Towards Real-Time Scheduling of Virtual Machines Without Kernel Modifications

Mikael Åsberg, Nils Forsberg, Thomas Nolte
MRTC/Mälardalen University
P.O. Box 883, SE-721 23,
Västerås, Sweden
mikael.asberg@mdh.se

Shinpei Kato
Carnegie Mellon University
Department of Electrical and Computer Engineering
Pittsburgh, USA

***Abstract*—Virtualization is a well used technique in the area of internet server systems for managing several (legacy) applications on a single physical machine. These applications do not have strict time deadlines, which also reflects how these applications are scheduled. Using virtualization in an embedded real-time systems context is of course attractive, since we want to pack as much software as possible on a, as small as possible, hardware platform. The problem is that this kind of software does not easily cope well together, in the aspect of time related properties. Hence, we need a new mechanism, i.e., a scheduler, that can satisfy the timing requirements of each application. However, scheduler implementations typically require modifications to middleware or kernel and this is not acceptable in the area embedded systems, due to stability and reliability reasons. Hence, in this paper, we propose a framework for scheduling (soft real-time) applications residing in separate operating systems (virtual machines) using hierarchical fixed-priority preemptive scheduling, without the requirement of kernel modifications.**[1]

I. Introduction

Virtualization originates back to the 1960's [1] and the background theory is that clients, denoted as *virtual machines*, are given the illusion of having total control of the the entire machine while they in fact only utilize part of the underlying physical machine [2]. The entity that facilitates this illusion is called a *Virtual Machine Monitor* (VMM), also known as a *hypervisor*. In todays internet server systems, it is not uncommon that a hypervisor facilitates several servers running in their own operating system, denoted *virtual machine* (VM), on the same physical machine [3]. This reduces the amount of desktop computers, and hence, it becomes cheaper and it decreases the heat (generated by the computers) which is a big problem in data centers. The basic idea of merging software onto hardware and thereby reducing the amount of hardware, is motivated by reducing heat, power and space consumption coming from the hardware. Also, as a bonus, it makes the software more modular, independent and reusable. This idea is also attractive in the area of (*hard*) real-time embedded systems [4]. However, this domain is sensitive when it comes to timing, making the shift to virtualization in this area more problematic.

[1]The work in this paper is supported by the Swedish Research Council, and Swedish Foundation for Strategic Research (SSF), via the research programme PROGRESS.

In this paper, we focus on soft real-time systems which has more relaxed time deadlines, i.e., the system has deadlines but it is sufficient if most of them are kept. An example of such a system is an IP-telephony media server which streams media packets [5]. Studies have shown [6] that applications can experience insufficient performance when executed on a virtualized environment due to I/O and scheduling. We are interested in the scheduling part but what is also important is that it must be free from kernel modifications, since reliability and stability are important requirements in embedded systems. We use hierarchical scheduling [7], [8], [9] to improve the performance of soft real-time virtual machines.

A. Preliminaries

a) Hierarchical Scheduling Framework: We have previously implemented a hierarchical scheduler (HSF) in the framework REal-time SCHeduler [10] (RESCH) in Linux. HSF schedules real-time Linux tasks (which themselves are periodic [11] through the use of RESCH) in groups. The group (server) is scheduled according to Fixed-Priority Preemptive Scheduling (FPPS), with respect to its defined interface: $period$, $budget$ and $priority$. A server executes $budget$ time units every $period$, and the order is determined by their $priority$. Inside each server, the group of tasks are scheduled according to FPPS.

b) Virtualization: There are 3 types of virtualization techniques: *full virtualization*, *para-virtualization* and *micro-kernel based virtualization*. Full virtualization means that the guest operating system (inside the VM) does not need any modifications. Examples of hypervisors with such a technique are VMware [12], VirtualBox[2] and KVM (Kernel Based Virtual Machine) [13].

