Report Number: WUCSE-2013-109

2013

Real-Time Multi-Core Virtual Machine Scheduling in Xen

Authors: Sisu Xi, Meng Xu, Chenyang Lu, Linh T.X. Phan, Christopher Gill, Olga Sokolsky, and Insup Lee

Recent years have witnessed two major trends in the development of complex real-time systems. First, to reduce cost and enhance flexibility, multiple systems are sharing common computing platforms via virtualization technology, instead of being deployed separately on physically isolated hosts. Second, multicore processors are increasingly being used in real-time systems. The integration of real-time systems as virtual machines (VMs) atop common multicore platforms raises significant new research challenges in meeting the real-time performance requirements of multiple systems.

2013-109

Real-Time Multi-Core Virtual Machine Scheduling in Xen

Authors: Sisu Xi, Meng Xu, Chenyang Lu, Linh T.X. Phan, Christopher Gill, Oleg Sokolsky, Insup Lee

Abstract: Recent years have witnessed two major trends in the development of complex real-time systems. First, to reduce cost and enhance flexibility, multiple systems are sharing common computing platforms via virtualization technology, instead of being deployed separately on physically isolated hosts. Second, multicore processors are increasingly being used in real-time systems. The integration of real-time systems as virtual machines (VMs) atop common multicore platforms raises significant new research challenges in meeting the real-time performance requirements of multiple systems.

Type of Report: Other

Real-Time Multi-Core Virtual Machine Scheduling in Xen

Sisu Xi[†] Meng Xu[‡] Chenyang Lu[†] Linh T.X. Phan[‡] Christopher Gill[†] Oleg Sokolsky[‡] Insup Lee[‡]

[†]*Washington University in St. Louis*

[‡]*University of Pennsylvania*

***Abstract*—Recent years have witnessed two major trends in the development of complex real-time systems. First, to reduce cost and enhance flexibility, multiple systems are sharing common computing platforms via virtualization technology, instead of being deployed separately on physically isolated hosts. Second, multicore processors are increasingly being used in real-time systems. The integration of real-time systems as virtual machines (VMs) atop common multicore platforms raises significant new research challenges in meeting the real-time performance requirements of multiple systems.**

This paper advances the state of the art in real-time virtualization by designing and implementing RT-Xen 2.0, a new real-time multicore VM scheduling framework in the popular Xen virtual machine monitor (VMM). RT-Xen 2.0 realizes a suite of real-time VM scheduling policies that span the design space. We implement both global and partitioned VM schedulers; each scheduler can be configured to support dynamic or static priorities and to run VMs as periodic or deferrable servers. We present a comprehensive experimental evaluation that provides important insights into real-time scheduling on virtualized multicore platforms: (1) both global and partitioned VM scheduling can be implemented in the VMM at moderate overhead; (2) at the VMM scheduler level, in compositional schedulability theory partitioned EDF (pEDF) is better than global EDF (gEDF) in schedulability guarantees, but in our experiments their actual performance is reversed in terms of the fraction of workloads that meet their deadlines on virtualized multicore platforms; (3) at the guest OS level, pEDF requests a smaller total VCPU bandwidth than gEDF, based on compositional schedulability analysis, and therefore using pEDF in the guest OS level led to more schedulable workloads in our experiments. The combination of pEDF in guest OS and gEDF in the VMM therefore resulted in the best experimental performance when using a periodic server; and (4) the global EDF scheduler running VMs as deferrable servers leads to low deadline miss ratios under system overload when compared to other real-time VM scheduling policies.

I. Introduction

Complex real-time systems are moving from physically isolated hosts towards common multicore computing platforms shared by multiple systems. Common platforms bring significant benefits including reduced cost and weight, as well as increased flexibility via dynamic resource allocation. However, the integration of real time systems as virtual machines (VM) on a common multicore computing platform brings significant challenges in simultaneously meeting the real-time performance requirements of multiple systems, which in turn require fundamental advances in the underlying VM scheduling at the virtual machine monitor (VMM) level.

As a step towards real-time virtualization technology for multicore processors, we have designed and implemented **RT-Xen 2.0**, a multicore real-time VM scheduling framework in Xen, an open-source VMM that has been widely adopted in both cloud computing and embedded systems [1], [2]. While earlier efforts on real-time scheduling in Xen (e.g., RT-Xen 1.0) focused on single-core schedulers [3], [4], RT-Xen 2.0 realizes a suite of multicore real-time VM scheduling policies spanning a large design space. We have implemented both global and partitioned VM schedulers (rt-global and rt-partition); each scheduler can be configured to support dynamic or static priorities and to run VMs as periodic or deferrable servers. Our scheduling framework therefore can support eight combinations of real-time VM scheduling policies, which enable us to perform comprehensive exploration of and experimentation with real-time VM scheduling on multicore processors. Moreover, RT-Xen 2.0 supports resource interfaces for VMs, which enable designers to calculate and specify the resource demands of VMs to the underlying RT-Xen 2.0 scheduler, thereby incorporating compositional schedulability theory [5]–[8] into a multicore virtualization platform.

We have conducted a series of experiments to evaluate the efficiency and real-time performance of RT-Xen 2.0 with LITMUS^{RT} as the guest OS. Our empirical results shed insights on the design and implementation of real-time VM scheduling:

- Both global and partitioned real-time VM scheduling can be realized within Xen at moderate overhead.
- At the VMM level, while compositional schedulability analysis shows that partitioned EDF (pEDF) outperforms global EDF (gEDF) in terms of schedulability guarantees for tasks running in VMs, experimentally gEDF often outperforms pEDF in terms of the fraction of workloads actually schedulable on a virtualized multicore processor.
- At the guest OS level, pEDF requests a smaller total VCPU bandwidth than gEDF based on compositional schedulability analysis, and therefore using pEDF in the guest OS leads to more schedulable workloads. The combination of pEDF in the guest OS and gEDF in the VMM therefore results in the best experimental performance when using a periodic server.
- When the guest OS scheduler uses gEDF, gEDF in the VMM when configured with deferrable servers leads to the lowest deadline miss ratio under system overload when compared to other real-time VM scheduling policies.

In the rest of this paper, in Section II we first introduce background on Xen scheduling and our earlier work on single-core real-time scheduling in RT-Xen 1.0. We then describe the design and implementation of the multicore real-time scheduling framework in RT-Xen 2.0 in Section III. Section IV presents our experimental evaluation. After reviewing other related work in Section V, we conclude the paper in Section VI.

II. Background

We first review the scheduling architecture in Xen, and then discuss the compositional schedulability analysis (CSA) that serves as the theoretical basis for the design of our multicore VMM scheduling framework, RT-Xen 2.0.

A. Scheduling in Xen and RT-Xen 1.0

Xen [9] is a popular open-source para-virtualization platform that has been developed over the past decade. It provides a layer called the virtual machine monitor (VMM) that allows multiple *domains* (VMs) to run different operating systems and to execute concurrently on shared hardware. Xen has a special domain, called *domain 0*, that is responsible for managing all other domains (called guest domains).

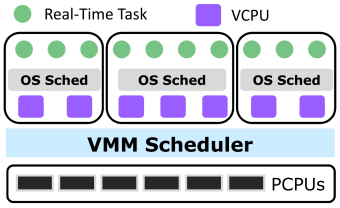


Fig. 1: Xen scheduling architecture.

Figure 1 illustrates the scheduling architecture in Xen. As shown in the figure, each domain has one or more virtual CPUs (VCPUs), and tasks within the domain are scheduled on these VCPUs by the domain's OS scheduler. The VCPUs of all domains are scheduled on the physical CPUs (PCPUs), or cores, by the VMM scheduler. At run time, each VCPU can be in either a *runnable* or *non-runnable* (blocked by I/O or idle) state. When there are no runnable VCPUs to execute, the VMM scheduler schedules an idle VCPU. By default, Xen boots one idle VCPU per PCPU.

