Journal of Software Engineering and Applications, **2012, 5, 277-290**
http://dx.doi.org/10.4236/jsea.2012.54033 Published Online April 2012 (http://www.SciRP.org/journal/jsea)

A State-of-the-Art Survey on Real-Time Issues in Embedded Systems Virtualization

Zonghua Gu, Qingling Zhao

College of Computer Science, Zhejiang University, Hangzhou, China.
Email: {zgu, ada_zhao}@zju.edu.cn

Received January 1st, 2012; revised February 5th, 2012; accepted March 10th, 2012

ABSTRACT

Virtualization has gained great acceptance in the server and cloud computing arena. In recent years, it has also been widely applied to real-time embedded systems with stringent timing constraints. We present a comprehensive survey on real-time issues in virtualization for embedded systems, covering popular virtualization systems including KVM, Xen, L4 and others.

Keywords: Virtualization; Embedded Systems; Real-Time Scheduling

1. Introduction

Platform virtualization refers to the creation of *Virtual Machines* (*VMs*), also called *domains*, *guest OSes*, or *partitions*, running on the physical machine managed by a *Virtual Machine Monitor* (*VMM*), also called a *hypervisor*. Virtualization technology enables concurrent execution of multiple VMs on the same hardware (single or multicore) processor. Virtualization technology has been widely applied in the enterprise and cloud computing space. In recent years, it has been increasingly widely deployed in the embedded systems domain, including avionics systems, industrial automation, mobile phones, etc. Compared to the conventional application domain of enterprise systems, virtualization in embedded systems must place strong emphasis on issues like real-time performance, security and dependability, etc.

A VMM can run either on the hardware directly (called *bare-metal*, or *Type*-1 virtualization), or run on top of a host operating system (called *hosted, or Type*-2 virtualization). Another way to classify platform-level virtualization technologies is full virtualization vs para-virtualization. Full virtualization allows the guest OS to run on the VMM without any modification, while para-virtualization requires the guest OS to be modified by adding hypercalls into the VMM. Representative Type-2, full virtualization solutions include KVM, VirtualBox, Microsoft Virtual PC, VMWare Workstation; Representative Type-1, paravirtualization solutions include Xen, L4, VMWare ESX. There are some research attempts at constructing Type-1, full virtualization solutions, e.g., Kinebuchi *et al*. [1] implemented such a solution by porting the QEMU machine emulator to run as an application on L4Ka::Pistachio microkernel; in turn, unmodified guest OS can run on top of QEMU; Schild *et al*. [2] used Intel VT-d HW extensions to run unmodified guest OS on L4. There are also Type-2, para-virtualization solutions, e.g., VMWare MVP (Mobile Virtualization Platform) [3], as well as some attempts at adding para-virtualization features to Type-2 virtualization systems to improve performance, e.g., task-grain scheduling in KVM [4].

Since embedded systems often have stringent timing and performance constraints, virtualization for embedded systems must address real-time issues. We focus on real-time issues in virtualization for embedded systems, and leave out certain topics that are more relevant to the server application space due to space constrains, including: power and energy-aware scheduling, dynamic adaptive scheduling, multicore scheduling on high-end NUMA (Non-Uniform Memory Access) machines, nested virtualization, etc. In addition, we focus on recent developments in this field, instead of presenting a historical perspective.

This paper is structured as follows: we discuss hard real-time virtualization solutions for safety-critical systems in Section 2; Xen-based solutions in Section 3, including hard real-time and soft real-time extensions to Xen; KVM-based solutions in Section 4; micro-kernel based solutions represented by L4 in Section 5; other virtualization frameworks for embedded systems in Section 6; OS virtualization in Section 7; task-grain scheduling in Section 8; the Lock-Holder Preemption problem in Section 9; and finally, a brief conclusion in Section 10. (The topics of Sections 8 and 9 are cross-cutting issues

that are not specific to any virtualization approach, but we believe they are of sufficient importance to dedicate separate sections to cover them.)

2. Hard Real-Time Virtualization for Safety-Critical Systems

The ARINC 653 standard [5] defines a software architecture for spatial and temporal partitioning designed for safety-critical IMA (Integrated Modular Avionics) applications. It defines services such as partition management, process management, time management, inter and intra-partition communication. In the ARINC 653 specification for system partitioning and scheduling, each partition (virtual machine) is allocated a time slice, and partitioned are scheduled in a TDMA (Time Division Multiple Access) manner. A related standard is the Multiple Independent Levels of Security (MILS) architecture, an enabling architecture for developing security-critical applications conforming to the *Common Criteria* security evaluation. The MILS architecture defines four conceptual layers of separation: separation kernel and hardware; middleware services; trusted applications; distributed communications. Authors from Lockheed Martin [6] presented a feasibility assessment toward applying the Microkernel Hypervisor architecture to enable virtualization for a representative set of avionics applications requiring multiple guest OS environments, including a mixture of safety-critical and non-safety-critical guest OSes. Several commercial RTOS products conform to the ARINC 653 standard, including LynuxWorks LynxOS-178, Green Hills INTEGRITY-178B, Wind River VxWorks 653, BAE Systems CsLEOS, and DDC-I DEOS, etc. Many vendors also offer commercial virtualization products for safety-critical systems, many by adapting existing RTOS products. For example, LynuxWorks implemented a virtualization layer to host their LynxOS product, called the LynxSecure hypervisor that supports MILS and ARINC 653. WindRiver [7] provides a hypervisor product for both its VxWorks MILS and VxWorks 653 platforms, including support for multicore. Other similar products include: Greenhills INTEGRITY MultiVisor [8], Real-Time Systems GmbH Hypervisor [9], Tenasys eVM for Windows [10], National Instruments Real-Time Hyper Hypervisor [11], Open Synergy COQOS [12], Enea Hypervisor [13], SysGO PikeOS [14], IBV Automation GmbH QWin etc. VanderLeest [15], from a company named DornerWorks, took a different approach by adapting the open-source Xen hypervisor to implement the ARINC 653 standard.

XtratuM [16] is a Type-1 hypervisor targeting safety critical avionics embedded systems. It runs on LEON3 SPARC V8 processor, a widely used CPU in space applications. It features temporal and spatial separation, efficient scheduling mechanism, low footprint with minimum computational overhead, efficient context switch of the partitions (domains), deterministic hypervisor system calls. Zamorano *et al*. [17,18] ported the Ada-based Open Ravenscar Kernel (ORK+) to run as a partition on XtratuM, forming a software platform conforming to the ARINC 653 standard. Campagna *et al*. [19] implemented a dual-redundancy system on XtratuM to tolerate transient faults like Single-Event-Upset, common on the high-radiation space environment. Three partitions are executed concurrently, two of them run identical copies of the application software, and the third checks consistency of their outputs. The *OVERSEE* (*Open Vehicular Secure Platform*) Project [20] aims to bring the avionics standard to automotive systems by porting FreeOSEK, an OSEK/VDX-compliant RTOS, as a paravirtualized guest OS running on top of the XtratuM hypervisor [21]. While most ARINC-653 compliant virtualization solutions are based on para-virtualization, Han *et al*. [22] presented an implementation of ARINC 653 based on Type-2, full virtualization architectures, including VM-Ware and VirtualBox.

Next, we briefly mention some virtualization solutions for safety-critical systems that are not specifically designed for the avionics domain, hence do not conform to the ARINC-653 standard. Authors from Indian Institute of Technology [23] developed *SParK* (*Safety Partition Kernel*), a Type-1 para-virtualization solution designed for safety-critical systems; an open-source RTOS uC/OS-II and a customized version of saRTL (stand-alone RT Linux) are ported as guest OSes on SParK. Authors from CEA (Atomic Energy Commission), France [24] developed *PharOS*, a dependable RTOS designed for automotive control systems featuring temporal and spatial isolation in the presence of a mixed workload of both time-triggered and event-triggered tasks. They adapted Trampoline, an OSEK/VDX-compliant RTOS, as a paravirtualized guest OS running on top of PharOS host OS to form a Type-2 virtualization architecture. To ensure temporal predictability, Trampoline is run as a time-triggered task within PharOS. Authors from PUCRS, Brazil [25] developed *Virtual-Hellfire Hypervisor*, a Type-1 virtualization system based on the microkernel HellfireOS featuring spatial and temporal isolation for safety-critical applications. The target HW platform is HERMES Network-on-Chip with MIPS-like processing elements [26].

3. Xen-Based Solutions

Cherkasova *et al*. [27] introduced and evaluated three CPU schedulers in Xen: *BVT* (*Borrowed Virtual Time*), *SEDF* (*Simple Earliest Deadline First*), and *Credit*. Since the BVT scheduler is now deprecated, we only discuss Credit and SEDF here.

The default scheduling algorithm in Xen is the *Credit*

Scheduler: it implements a proportional-share scheduling strategy where a user can adjust the CPU share for each VM. It also features automatic workload balancing of virtual CPUs (vCPUs) across physical cores (pCPUs) on a multicore processor. This algorithm guarantees that no pCPU will idle when there exists a runnable vCPU in the system. Each VM is associated with a *weight* and a *cap*. When the cap is 0, VM can receive extra CPU time unused by other VMs, *i.e*., *WC* (*Work-Conserving*) mode; when the cap is nonzero (expressed as a percentage), it limits the amount of CPU time given to a VM to not exceed the cap, *i.e*., *NWC* (*Non-Work-Conserving*) mode. By default the credits of all runnable VMs are recalculated in intervals of 30 ms in proportion to each VM's *weight* parameter, and the scheduling time slice is 10 ms, *i.e*., the credit of the running virtual CPU is decreased every 10 ms.