Para-virtualization means that the guest operating system needs modifications to be able to run on top of a hypervisor. XEN [14] is an example of such a VMM. The guest operating system (OS) needs to replace some instructions with system calls (called hyper calls) in order to be able to run on XEN. Micro-kernel based virtualization, as opposed to the hypervisor approach, is to run a micro-kernel directly on hardware beneath an OS. A micro-kernel is a small-sized software

[2]VirtualBox: http://www.virtualbox.org/

layer that abstracts the minimal and necessary low-level functionality. Hence, much of the functionality implemented in an OS is left for the user to implement. This is done by forwarding low-level events such as hardware interrupts as IPC messages to user-level handlers. In essence, a micro-kernel abstracts low-level hardware to a higher level. Any software can be implemented on top of a micro-kernel, including entire OS's [15]. However, it is more common that the OS itself is modified rather than the micro-kernel [16].

Another type of classification is whether the hypervisor resides directly on top of hardware, i.e., type-1 hypervisors (XEN and XtratuM [17]), or on top of an OS which is then denoted as the *host OS* (type-2 hypervisor, i.e., KVM, VirtualBox and VMware).

B. Outline

The outline of this paper is as follows: Section II presents related work and in Section III we present the proposed framework. In Section IV we present some preliminary experiments, and finally in Section V, we discuss our approach and the future work to be done.

II. Related work

A. Hierarchical scheduling in Linux

SCHED_DEADLINE is an implementation [18] of the Earliest Deadline First (EDF) scheduling algorithm for Linux. The scheduler modifies the kernel and brings support for server-based scheduling, where a period, budget and a task associated with the EDF scheduled server. The scheduling framework AQuoSA [19] is a hierarchical scheduler (consisting of kernel modifications) which uses the Constant Bandwidth Server (CBS) algorithm for the scheduling of servers. SCHED_SPORADIC [20] is a (POSIX compliant) sporadic server implementation that improves the response time of aperiodic tasks.

B. Real-time virtualization

Many scheduler implementations in the domain of virtualization are based on XEN [14]. XEN defers all I/O to a special VM called Dom0, this VM (usually Linux) requires modifications, i.e., it is para-virtualized. It is also common that the guest OS's (VM) are required to be modified as well. All scheduler implementations requires XEN to be modified and re-compiled. The following are examples of XEN-based schedulers. Recently, the authors in [21] implemented FPPS of virtual machines. An implementation of a feedback controlled EDF scheduler in XEN was presented in [22]. There has also been work on message passing between XEN and the guest OS's [3]. This is done in order for XEN to avoid deadline misses of real-time tasks residing in guests. [23] improve the real-time performance of the default XEN scheduler (Credit Based Scheduler) by making it preemptive. [24] modifies the Credit Based Scheduler in order to improve the performance of soft real-time tasks. A new scheduler is presented in [25], based on XEN. The scheduler, called PSEDF (priority-based scheduling plus Simple-EDF), gives higher priority to real-time guest OS's and schedules these with fixed priority. The non-real-time domains are scheduled with the Simple EDF scheduler (SEDF), which is one of the default XEN schedulers. There has also been work on improving the I/O performance [26], [27], [28] in XEN.

Similar to XEN, SPUMONE is a type-1 virtualization layer which schedules guest OS's with FPPS [29]. Another type-1 hypervisor is presented in [30] which is specialized for mobile phones. XtratuM [17] is also based on a type-1 hypervisor which has evolved towards hard-real time applications.

There exists also implementations based on type-2 hypervisors. In [31] the authors improve the performance of guest OS's executed by KVM. This is done by applying real-time priorities and the preempt-rt patch (from Ingo Molnar of Red Hat, Inc.) to the host Linux OS, thereby improving schedulability of guest OS's (which are actually Linux processes). The authors in [32] schedule KVM-based OS guests with the CBS scheduling algorithm and each guest schedules tasks with FPPS. The authors show improved task response times (compared to using the default scheduling of KVM guests). [33] presents an interesting solution which is related to our work since their solution does not either require kernel modifications. They let a high priority real-time Linux task schedule other real-time tasks according to EDF. The scheduled real-time tasks contain a type-2 hypervisor (VMware). The main difference between their approach and ours is that our scheduler is a kernel module (where all the code reside in interrupt routines) while their scheduler is a high-priority real-time task. The benefit with our solution is that it should generate less scheduling overhead since it is less time consuming to switch between interrupts and tasks as opposed to context switching between tasks.