VMM schedulers in Xen. There are three VMM schedulers in Xen: credit, credit2 and simple EDF (SEDF). The credit scheduler is the default one, and uses a proportional share scheme, where each domain is associated with a weight (which encodes the CPU resource share it will receive relative to other domains), and a cap (which encodes the maximum CPU resources it will receive). The credit scheduler is implemented using a partitioned queue, and uses a heuristic load balancing scheme: when the PCPU's RunQ only has one VCPU with credit left, the PCPU "steals" VCPUs from other RunQs to rebalance the load. The credit2 scheduler also assigns a weight per domain. The credit2 scheduler is still in an experimental phase, does not support caps and is not CPU-mask aware. As a result, it cannot limit each domain's CPU resource, nor can it dedicate cores to domains. Unlike the credit and credit2 schedulers, the SEDF scheduler schedules domains based on their interfaces (defined as period and budget) using an EDF-like scheduling policy. However, SEDF is not in active development and is planned to be "phased out and eventually removed" from Xen [10].

In our prior work, we developed RT-Xen 1.0 [3], a real-time scheduling platform based on Xen. RT-Xen 1.0 offered single-core static-priority real-time scheduling, where each domain is captured using a periodic resource model (PRM) [11] interface including a period and a budget within each period. It supports four server mechanisms (deferrable server, periodic server, polling server, and sporadic server) for implementing the domains' VCPUs, as well as two enhanced periodic servers [4] for overall system performance optimization. RT-Xen 1.0, however, only supported single-core real-time scheduling.

B. Compositional Schedulability Analysis

RT-Xen 2.0 supports resource interfaces for VMs based on multicore compositional schedulability analysis (CSA) [8] for resource allocation. This is enabled by a natural mapping between Xen and a two-level compositional scheduling hierarchy in CSA: each domain corresponds to an elementary component, and the system (a composition of the domains under the VMM scheduler) corresponds to the root component in CSA. Using this approach, we can represent the resource requirements of each domain as a multiprocessor periodic resource (MPR) interface, $\mu = \langle \Pi, \Theta, m' \rangle$, which specifies a resource allocation that provides a total of Θ execution time units in each period of Π time units, with a maximum level of parallelism of m'. Each interface $\mu = \langle \Pi, \Theta, m' \rangle$ can be mapped to m' VCPUs whose total resource demand is at least equal to the resource supply of the interface. The VCPUs of the domains are then scheduled together on the PCPUs by the VMM scheduler. Based on the VCPUs and the VMM scheduling semantics, we can also derive an interface for the system (i.e., root component), which can be used to check the schedulability of the system or to determine the minimum number of cores needed to schedule the system feasibly.

The interfaces of domains under pEDF or partitioned Deadline Monotonic (pDM) can be computed using the uniprocessor interface computation method in [5]. The interfaces under gEDF can be computed using the multicore interface computation method described in [6]. Finally, the interfaces under gDM (global DM) can be computed using the interface computation method described in [7].

III. Design and Implementation

In this section, we first describe the design principles behind the multicore real-time scheduling framework of RT-Xen 2.0. We then discuss our implementation in detail.

A. Design Principles

To leverage multicore platforms effectively in real-time virtualization, we re-designed RT-Xen 2.0 to cover the design space of multicore VM scheduling in three dimensions: global and partitioned scheduling; dynamic and static priority schemes; and two server schemes (deferrable and periodic) for running the VMs. In summary, RT-Xen 2.0 supports:

- a scheduling interface that is compatible with a range of resource interfaces (e.g., [6], [12], [13]) used in compositional schedulability theory;
- both global and partitioned schedulers, called rt-global and rt-partition, respectively;
- EDF and DM priority schemes for both schedulers; and
- configurability of each scheduler to use either a work-conserving deferrable server or a periodic server.

We next discuss each dimension of the design space, focusing on how theory and platform considerations influenced our design decisions.

Scheduling interface. In RT-Xen, the scheduling interface of a domain specifies the amount of resource allocated to the domain by the VMM scheduler. In a single core virtualization setting, each domain has only one VCPU and thus, its scheduling interface can be defined by a *budget* and a *period* [4]. In contrast, each domain in a multicore virtualization setting can have multiple VCPUs. As a result, the scheduling interface needs to be sufficiently expressive to enable resource allocation for different VCPUs at the VMM level. At the same time, it should be highly flexible to support a broad range of resource interfaces, as well as different distributions of interface bandwidth to VCPUs (to be scheduled by the VMM scheduler), according to CSA theory.

Towards the above, RT-Xen 2.0 defines the scheduling interface of a domain to be a set of VCPU interfaces, where each VCPU interface is represented by a *budget*, a *period*, and a *cpu_mask* (which gives the subset of PCPUs on which the VCPU is allowed to run), all of which can be set independently of other VCPUs. This scheduling interface has several benefits: (1) it can be used directly by the VMM scheduler to schedule the VCPUs of the domains; (2) it can be configured to support different theoretical resource interfaces, such as MPR interfaces [6], deterministic MPR interfaces [12], and multi-supply function interfaces [13]; (3) it is compatible with different distributions of interface bandwidth to VCPU budgets, such as one that distributes budget equally among the VCPUs [6], or one that provides the maximum bandwidth (equal to 1) to all but one VCPU [12]; and finally (4) it enables the use of CPU-mask-aware scheduling strategies, such as one that dedicates a subsets of PCPUs to some VCPUs and schedules the rest of the VCPUs on the remaining PCPUs [12].

Global vs. partitioned schedulers. Different multicore schedulers require different implementation strategies and provide different performance advantages. A partitioned scheduler only schedules VCPUs in its own core's run queue and hence is simpler; in contrast, a global scheduler schedules all VCPUs in the system and thus is more complex but can provide better resource utilization. We support both by implementing two schedulers in RT-Xen 2.0: rt-global and rt-partition.[1] The rt-partition scheduler uses a partitioned queue scheme, whereas the rt-global scheduler uses a global shared run queue that is protected by a spin-lock.[2] For each scheduler, users can switch between dynamic priority (EDF) and static priority (DM) schemes on the fly.

Server mechanisms. Each VCPU in RT-Xen 2.0 is associated with a period and a budget, and is implemented as a server: the VCPU is released periodically and its budget is replenished at the beginning of every period, it consumes its budget when running, and it stops running when its budget is exhausted. Different server mechanisms provide different ways to schedule the VCPUs when the current highest priority VCPU is not runnable (i.e., has no jobs to execute) but still has unused budget. For instance, when implemented as a deferrable server, the current VCPU defers its unused budget to be used at a later time within its current period if it becomes runnable, and the highest-priority VCPU among the runnable VCPUs is scheduled to run. In contrast, when implemented as a periodic server, the current VCPU continues to run and consume its budget (as if it had a background task executing within it). The schedulability results can be quite different when different servers are used even if the scheduler is the same, shown in our experimental results presented in Section IV-F. We implemented both rt-global and rt-partition as deferrable servers, and can configure them as periodic servers by running a lowest priority CPU-intensive task in a guest VCPU.

[1]In our current platform, all cores share an L3 cache, thus limiting the potential benefits of cluster-based schedulers; however, we plan to consider cluster-based schedulers in our future work on new platforms with multiple multicore sockets.

[2]An alternative approach to approximate a global scheduling policy is to employ partitioned queues that can push/pull threads from each other [14] (as adopted in the Linux Kernel). We opt for a simple global queue design because the locking overhead for the shared global queue is small given the relatively small number of VMs usually on a host in practice.

