributed testbed using VMs to increase scalability. PlanetLab [5] tries to address this problem by combining a proportional share scheduler with a mechanism that limits the maximum amount of time available for each VM. However, additional experiments [6] show that the scheduler used in PlanetLab is not able to fully isolate the temporal behaviours of the various VMs, and the authors propose to implement hard reservations.

If virtual machines are scheduled using proper real-time algorithms, the system can be modelled as a hierarchy of schedulers, and its real-time performance can be evaluated by using hierarchical scheduling analysis techniques. For example, Saewong and Rajkumar extended the so called *resource reservation* (see Section 4) to support hierarchical reservations [19]. Shin and Lee proposed a different approach based on a compositional real-time scheduling framework [20], where the timing requirements of complex real-time components are analysed in isolation and subsumed into an abstract specification called *interface*, then combined to check schedulability of the overall system.

Mok and others [16, 10] presented a general methodology for hierarchical partitioning of a computational resource, where schedulers may be composed at arbitrary nesting levels. Specifically, they associate to each resource partition a *characteristic function* that identifies, for each time window of a given duration, the minimum time that the processor is allocated to the partition. On the other hand, Lipari and Bini [13] addressed the problem of how to optimally tune the scheduling parameters for a partition, in order to fulfil the demand of contained real-time task sets. The latter techniques will be applied in Section 4 to analyse the schedulability of real-time tasks running in a VM, and to compute suitable scheduling parameters at the root scheduling level.

3 Problem presentation

In this paper, the term *virtualisation* refers to the ability, 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 host may also seamlessly run within the emulated machine.

A host is modelled as a set of guest VMs $\{VM^k : k = a, b, \ldots\}$ scheduled by a *global scheduler*[2]. All the tasks $\tau_i^k \in \mathcal{T}^k$ are scheduled by a *local scheduler* running in VM^k. Such a scheduling system is denoted from here on by the X/Y notation, where X denotes the root scheduling strategy, while Y denotes the local scheduling strategy.

Each VM VM^k is modelled as a real-time system composed by a set $\mathcal{T}^k$ of real-time tasks, with $\mathcal{T}^k = \{\tau_i^k : i = 1, 2, \ldots\}$. Each task is a sequence of jobs $J_{i,j}^k$, characterised by an arrival time $r_{i,j}^k$, an execution time $c_{i,j}^k$, a finishing time $f_{i,j}^k$ and an absolute deadline $d_{i,j}^k$. For the sake of simplicity, only periodic RT tasks[3] $\tau_i^k = (C_i^k, T_i^k)$ are considered, with Worst Case Execution Time (WCET) $C_i^k = max_j\{c_{i,j}^k\}$ and $d_{i,j}^k = r_{i,j+1}^k = r_{i,j}^k + T_i^k$.

If the X/Y scheduling system is not properly designed, the temporal behaviour of the guest is hardly predictable. For example, consider the periodic task set $\mathcal{T} = \{\tau_1 = (30ms, 150ms), \tau_2 = (50ms, 200ms)\}$: on real hardware, real-time scheduling theory guarantees that, if tasks are scheduled with fixed priorities assigned according to Rate Monotonic (RM) [14], then all the deadlines are respected. This has been verified by running $\mathcal{T}^k$ on a real PC (with execution times forced as equal as possible to the mentioned WCET values). The left side of Figure 1 shows the Cumulative Distribution Function (CDF) $C(x) = P\{\rho_i < x\}$ of the response times $\rho_{i,j} = f_{i,j} - r_{i,j}$ of the two tasks, showing that $C(x)$ arrives to 1 before the deadline of the task, hence all the deadlines are respected.

When the same task set $\mathcal{T}$ is run inside a VM (in this example, KVM [12] on Linux has been used, as described in the following), most of the deadlines are easily missed, as shown in the right side of Figure 1. Such a behaviour occurs every time a general-purpose scheduler is used to schedule concurrent VMs, because of the unpredictability of the temporal interferences that each VM experiences due

[2]The global scheduler may either be implemented in a host OS so to perform inter-VM scheduling (e.g., the KVM [12] approach), or it may be implemented in the virtualisation layer (i.e., the Xen approach).