In the SEDF scheduler, each domain can specify a lower bound on the CPU reservations that it requests by specifying a tuple of (*slice*, *period*), so that the VM will receive at least *slice* time units in each *period* time units. A Boolean flag indicates whether the VM is eligible to receive extra CPU time. If true, then it is WC (Work-Conserving) mode, and any available slack time is distributed in a fair manner after all the runnable VMs have received their specified slices. Unlike the Credit Scheduler, SEDF is a partitioned scheduling algorithm that does not allow VM migration across multiple cores, hence there is no global workload balancing on multicore processors.

Masrur *et al*. [28] presented improvements to Xen's SEDF scheduler so that a domain can utilize its whole budget (slice) within its period even if it blocks for I/O before using up its whole slice (in the original SEDF scheduler, the unused budget is lost once a task blocks). In addition, certain critical domains can be designated as real-time domains and be given higher fixed-priority than other domains scheduled with SEDF. One limitation is that each real-time domain is constrained to contain a single real-time task. Masrur *et al*. [29] removed this limitation, and considered a hierarchical scheduling architecture in Xen, where both the Xen hypervisor and guest VMs adopt deadline-monotonic Fixed-Priority scheduling with a tuple of (*period*, *slice*) parameters (similar to SEDF, a VM is allowed to run for a maximum length of *slice* time units within each *period*), and proposed a method for selecting optimum time slices and periods for each VM in the system to achieve schedulability while minimizing the lengths of time slices.

The *RT-Xen* project [30,31] implemented a compositional and hierarchical scheduling architecture based on fixed-priority scheduling within Xen, and extensions of compositional scheduling framework [32] and periodic server design for fixed-priority scheduling. Similarly, Yoo *et al*. [33] implemented the Compositional Scheduling Framework [32] in Xen-ARM. Jeong *et al*. [34] developed PARFAIT, a hierarchical scheduling framework in Xen-ARM. At the bottom level (near the HW), SEDF is used to provide CPU bandwidth guarantees to Domain0 and real-time VMs; at the higher level, BVT (Borrowed Virtual Time) is used to schedule all non-real-time VMs to provide fair distribution of CPU time among them, by mapping all vCPUs in the non real-time VMs to a single abstract vCPU scheduled by the underlying SEDF scheduler. Lee *et al*. [30], Xi *et al*. [31], Yoo *et al*. [33], Jeong *et al*. [34] only addressed CPU scheduling issues, but did not consider I/O scheduling.

Xen has a *split-driver* architecture for handling I/O, where a special driver domain (Domain0) contains the backend driver, and user domains contain the frontend driver. Physical interrupts are first handled by the hypervisor, which then notifies the target guest domain with virtual interrupts, handled when the guest domain is scheduled by the hypervisor scheduler. Hong *et al*. [35] improved network I/O performance of Xen-ARM by performing dynamic load balancing of interrupts among the multiple cores by modifying the ARM11 MPCore interrupt distributor. In addition, each guest domain contains its own native device drivers, instead of putting all device drivers in Domain0. Yoo *et al*. [36] proposed several improvements to the Xen Credit Scheduler to improve interrupt response time, including: do not deschedule the driver domain when it disables virtual interrupts; exploit ARM processor support for FIQ, which has higher priority than regular IRQ, to support real-time I/O devices; ensure that the driver domain is always scheduled with the highest priority when it has pending interrupts.

Lee *et al*. [37] enhanced the soft real-time performance of the Xen Credit Scheduler. They defined a *laxity* value as the target scheduling latency that the workload desires, and preferentially schedule the vCPU with smallest laxity. Furthermore, they take cache-affinity into account for real time tasks when performing multicore load-balancing to prevent cache thrashing.

Yu *et al*. [38] presented real-time improvements to the Xen Credit Scheduler: real-time vCPUs are always given priority over non real-time vCPUs, and can preempt any non real-time vCPUs that may be running. When there are multiple real-time vCPUs, they are load-balanced across multiple cores to reduce their mutual interference. Wang *et al*. [39] proposed to insert a workload monitor in each guest VM to monitor and calculate both the CPU and GPU resource utilization of all active guest VMs, and give priority to guest VMs that are GPU-intensive in the Xen Credit Scheduler, since those are likely the interactive tasks that need the shortest response time.

Chen *et al*. [40] improved the Xen Credit Scheduler to

have better audio-playing performance. The real-time vCPUs are assigned a shorter time slice, say 1 ms, while the non-real-time vCPUs keep the regular time slice of 30 ms. The real-time vCPUs are also allowed to stay in the BOOST state for longer periods of time, while the regular Credit Scheduler only allows a vCPU to stay in the BOOST state for one time slice before demoting it to UNDER state. Xen provides a mechanism to pin certain vCPUs on physical CPUs (pCPUs). Chen *et al*. [41] showed that when the number of vCPUs in a domain is greater than that of pinned pCPUs, the Xen Credit Scheduler may seriously impair the performance of I/O-intensive applications. To improve performance of pinned domains, they added a periodic runtime monitor in the VMM to monitor the length of time when each domain's vCPUs go into the BOOST state, as well as the bus transactions with the help of HW Performance Monitoring Unit, in order to distinguish between CPU-intensive and I/O intensive domains. The former are assigned larger time slices, and the latter are assigned smaller time slices to improve their responsiveness.

Gupta *et al*. [42] presented a set of techniques for enforcing performance isolation among multiple VMs in Xen: *XenMon* measures per-VM resource consumption, including work done on behalf of a particular VM in Xen's driver domains; a *SEDF-DC* (*Debt Collector*) scheduler periodically receives feedback from XenMon about the CPU consumed by driver domains for I/O processing on behalf of guest domains, and charges guest domains for the time spent in the driver domain on their behalf when allocating CPU time; *ShareGuard* limits the total amount of resources consumed in driver domains based on administrator-specified limits.

Ongaro *et al*. [43] studied impact of the Xen VMM scheduler on performance using multiple guest VMs concurrently running different types of application workloads, *i.e*., different combinations of CPU-intensive, I/O-intensive, and latency-sensitive applications. Several optimizations are proposed, including disabling preemption for the privileged driver domain, and optimizing the Credit Scheduler by sorting VMs in run queue based on remaining credits to give priority to short-running I/O VMs.

Govindan *et al*. [44] presented a communication-aware CPU scheduling algorithm and a CPU usage accounting mechanism for Xen. They modified the Xen SEDF scheduler by preferentially scheduling I/O-intensive VMs over their CPU-intensive VMs, by counting the number of packets flowing into or out of each VM, and selecting the VM with the highest count that has not yet consumed its entire slice during the current period.

Hu *et al*. [45] proposed a method for improving I/O performance of Xen on a multicore processor. The cores in the system are divided into 3 subsets, including the *driver core* hosting the single driver domain, hence it does not need VMM scheduling; *fast-tick cores* for handling I/O events, which employs preemptive scheduling with small time slices to ensure fast response to I/O events; and *general cores* for computation-intensive workloads, which use the default Xen Credit Scheduler for fair CPU sharing.

Lee *et al*. [46] improved performance of multimedia applications in Xen by assigning higher fixed-priorities to Domain0 and other real-time domains than non-real-time domains, which are scheduled by the default credit scheduler.

4. KVM-Based Solutions

KVM is a Type-2 virtualization solution that uses Linux as the host OS, hence any real-time improvements to the Linux host OS kernel directly translate into real-time improvements to the KVM VMM. For example, real-time kernel patches like Ingo Milnar's PREEMPT_RT patch, can be applied to improve the real-time performance and predictability of the Linux host OS.

Kiszka [4] presented some real-time improvements to KVM. Real-time guest VM threads are given real-time priorities, including the main I/O thread and one or more vCPU threads, at the expense of giving lower priorities to threads in the host Linux kernel for various system-wide services. A paravirtualized scheduling interface was introduced to allow task-grain scheduling by introducing two hypercalls for the guest VM to inform the host VMM about priority of the currently running task in the VM, as well as when an interrupt handler is finished execution in the VM. Introduction of these hypercalls implies that the resulting system is no longer a strict full-virtualization system as the conventional KVM. Zhang *et al*. [47] presented two real-time improvements to KVM with coexisting RTOS and GPOS guests: giving the guest RTOS vCPUs higher priority than GPOS vCPUs; use CPU shielding to dedicate one CPU core to the RTOS guest and shield it from GPOS interrupts, by setting the CPU affinity parameter of both host OS processes and interrupts. Experimental results indicate that the RTOS interrupt response latencies are reduced. Based on the work of [47], Zuo *et al*. [48] presented additional improvements by adding two hypercalls that enable the guest OS to boost priority of its vCPU when a high-priority task is started, e.g., like an interrupt handler, and deboost it when the high-priority task is finished.