B. Implementation

We first introduce the run queue structure of the rt-global scheduler, followed by that of the rt-partition scheduler which has a simpler run queue structure. We then describe the key scheduling functions in both schedulers.

Run queue structure. Figure 2 shows the structure of the global run queue of the rt-global scheduler (RunQ), which is shared by all physical cores. The RunQ holds all the runnable VCPUs, and is protected by a global spin-lock. Within this queue, the VCPUs are divided into two sections: the first consists of the VCPUs with remaining budget, followed by the second which consists of VCPUs that have no remaining budget. Within each section, the VCPUs are sorted based on their priorities (determined by a chosen priority assignment scheme). We implemented both EDF and DM priority schemes in RT-Xen 2.0. A run queue of the rt-partition scheduler follows the same structure as the RunQ, except that it is not protected by a spin-lock, and the rt-partition scheduler maintains a separate run queue for each core.

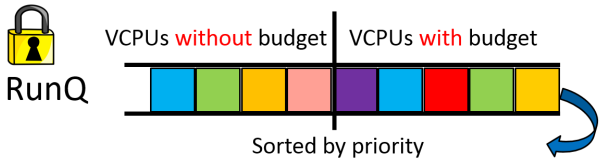


Fig. 2: rt-global run queue structure.

Scheduling functions: The scheduling procedure consists of two steps: first, the scheduler triggers the scheduler-specific *do_schedule()* function to make scheduling decisions; then, if necessary, it triggers the *context_switch()* function to switch the VCPUs.

Algorithm 1 shows the pseudo-code of the *do_schedule()* function of the rt-global scheduler under an EDF priority scheme (i.e., gEDF). In the first `for loop` (Lines 5–10), it replenishes the budgets of the VCPUs and rearranges the VCPUs in the RunQ appropriately. In the second `for loop` (Lines 11–18), it selects the highest-priority runnable VCPU (`snext`) that can be executed. Finally, it compares the deadline of the selected VCPU (`snext`) with that of the currently running

VCPU (scurr), and returns the VCPU to be executed next (Lines 19–24).

Algorithm 1 *do_schedule()* function for rt-global under EDF.

```
 1: scurr ← the currently running VCPU on this PCPU
 2: idleVCPU ← the idle VCPU on this PCPU
 3: snext ← idleVCPU
 4: burn_budget(scurr)
 5: for all VCPUs in the RunQ do
 6:    if VCPU's new period starts then
 7:       reset VCPU.deadline, replenish VCPU.budget
 8:       move VCPU to the appropriate place in the RunQ
 9:    end if
10: end for
11: for all VCPUs in the RunQ do
12:    if VCPU.cpu_mask & this PCPU ≠ 0 then
13:       if VCPU.budget > 0 then
14:          snext ← VCPU
15:          break
16:       end if
17:    end if
18: end for
19: if (snext = idleVCPU or snext.deadline > scurr.deadline)
       and (scurr ≠ idleVCPU) and (scurr.budget > 0)
       and vcpu_runnable(scurr) then
20:    snext ← scurr
21: end if
22: if snext ≠ scurr then
23:    remove snext from the RunQ
24: end if
25: return snext to run for 1 ms
```

There are two key differences between this algorithm and the single-core scheduling algorithm in RT-Xen 1.0 in [3]: (1) the second `for loop` (Lines 11–18) guarantees that the scheduler is CPU-mask aware; and (2) if the scheduler decides to switch VCPUs (Lines 22–24), the currently running VCPU (scurr) is **not** inserted back into the run queue; otherwise, it could be grabbed by another physical core before its context is saved (since the run queue is shared among all cores), which would then make the VCPU's state inconsistent. For this reason, Xen adds another scheduler-dependent function named *context_saved()*, which is invoked at the end of a *context_switch()* to insert scurr back into the run queue if it is still runnable. Note that both *do_schedule()* and *context_saved()* need to grab the spin-lock before running; since this is done in the Xen scheduling framework, we do not show this in Algorithm 1.

Another essential function of the scheduler is the *wake_up()* function, which is called when a domain receives a packet or a timer fires within it. In the *wake_up()* function of the rt-global scheduler, we only issue an interrupt if there is a currently running VCPU with a lower priority than the domain's VCPUs, so as to reduce overhead and potential priority inversions. We also implemented a simple heuristic to minimize cache overhead due to VCPU migrations: whenever there are multiple cores available, we assign the previously scheduled core first to reduce cache misses.

The *do_schedule()* function of the rt-partition scheduler is similar to that of the rt-global scheduler, except that (1) it does not need to consider the CPU mask when operating on a local run queue (because VCPUs have already been partitioned and allocated to PCPUs based on the CPU mask), and (2) if the scheduler decides to switch VCPUs, the currently running VCPU scurr will be immediately inserted back into the run queue. In addition, in the *wake_up()* function, we only compare the wake_up VCPU's priority with that of the currently running VCPU, and perform a switch if necessary.

Both rt-global and rt-partition schedulers were implemented in C. We also patched the Xen tool for adjusting the parameters of a VCPU on the fly. Our modifications were done solely within Xen. The source code of RT-Xen 2.0 and the data used in our experiments are both available via the RT-Xen website: *https://sites.google.com/site/realtimexen*.

IV. Empirical Evaluation

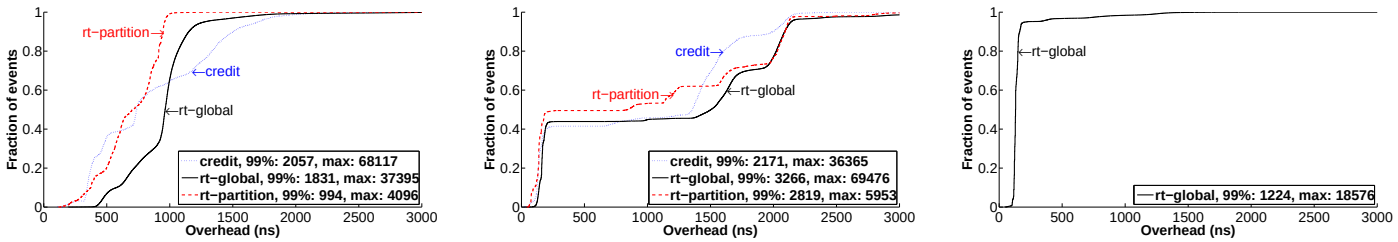
In this section, we present the experimental evaluation of RT-Xen 2.0. We have four main objectives for our evaluation: (1) to evaluate the scheduling overhead of the rt-global and rt-partition schedulers compared to the Xen credit scheduler; (2) to evaluate experimentally system schedulability under different combinations of schedulers at the guest OS and VMM levels; (3) to evaluate the real-time performance in overload situations of both schedulers under the EDF and DM priority schemes; and (4) to compare the performance of the deferrable server scheme and the periodic server scheme.

A. Experiment Setup

We performed our experiments on an Intel i7 x980 machine, with six cores (PCPUs) running at 3.33 GHz. We disabled hyper-threading and SpeedStep to ensure constant CPU speed, and shut down all other non-essential processes during our experiments to minimize interference. The scheduling quantum for RT-Xen 2.0 was set to 1 ms. The latest version of Xen 4.3 was patched with RT-Xen 2.0 and installed, with a 64-bit Linux 3.90 Kernel as domain 0. The guest domain image was installed with a 64-bit Ubuntu para-virtualized Kernel. For all experiments, we booted domain 0 with one VCPU and pinned this VCPU to one core, and the remaining five cores were used to run four guest domains. In addition, we configured the EDF priority scheme for the guest OS scheduler (via the LITMUS^{RT} [15] patch based on Linux 3.0), while focusing on evaluating the VMM schedulers under both EDF and DM schemes. Our evaluation focuses on CPU-intensive workloads. The impact of cache on scheduling is an important issue, which we plan to investigate in our future work.[3]

In our experiments, tasks were created based on the *base_task* provided by LITMUS^{RT}. To emulate a desirable execution time for a task in RT-Xen 2.0, we first calibrated a CPU-intensive job to take 1 ms in the guest OS (when running without any interference), then scaled it to the desirable execution time. For each task set, we ran each experiment for 60 seconds, and recorded the deadline miss ratio for each task using the *st_trace* tool provided by LITMUS^{RT}.