[3]Note that the techniques and results described in this paper can be extended to sporadic real-time tasks, and to tasks with relative deadline different from the period.

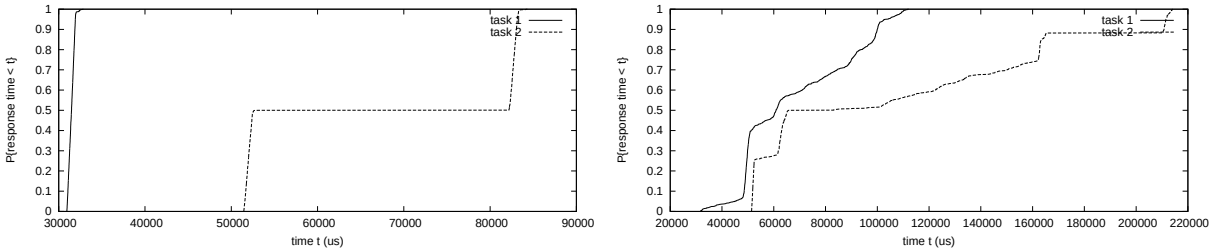

Figure 1. CDF of the response times of tasks in $\mathcal{T}$ executed on real hardware (left) and KVM (right).

to the behaviour of the other VMs on the same physical host. In the example in Figure 1, the two VMs were scheduled by using the standard completely fair scheduler currently present in the Linux kernel.

4 Scheduling

The just exposed problem may be faced with by adopting an appropriate soft real-time scheduling strategy among the VMs running in the host. A first approach could be to schedule the various VMs with fixed real-time priorities. However, this solution can be problematic, because if a higher priority VM consumes more than expected, it can stall the lower priority VMs.

An alternative approach is based on *resource reservations*, which allow to reserve a hardware resource (the CPU, in this case) to a task for Q time units (a.k.a., *reservation budget*) in every window of P time units (a.k.a., *reservation period*). *Hard* [18] reservations guarantee that no more than Q time units every P are allocated to the task, whilst *soft* reservations do not pose this restriction.

Although this abstraction can be very effective for serving real-time virtual machines, it is important that the reservation mechanism be designed so as to correctly cope with aperiodic activations. Most of the reservation techniques previously used to schedule VMs exhibit the same behaviour of a Deferrable Server [21] (also the CPU throttling mechanism available in the Linux kernel exhibits such behaviour), which is well-known for the restrictions that it imposes on the schedulability of tasks that activate and deactivate dynamically [2]. Moreover, *hard* reservations are more appropriate for scheduling VMs: in fact, most of the hierarchical scheduling analysis for reservation-based systems is based on the assumption that a reservation provides *exactly* Q time units every T time units, and using a hard reservation algorithm is the easiest way to enforce this requirement.

For these reasons, this paper focuses on a variant of the **CBS** algorithm [1], which has a strong theoretical foundation in the area of real-time scheduling, and can easily cope with aperiodic arrivals. Actually, a **hard reservation behaviour** variant of the CBS has been used, as implemented by the same authors in the AQuoSA framework [7] for the Linux kernel. This mechanism has been used for providing temporal isolation among concurrently running VMs, while gathering the experimental data reported in Section 5.

With proper real-time scheduling algorithms at the guest and in the host, it is possible to apply well-known schedulability techniques to the analyse the system, as shown next.

Fixed Priority (FP) inter-VM scheduling If the global scheduler is a fixed priority scheduler (so, the hierarchy is FP/FP) giving priority to VM^a over VM^b, to VM^b over VM^c, etc..., then every task of $\mathcal{T}^a$ has priority over all the tasks of $\mathcal{T}^b$, $\mathcal{T}^c$, etc.... As a result, the system behaves as if the tasks from all VMs are globally scheduled through FP, but with an equivalent global priority assignment that may generally differ from the optimum Rate Monotonic (RM) one. This means, from a real-time schedulability perspective, that the overall system utilisation for real-time tasks, respecting schedulability, will be lower, and response times will be higher, than if a global RM assignment were used.

For example, if VM^a has priority over VM^b, then τ_i^a has priority over τ_j^b even if $T_i^a > T_j^b$.