Cucinotta *et al*. [49] presented the IRMOS scheduler, which implements a hard reservation variant of CBS (Constant Bandwidth Server) on top of the EDF scheduler in Linux by extending the Linux CGroup interface. When applied in the context of KVM, inter-VM scheduling is CBS/EDF, while intra-VM task scheduling is Fixed-Priority. While Cucinotta *et al*. [49] only addressed

CPU scheduling, Cucinotta *et al*. [50] addressed I/O and networking issues by grouping both guest VM threads and interrupt handler threads in the KVM host OS kernel into the same CBS reservation, and over-provisioning the CPU budget assigned to each VM by an amount that is dependent on the overall networking traffic performed by VMs hosted on the same system. Cucinotta *et al*. [51] improved the CBS scheduler in the KVM host OS for a mixed workload of both compute-intensive and network-intensive workloads. In addition to the regular CPU CBS reservation of (Q, P), where the VM is guaranteed to receive budget Q time units of CPU time out of every P time units, a spare CPU reservation of (Q_s, P_s) is attached to the VM and dynamically activated upon a new packet arrival. The spare reservation (Q_s, P_s) has much smaller budget and period than the regular reservation (Q, P) to provide fast networking response time. Checconi *et al*. [52] addressed real-time issues in live migration of KVM virtual machines by using a probabilistic model of the migration process to select an appropriate policy for memory page migration, either based on a simple LRU (Least-Recently Used) order, or a more complex one based on observed page access frequencies in the past VM history.

RESCH (*REal-time SCHeduler framework*) [53] is an approach to implementing new scheduling algorithms within Linux without modifying the kernel. In contrast to typical user-level schedulers [54], RESCH consists of a loadable kernel module called RESCH core, and a user library called RESCH library. In addition to the built-in Fixed-Priority (FP) scheduling algorithm in Linux, 4 new real-time FP-based scheduling algorithms have been implemented with RESCH, including: FP-FF (partitioned scheduling with First-Fit heuristic for mapping tasks to processors), FP-PM (when a task cannot be assigned to any CPU by first-fit allocation, the task can migrate across multiple CPUs with highest priority on its allocated CPU), G-FP (Global FP scheduling, where tasks can freely migrate among different processors, and the task with globally highest priority is executed), FP-US (global FP scheduling, where tasks are classified into heavy tasks and light tasks, based on their utilizations factors. If the CPU utilization of a task is greater than or equal to $m/(3m-2)$, where m is the number of processors, it is a heavy task. Otherwise, it is a light task. All heavy tasks are statically assigned the highest priorities, while light tasks have the original priorities). Other scheduling algorithms can also be implemented. Asberg *et al*. [55] applied RESCH to KVM to implement a real-time hierarchical scheduling framework in a uni-processor environment, where RESCH is used in both the host OS and guest OS to implement FP scheduling. Asberg *et al*. [56] used RESCH to implement 4 variants of multi-core global hierarchical scheduling algorithms in Linux.

Lin *et al*. [54] presented *VSched*, an user-space implementation of EDF-based scheduling algorithm in the host OS of a Type-2 virtualization system VMware GSX Server. The scheduler is a user-level process with highest priority, and Linux SIGSTOP/SIGCONT signals are used to implement (optional) hard resource reservations, hence it does not require kernel modification, but carries large runtime overhead due to excessive context-switches between the scheduler process and application processes compared to RESCH. (Although VSched is not implemented on KVM, we place it here due to its similarity to RESCH.)

5. MicroKernel-Based Solutions

*L*4 is a representative microkernel operating system, extended to be a Type-1 virtualization architecture. There are multiple active variants of L4. **Figure 1** shows the evolution lineage of L4 variants.

Authors from Avaya Labs [57] implemented a para-virtualized Android kernel, and ran it alongside a para-virtualized Linux kernel from OK Labs, L4Linux, on an ARM processor. Iqbal *et al*. [58] presented a detailed comparison study between microkernel-based approach, represented by L4, and hypervisor-based approach, represented by Xen, to embedded virtualization, and concluded that L4-based approaches have better performance and security properties. Gernot Heiser [59], founder of OKL4, argued for microkernel-based virtualization for embedded systems. Performance evaluation of OKL4 hypervisor on Beagle Board based on 500 MHz Cortex A8 ARMv7 processor, running *netperf* benchmark shows that the TCP and UDP throughput degradation due to virtualization is only 3% - 4%. Bruns *et al*. [60] and

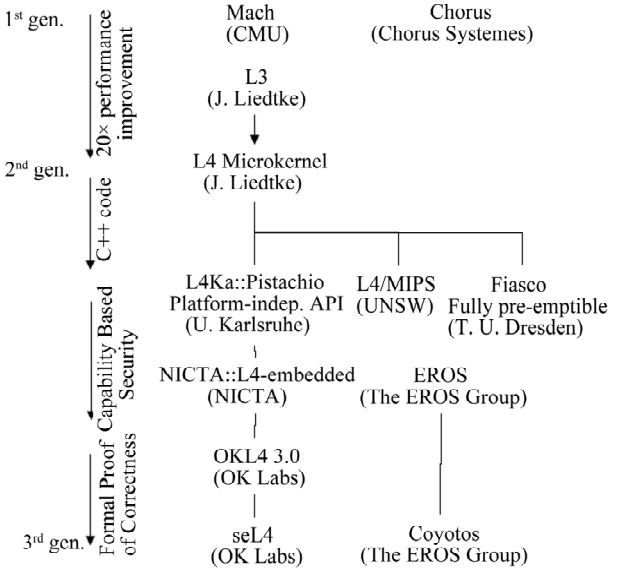

Figure 1. **L4 evolution lineage (from Acharya *et al.* [57]).**

Lackorzynski *et al*. [61] performed performance evaluation of L4Linux. Using L4/Fiasco as the hypervisor, L4Linux as the guest OS, an ARM-based Infineon X-GOLD618 processor for mobile phones, Bruns *et al*. [60] compared thread context-switching times and interrupt latencies of L4Linux to those of a stand-alone RTOS (FreeRTOS), and demonstrated that L4-based system has relatively small runtime overhead, but requires significantly more cache resources than an RTOS, with cache contention as the main culprit for performance degradations.

One important application area of L4 is mobile phone virtualization. A typical usage scenario is co-existence of multiple OSes on the same HW platform, e.g., have a "rich" OS like Android or Windows for handling user interface and other control-intensive tasks, and a "simple" real-time OS for handling performance and timing-critical tasks like baseband signal processing. Another goal is to save on hardware bill-of-material costs by having a single core instead of two processor cores. Open Kernel Labs implemented the first commercial virtualized phone in 2009 named the Motorola Evoke QA4, which runs two VMs on top of L4 hypervisor, one is Linux for handling user interface tasks, and the other is BREW (Binary Runtime Environment for Wireless) for handling baseband signal processing tasks on a single ARM processor.

The company *SysGo AG* developed *PikeOS*, which evolved from a L4-microkernel, designed for safety-critical applications. Kaiser *et al*. [62] described the scheduling algorithm in PikeOS, which is a combination of priority-based, time-driven and proportional share scheduling. Each real-time VM is assigned a fixed-priority decided by the designer, hence a time-driven VM can have lower or higher priority values than an event-driven VM. Non real-time VMs execute in the background when real-time VMs are not active. Yang *et al*. [63] developed a two-level *Compositional Scheduling Architecture* (*CSA*), using the L4/Fiasco microkernel as a VMM and L4Linux as a VM.

The NOVA Hypervisor [64] from Technical University of Dresden is a Type-1 hypervisor focusing on security issues that shares many similarities with L4, but it is a full-virtualization solution based on processor HW support that does not require modification of the guest OS.

6. Other Virtualization Frameworks for Embedded Systems

Authors from Motorola [65] argued for the use of Type-1 hypervisors as opposed to Type-2 hypervisors for mobile phone virtualization for their security benefits due to a small TCB (Trusted Computing Base). Representative mobile virtualization solutions include MVP (Mobile Virtualization Platform) from VMWare [3,66], VLX from Red Bend [67,68], Xen-ARM from Samsung [69], etc. L4, VLX and Xen-ARM are all Type-1 hypervisors. VMWare MVP is a Type-2 virtualization solution with Linux as both host and guest OSes implemented on ARMv7 processor. It adopts a lightweight para-virtualization approach: 1) the entire guest OS is run in CPU user-mode; most privileged instructions are handled via trap-and-emulate, e.g., privileged coprocessor access instructions *mcr* and *mrc*; other sensitive instructions are replaced with hypercalls; 2) all devices are para-virtualized; especially, the paravirtualized *TCP/IP* is different from traditional para-virtualized networking by operating at the socket system call level instead of at the device interface level: When a guest VM calls a syscall to open a socket, the request is made directly to the *offload engine* in the host OS, which opens a socket and performs TCP/IP protocol processing within the host OS kernel.

SPUMONE (Software Processing Unit, Multiplexing ONE into two or more) [70,71] is a lightweight Type-1 virtualization software implemented on a SH-4A processor with the goal of running a GPOS (General-Purpose OS) like Linux and an RTOS like *TOPPERS* (an open source implementation of the ITRON RTOS standard [72]) on the same HW processor. It can either use fixed-priority scheduling, and always assign the RTOS higher priority than GPOS, or adopt task-grain scheduling [73], where each task can be assigned a different priority to allow more fine-grained control, e.g., an important task in the GPOS can be assigned a higher priority than a less important task in the RTOS. Inter-OS communication is achieved with IPI (Inter-Processor Interrupts) and shared memory. Aalto [74] developed *DynOS SPUMONE*, an extension to SPUMONE to enable runtime migration of guest OSes to different cores on a 4-core RP1 processor from Renesas. Lin *et al*. [75] presented a redesign of SPUMONE as a *multikernel* architecture. (*Multikernel* means that a separate copy of the kernel or VMM runs on each core on a multicore processor.) Each processor core has its private local memory. To achieve better fault isolation, the VMM runs in the local memory, while the guest OSes run in the global shared main memory; both run in kernel mode. To enhance security, Li *et al*. [76] added a small trusted OS called *xv6* to run a monitoring service to detect any integrity violations of the large and untrusted Linux OS. Upon detection of such integrity violations, the monitoring service invokes a recovery procedure to recover the integrity of the data structures, and if that fails, reboot Linux.