[3]The impact of cache has received significant attention in the context of one level scheduling (i.e., the OS level) from both theory and system perspectives [16]–[25]. We plan to build on these approaches and develop a cache-aware real-time scheduler for virtualized platforms in our future work.



(a) *do_schedule* (b) *context_switch* (c) *context_saved*

Fig. 3: CDF plot for scheduling overhead for different schedulers over 30 seconds.

B. Task Sets

Our evaluation is based on a set of synthetic real-time task sets. The tasks' periods were chosen uniformly at random between 350ms and 850ms, and the tasks' deadlines were equal to their periods. The tasks' utilizations followed one of three bimodal distributions, where the utilizations were distributed uniformly over either [0.0001, 0.5) or [0.5, 0.9], with respective probabilities of 4/9 and 5/9 (heavy), 6/9 and 3/9 (medium) and 8/9 and 1/9 (light). We followed the method in [16] to generate the bimodal distributions. Since there were five cores for running the guest domains, we generated task sets with total utilization ranging from 1.1 to 4.9, with a step of 0.2. To generate a task set at a specific utilization, we first generated tasks until we exceeded the specified total task utilization, then we discarded the last generated task and used a "pad" task to make the task set utilization match exactly the specified utilization. For each of the 20 task set utilizations, we used 25 random seeds to generate 25 task sets for each bimodal distribution. In total, there were 3 (distribution types) $\times$ 20 (utilization values) $\times$ 25 (random seeds) $=$ 1,500 task sets in our experiments.

Each generated task set was then distributed uniformly into four different domains. We applied compositional schedulability analysis to compute the interface of each domain based on its tasks, as well as to transform the computed interface into a set of VCPUs to be scheduled by the VMM scheduler. In our evaluation, we assume harmonic periods for all VCPUs, so as to achieve a higher schedulability bound. Note that it is common for a system to only support tasks with harmonic periods [26]; for instance, in avionics systems [27] and ARINC 653 [28] systems (which employ a two-level scheduling hierarchy), the periods of the tasks or partitions are usually harmonic. We therefore configure the VCPUs to run at harmonic periods.

For the partitioned scheduler at the guest OS level and the VMM level, we used a variant of the best-fit bin-packing algorithm for assigning tasks to VCPUs and VCPUs to cores, respectively. Specifically, for each domain, we assigned a task to the VCPU with the *largest* current bandwidth[4] among all existing VCPUs of the domain that can feasibly schedule the task. Since the number of VCPUs of the domain is unknown, we started with one VCPU for the domain, and added a new VCPU when the current task could not be packed into any existing VCPU. At the VMM level, we assigned VCPUs to the available cores in the same manner, except that (1) we aimed to minimize the number of VCPUs of the same domain that are assigned to the same core to maximize the parallelism available to each domain, and (2) under an overload condition (i.e., the current VCPU cannot be feasibly scheduled on any core), we assigned it to the core with the *smallest* current bandwidth, so as to balance the load among cores.

We performed the same experiments as above for the credit scheduler. Under this scheduler, both the weight and the cap of each domain were configured to be the total bandwidth of its VCPUs. (Recall that the bandwidth of a VCPU is the ratio of its budget to its period.) The CPU-mask of each VCPU was configured to be 1-5 (same as in the rt-global scheduler).

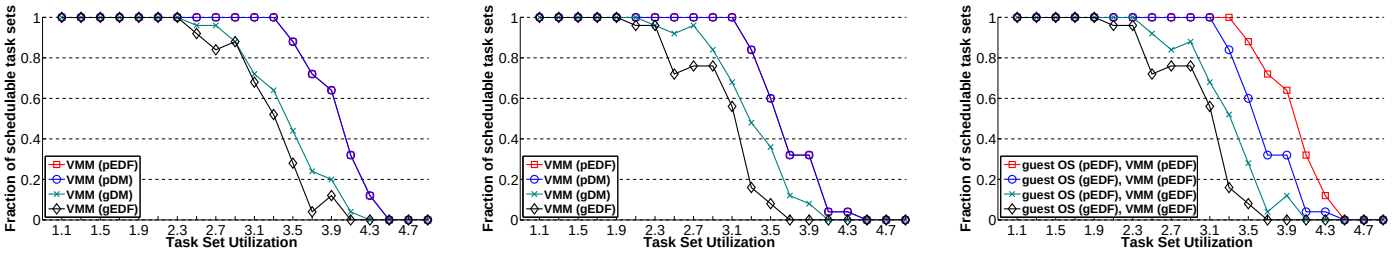
C. Scheduling Overhead Measurement

In order to measure the overheads for different schedulers, we boot 4 domains, each with 4 VCPUs. We set each VCPU's share to 20%, and evenly distributed the VCPUs to 5 PCPUs for the rt-partition scheduler; for the rt-global and credit schedulers, we allowed all guest VCPUs to run on all 5 PCPUs. We ran a CPU intensive workload with utilization of 3.10. We used the EDF scheme in both rt-global and rt-partition schedulers, as the different priority schemes only differ in their placement of a VCPU in the RunQ. In the Xen scheduling framework, there are three key functions related to schedulers: (1) at each scheduling quantum, the *do_schedule* function is triggered to make a scheduling decision (depends on the scheduler) – we define the time spent in the *do_schedule* function as *scheduling latency*; (2) if the next-to-run VCPU is different from the currently running one, the *context_switch* function is triggered; and (3) after the context switch, Xen calls the *context_saved* function to put the currently running VCPU back into the run queue. Note that *context_saved* is necessary only in rt-global schedulers, as they have shared queues. For rt-partition and credit schedulers, this function is NULL. To record these overheads, we patched *xentrace* [29] and recorded data for 30 seconds.

Figure 3 shows the CDF plots of the time spent in the three functions for different schedulers. Since 99% of the values were smaller than 3 microseconds (except rt-global in the *context_switch* function, which is 266 nanoseconds more than 3 microseconds), we cut the X-axis at 3 microseconds for a clear view, and include the 99% and maximum values in the legend for each scheduler. We observe the following:

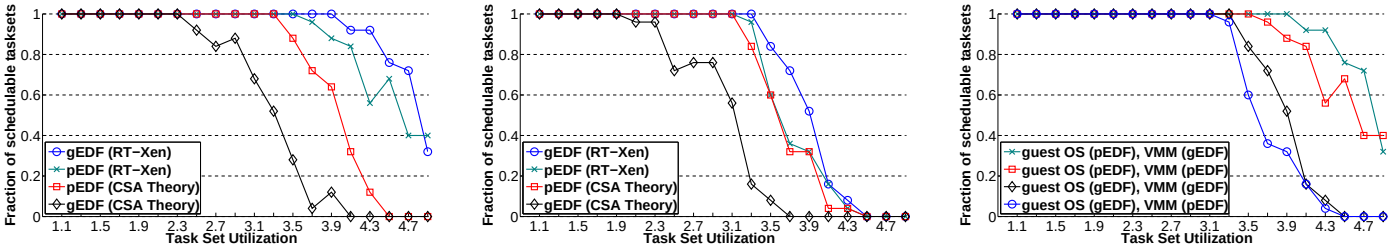
First, as is shown in Figure 3a, the rt-global scheduler incurred a higher scheduling latency than the rt-partition

[4]The maximum bandwidth of a VCPU is 1, since we assume that it can only execute on one core at a time.