In such case, response times R_i^k of the tasks τ_i^k may be computed by solving the well-known implicit equations:

$$R_i^k = C_i^k + \sum_{j<i} \left\lceil \frac{R_i^k}{T_j^k} \right\rceil C_j^k + \sum_{h<k} \sum_{\forall j} \left\lceil \frac{R_i^k}{T_j^h} \right\rceil C_j^h. \quad (1)$$

Then, schedulability of the task set is verified by checking that the obtained response times are below the assigned relative deadlines.

Reservation-Based inter-VM scheduling If a CBS/FP hierarchy is used and VM^k is scheduled through a reservation $RSV^k = (Q^k, P^k)$, then the response time R_i^k only depends on RSV^k and on the higher priority tasks $\tau_j^k : j < i$ contained in $\mathcal{T}^k$.

This kind of systems can be analysed by using the technique shown in [13], based on the extension of well-known results for schedulability analysis of non-hierarchical systems. For fixed priority scheduling of tasks inside a VM

VM^k served by a reservation $RSV^k = (Q^k, P^k)$, schedulability is guaranteed if and only if:

$$\forall i \, \exists t \in \mathcal{P}^k : C_i^k + \sum_{j<i} \left\lceil \frac{t}{T_j^k} \right\rceil C_j^k \leq Z^k(t), \qquad (2)$$

where tasks are ordered by decreasing priority, $Z^k(t)$ is a *characteristic function* indicating the amount of time dedicated to the VM by the root scheduler, and $\mathcal{P}^k$ is a set of appropriate *scheduling points*.

This paper also proves that checking Condition (2) with the original sets of scheduling points $\mathcal{P}^k$ as described in [13], constitutes only a *sufficient* condition for schedulability of the tasks inside the VM, but unfortunately it is not *necessary*, as claimed in the original paper. In fact, generating 100000 task sets randomly, and verifying schedulability for all points up to each task deadline, it has been found that 1189 task sets were actually schedulable, but only 1009 passed the test as presented in [13]. Therefore, nearly 16% of the task sets would have been rejected by the test, but they are actually schedulable.

The reason for which this happens is that the test has been adapted from a prior test that was relying on a perfectly linear shape for the $Z^k(t)$ function, but in the adaptation made in [13] the slope changes of the $Z^k(t)$ have not been considered. Therefore, it is proposed that the technique be fixed by adding to $\mathcal{P}^k$ also the points of slope change of $Z^k(t)$ before the deadline of the task being checked.

The test in Equation 2 with the enriched sets $\mathcal{P}^k$ as just described should constitute again a necessary and sufficient condition for schedulability of the task set. In fact, in the example above, the modified test managed to successfully identify all of the schedulable task sets. However, a formal proof is reserved for future work.

Comparison of the approaches To compare the schedulability of the different scheduling hierarchies, a large number of systems composed by multiple VMs has been randomly generated (every VM VM^k contains a randomly generated task set $\mathcal{T}^k$) and their schedulability has been checked by using the analysis techniques described above. The schedulability for CBS/FP has been tested by using Equation 2 to compute the set of reservations $RSV^k = (Q^k, P^k)$ that allow to properly serve each VM VM^k, and by checking if $\sum_k Q^k/P^k \leq 1$.

The results of this experiment indicate that 88.4% of the generated systems are schedulable with CBS/FP, and only 72% of them are schedulable with FP/FP.

5 Experimental Results

The approach presented in the previous sections has been tested by first verifying that it allows to respect temporal constraints, and then evaluating how it allows to control the performance in a SOA environment.

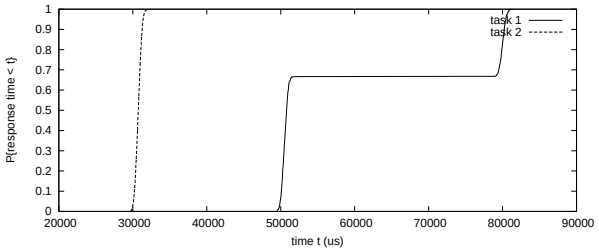

Figure 2. CDF of the response times on VM^a when the host is unloaded.