SIGMA System [77] assigns a dedicated CPU core to each OS, to build a multi-OS environment with native performance. Since there is no virtualization layer, guest OSes cannot share devices and interrupts among them, hence all interrupts are bound and directly delivered to

their target OSes, e.g., for the Intel multiprocessor architecture, APIC (Advanced Peripheral Interrupt Controller) can be used to distribute I/O interrupts among processors by interrupt numbers.

Gandalf [78,79] is a lightweight Type-1 VMM designed for resource-constrained embedded systems. It is called *meso-virtualization*, which is more lightweight and requires less modification to the guest OS than typical para-virtualization solutions. Similar to SIGMA, interrupts are not virtualized, but instead delivered to each guest OS directly.

Yoo *et al*. [80] developed *MobiVMM*, a lightweight virtualization solution for mobile phones. It uses preemptive fixed-priority scheduling and assigns the highest priority to the real-time VM (only a single real-time VM is supported), and uses *pseudo-polling* to provide low-latency interrupt processing. Physical interrupts are stored temporarily inside the VMM, and delivered to the target guest VM when it is scheduled to run. Real-time schedulability test is used to determine if certain interrupts should be discarded to prevent overload. Yoo *et al*. [81] improved I/O performance of MobiVMM, by letting the non-real-time VM preempt the real-time VM whenever a device interrupt occurs targeting the non real-time VM.

Li *et al*. [82] developed Quest-V, a virtualized multikernel for high-confidence systems. Memory virtualization with shadow paging is used for fault isolation. It does not virtualize any other resources like CPU or I/O, *i.e*., processing or I/O handling are performed within the kernels or user-spaces on each core without intervention from the VMM. vCPUs are the fundamental kernel abstraction for scheduling and temporal isolation. A hierarchical approach is adopted where vCPUs are scheduled on pCPUs and threads are scheduled on vCPUs [83] Quest-V defines two classes of vCPUs: Main vCPUs, configured as sporadic servers by default, are used to schedule and track the pCPU usage of conventional software threads, while I/O vCPUs are used to account for, and schedule the execution of, interrupt handlers for I/O devices. Each device is assigned a separate I/O vCPU, which may be shared among multiple main vCPUs. I/O vCPUs are scheduled with a Priority-Inheritance Bandwidth-Preserving (PIBS) server algorithm, where each shared I/O vCPU inherits its priority from the main vCPUs of tasks responsible for I/O requests, and is assigned a certain CPU bandwidth share that should not be exceeded.

7. OS Virtualization

OS virtualization, also called *container-based virtualization*, allows to partition an OS environment into multiple domains with independent name spaces to achieve a certain level of security and protection between different domains. Examples include *FreeBSD Jails* [84], *OpenVZ* [85], *Linux VServer* [86], *Linux Containers LXC* and *Solaris Zones*. The key difference between OS virtualization and system-level virtualization is that the former only has a single copy of OS kernel at runtime shared among multiple domains, while the latter has multiple OS kernels at runtime. As a result, OS virtualization is more lightweight, but can only run a single OS, e.g., Linux. Commercial products that incorporate OS virtualization technology include: *Parallels Virtuozzo Containers* [87] is a commercial version of the open-source OpenVZ software; the *MontaVista Automotive Technology Platform* [88] adopts Linux Containers in combination with SELinux (Security Enhanced Linux) to run multiple Linux or Android OSes on top of MontaVista Linux host OS.

Resource control and isolation in OS virtualization are often achieved with the Linux kernel mechanism *CGroup*. For example, *Linux VServer* [86] enforces CPU isolation by overlaying a token bucket filter (TBF) on top of the standard O(1) Linux CPU scheduler. Each VM has a token bucket that accumulates incoming tokens at a specified rate; every timer tick, the VM that owns the running process is charged one token. The authors modified the TBF to provide fair sharing and/or work-conserving CPU reservations. CPU capacity is effectively partitioned between two classes of VMs: VMs with *reservations* get what they have reserved, and VMs with *shares* split the unreserved CPU capacity proportionally. For network I/O, the Hierarchical Token Bucket (HTB) queuing discipline of the Linux Traffic Control facility is used to provide network bandwidth reservations and fair service among VMs. Disk I/O is managed using the standard Linux CFQ (Completely-Fair Queuing) I/O scheduler, which attempts to divide the bandwidth of each block device fairly among the VMs performing I/O to that device.

OS virtualization has been traditionally applied in server virtualization, since it is essentially a name-space separation technique, but has limited support for I/O device virtualization (typically only disk and network devices). Recently, Andrus *et al*. [89] developed *Cells*, an OS virtualization architecture for enabling multiple virtual Android smart phones to run simultaneously on the same physical cell phone in an isolated, secure manner. The main technical challenges is to develop kernel-level and user-level device virtualization techniques for a variety of handset devices like GPU, touch screen, GPS, etc.

8. Task-Grain Scheduling

Conventional VMM schedulers view each guest VM as an opaque blackbox and unaware of individual tasks within each VM. This is true for both full-virtualization solutions like KVM and para-virtualization solutions like Xen. This opacity is beneficial for system modularity, but can sometimes be a impediment to optimal perform-

ance. Some authors have implemented task-grain scheduling schemes, where the VMM scheduler has visibility into the internal task-grain details within each VM. This work can be further divided into two types: one approach is to modify the guest OS to *inform* the VMM about its internal tasks via hypercalls in a paravirtualization architecture; another approach is to let the VMM *infer* the task-grain information transparently without guest OS modification in a full-virtualization architecture.

8.1. Approaches That Require Modifying the Guest OS

Augier *et al*. [90] implemented task-grain scheduling algorithm within VirtualLogix VLX, by removing the scheduler from each guest OS and letting the hypervisor handle scheduling of all tasks.

KimD *et al*. [91] presented a guest-aware, task-grain scheduling algorithm in Xen by selecting the next VM to be scheduled based on the highest-priority task within each VM and the I/O behavior of the guest VMs via inspecting the I/O pending status.

Kiszka [4] developed a para-virtualized scheduling interface for KVM. Two hypercalls are added: one for letting the guest OS inform the VMM about the priority of currently running task in the guest OS, and the VMM schedules the guest VMs based on task priority information; another for the guest OS to inform the VMM that it has just finished handling an interrupt, and its priority should be lowered to its nominal priority.

Xia *et al*. [92] presented PaS (Preemption-aware Scheduling) with two interfaces: one to register VM preemption conditions, the other to check if a VM is preempting. VMs with incoming pending I/O events are given higher priorities and allowed to preempt the currently running VM scheduled by the Credit Scheduler. Instead of adding hypercalls, the PaS interface is implemented by adding two fields in a data structure which is shared between the hypervisor and guest VM.

Wang *et al*. [93] implemented task-grain scheduling within Xen by adding a hypercall to inform the VMM about the timing attributes of a task at its creation time, and using either fixed-priority or EDF to schedule the tasks directly, bypassing the guest VM scheduler.

Kim *et al*. [94] presented a feedback control architecture for adaptive scheduling in Xen, by adding two hypercalls for the guest VMs to increase or decrease the CPU budget (slice size) requested per period based on CPU utilization and deadline miss ratios in the guest VMs.

8.2. Approaches That Do Not Require Modifying the Guest OS

To avoid modifying the guest OS, the VMM attempts to *infer* graybox knowledge about a VM by monitoring certain HW events. Kim *et al*. [95] presented task-aware virtual machine scheduling for I/O Performance by introducing a *partial boosting* mechanism into the Xen Credit Scheduler, which enables the VMM to boost the priority of a vCPU that contains an *inferred* I/O-bound task in response to an incoming event. The correlation mechanism for block and network I/O events is best-effort and lightweight. It works by monitoring the *CR3* register on x86 CPU for addressing the MMU, in order to capture process context-switching events, which are classified into disjoint classes: *positive evidence, negative evidence, and ambiguity*, to determine a *degree of belief* on the I/O boundedness of tasks. Kim *et al*. [96] presented an improved credit scheduler that transparently estimates multimedia quality by *inferring* frame rates from low-level hardware events, and dynamically adjusts CPU allocation based on the estimates as feedback to keep the video frame rate around a set point *Desired Frame Rate*. For memory-mapped display, the hypervisor monitors the frequency of frame buffer writes in order to estimate the frame rates; for GPU-accelerated display, the hypervisor monitors the rate of interrupts raised by a video device, which is shown to exhibit good linear correlation with frame rate.

Tadokoro *et al*. [97] implemented the Monarch scheduler in the Xen VMM, which works by monitoring and manipulating the run queue and the process data structure in guest VMs without modifying guest OSes, using *Virtual Machine Intraspection* (*VMI*) to monitor process execution in the guest VMs, and *Direct Kernel Object Manipulation* (*DKOM*) to modify the scheduler queue data structure in the guest VMs.