III. Proposed solution

We propose a setup illustrated in Figure 1. The core scheduling functionality comes from the scheduling framework RESCH [10]. Currently, RESCH supports any Linux kernel which has real-time task support (from version 2.6.23 and beyond) and it does not require any kernel modifications. It has more flexible scheduling than group scheduling in Linux, i.e., *control groups* and *sched-rt-group*. The base functionality of RESCH is its support for FPPS of periodic real-time tasks in Linux. It also has several scheduler plugins, including multi-core scheduling algorithms (global, partitoned and semi-partitioned) and a 2-level hierarchical FPPS scheduler (HSF). As described in Section I, several real-time Linux tasks can reside in a server and they can in turn be scheduled internally with FPPS by the RESCH core scheduler. However, our proposition (Figure 1) suggests that one task should reside per server, where this task contains a VM (hosted by VirtualBox, KVM, VMware or any other type-2 hypervisor). Further, each VM could host a Linux operating system installed with RESCH, giving rise to FPPS scheduling of periodic tasks inside each VM. In this way we get a 2-level

FPPS scheduling framework with virtual machines, without any kernel modifications.

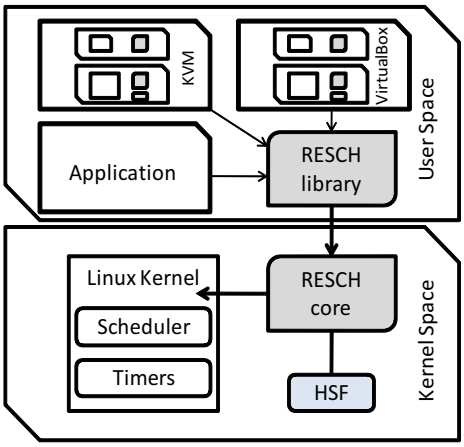

Fig. 1. The proposed framework

Multimedia intensive applications can gain performance with a HSF scheduler since the CPU availability for video and audio processing can be controlled in detail.

IV. Experiments

We have conducted some preliminary tests by executing the proposed framework on a desktop computer (Intel Pentium Dual-Core, E5300 2,6GHz) equipped with Ubuntu 9.10 Linux (kernel version 2.6.31.9). Since we assume uni-core the configuration in RESCH was set to load-balancing disabled, i.e., no task migration was allowed to balance the load between the cores. Ubuntu Linux was used both on the host and on the guest OS. We used both VirtualBox and KVM to virtualize Ubuntu as a Linux process. During the execution monitoring, using Ftrace [34], we found that VirtualBox VM's were scheduled with a period of 12-13 ticks (1 tick = 4 milliseconds), while the KVM scheduler scheduled their VM's with a frequency of 7-8 ticks. Both virtualization schedulers used a budget of less than 1 tick. We ran two RESCH-based tasks inside a VM with task periods 55 and 5 ticks, execution time 20 and 2 ticks respectively and we used rate-monotonic priorities (lower period value = higher priority). In the traces presented (Figure 2-5) we used the KVM virtualization. Figure 2 shows the execution trace (Ftrace) visualized with the Grasp tool [35]. Observe that the high priority task rt_task1 should have a clean periodic pattern (since it has highest priority) but due to the interference from the KVM scheduler (scheduling the VM itself), there are some gaps (time 270 and 380) present due to that the VM had long idle periods. Figure 3 shows the trace of the two tasks executing in a VM while it, in turn, was scheduled by HSF. We configured the VM with period 3, budget 1 (ticks) and highest real-time priority. Since the CPU allocation of the VM was greater, task rt_task1 showed a more even pattern.

Figure 4 shows a trace of the KVM guest OS when it was only scheduled by the KVM scheduler and Figure 5 shows the pattern when it was scheduled by the HSF scheduler (simultaneous with the KVM scheduler).