Respecting Temporal Constraints Some randomly generated task sets have been run in some VMs (executing in parallel), and the results expected from the theoretical analysis have been compared with these experimental results. Here it is reported an example with a simplified system (composed only by two task sets) that has been designed to be easily understandable. The example system is composed by two sets of periodic tasks $\mathcal{T}^a = \{\tau_1^a = (30ms, 150ms), \tau_2^a = (50ms, 200ms)\}$ and $\mathcal{T}^b = \{\tau_1^b = (30ms, 120ms), \tau_2^b = (40ms, 240ms)\}$ scheduled in two KVM guests VM^a and VM^b. To simplify the analysis, all the tasks have offset equal to 0 (that is, $r_{1,1}^a = r_{2,1}^a$ and $r_{1,1}^b = r_{2,1}^b$). The tasks are scheduled with fixed priorities (assigned according to RM) inside their VMs (using the `SCHED_FIFO` scheduling policy).

Figure 2 shows the CDF of the response times $\rho_{i,j}$ for the two tasks running in VM^a when the host is unloaded. The worst case response times for the two tasks are about $R_1^a = 31ms$ and $R_2^a = 81ms$ (the unexpected increase of $1ms$ in the response times is due to the accuracy of the timers in the host and in the guest); hence, all the deadlines are respected (for the sake of brevity, the same figure for VM^b is omitted, but it shows similar results).

However (as already shown in Section 3), if a VM is not properly scheduled then the real-time tasks running in it can easily miss their deadlines: in fact, when the two virtual machines are simultaneously executed on the same host, the real-time tasks are not able to respect their deadline, as shown in Figure 3. Scheduling each VM through hard CBS servers allows to solve this problem: for example, when VM^a is scheduled through a CBS $RSV^a = (28ms, 50ms)$[4] and VM^b is scheduled through a CBS $RSV^b = (52ms, 120ms)$, all the tasks are able to respect all their deadlines. This happens even if the two VMs

[4] A schedulability analysis of the schedulers hierarchy allows to see that a $(27ms, 50ms)$ reservation would be sufficient for VM^a, but the reserved time has been increased to take in account the virtualisation overhead, and the execution of KVM code.

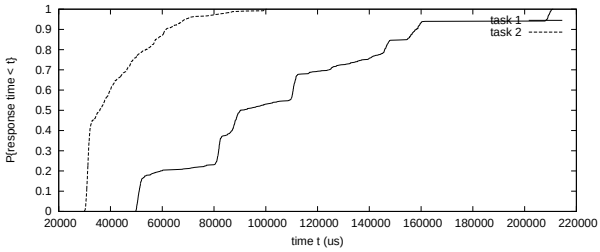

Figure 3. CDF of the response times on VM^a when it is scheduled together with VM^b.

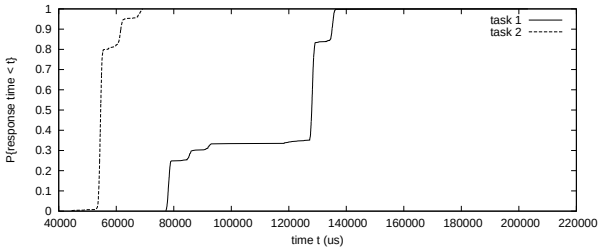

Figure 4. CDF of the response times on VM^a with a $(28, 50)$ reservation when it is scheduled together with VM^b.

are executed concurrently, as shown in Figure 4 (the figure shows the CDF for VM^a, but similar results can be obtained for VM^b too).

Controlling the Performance Being able to provide accurate estimations of the response times is very important in a SOA environment, which is moving away from the old best-effort Internet model. In fact, a service provider has to take QoS into account, for example in order to meet business policies or because QoS guarantees are required by consumers. Hence, the proposed approach has been evaluated in a typical SOA scenario, with multiple web servers executed in different VMs. These experiments use two VMs VM^a and VM^b running an Apache 2 web server and some CGI scripts for rotating large images (a computation intensive task). Two kinds of requests are performed by using the Apache 'ab' program: $req1$ (consisting in the rotation of a 1000x1000 image by an angle $\alpha = 20°$) and $req2$ (consisting in the rotation of a 2000x2000 image by an angle $\alpha = 20°$).