9. The Lock-Holder Preemption Problem

Spinlocks are used extensively in the Linux kernel for synchronizing access to shared data. When one thread is in its critical section holding a spinlock, another thread trying to acquire the same spinlock to enter the critical section will busy-wait (spin in a polling loop and continuously check the locking condition) until the lock-holding thread releases the spinlock. Since the typical critical sections protected by the spinlocks are kept very short, the waiting times are acceptable for today's Linux kernels, especially with real-time enhancements like the PREEMPT_RT patch. However, in a virtualized environment, *Lock Holder Preemption* (*LHP*) may cause excessively long waiting times for guest OSes configured with SMP support with multiple vCPUs in the same guest VM. A VM is called *overcommitted* when its number of vCPUs exceeds the available number of physical cores. Since the VMM scheduler is not aware of any spinlocks held within the guest OSes, when thread A in a

guest OS holding a spinlock being waited for by another thread B on a different vCPU in the same guest OS is preempted by the VMM scheduler to run another vCPU, possibly in a different guest OS, Thread B will busy-wait for a long time until the VMM scheduler switches back to this guest OS to allow thread A to continue execution and release the spinlock. When LHP occurs, the waiting times can be tens of milliseconds instead of the typical tens of microseconds in the Linux kernel without virtualization.

There are two general approaches to addressing the LHP problem: *preventing* it from occurring, or *mitigating* it after detecting that it has occurred.

Pause Loop Exit (*PLE*) is an HW mechanism built-in the latest Intel processors for detecting guest spin-lock. It can be used by the VMM software to mitigate LHP after detection. It enables the VMM to detect spin-lock by monitoring the execution of PAUSE instructions in the VM through the *PLE_Gap* and *PLE_Window* values. PLE_Gap denotes upper bound on the amount of time between two successive executions of PAUSE in a loop. PLE_Window denotes upper bound on the amount of time a guest is allowed to execute in a PAUSE loop. If the amount of time between this execution of PAUSE and previous one exceeds the PLE_Gap, then processor consider this PAUSE belongs to a new loop. Otherwise, processor determines the total execution time of this loop (since 1st PAUSE in this loop), and triggers a VM exit if total time exceeds PLE_Window. According to KVM's source code, PLE_Gap is set to 41 and PLE_Window is 4096 by default, which means that this approach can detect a spinning loop that lasts around 55 microseconds on a 3GHz CPU. AMD processors has similar HW support called *Pause-Filter-Count*. (Since these HW mechanisms are relatively new, the techniques discussed below generally do not make use of them.)

Uhlig *et al*. [98] presented techniques to address LHP for both fully virtualized and para-virtualized environments. In the paravirtualization case, the guest OS is modified to add a *delayed preemption* mechanism: before acquiring a spinlock, the guest OS invokes a hypercall to indicate to the VMM that it should not be preempted for the next *n* microseconds, where *n* is a configurable parameter depending how long the guest OS expects to be holding a lock. After releasing the lock, the guest OS indicates its willingness to be preempted again. In the full virtualization case where the guest OS cannot be modified, the VMM transparently monitors guest OS switches between user space and kernel space execution, and prevents a vCPU from being preempted when it is running in kernel space and may be holding kernel spinlocks. (The case when a user-level application is holding a spinlock is not handled, and it is assumed that the application itself is responsible for handling this case.) The implementation is based on a L4-ka microkernel-based virtualization system with a paravirtualized Linux as guest OS, although their techniques do not depend on the specific virtualization software.

Friebel [99] extended the spinlock waiting code in the guest OS to issue a *yield*() hypercall when a vCPU has been waiting longer than a certain threshold of 2^{16} cycles. Upon receiving the hypercall, the VMM schedules another VCPU of the same guest VM, giving preference to vCPUs preempted in kernel mode because they are likely to be preempted lock-holders.

Coscheduling, also called Gang Scheduling, is a common technique for preventing the LHP problem, where all vCPUs of a VM are simultaneously scheduled on physical processors (PCPUs) for an equal time slice. Strict co-scheduling may lead to performance degradation due to fragmentation of processor availability. To improve runtime efficiency, VMWare ESX 4.x implements relaxed co-scheduling to allow vCPUs to be co-scheduled with slight skews. It maintain synchronous progress of vCPU siblings by deferring the advanced vCPUs until the slower ones catch up. Several authors developed various versions of relaxed co-scheduling, as we discuss next.

Jiang *et al*. [100] presented several optimizations to the CFS (Completely Fair Scheduler) in KVM to help prevent or mitigate LHP. Three techniques were proposed for LHP prevention: 1) add two hypercalls that enable the guest VM to inform the host VMM that one of its vCPU threads is currently holding a spinlock or just released a spinlock. The host VMM will not de-schedule a vCPU that is currently holding a spinlock; 2) add two hypercalls for one vCPU thread to boost its priority when acquiring a spinlock, and resume its nominal priority when releasing it; 3) implement *approximate co-scheduling* by temporarily changing a VM's scheduling policy from SCHED_OTHER (default CFS scheduling class) to SCHED_RR (real-time scheduling class) when the VM has high degree of internal concurrency, and changing it back a short period of time later. Two techniques were proposed for LHP mitigation after it occurs: 1) Add a hypercall *yield*() in the vCPU thread source code to let it give up the CPU when it has been spin-waiting for a spinlock for too long, by adding a counter in the source code to record the spin-waiting time; 2) implement co-descheduling by slowing down or stopping all other vCPUs of the same VM if one of its vCPUs is descheduled, since if a vCPU thread requires a spinlock, it is possible for the other vCPUs to require the same spinlock later.

Weng *et al*. [101] presented a *hybrid scheduling* framework in Xen by distinguishing between two types of VMs: the *high-throughput* type and the *concurrent* type. A VM is set as the concurrent type when the majority of

its workload is concurrent applications in order to reduce the cost of synchronization, and otherwise set as the high-throughput type by default. Weng *et al*. [102] implemented dynamic co-scheduling by adding *vCPU Related Degree* (*VCRD*) to reflect the degree of relatedness of vCPUs in a VM. When long waiting times occur in a VM due to spinlocks, the VCRD of the VM is set as HIGH, and all vCPUs should be co-scheduled; otherwise, the VCRD of the VM is set as LOW, and vCPUs can be scheduled asynchronously. A monitoring module is added in each VM to detect long waiting times due to spinlocks, and uses a hypercall to inform the host VMM about the VCRD values of the guest VM.

Bai *et al*. [103] presented a task-aware co-scheduling method, where the VMM infers gray-box knowledge about the concurrency and synchronization of tasks within guest VMs. By tracking the states of tasks within guest VMs and combining this information with states of vCPUs, they are able to *infer* occurrence of spinlock waiting without modifying the guest OS.

Yu *et al*. [104] presented two approaches: *Partial Co-scheduling* and *Boost Co-scheduling*, both implemented within the Xen credit scheduler. Partial Co-scheduling allows multiple VMs to be co-scheduled concurrently, but only raises co-scheduling signals to the CPUs whose run-queues contain the corresponding co-scheduled VCPUs, while other CPUs remain undisturbed; Boost Co-scheduling promotes the priorities of vCPUs to be co-scheduled, instead of using co-scheduling signals to force them to be co-scheduled. They performed performance comparisons with hybrid co-scheduling [101] and co-de-scheduling [100] and showed clear advantages.

Sukwong *et al*. [105] presented a *balance scheduling* algorithm which attempts to balance vCPU siblings in the same VM on different physical CPUs without forcing the vCPUs to be scheduled simultaneously by dynamically setting CPU affinity of vCPUs so that no two vCPU siblings are in the same CPU run-queue. Balance scheduling is shown to significantly improve application performance without the complexity and drawbacks found in co-scheduling (CPU fragmentation, priority inversion and execution delay). Balance scheduling is similar to *relaxed co-scheduling* in VMWare ESX 4.x [106], but it never delays execution of a vCPU to wait for another vCPU in order to maintain synchronous progress of vCPU siblings. It differs from the *approximate co-scheduling* algorithm in [100] in that it does not change vCPU scheduling class or priorities.

SPUMONE [71] adopts a unique runtime migration approach to address the LHP (Lock-Holder Preemption) problem by migrating a vCPU away from a core where LHP may occur with another co-located vCPU: if vCPUs of a GPOS and an RTOS share the same pCPU core on a multicore processor, and the thread running on the vCPU of GPOS issues a system call, or when an interrupt is raised to the vCPU of GPOS, the vCPU of GPOS is immediately migrated away from to another core that does not host any RTOS vCPUs, thus achieving the effect of co-scheduling on different cores. This is a pessimistic approach, since the migration takes place even when there may not be any lock contention.

10. Conclusion

In this paper, we have presented a comprehensive survey on real-time issues in embedded systems virtualization. We hope this article can serve as a useful reference to researchers in this area.

11. Acknowledgements

This work was supported by MoE-Intel Information Technology Special Research Fund under Grant Number MOE-INTEL-10-02; NSFC Project Grant #61070002; the Fundamental Research Funds for the Central Universities.

REFERENCES

[1] Y. Kinebuchi, H. Koshimae and T. Nakajima, "Constructing Machine Emulator on Portable Microkernel," *Proceedings of the* 2007 *ACM Symposium on Applied Computing*, 11-15 March 2007, Seoul, 2007, pp. 1197-1198. doi:10.1145/1244002.1244261

[2] H. Schild, A. Lackorzynski and A. Warg, "Faithful Virtualization on a Realtime Operating System," 11*th Real-Time Linux Workshop*, Dresden, 28-30 September 2009.

[3] K. Barr, P. P. Bungale, S. Deasy, V. Gyuris, P. Hung, C. Newell, H. Tuch and B. Zoppis, "The VMware Mobile Virtualization Platform: Is That a Hypervisor in Your Pocket?" *Operating Systems Review*, Vol. 44, No. 4, 2101, pp. 124-135. doi:10.1145/1899928.1899945

[4] J. Kiszka, "Towards Linux as a Real-Time Hypervisor," 11*th Real-Time Linux Workshop*, Dresden, 28-30 September 2009.