V. Discussion

We believe that this proposed framework will bring a new dimension to the RESCH framework itself. Most importantly, we will continue this work towards a measurement comparison with other solutions, both such that are modification-free (Vsched [33]) and such that are not [32]. It will also be valuable to compare the overhead of this approach to type-1 hypervisor solutions, such as those scheduler implementations presented in Section II which are based on XEN [21], [22], [25]. Conclusively, it would be interesting to see the performance penalty by using a modification-free solution (as opposed to modifying the kernel), as well as the performance difference between type-1 and type-2 hypervisor-based schedulers. The combination of modification-free solutions (like RESCH or Vsched) together with type-2 hypervisors results in an embedded-systems friendly approach (since it is 100% free from kernel modifications). However, the performance loss should not be too great in order for this approach to retain attractive. Further investigations will reveal how practical this type-2 hypervisor solution is.

References

[1] R. J. Creasy, "The Origin of the VM/370 Time-Sharing System," *IBM J. Res. Dev.*, vol. 25, no. 5, pp. 483–490, 1981.
[2] G. J. Popek and R. P. Goldberg, "Formal Requirements for Virtualizable Third Generation Architectures," *Commun. ACM*, vol. 17, no. 7, pp. 412–421, 1974.
[3] Y. Wang, J. Zhang, L. Shang, X. Long, and H. Jin, "Research of Real-time Task in Xen Virtualization Environment," in *ICCAE'10*, 2010.
[4] T. Gaska, B. Werner, and D. Flagg, "Applying Virtualization to Avionics Systems - The Integration Challenges," in *DASC'10*, 2010.
[5] D. Patnaik, A. Bijlani, and V. Singh, "Towards High-Availability for IP Telephony Using Virtual Machines," in *IMSAA'10*, 2010.
[6] D. Patnaik, A. S. Krishnakumar, P. Krishnan, N. Singh, and S. Yajnik, "Performance Implications of Hosting Enterprise Telephony Applications on Virtualized Multi-Core Platforms," in *IPTComm'09*, 2009.
[7] P. Goyal, X. Guo, and H. M. Vin, "A Hierarchical CPU Scheduler for Multimedia Operating Systems," in *OSDI'96*, 1996.
[8] Z. Deng and J. W.-S. Liu, "Scheduling Real-time Applications in an Open Environment," in *RTSS'97*, 1997.
[9] J. Regehr and J. A. Stankovic, "HLS: A Framework for Composing Soft Real-Time Schedulers," in *RTSS'01*, 2001.
[10] S. Kato, R. Rajkumar, and Y. Ishikawa, "A Loadable Real-Time Scheduler Suite for Multicore Platforms," Technical Report CMU-ECE-TR09-12, 2009. [Online]. Available: http://www.contrib.andrew.cmu.edu/~shinpei/papers/techrep09.pdf
[11] C. Liu and J. Layland, "Scheduling Algorithms for Multi-Programming in a Hard-Real-Time Environment," *ACM*, vol. 20, no. 1, pp. 46–61, 1973.
[12] A. Muller, S. Wilson, D. Happe, and G. J. Humphrey, *Virtualization with VMware ESX Server*. Rockland, Ma.: Syngress Publ. Inc., 2005.
[13] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori, "KVM: The Linux Virtual Machine Monitor," in *OLS'07*, 2007.
[14] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the Art of Virtualization," in *SOSP'03*, 2003.
[15] H. Härtig, M. Hohmuth, J. Liedtke, J. Wolter, and S. Schönberg, "The Performance of µ-Kernel-Based Systems," in *SOSP'97*, 1997.
[16] F. Armand and M. Gien, "A Practical Look at Micro-Kernels and Virtual Machine Monitors," in *CCNC'09*, 2009.
[17] A. Crespo, I. Ripoll, and M. Masmano, "Partitioned Embedded Architecture Based on Hypervisor: The XtratuM Approach," in *EDCC'10*, 2010.