To reproduce a realistic scenario, each VM has been put through a different workload, obtained by varying the number of concurrent clients. In particular, VM^a has been tested in serving 10 concurrent clients and VM^b has been tested in serving 20 concurrent clients. Half of the clients of each VM performed 10 requests for the $req1$ service, and the other half performed 10 requests for the $req2$ service.

When each VM is executed alone on the host, VM^a has an average service time of $0.768s$ and a maximum service time of $7.7s$, with a standard deviation of $1.547s$. The service time for VM^b has an average of $1.603s$, a maximum value of $14.114s$, and a standard deviation of $2.754s$ (the 90% confidence interval is 5.4% of the average value). All these values increase in an almost unpredictable way when the two VMs are executed simultaneously on the same host: the service times for VM^a have an average of $1.564s$, a maximum of $13.253s$, and a standard deviation of $2.434s$ while the service times for VM^b have an average of $2.416s$, a maximum of $23.056s$, and a standard deviation of $4.409s$ (in this case, the 90% confidence intervals are about 6.9% of the average values).

This experiment shows that the behaviour of each VM is affected by the interference from the other VM: as a result, average and maximum response times increase in a remarkable way. The standard deviations also increase, indicating that fluctuations from average values are large and frequent. As a result, it is not possible to control the response times for the hosted virtualised services.

This problem can be avoided by attaching each VM to a hard reservation, in order to provide temporal isolation between different VMs. This has been verified by measuring the response times when VM^a is served by a $(3ms, 10ms)$ hard CBS and VM^b is served by a $(6ms, 10ms)$ hard CBS (the server parameters have been selected to achieve short response times inside each VM while setting the utilisation according to the request pattern). With this setup, the response times for VM^a have an average of $4.602s$, a maximum of $5.990s$, and a standard deviation of $0.470s$ while the service times for VM^b have an average of $3.881s$, a maximum of $6.793s$, and a standard deviation of $0.977s$ (in this case, the 90% confidence intervals are about 1.42% of the average values).

Note that average response times are increased respect to the previous example, but maximum response times are drastically reduced (running a larger set of experiments, the maximum values are decreased from 20% to 45% of the previous values). Standard deviation values are also very low, indicating that response times do not deviate too much from average values: as a result, in this case response times can be estimated with a higher degree of accuracy.

When using reservations to serve a VM, it is also possible to apply more flexible policies in resource provisioning. For example, it is possible to give more importance to requests towards VM^b by increasing the amount of time reserved to it and decreasing the amount of time reserved to VM^a (for instance, by assigning a reservation $(2ms, 10ms)$ to VM^a and a reservation $(7ms, 10ms)$ to VM^b). Some experiments (omitted because of space constraints) showed that this mechanism is effective in controlling the relative QoSs of the various VMs. For example,

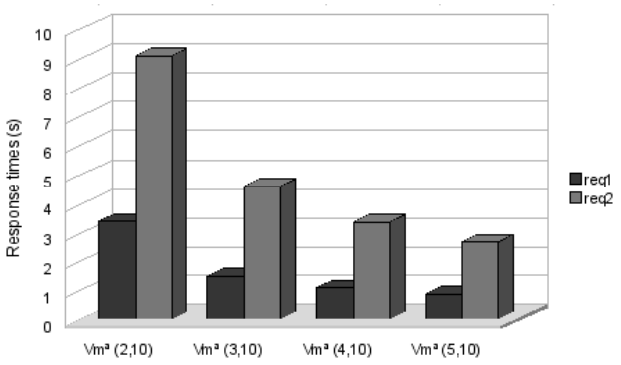

Figure 5. Response times varying Q

Figure 5 shows how the average response times of VM^a change at varying assignments of the maximum budget Q^a, keeping the reservation period constant.

6 Conclusions and Future Work

This paper presented how techniques designed according to hierarchical real-time theory may be effectively applied to the problem of providing proper service-level guarantees to virtualised software components. A few alternatives in the types of schedulers to adopt have been discussed, and hard resource reservations have been proved to be effective for the considered problem, by experimental evaluations.