(a) Guest OS (pEDF) (b) Guest OS (gEDF) (c) EDF for both Guest OS and VMM

Fig. 4: Theoretical results: schedulability of different schedulers at the guest OS and the VMM levels.



(a) Guest OS (pEDF) (b) Guest OS (gEDF) (c) EDF for both Guest OS and VMM (RT-Xen)

Fig. 5: Experimental vs. theoretical results: schedulability of different schedulers at the guest OS and the VMM levels.

scheduler. This is because the rt-global scheduler experienced the overhead to grab the spinlock, and it had a run queue that was 5 times longer than that of the rt-partition scheduler. The credit scheduler performed better than the rt-global scheduler in the lower 60%, but performed worse in the higher 40% of our measurements. We attribute this to the load balancing scheme in the credit scheduler, which must check all other PCPUs' RunQs to "steal" VCPUs.

Second, Figure 3b shows that the context switch overheads for all three schedulers were largely divided into two phases: approximately 50% of the overhead was around 200 nanoseconds, and the remaining was more than 1500 nanoseconds. Based on the detailed context switches of all three schedulers, we observe that the first 50% (with lower overhead) ran without actually performing context switches, since Xen defers the actual context switch until necessary; therefore, when the scheduler switched from a guest VCPU to the IDLE VCPU, or from the IDLE VCPU to a guest VCPU with its context still intact, the time spent in the *context_switch* function was much shorter than a context switch between two different guest VCPUs. This also illustrates the benefit of dedicating a core to a VCPU, since all context switches will then happen only between the IDLE VCPU and the guest VCPU.

Third, Figure 3c shows the time spent in the *context_saved* function for the rt-global scheduler. Recall that this function is NULL in the rt-partition and credit schedulers, since the current VCPU is already inserted back into the run queue by the *do_schedule* function. We observe that, for the rt-global scheduler, around 90% of the overhead was within 200 nanoseconds, and the 99% value was only 1224 nanoseconds. We attribute this to the extra overhead to grab the spinlock to access the shared run queue in the rt-global scheduler.

Overall, 99% of the overhead in all three functions (do_schedule, context_switch, and context_saved) for all schedulers were smaller than 4 microseconds. Since we used a 1 ms scheduling quantum in both rt-global and rt-partition schedulers, an overhead of 4 microseconds means only 0.4% resource loss per scheduling quantum. Notably, in contrast to an OS scheduler which is expected to handle a large number of tasks, the VMM scheduler usually runs fewer than 100 VCPUs as each VCPU typically demands much more resources than a single task; as a result, both the typical run queue length and the overhead to grab the lock in a shared RunQ are usually smaller than an OS scheduler.

D. Comparing Theoretical vs. Experimental Schedulability

In this experiment, our goal is to evaluate the performance of the different schedulers in terms of schedulability on RT-Xen 2.0, as well as to compare it with the performance predicted in theory. Since there were multiple possible combinations of schedulers both at the guest OS and the VMM levels, we first performed a numerical study of the relative performance of the different schedulers based on the compositional schedulability analysis (CSA) theory. Based on the theoretical results, we selected the best and the worst schedulers, and evaluated them experimentally on RT-Xen 2.0 on the physical machine. In both theoretical and experimental evaluations, we used the medium-bimodal distribution, and performed the experiments for all 25 task sets per utilization under the rt-global and rt-partition schedulers.

Theoretical numerical evaluation. To evaluate the relative performance of the four scheduling policies at the VMM level, we fixed the guest OS scheduler to be either pEDF or gEDF, and we varied the VMM scheduler among the four schedulers, pEDF, gEDF, pDM and gDM. For each configuration, we performed the schedulability test for every task set.

Performance of the four schedulers at the VMM level: Figures 4(a) and 4(b) show the fraction of schedulable task sets for the four schedulers at the VMM level with respect to the task set utilization when fixing pEDF or gEDF as the guest OS scheduler, respectively. The results show that when we fix the guest OS scheduler, the pEDF and pDM schedulers at the VMM level can schedule more task sets than the gDM scheduler, which in turn can schedule more task sets than the gEDF scheduler, for all utilizations.[5] The results also show that partitioned schedulers always outperformed global schedulers.

Combination of EDF schedulers at both levels: Figure 4(c) shows the fraction of schedulable task sets for each task set utilization under four different combinations of the EDF priority assignment at the guest OS and the VMM levels. The results show a clear relative performance ordering among the four combinations (from best to worst): (pEDF, pEDF), (gEDF, pEDF), (pEDF, gEDF), and (gEDF, gEDF).

Experimental evaluation on RT-Xen 2.0. From the above theoretical results, we observed that pEDF and gEDF have the best and the worst theoretical performance at either level; therefore, we focused on EDF for our experiments on RT-Xen 2.0 on the physical host.

Comparing theoretical prediction and experimental measurements: Figures 5(a) and 5(b) show the fractions of schedulable task sets that were predicted by the CSA theory and that were observed on RT-Xen 2.0 for the two EDF schedulers at the VMM level, when fixing pEDF or gEDF as the guest OS scheduler, respectively. We examined all 25 task sets for each task set utilization, and confirmed that whenever a task set used in our evaluation was schedulable according to the theoretical analysis, it was also schedulable under the corresponding scheduler on RT-Xen 2.0 in our experiments. In addition, for both pEDF and gEDF schedulers, the fraction of schedulable task sets observed on RT-Xen 2.0 was always larger than or equal to that was predicted by the theoretical analysis. The results also show that, in contrast to the trend predicted in theory, the gEDF scheduler at the VMM level was able to schedule more task sets on RT-Xen 2.0 than the pEDF scheduler in most utilizations. We attribute this to the pessimism of the gEDF schedulability test when applied to the VMM level.

Combination of EDF schedulers at both levels: Figure 5(c) shows the fraction of schedulable task sets observed on RT-Xen 2.0 for each task set utilization under four different combinations of EDF priority assignment at the guest OS and VMM levels. The results show that, at the guest OS level, the pEDF scheduler always outperformed the gEDF scheduler. Further, if we fixed pEDF (gEDF) for the guest OS scheduler, the gEDF scheduler at the VMM level was able to schedule more task sets than the pEDF scheduler for most utilizations.

To explain the relative performance of pEDF and gEDF in a two-level scheduling hierarchy, we investigated the corresponding set of VCPUs that were scheduled by the VMM when varying the guest OS scheduler. For the same task set, the VCPUs of a domain under the pEDF and gEDF schedulers can be different and hence, the set of VCPUs to be scheduled by the VMM can also be different. Figure 6 shows the total bandwidth of all the VCPUs that were scheduled by the VMM – averaged across all 25 task sets – at each task set utilization for the pEDF and gEDF schedulers at the guest OS level. The horizontal black line represents the total resource bandwidth available (with 5 cores).

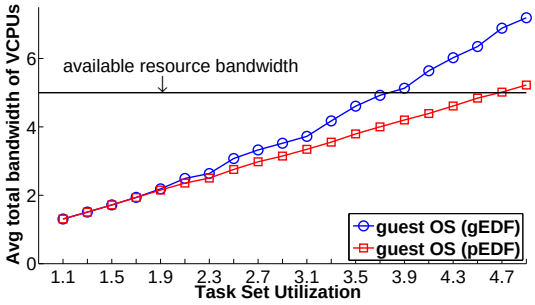


Fig. 6: Average total VCPU bandwidth comparison.