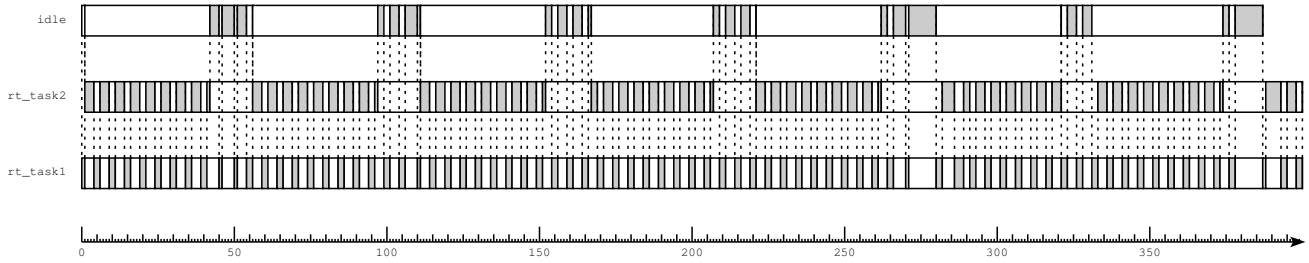

Fig. 2. Example trace of tasks running in the VM (without HSF)

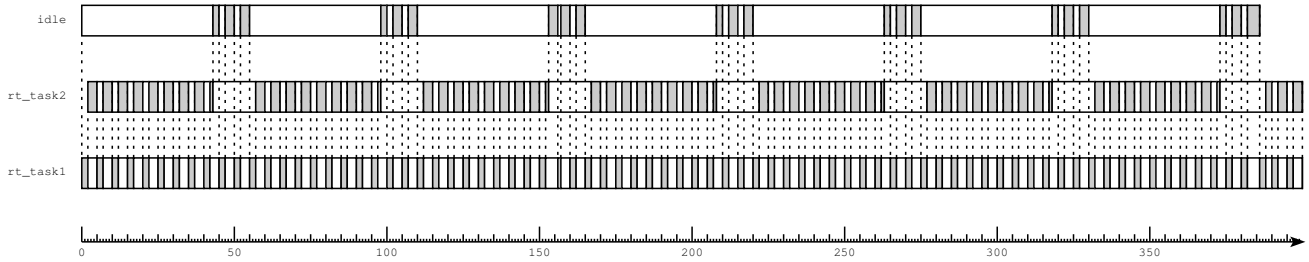

Fig. 3. Example trace of tasks running in the VM (with HSF)

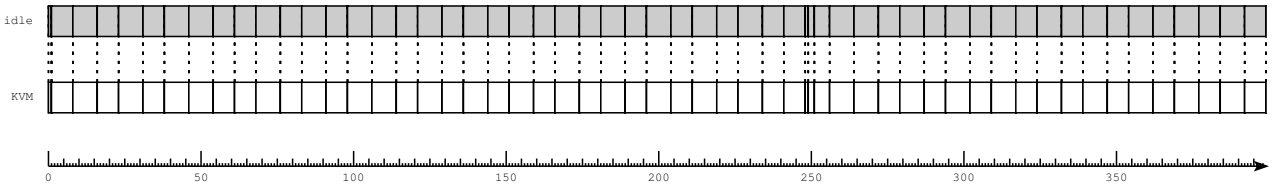

Fig. 4. Example trace of the VM (without HSF)

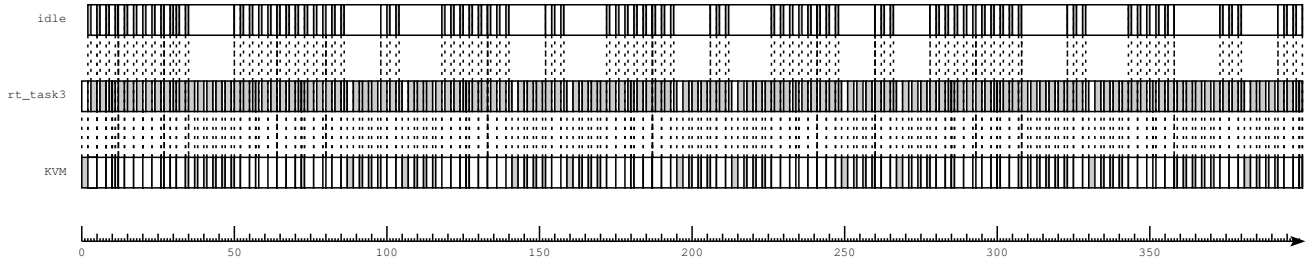

Fig. 5. Example trace of the VM (with HSF)

[18] D. Faggioli and F. Checconi, "An EDF Scheduling Class for the Linux Kernel," in *RTLWS'09*, 2009.