[5] Airlines Electronic Engineering Committee, "ARINC 653—Avionics Application Software Standard Interface," 2003

[6] T. Gaska, B. Werner and D. Flagg, "Applying Virtualization to Avoinics Systems—The Integration Challenge," *Proceedings of the* 29*th Digital Avionics Systems Conference of the IEEE/AIAA*, Salt Lake City, 3-7 October 2010, pp. 1-19. doi:10.1109/DASC.2010.5655297

[7] Wind River Hypervisor Product Overview. http://www.windriver.com/products/hypervisor

[8] INTEGRITY Multivisor Datasheet. http://www.ghs.com

[9] Real-Time Systems GmbH. http://www.real-time-systems.com

[10] Tenasys eVM for Windows. http://www.tenasys.com/products/evm.php

[11] Quick Start Guide, NI Real-Time Hypervisor.

http://www.ni.com/pdf/manuals/375174b.pdf.

[12] OpenSynergy COQOS. http://www.opensynergy.com/en/Products/COQOS

[13] Enea Hypervisor. http://www.enea.com/software/products/hypervisor

[14] SysGO GmbH. http://www.sysgo.com

[15] S. H. VanderLeest, "ARINC 653 Hypervisor," *Proceedings of 29th Digital Avionics Systems Conference of IEEE/AIAA*, Salt Lake City, 3-7 October 2010, pp. 1-20. doi:10.1109/DASC.2010.5655298

[16] M. Masmano, I. Ripoll, A. Crespo and J. J. Metge "XtratuM: A Hypervisor for Safety Critical Embedded Systems," *Proceedings of Real-Time Linux Workshop*, Dresden, 28-30 September 2009.

[17] J. Zamorano and J. A. de la Puente, "Open Source Implementation of Hierarchical Scheduling for Integrated Modular Avionics," *Proceedings of Real-Time Linux Workshop*, Nairobi, 25-27 October 2010.

[18] Á. Esquinas, J. Zamorano, J. Antonio de la Puente, M. Masmano, I. Ripoll and A. Crespo, "ORK+/XtratuM: An Open Partitioning Platform for Ada," *Lecture Notes in Computer Science*, Vol. 6652, pp. 160-173. doi:10.1007/978-3-642-21338-0_12

[19] S. Campagna, M. Hussain and M. Violante, "Hypervisor-Based Virtual Hardware for Fault Tolerance in COTS Processors Targeting Space Applications," *Proceedings of the* 2010 *IEEE 25th International Symposium on Defect and Fault Tolerance in VLSI Systems*, Kyoto, 6-8 October 2010, pp. 44-51. doi:10.1109/DFT.2010.12

[20] N. McGuire, A. Platschek and G. Schiesser, "OVERSEE—A Generic FLOSS Communication and Application Platform for Vehicles," *Proceedings of 12th Real-Time Linux Workshop*, Nairobi, 25-27 October 2010.

[21] A. Platschek and G. Schiesser, "Migrating a OSEK Runtime environment to the OVERSEE Platform," *Proceedings of 13th Real-Time Linux Workshop*, Prague, 22-22 October 2011.

[22] S. Han and H. W. Jin, "Full Virtualization Based ARINC 653 Partitioning," *Proceedings of the 30th Digital Avionics Systems* (*DASC*) *Conference of the IEEE/AIAA*, Seattle, 16-20 October 2011, pp. 1-11. doi:10.1109/DASC.2011.6096132

[23] S. Ghaisas, G. Karmakar, D. Shenai, S. Tirodkar and K. Ramamritham, "SParK: Safety Partition Kernel for Integrated Real-Time Systems," *Lecture Notes in Computer Science*, Vol. 6462, 2010, pp. 159-174. doi:10.1007/978-3-642-17226-7_10

[24] M. Lemerre, E. Ohayon, D. Chabrol, M. Jan and M.-B. Jacques, "Method and Tools for Mixed-Criticality Real-Time Applications within PharOS," *Proceedings of IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing Workshops*, Newport Beach, 28-31 March 2011, pp. 41-48. doi:10.1109/ISORCW.2011.15

[25] A. Aguiar and F. Hessel, "Virtual Hellfire Hypervisor: Extending Hellfire Framework for Embedded Virtualization Support," *Proceedings of the 12th International Symposium on Quality Electronic Design* (*ISQED*), Santa Clara, 14-16 March 2011, pp. 129-203. doi:10.1109/ISQED.2011.5770725

[26] F. Moraes, N. Calazans, A. Mello, L. Möller and L. Ost, "HERMES: An Infrastructure for Low Area Overhead Packet-Switching Networks on Chip," *Integration, the VLSI Journal*, Vol. 38, No. 1, 2004, pp. 69-93. doi:10.1106/j.vlsi.2004.03.003

[27] L. Cherkasova, D. Gupta and A. Vahdat, "Comparison of the Three CPU Schedulers in Xen," *ACM SIGMETRICS Performance Evaluation Review*, Vol. 35, No. 2, 2007, pp. 42-51. doi:10.1145/1330555.1330556

[28] A. Masrur, S. Drössler, T. Pfeuffer and S. Chakraborty, "VM-Based Real-Time Services for Automotive Control Applications," *Proceedings of the* 2010 *IEEE 16th International Conference on Embedded and Real-Time Computing Systems and Applications*, Macau, 23-25 August 2010, pp. 218-223. doi:10.1109/RTCSA.2010.38

[29] A. Masrur, T. Pfeuffer, M. Geier, S. Drössler and S. Chakraborty, "Designing VM Schedulers for Embedded Real-Time Applications," *Proceedings of the 7th IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, Taipei, 9-14 October 2011, pp. 29-38. doi:10.1145/2039370.2039378

[30] J. Lee, S. S. Xi, S. J. Chen, L. T. X. Phan, C. Gill, I. Lee, C. Y. Lu and O. Sokolsky, "Realizing Compositional Scheduling through Virtualization," *Technical Report*, University of Pennsylvania, Philadelphia, 2011.

[31] S. S. Xi, J. Wilson, C. Y. Lu and C. Gill, "RT-Xen: Towards Real-Time Hypervisor Scheduling in Xen," *Proceedings of the* 2011 *International Conference on Embedded Software*, Taipei, 9-14 October 2011, pp. 39-48.

[32] I. Shin and I. Lee, "Compositional Real-Time Scheduling Framework with Periodic Model," *ACM Transactions on Embedded Computing Systems*, Vol. 7, No. 3, 2008. doi:10.1145/1347375.1347383

[33] S. Yoo, Y.-P. Kim and C. Yoo, "Real-time Scheduling in a Virtualized CE Device," *Proceedings of* 2010 *Digest of Technical Papers International Conference on Consumer Electronics* (*ICCE*), Las Vegas, 9-13 January 2010, pp. 261-262. doi:10.1109/ICCE.2010.5418991

[34] J.-W. Jeong, S. Yoo and C. Yoo, "PARFAIT: A New Scheduler Framework Supporting Heterogeneous Xen-ARM Schedulers," *Proceedings of* 2011 *Consumer Communications and Networking Conference of the IEEE CCNC*, Las Vegas, 9-12 January 2011, pp. 1192-1196. doi:10.1109/CCNC.2011.5766431

[35] C.-H. Hong, M. Park, S. Yoo, C. Yoo and H. D. Considering, "Hypervisor Design Considering Network Performance for Multi-Core CE Devices," *Proceedings of* 2010 *Digest of Technical Papers International Conference on Consumer Electronics of the IEEE ICCE*, Las Vegas, 9-13 January 2010, pp. 263-264. doi:10.1109/ICCE.2010.5418708

[36] S. Yoo, K.-H. Kwak, J.-H. Jo and C. Yoo, "Toward Under-Millisecond I/O Latency in Xen-ARM," *The 2nd ACM SIGOPS Asia-Pacific Workshop on Systems*, Shanghai, 11-12 July 2011, p. 14.

[37] M. Lee, A. S. Krishnakumar, P. Krishnan, N. Singh and S. Yajnik, "Supporting Soft Real-Time Tasks in the Xen Hypervisor," *Proceedings of the 6th ACM SIGPLAN/SIGO-*

PS International Conference on Virtual Execution Environments, Pittsburgh, March 2010, pp. 97-108. doi:10.1145/1837854.1736012

[38] P. J. Yu, M. Y. Xia, Q. Lin, M. Zhu, S. Gao, Z. W. Qi, K. Chen and H. B. Guan, "Real-Time Enhancement for Xen Hypervisor," *Proceedings of the 8th International Conference on Embedded and Ubiquitous Computing of the IEEE/IFIP*, Hong Kong, 11-13 December 2010, pp. 23-30.

[39] Y. X. Wang, X. G. Wang and H. Guo, "An Optimized Scheduling Strategy Based on Task Type In Xen," *Lecture Notes in Electrical Engineering*, Vol. 123, 2011, pp. 515-522. doi:10.1007/978-3-642-25646-2_67

[40] H. C. Chen, H. Jin, K. Hu and M. H. Yuan, "Adaptive Audio-Aware Scheduling in Xen Virtual Environment," *Proceedings of the International Conference on Computer Systems and Applications* (*AICCSA*) *of the IEEE/ACS*, Hammamet, 16-19 May 2010, pp. 1-8. doi:10.1109/AICCSA.2010.5586974

[41] H. C. Chen, H. Jin and K. Hu, "Affinity-Aware Proportional Share Scheduling for Virtual Machine System," *Proceedings of the 9th International Conference on Grid and Cooperative Computing* (*GCC*), Nanjing, 1-5 November 2010, pp. 75-80. doi:10.1109/GCC.2010.27

[42] D. Gupta, L. Cherkasova, R. Gardner and A. Vahdat, "Enforcing Performance Isolation across Virtual Machines in Xen," *Proceedings of the ACM/IFIP/USENIX* 2006 *International Conference on Middleware*, Melbourne, 27 November-1 December 2006, pp. 342-362.