It can be observed from the figure that gEDF as the guest OS scheduler resulted in a higher average total VCPU bandwidth compared to pEDF; therefore, the extra resource that the VMM allocated to the VCPUs (compared to that was actually required by their tasks) was higher under gEDF. Since the resource that was unused by tasks of a higher priority VCPU cannot be used by tasks of lower-priority VCPUs[6], more resource was wasted under gEDF. In an overloaded situation, where the underlying platform cannot provide enough resources at the VMM level, the lower priority VCPUs will likely miss deadlines. This also indicates that a work-conserving design like [4] might help, which we plan to study as future work.

In contrast, for the same task set, when we fixed the guest OS scheduler to be either pEDF or gEDF, the set of VCPUs that was scheduled by the VMM is also fixed. As a result, we observed more VCPUs being schedulable on RT-Xen 2.0 under the gEDF scheduler than under the pEDF scheduler at the VMM level (c.f., Figure 5(c)). This is consistent with our observation, in a single level of scheduling setting, that the gEDF scheduler can often schedule more task sets than the pEDF scheduler experimentally.

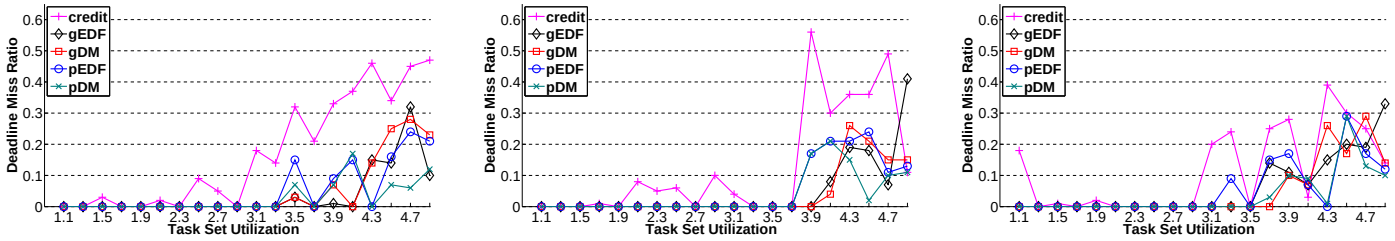
E. RT-Xen 2.0 vs. Credit

This set of experiments are designed to compare the credit scheduler with the rt-global and rt-partition schedulers on multicore processors. All three bimodal distributions (heavy, medium, light) were evaluated. We ran each task utilization with 1 task set, and plot the total deadline miss ratio for all the tasks across 4 domains (the number of jobs missed deadlines / the total number of jobs).

Figure 7 shows the results, from which we can draw several key conclusions: (1) the credit scheduler performs poorly, starting to miss deadlines at utilizations of 1.5 in heavy, 1.7 in medium, and even 1.1 in the light case. When the task set utilization is larger than 3.0, in most cases more than 20% of

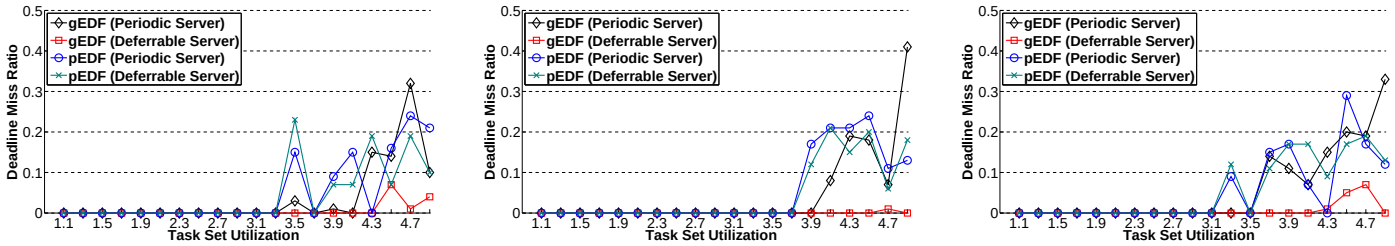
[5]Note that the fraction of schedulable task sets of the pEDF scheduler is the same as that of the pDM scheduler. This is because the set of VCPUs to be scheduled by the VMM is the same for both pDM and pEDF schedulers (since we fixed the guest OS scheduler), and these VCPUs have harmonic periods; as a result, the utilization bounds under both schedulers are both equal to 1 [30].

[6]The VCPUs were implemented as periodic servers in this experiment.



(a) Heavy Bimodal (b) Medium Bimodal (c) Light Bimodal

Fig. 7: Deadline miss ratio comparison for gEDF, gDM, pDM, pEDF, and credit schedulers (gEDF in guest OS).



(a) Heavy Bimodal (b) Medium Bimodal (c) Light Bimodal

Fig. 8: Deadline miss ratio comparison for for deferrable server and periodic server (gEDF in guest OS).

deadlines were missed in the credit scheduler; We examined the results for the credit scheduler for small utilizations (1.5 in heavy, 1.7 in medium, 1.1 in light case) and found that in those cases, due to the bimodal distribution, there is one domain with heavy tasks and thus requesting much more share compared to other domains, and most of the deadline misses happen in that domain. The proportional share scheme performs poorly under such uneven share cases. (2) both rt-global and rt-partition schedulers under DM and EDF priority schemes are consistent with the theory: when the theory predicts they are schedulable, they had no deadline misses; and (3) under high utilizations, there is no clear winner among the four RT-Xen 2.0 schedulers in terms of deadline miss ratio.

F. Periodic Server vs. Deferrable Server

It is well known that the deferrable server scheme has the *back-to-back* effect, in which higher-priority servers can preempt lower priority servers back to back, which may result in poor schedulability of the lower-priority servers. However, due to its work-conserving behavior, deferrable servers can also help improve lower-priority servers' response times. To compare the real-time performance of the periodic server and deferrable server schemes, we repeated the experiments in Section IV-E with the deferrable server and the EDF priority scheme, and compared its performance to that of a periodic server (see Section IV-E).

Figure 8 shows the real-time performance of the two servers for the pEDF and gEDF schedulers. It can be observed from the figure that the gEDF scheduler with a deferrable server performs best among all the schedulers; we attribute this to the fact that gEDF with a deferrable server can fully utilize all "slack" times from all VCPUs. For the pEDF scheduler, we observe no clear performance order between the deferrable server and periodic server schemes. This is because each VCPU of a domain, generated using compositional schedulability analysis, often requests more than half of the available bandwidth of a PCPU; as a result, in partitioned scheduling, every PCPU is either able to feasibly schedule all tasks (if it executes only one VCPU) or heavily overloaded (if it executes two or more VCPUs). In the former case, there is no deadline miss on the PCPU under both deferrable server and periodic server schemes; in the latter, using deferrable server cannot help improve the deadline miss ratio much, since there is often no slack available when the PCPU is heavily overloaded.

To further understand the back-to-back effect of the deferrable server scheme, we patched the *xentrace* tool to measure how many times it happens. We defined the number of back-to-back effect occurrences as the number of times when the VCPU's budget was replenished within consecutive executing time windows. For this experiment, we used a taskset with medium-bimodal distribution and a utilization of 3.10, and the task set was schedulable with the periodic server scheme for all four schedulers. We measured the number of back-to-back effect occurrences for 30 seconds, and calculated the ratio by dividing it by the total number of VCPU replenishments.

TABLE I: Back-to-back effects for different schedulers

Scheduler	gEDF	gDM	pEDF	pDM
Ratio	11.4%	11.2%	11.1%	10.9%

Table I shows the ratios of the back-to-back effect occurrences for the four schedulers. It can be observed from the table that the *back-to-back* effect only happens approximately 11% of the time, for all four schedulers. As a result, the impact of back-to-back effects was offset by the benefits of the work-conserving nature of deferrable servers, particularly under gEDF scheduling.