[19] L. Palopoli, T. Cucinotta, L. Marzario, and G. Lipari, "AQuoSA Adaptive Quality of Service Architecture," *Softw. Pract. Exper.*, vol. 39, no. 1, pp. 1–31, 2009.

[20] D. Faggioli, A. Mancina, F. Checconi, and G. Lipari, "Design and Implementation of a POSIX Compliant Sporadic Server for the Linux Kernel," in *RTLWS'08*, 2008.

[21] S. Xi, J. Wilson, C. Lu, and C. Gill, "RT-Xen: Real-Time Virtualization Based on Fixed Priority Hierarchical Scheduling," Technical Report WUCSE-2010-38, 2010. [Online]. Available: http://students.cec.wustl.edu/~sx1/RT-XEN/emsoft11_rtxen.pdf

[22] D. Gupta, L. Cherkasova, R. Gardner, and A. Vahdat, "Enforcing Performance Isolation Across Virtual Machines in XEN," in *MIDDLEWARE'06*, 2006.

[23] P. Yu, M. Xia, Q. Lin, M. Zhu, S. Gao, Z. Qi, K. Chen, and H. Guan, "Real-Time Enhancement for XEN Hypervisor," in *ICEUC'10*, 2010.

[24] M. Lee, A. S. Krishnakumar, P. Krishnan, N. Singh, and S. Yajnik, "Supporting Soft Real-Time Tasks in the XEN Hypervisor," in *VEE'10*, 2010.

[25] A. Masrur, S. Drossler, T. Pfeuffer, and S. Chakraborty, "VM-Based Real-Time Services for Automotive Control Applications," in *RTCSA'10*, 2010.

[26] S. Govindan, A. R. Nath, A. Das, B. Urgaonkar, and A. Sivasubramaniam, "Xen and Co.: Communication-Aware CPU Scheduling for Consolidated XEN-Based Hosting Platforms," in *VEE'07*, 2007.

[27] D. Ongaro, A. L. Cox, and S. Rixner, "Scheduling I/O in Virtual Machine Monitors," in *VEE'08*, 2008.

[28] H. Kim, H. Lim, J. Jeong, H. Jo, and J. Lee, "Task-Aware Virtual Machine Scheduling for I/O Performance," in *VEE'09*, 2009.

[29] Y. Kinebuchi, W. Kanda, Y. Yumura, K. Makijima, and T. Nakajima, "A Hardware Abstraction Layer for Integrating Real-Time and General-Purpose with Minimal Kernel Modification," in *STFSSD'09*, 2009.

[30] S. Yoo, Y. Liu, C.-H. Hong, C. Yoo, and Y. Zhang, "MobiVMM: A Virtual Machine Monitor for Mobile Phones," in *MobiVirt'08*, 2008.

[31] J. Zhang, K. Chen, B. Zuo, R. Ma, Y. Dong, and H. Guan, "Performance Analysis Towards a KVM-Based Embedded Real-Time Virtualization Architecture," in *ICCIT'10*, 2010.

[32] T. Cucinotta, G. Anastasi, and L. Abeni, "Respecting Temporal Constraints in Virtualised Services," in *COMPSAC'09*, 2009.

[33] B. Lin and P. A. Dinda, "VSched: Mixing Batch and Interactive Virtual Machines Using Periodic Real-time Scheduling," in *SC'05*, 2005.

[34] T. Bird, "Measuring Function Duration with Ftrace," in *Proc. of the Japan Linux Symposium*, 2009.

[35] M. Holenderski, M. M. H. P. van den Heuvel, R. J. Bril, and J. J. Lukkien, "Grasp: Tracing, Visualizing and Measuring the Behavior of Real-Time Systems," in *WATERS'10*, 2010.