The presented work will possibly shed some light on how to build low-level mechanisms for temporal isolation of VMs, in the emerging and challenging research area of RT support for SOAs.

In the future, the authors intend to evaluate the impact of virtualised I/O on the temporal behaviour of the running services. Preliminary experiments (that cannot be reported due to the lack of space) show that para-virtualisation of the networking adaptors (as provided e.g., by the `virtio` KVM capability) provides not only a reduction in the processing overhead associated to intensive I/O traffic to/from VMs, but also a dramatic reduction in its fluctuations, improving generally predictability and analysability of the system.

Furthermore, the virtualisation mechanism used in this paper (based on KVM) will be compared with more lightweight approaches (such as OS-level virtualisation), and the effect of emulating multiple resources (CPU, network, disk) will be better investigated.

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] L. Abeni and G. Lipari. Implementing resource reservations in linux. In *RTLW*, Boston (MA), Dec. 2002.

[3] L. Almeida et al. Solutions for supporting composition of service-based rt applications. In *Proceedings of the* 11^{th} *IEEE Symposium on Object Oriented Real-Time Distributed Computing*, pages 42–49, Washington, DC, USA, 2008.

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

[5] A. Bavier et al. OS support for planetary-scale services. In *NSDI Design and Implementation (NSDI)*, March 2004.

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

[7] T. Cucinotta et al. AQuoSA – adaptive quality of service architecture. *Software – Practice and Experience*, 39(1):1–31, 2009.

[8] A. Cuzzocrea. Towards RT data transformation services over grids. In *Proceedings of the* 32^{nd} *Annual IEEE Internat. Computer Software and Applic. Conf.*, pages 1143–1149, 2008.

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

[10] X. Feng and A. K. Mok. A model of hierarchical real-time virtual resources. In *Proc.* 23^{rd} *IEEE Real-Time Systems Symposium*, 2002.

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

[12] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori. kvm: the linux virtual machine monitor. volume Proc. of the Linux Symposium, Ottawa, Ontario, Canada, 2007.

[13] G. Lipari and E. Bini. A methodology for designing hierarchical scheduling systems. *Journal of Embedded Computing*, 1(2), 2004.

[14] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the Association for Computing Machinery*, 20(1):46–61, Jan. 1973.

[15] C. McGregor and J. M. Eklund. RT SOAs to support remote critical care: Trends and challenges. In *COMPSAC '08: Proceedings of the 2008 32nd Annual IEEE International Computer Software and Applications Conference*, pages 1199–1204, Washington, DC, USA, 2008.

[16] A. K. Mok and X. A. Feng. Towards compositionality in real-time resource partitioning based on regularity bounds. In *Proceedings of the 22nd IEEE Real-Time Systems Symposium*, 2001.

[17] L. Peterson, T. Anderson, D. Culler, and T. Roscoe. A blueprint for introducing disruptive technology into the internet. In *Proceedings of the 1st ACM Work on Hot Topics in Networks*, October 2002.

[18] R. Rajkumar, K. Juvva, A. Molano, and S. Oikawa. Resource Kernels: A Resource-Centric Approach to Real-Time and Multimedia Systems. In *Proc. Conf. on Multimedia Comp. and Netw.*, 1998.

[19] S. Saewong, R. Rajkumar, J. P. Lehoczky, and M. H. Klein. Analysis of hierarchical fixed-priority scheduling. In *Proceedings of the IEEE Euromicro Conference on Real-Time Systems*, June 2002.

[20] I. Shin and I. Lee. Compositional real-time scheduling framework. In *Proceedings of the* 25^{th} *IEEE International Real-Time Systems Symposium*, pages 57–67, December 2004.

[21] J. K. Strosnider, J. P. Lehoczky, and L. Sha. The deferrable server algorithm for enhanced aperiodic responsiveness in hard-real-time environments. *IEEE Transactions on Computers*, 4(1), January 1995.

[22] W. Tsai, Y.-H. Lee, Z. Cao, Y. Chen, and B. Xiao. RTSOA: Real-time service-oriented architecture. *Service-Oriented System Engineering, IEEE International Workshop on*, 0:49–56, 2006.