V. Related Work

Since our RT-Xen 2.0 scheduling framework is shaped by both theoretical and practical considerations, we highlight related work from both theory and systems perspectives.

A. Theory perspective

There exists a large body of work on hierarchical scheduling for single processor platforms (see e.g., [5], [31]–[33]), which consider both DM and EDF schemes. In RT-Xen 2.0, we use the method in [5] to derive the interfaces under partitioned DM and partitioned EDF schedulers.

Recently, a number of compositional scheduling techniques for multicore platforms have been developed (e.g., [6], [12], [13], [34], [35]), which provide different resource models for representing the interfaces. For instance, an MPR interface [6] abstracts a component using a period, a budget within each period, and a maximum level of concurrency, whereas a multi supply function interface [13] uses a set of supply bound functions. Our evaluations on RT-Xen 2.0 were performed using the MPR model for the domains' interfaces [6], but the RT-Xen 2.0 scheduling framework is compatible with most other interfaces, such as [12], [13] and their variations as discussed in Section III. In the future RT-Xen 2.0 can serve as an open source platform for the community to experiment with different hierarchical schedulability analysis.

B. Systems perspective

The implementation of hierarchical scheduling has been investigated for various platforms, including (1) native platforms, (2) Xen, and (3) other virtualization platforms. We describe each category below:

Native platforms: There are several implementations of hierarchical scheduling within middleware or OS kernels [36]–[41]. In these implementations, all levels of the scheduling hierarchy are implemented in one (user or kernel) space. In contrast, RT-Xen 2.0 implements the the schedulers in the VMM level, and leverage existing real-time schedulers in guest OS, thereby achieving a clean separation between the two levels of scheduling. Furthermore, leveraging compositional schedulability analysis, RT-Xen 2.0 also enables guest domains to hide their task-level details from the underlying platform, since it only requires a minimal scheduling interface abstraction from the domains.

Other Xen approaches: There were recent effort on adding real-time capabilities to Xen. For example, Lee et al. [42] enhanced the credit scheduler by improving the responsiveness of domains that run both CPU-intensive and I/O-intensive workloads, and Govindan et al. [43] mitigates priority inversions when guest domains are co-scheduled with domain 0 on a single core. Yoo et al. [44] used similar ideas to improve the credit scheduler on the Xen ARM platform. While these works employed heuristics to enhance the existing schedulers, RT-Xen 2.0 delivers a real-time performance based on compositional schedulability analysis and provides a new real-time scheduling framework that is separate from the existing schedulers.

Other virtualization approaches: There are other virtualization technologies that use different architecture than Xen. For instance, KVM [45] integrates the VMM with the host OS, and schedules VCPUs together with other tasks in host OS. Hence, in principle, any real-time multicore Linux scheduler [16], [46]–[49] could be configured to apply two-level hierarchical scheduling in KVM, but with limited server mechanism support. As an example, Checconi et al. [50] implemented a partitioned queue EDF scheduler for scheduling multicore KVM virtual machines, using hard constant-bandwidth servers for the VCPUs and global fixed-priority scheduling for the guest OS scheduler. The same group also investigated the scheduling of real-time workloads in virtualized cloud infrastructures [51]. Besides KVM, the micro-kernel like L4/Fiasco [52] can also be used to achieve hierarchical scheduling, as demonstrated by Yang et al. [53] using a periodic server implementation. Crespo et al. [54] also proposed a bare-metal VMM based on a para-virtualization technology similar to Xen for embedded platforms, which uses static (cyclic) scheduling. In contrast, RT-Xen 2.0 provides a scheduling framework spanning the design space in terms of global and partitioned scheduling, dynamic and static priority, periodic and deferrable servers. We also provide an comprehensive experimental study of different combinations of scheduling designs on a virtualized multicore processor.

VI. Conclusions and Future Work

We have designed and implemented **RT-Xen 2.0**, a new real-time multicore VM scheduling framework in Xen virtual machine monitor (VMM). RT-Xen 2.0 realizes global and partitioned VM schedulers, and each scheduler can be configured to support dynamic or static priorities, and to run VMs as periodic or deferrable servers. Through a comprehensive experimental study, we show that both global and partitioned VM scheduling can be implemented in the VMM at moderate overhead. Moreover, at the VMM scheduler level, in compositional schedulability theory pEDF is better than gEDF in schedulability guarantees, but in our experiments their actual performance is reversed in terms of fraction of workloads that meet their deadlines on virtualized multicore platforms. At the guest OS level, pEDF requests a smaller total VCPU bandwidth than gEDF based on compositional schedulability analysis, and therefore using pEDF in the guest OS level leads to more schedulable workloads on a virtualized multicore processor. The combination of pEDF in guest OS and gEDF in the VMM therefore resulted the best experimental performance when using a periodic server. Finally, gEDF scheduler runnings VMs as deferrable servers leads to the lowest deadline miss ratios under system overload when compared to other real-time VM scheduling policies.

While this work focuses on CPU intensive workloads, in the future we plan to develop cache-aware real-time VMM schedulers for multicore processors. Moreover, we would like to investigate real-time resource management approaches for other resources such as I/O and memory in virtualized platforms.

References

[1] "The Xen Project's Hypervisor for the ARM architecture," http://www.xenproject.org/developers/teams/arm-hypervisor.html.

[2] "The Xen Project is Built for Cloud Computing," http://www.xenproject.org/users/cloud.html.

[3] S. Xi, J. Wilson, C. Lu, and C. Gill, "RT-Xen: Towards Real-Time Hypervisor Scheduling in Xen," in *EMSOFT*. IEEE, 2011.

[4] J. Lee, S. Xi, S. Chen, L. T. Phan, C. Gill, I. Lee, C. Lu, and O. Sokolsky, "Realizing Compositional Scheduling Through Virtualization," in *RTAS*. IEEE, 2012.

[5] I. Shin and I. Lee, "Compositional Real-Time Scheduling Framework," in *RTSS*, 2004.

[6] A. Easwaran, I. Shin, and I. Lee, "Optimal virtual cluster-based multiprocessor scheduling," *Real-Time Syst.*, vol. 43, no. 1, pp. 25–59, Sep. 2009. [Online]. Available: http://dx.doi.org/10.1007/s11241-009-9073-x

[7] M. Xu, L. T. X. Phan, I. Lee, and O. Sokolsky, "Compositional analysis under global deadline monotonic," http://www.cis.upenn.edu/~linhphan/papers/rtas14-tr.pdf, University of Pennsylvania, Tech. Rep., 2013.

[8] A. Easwaran, I. Shin, and I. Lee, "Optimal Virtual Cluster-Based Multiprocessor Scheduling," *Real-Time Systems*, 2009.

[9] 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," *ACM SIGOPS Operating Systems Review*, 2003.

[10] "Credit Scheduler," http://wiki.xen.org/wiki/Credit_Scheduler.

[11] I. Shin and I. Lee, "Periodic Resource Model for Compositional Real-Time Guarantees," in *RTSS*, 2003.

[12] M. Xu, L. T. Phan, I. Lee, O. Sokolsky, S. Xi, C. Lu, and C. D. Gill, "Cache-Aware Compositional Analysis of Real-Time Multicore Virtualization Platforms," in *Proceedings of the IEEE Real-Time Systems Symposium (RTSS)*, 2013.

[13] E. Bini, G. Buttazzo, and M. Bertogna, "The Multi Supply Function Abstraction for Multiprocessors," in *RTCSA*. IEEE, 2009.

[14] S. Barush and B. Brandenburg, "Multiprocessor feasibility analysis of recurrent task systems with specified processor affinities," in *Proceedings of the IEEE Real-Time Systems Symposium (RTSS)*, 2013.