[43] D. Ongaro, A. L. Cox and S. Rixner, "Scheduling I/O in Virtual Machine Monitors," *Proceedings of the 4th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, Seattle, 5-7 March 2008, pp. 1-10. doi:10.1145/134256.1246258

[44] S. Govindan, J. Choi, A. R. Nath, A. Das, B. Urgaonkar, and A. Sivasubramaniam, "Xen and Co.: Communication-Aware CPU Management in Consolidated Xen-Based Hosting Platforms," *IEEE Transactions on Computers*, Vol. 58, No. 8, 2009, pp. 1111-1125. doi:10.1109/TC.2009.53

[45] Y. Y. Hu, X. Long, J. Zhang, J. He and L. Xia, "I/O Scheduling Model of Virtual Machine Based on Multi-Core Dynamic Partitioning," *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, Chicago, 21-25 June 2010, pp. 142-154. doi:10.1145/1851476.1851494

[46] J. G. Lee, K. W. Hur and Y. W. Ko, "Minimizing Scheduling Delay for Multimedia in Xen Hypervisor," *Communications in Computer and Information Science*, Vol. 199, 2011, pp. 96-108. doi:10.1007/978-3-642-23312-8_12

[47] J. Zhang, K. Chen, B. J. Zuo, R. H. Ma, Y. Z. Dong and H. B. Guan, "Performance Analysis towards a KVM-Based Embedded Real-Time Virtualization Architecture," *Proceedings of the 5th International Conference on Computer Sciences and Convergence Information Technology* (*ICCIT*), Seoul, 30 November-2 December 2010, pp. 421-426. doi:10.1109/ICCIT.2010.5711095

[48] B. J. Zuo, K. Chen, A. Liang, H. B. Guan, J. Zhang, R. H. Ma and H. B. Yang, "Performance Tuning towards a KVM-Based Low Latency Virtualization System," *Proceedings of the 2nd International Conference on Information Engineering and Computer Science* (*ICIECS*), Wuhan, 25-26 December 2010, pp. 1-4. doi:10.1109/ICIECS.2010.5678357

[49] T. Cucinotta, G. Anastasi and L. Abeni, "Respecting Temporal Constraints in Virtualised Services," *Proceedings of the 33rd Annual IEEE International Conference on Computer Software and Applications*, Seattle, Washington DC, 20-24 July 2009. pp. 73-78. doi:10.1109/COMPSAC.2009.118

[50] T. Cucinotta, D. Giani, D. Faggioli and F. Checconi, "Providing Performance Guarantees to Virtual Machines Using Real-Time Scheduling," *Proceedings of Euro-Par Workshops*, Naples, 31 August-3 September 2010, pp. 657-664.

[51] T. Cucinotta, F. Checconi and D. Giani, "Improving Responsiveness for Virtualized Networking under Intensive Computing Workloads," *Proceedings of the 13th Real-Time Linux Workshop*, Prague, 20-22 October 2011.

[52] F. Checconi, T. Cucinotta and M. Stein, "Real-Time Issues in Live Migration of Virtual Machines," *Proceedings of Euro-Par Workshops*, Delft, 29-28 August 2009, pp. 454-466.

[53] S. Kato, R. (Raj) Rajkumar and Y. Ishikawa, "A Loadable Real-Time Scheduler Framework for Multicore Platforms," *Proceedings of the 6th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications* (*RTCSA*), Macau, 23-25 August 2010.

[54] B. Lin and P. A. Dinda, "VSched: Mixing Batch And Interactive Virtual Machines Using Periodic Real-time Scheduling," *Proceedings of the ACM/IEEE SC* 2005 *Conference Supercomputing*, Seattle, 12-18 November 2005, p. 8. doi:10.1109/SC.2005.80

[55] M. Asberg, N. Forsberg, T. Nolte and S. Kato, "Towards Real-Time Scheduling of Virtual Machines without Kernel Modifications," *Proceedings of the* 2011 *IEEE 16th Conference on Emerging Technologies & Factory Automation* (*ETFA*), Toulouse, 5-9 September 2011, pp. 1-4. doi:10.1109/ETFA.2011.6059185

[56] M. Asberg, T. Nolte and S. Kato, "Towards Hierarchical Scheduling in Linux/Multi-Core Platform," *Proceedings of the* 2010 *IEEE Conference on Emerging Technologies and Factory Automation* (*ETFA*), Bilbao, 13-16 September 2010, pp. 1-4. doi:10.1109/ETFA.2010.5640999

[57] A. Acharya, J. Buford and V. Krishnaswamy, "Phone Virtualization Using a Microkernel Hypervisor," *Proceedings of the* 2009 *IEEE International Conference on Internet Multimedia Services Architecture and Applications* (*IMSAA*), Bangalore, 9-11 December 2009, pp. 1-6. doi:10.1109/IMSAA.2009.5439460

[58] A. Iqbal, N. Sadeque and R. I. Mutia, "An Overview of Microkernel, Hypervisor and Microvisor Virtualization Approaches for Embedded Systems," *Technical Report*, Lund University, Lund, 2009.

[59] G. Heiser, "Virtualizing Embedded Systems: Why Bother?" *Proceedings of the* 2011 *48th ACM/EDAC/IEEE Conference on Design Automation*, San Diego, 5-9 June 2011, pp. 901-905.

[60] F. Bruns, S. Traboulsi, D. Szczesny, M. E. Gonzalez, Y. Xu and A. Bilgic, "An Evaluation of Microkernel-Based

Virtualization for Embedded Real-Time Systems," *Proceedings of the 22nd Euromicro Conference on Real-Time Systems* (*ECRTS*), Dublin, 6-9 July 2010, pp. 57-65. doi:10.1109/ECRTS.2010.28

[61] A. Lackorzynski, J. Danisevskis, J. Nordholz and M. Peter, "Real-Time Performance of L4Linux," *Proceedings of the 13th Real-Time Linux Workshop*, Prague, 20-22 October 2011.

[62] R. Kaiser, "Alternatives for Scheduling Virtual Machines in Real-Time Embedded Systems," *Proceedings of the 1st Workshop on Isolation and Integration in Embedded Systems*, Glasgow, 1 April 2008, pp. 5-10. doi:10.1145/1435458.1435460

[63] J. Yang, H. Kim, S. Park, C. K. Hong and I. Shin, "Implementation of Compositional Scheduling Framework on Virtualization," *SIGBED Review*, Vol. 8, No. 1, 2011, pp. 30-37. doi:10.1145/1967021.1967025

[64] U. Steinberg and B. Kauer, "NOVA: A Microhypervisor-Based Secure Virtualization Architecture," *Proceedings of the 5th European Conference on Computer Systems*, Paris, 13-16 April 2010, pp. 209-222. doi:10.1145/1755913.1755935

[65] K. Gudeth, M. Pirretti, K. Hoeper and R. Buskey, "Delivering Secure Applications on Commercial Mobile Devices: The Case for Bare Metal Hypervisors," *Proceedings of the 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices*, Chicago, 17-21 October 2011, pp. 33-38. doi:10.1145/2046614.2046622

[66] VMware MVP (Mobile Virtualization Platform) http://www.vmware.com/products/mobile

[67] F. Armand and M. Gien, "A Practical Look at Micro-Kernels and Virtual Machine Monitors," *Proceedings of the 6th IEEE Consumer Communications and Networking Conference*, Las Vegas, 10-13 January 2009, pp. 1-7. doi:10.1109/CCNC.2009.4784874

[68] Red Bend VLX for Mobile Handsets. http://www.virtuallogix.com/products/vlx-for-mobile-handsets.html

[69] J.-Y. Hwang, S.-B. Suh, S.-K. Heo, C.-J. Park, J.-M. Ryu, S.-Y. Park and C.-R. Kim, "Xen on ARM: System Virtualization Using Xen Hypervisor for ARM-Based Secure Mobile Phones," *Proceedings of the 5th IEEE Consumer Communications and Networking Conference*, Las Vegas, 10-12 January 2008, pp. 257-261. doi:10.1109/ccnc08.2007.64

[70] W. Kanda, Y. Yumura, Y. Kinebuchi, K. Makijima and T. Nakajima, "SPUMONE: Lightweight CPU Virtualization Layer for Embedded Systems," *Proceedings of the IEEE/IFIP International Conference on Embedded and Ubiquitous Computing*, Shanghai, 17-20 December 2008, pp. 144-151. doi:10.1109/EUC.2008.157

[71] H. Mitake, Y. Kinebuchi, A. Courbot and T. Nakajima, "Coexisting Real-Time OS and General Purpose OS on an Embedded Virtualization Layer for a Multicore Processor," *Proceedings of the* 2011 *ACM Symposium on Applied Computing*, Taichung, 21-24 March 2011, pp. 629-630. doi:10.1145/1982185.1982322

[72] H. Tadokoro, K. Kourai and S. Chiba, "A Secure System-wide Process Scheduler across Virtual Machines," *Proceedings of the 16th Pacific Rim International Symposium on Dependable Computing* (*PRDC*), Tokyo, 13-15 December 2010, pp. 27-36. doi:10.1109/PRDC.2010.34

[73] Y. Kinebuchi, M. Sugaya, S. Oikawa and T. Nakajima, "Task Grain Scheduling for Hypervisor-Based Embedded System," *Proceedings of the 10th IEEE International Conference on High Performance Computing and Communications*, Dalian, 25-27 September 2008, pp. 190-197. doi:10.1109/HPCC.2008.144

[74] A. Aalto, "Dynamic Management of Multiple Operating Systems in an Embedded Multi-Core Environment," Master's Thesis, Aalto University, Finland, 2010.