[15] "LITMUS-RT," http://www.litmus-rt.org/.

[16] A. Bastoni, B. B. Brandenburg, and J. H. Anderson, "An Empirical Comparison of Global, and Clustered Multiprocessor EDF Schedulers," in *RTSS*. IEEE, 2010.

[17] W. Lunniss, S. Altmeyer, C. Maiza, and R. I. Davis, "Integrating Cache Related Pre-emption Delay Analysis into EDF Scheduling," in *RTAS*, 2013.

[18] S. Altmeyer, R. I. Davis, and C. Maiza, "Cache Related Pre-emption Delay Aware Response Time Analysis for Fixed Priority Pre-emptive Systems," in *RTSS*, 2011.

[19] S. Altmeyer, R.I. Davis, and C. Maiza, "Improved Cache Related Pre-emption Delay Aware Response Time Analysis for Fixed Priority Pre-emptive Systems," *Real-Time Systems*, 2012.

[20] S. K. Baruah, A. K. Mok, and L. E. Rosier, "Preemptively Scheduling Hard-Real-Time Sporadic Tasks on One Processor," in *RTSS*, 1990.

[21] S. K. Baruah, L. E. Rosier, and R. R. Howell, "Algorithms and Complexity Concerning the Preemptive Scheduling of Periodic, Real-Time Tasks on One Processor," *Real-Time Systems*, 1990.

[22] M. Bertogna and S. Baruah, "Limited Preemption EDF Scheduling of Sporadic Task Systems," *Industrial Informatics, IEEE Transactions on*, 2010.

[23] J. V. Busquets-Mataix, J. J. Serrano, R. Ors, P. Gil, and A. Wellings, "Adding Instruction Cache Effect to Schedulability Analysis of Preemptive Real-Time Systems," in *RTAS*, 1996.

[24] H. Yun, G. Yao, R. Pellizzoni, M. Caccamo, and L. Sha, "MemGuard: Memory Bandwidth Reservation System for Efficient Performance Isolation in Multi-core Platforms," in *RTAS*, 2013.

[25] H. Yun, G. Yao, R. Pellizzoni, M. Caccamo, Marco and L. Sha, "Memory Access Control in Multiprocessor for Real-Time Systems with Mixed Criticality," in *ECRTS*, 2012.

[26] M. B. Jones, D. Roşu, and M.-C. Roşu, "Cpu reservations and time constraints: Efficient, predictable scheduling of independent activities," in *ACM SIGOPS Operating Systems Review*. ACM, 1997.

[27] C. D. Gill, J. M. Gossett, D. Corman, J. P. Loyall, R. E. Schantz, M. Atighetchi, and D. C. Schmidt, "Integrated adaptive qos management in middleware: A case study," *Real-Time Systems*, 2005.

[28] "ARINC Standards," http://www.aviation-ia.com/standards/index.html.

[29] D. Gupta, R. Gardner, and L. Cherkasova, "Xenmon: Qos Monitoring and Performance Profiling Tool," *Hewlett-Packard Labs, Tech. Rep. HPL-2005-187*, 2005.

[30] G. C. Buttazzo, "Rate monotonic vs. edf: judgment day," *Real-Time Syst.*, vol. 29, no. 1, pp. 5–26, Jan. 2005. [Online]. Available: http://dx.doi.org/10.1023/B:TIME.0000048932.30002.d9

[31] Z. Deng and J. Liu, "Scheduling Real-Time Applications in an Open Environment," in *RTSS*, 1997.

[32] A. Easwaran, M. Anand, and I. Lee, "Compositional Analysis Framework Using EDP Resource Models," in *RTSS*, 2007.

[33] M. Behnam, I. Shin, T. Nolte, and M. Nolin, "SIRAP: A Synchronization Protocol for Hierarchical Resource Sharing in Real-Time Open Systems," in *EMSOFT*, 2007.

[34] H. Leontyev and J. H. Anderson, "A Hierarchical Multiprocessor Bandwidth Reservation Scheme with Timing Guarantees," *Real-Time Systems*, 2009.

[35] G. Lipari and E. Bini, "A Framework for Hierarchical Scheduling on Multiprocessors: From Application Requirements to Run-Time Allocation," in *RTSS*. IEEE, 2010.

[36] J. Regehr and J. Stankovic, "HLS: A Framework for Composing Soft Real-Time Schedulers," in *RTSS*, 2001.

[37] Y. Wang and K. Lin, "The Implementation of Hierarchical Schedulers in the RED-Linux Scheduling Framework," in *ECRTS*, 2000.

[38] F. Bruns, S. Traboulsi, D. Szczesny, E. Gonzalez, Y. Xu, and A. Bilgic, "An Evaluation of Microkernel-Based Virtualization for Embedded Real-Time Systems," in *ECRTS*, 2010.

[39] T. Aswathanarayana, D. Niehaus, V. Subramonian, and C. Gill, "Design and Performance of Configurable Endsystem Scheduling Mechanisms," in *RTAS*, 2005.

[40] B. Lin and P. A. Dinda, "Vsched: Mixing batch and interactive virtual machines using periodic real-time scheduling," in *Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, 2005.

[41] M. Danish, Y. Li, and R. West, "Virtual-CPU Scheduling in the Quest Operating System," in *RTAS*, 2011.

[42] M. Lee, A. Krishnakumar, P. Krishnan, N. Singh, and S. Yajnik, "Supporting Soft Real-Time Tasks in the Xen Hypervisor," in *ACM Sigplan Notices*, 2010.

[43] 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*, 2007.

[44] S. Yoo, K.-H. Kwak, J.-H. Jo, and C. Yoo, "Toward under-millisecond i/o latency in xen-arm," in *Proceedings of the Second Asia-Pacific Workshop on Systems*. ACM, 2011, p. 14.

[45] "Kernel Based Virtual Machine," http://www.linux-kvm.org/page/Main_Page.

[46] G. Gracioli, A. A. Fröhlich, R. Pellizzoni, and S. Fischmeister, "Implementation and evaluation of global and partitioned scheduling in a real-time os," *Real-Time Systems*, 2013.

[47] B. B. Brandenburg and J. H. Anderson, "On the Implementation of Global Real-Time Schedulers," in *RTSS*. IEEE, 2009.

[48] J. M. Calandrino, H. Leontyev, A. Block, U. C. Devi, and J. H. Anderson, "LITMUS-RT: A Testbed for Empirically Comparing Real-Time Multiprocessor Schedulers," in *RTSS*, 2006.

[49] T. Cucinotta, G. Anastasi, and L. Abeni, "Respecting Temporal Constraints in Virtualised Services," in *COMPSAC*, 2009.

[50] F. Checconi, T. Cucinotta, D. Faggioli, and G. Lipari, "Hierarchical Multiprocessor CPU Reservations for the Linux Kernel," *OSPERT*, 2009.

[51] T. Cucinotta, F. Checconi, G. Kousiouris, D. Kyriazis, T. Varvarigou, A. Mazzetti, Z. Zlatev, J. Papay, M. Boniface, S. Berger *et al.*, "Virtualised e-learning with real-time guarantees on the irmos platform," in *Service-Oriented Computing and Applications (SOCA)*. IEEE, 2010.

[52] "Fiasco micro-kernel," http://os.inf.tu-dresden.de/fiasco/.

[53] J. Yang, H. Kim, S. Park, C. Hong, and I. Shin, "Implementation of compositional scheduling framework on virtualization," *SIGBED Rev*, 2011.

[54] A. Crespo, I. Ripoll, and M. Masmano, "Partitioned Embedded Architecture Based on Hypervisor: the XtratuM Approach," in *EDCC*, 2010.