[75] T.-H. Lin, Y. Kinebuchi, H. Shimada, H. Mitake, C.-Y. Lee and T. Nakajima, "Hardware-Assisted Reliability Enhancement for Embedded Multi-Core Virtualization Design," *Proceedings of the 17th International Conference on Embedded and Real-Time Computing Systems and Applications of the IEEE RTCSA*, Toyama, 28-31 August 2011, pp. 241-249. doi:10.1109/RTCSA.2011.24

[76] N. Li, Y. Kinebuchi and T. Nakajima, "Enhancing Security of Embedded Linux on a Multi-Core Processor," *Proceedings of the 17th International Conference on Embedded and Real-Time Computing Systems and Applications of the IEEE RTCSA*, Toyama, 28-31 August 2011, pp. 117-121. doi:10.1109/RTCSA.2011.36

[77] W. Kanda, Y. Murata and T. Nakajima, "SIGMA System: A Multi-OS Environment for Embedded Systems," *Journal Of Signal Processing Systems*, Vol. 59, No. 1, 2010, pp. 33-43. doi:10.1007/s11265-008-0272-9

[78] M. Ito and S. Oikawa, "Mesovirtualization: Lightweight Virtualization Technique for Embedded Systems," *Sociedad Española de UitraSonidos*, Vol. 4761, 2007, pp. 496-505.

[79] M. Ito and S. Oikawa, "Making a Virtual Machine Monitor Interruptible," *Journal of Information Processing*, Vol. 19, 2011, pp. 411-420.

[80] S.-H. Yoo, Y. X. Liu, C.-H. Hong, C. Yoo and Y. G. Zhang, "MobiVMM: A Virtual Machine Monitor for Mobile Phones," *Proceedings of the 1st Workshop on Virtualization in Mobile Computing*, Breckenridge, 17 June 2008, pp. 1-5.

[81] S. Yoo, M. Park and C. Yoo, "A Step to Support Real-Time in Virtual Machine," *Proceedings of the 6th International Conference on Consumer Communications and Networking of the IEEE CCNC*, Las Vegas, 10-13 January 2009, pp. 1-7. doi:10.1109/CCNC.2009.4784876

[82] Y. Li, M. Danish and R. West, "Quest-V: A Virtualized Multikernel for High-Confidence Systems," *Technical Report*, Boston University, Boston, 2011.

[83] M. Danish, Y. Li and R. West, "Virtual-CPU Scheduling in the Quest Operating System," *Proceedings of the 17th International Conference on Real-Time and Embedded Technology and Applications Symposium of the IEEE RTAS*, Chicago, 11-14 April 2011, pp. 169-179. doi:10.1109/RTAS.2011.24

[84] P.-H. Kamp and R. N. M. Watson, "Jails: Confining the Omnipotent Root," *Proceedings of the 2nd International System Administration and Networking Conference*, Maas-

tricht, 22-25 May 2000.

[85] OpenVZ Linux Containers. http://wiki.openvz.org

[86] S. Soltesz, H. Pötzl, M. E. Fiuczynski, A. C. Bavier and L. L. Peterson, "Container-Based Operating System Virtualization: A Scalable, High-Performance Alternative to Hypervisors," *Proceedings of EuroSys* 2007, Lisbon, 21-23 March 2007, pp. 275-287.

[87] Parallels Virtuozzo Containers. http://www.parallels.com/products/pvc

[88] MontaVista Automotive Technology Platform. http://mvista.com/sol_detail_ivi.php

[89] J. Andrus, C. Dall, A. V. Hof, O. Laadan and J. Nieh, "Cells: A Virtual Mobile Smartphone Architecture," *Columbia University Computer Science Technical Reports*, Columbia University, Columbia, 2011, pp. 173-187.

[90] C. Augier, "Real-Time Scheduling in a Virtual Machine Environment," *Proceedings of Junior Researcher Workshop on Real-Time Computing* (*JRWRTC*), Nancy, 29-30 March 2007.

[91] D. Kim, H. Kim, M. Jeon, E. Seo and J. Lee, "Guest-Aware Priority-Based Virtual Machine Scheduling for Highly Consolidated Server," *Proceedings of the* 14*th International Euro-Par Conference on Parallel Processing*, Las Palmas de Gran Canaria, 26-29 August 2008, pp. 285-294. doi:10.1007/978-3-540-85451-7_31

[92] Y. B. Xia, C. Yang and X. Cheng, "PaS: A Preemption-Aware Scheduling Interface for Improving Interactive Performance in Consolidated Virtual Machine Environment," *Proceedings of the* 15*th International Conference on Parallel and Distributed Systems* (*ICPADS*), Shenzhen, 8-11 December 2009, pp. 340-347. doi:10.1109/ICPADS.2009.51

[93] Y. D. Wang, J. Zhang, L. H. Shang, X. Long and H. H. Jin, "Research of Real-Time Task in Xen Virtualization Environment," *Proceedings of the* 2*nd International Conference on Computer and Automation Engineering* (*ICCAE*), Mumbai, 26-28 February 2010, pp. 496-500. doi:10.1109/ICCAE.2010.5451903

[94] B. K. Kim, K. W. Hur, J. H. Jang and Y. W. Ko, "Feedback Scheduling for Realtime Task on Xen Virtual Machine," Communication and Networking, Vol. 266, 2011, pp. 283-291. doi:10.1007/978-3-642-27201-1_32

[95] H. Kim, H. Lim, J. Jeong, H. Jo and J. Lee, "Task-Aware Virtual Machine Scheduling for I/O Performance," *Proceedings of the* 2009 *ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, Washington DC, 11-13 March 2009, pp. 101-110. doi:10.1145/1508293.1508308

[96] H. Kim, J. Jeong, J. Hwang, J. Lee and S. Maeng, "Scheduler Support for Video-Oriented Multimedia on Client-Side Virtualization," *Proceedings of the* 3*rd Multimedia Systems Conference*, Orange County, 22 February 2012. doi:10.1145/2155555.2155566

[97] H. Tadokoro, K. Kourai and S. Chiba, "A Secure System-wide Process Scheduler across Virtual Machines," *Proceedings of the* 16*th Pacific Rim International Symposium on Dependable Computing of the IEEE PRDC*, Tokyo, 13-15 December 2010, pp. 27-36. doi:10.1109/PRDC.2010.34

[98] V. Uhlig, J. LeVasseur, E. Skoglund and U. Dannowski, "Towards Scalable Multiprocessor Virtual Machines," *Proceedings of the* 3*rd Conference on Virtual Machine Research and Technology Symposium*, San Jose, 6-12 May 2004, pp. 43-56.

[99] T. Friebel and S. Biemueller, "How to Deal with Lock Holder Preemption," *Proceedings of the Xen Summit*, Boston, 23-24 June 2008.

[100] W. Jiang, Y. S. Zhou, Y. Cui, W. Feng, Y. Chen, Y. C. Shi and Q. B. Wu, "CFS Optimizations to KVM Threads on Multi-Core Environment," *Proceedings of the* 15*th International Conference on Parallel and Distributed Systems of the IEEE ICPADS*, Shenzhen, 8-11 December 2009, pp. 348-354. doi:10.1109/ICPADS.2009.83

[101] C. L. Weng, Z. G. Wang, M. L. Li and X. D. Lu, "The Hybrid Scheduling Framework for Virtual Machine Systems," *Proceedings of the* 2009 *ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, Washington DC, 11-13 March 2009, pp. 111-120. doi:10.1145/1508293.1508309

[102] C. L. Weng, Q. Liu, L. Yu and M. L. Li, "Dynamic Adaptive Scheduling for Virtual Machines," *Proceedings of the* 20*th International Symposium on High Performance Distributed Computing*, San Jose, 8-11 June 2011, pp. 239-250. doi:10.1145/1996130.1996163

[103] Y. B. Bai, C. Xu and Z. Li, "Task-Aware Based Co-Scheduling for Virtual Machine System," *Proceedings of the* 2010 *ACM Symposium on Applied Computing*, Sierre, 22-26 March 2010, pp. 181-188. doi:10.1145/1774088.1774126

[104] Y. L. Yu, Y. X. Wang, H. Guo and X. B. He, "Hybrid Co-Scheduling Optimizations for Concurrent Applications in Virtualized Environments," *Proceedings of the 6th IEEE International Conference on Networking, Architecture and Storage* (*NAS*), Dalian, 28-30 July 2011, pp. 20-29. doi:10.1109/NAS.2011.30

[105] O. Sukwong and H. S. Kim, "Is Co-Scheduling Too Expensive for SMP VMs?" *Proceedings of EuroSys* 2011, Salzburg, 10-13 April 2011, pp. 257-272.

[106] VMWARE White Paper, Performance Tuning Best Practices for ESX Server, 2007. http://www.vmware.com/pdf/vi_performance_tuning.pdf

