PrVM: A Multicore Real-Time Virtualization Scheduling Framework With Probabilistic Timing Guarantees

Kevin Burns*
ECE, Virginia Tech
kevinpb@vt.edu

Vincent Legout†
ECE, Virginia Tech
vlegout@vt.edu

Antonio Barbalace‡
ECE, Virginia Tech
antoniob@vt.edu

Binoy Ravindran
ECE, Virginia Tech
binoy@vt.edu

ABSTRACT

We present PrVM, a framework for scheduling real-time VMs on multicore hardware. It addresses the intersection of the following problems: probabilistic real-time scheduling, VM scheduling, and full virtualization. Though each of these problems have been studied, their intersection – motivated by the need to consolidate multiple real-time software stacks, whose applications can be defined via probabilistic timing properties, onto a single embedded platform – is empty. PrVM uses a probabilistic model and timeliness optimality criterion. PrVM schedules VMs as server-like processes, computes time budgets using probabilistic methods, and aggregates task time budgets into VM time budgets. Experimental evaluations, using simulations and a concrete implementation, confirm the framework's effectiveness for synthetic benchmarks and multimedia applications.

1 INTRODUCTION

With the introduction of multicore processors among all computer markets embedded systems faced a paradigm shift: from single-purpose platforms to platforms that consolidate multiple embedded systems, each serving a single purpose. Virtualization technologies enable the mutual isolation of these different embedded systems on a single platform. Example scenarios in which this is already the practice or under research are industrial control systems, big-physics control and data acquisition systems, vehicle control system, etc. Consolidating multiple software stacks into a single embedded platform has multiple advantages, such as cost, programmability, ease of maintenance, drastically reduced communication latencies between the consolidated embedded systems, etc. However, for real-time applications, applications' time constraints must (continue to) be satisfied even when multiple embedded systems co-run on the same platform via virtualization. This requires hypervisor-level scheduling policies that ensure temporal isolation of the guest VMs, especially when each VM cannot be exclusively assigned to a subset of processor cores but multiple VMs share the same subset.

*The author is now with Northrop Grumman.
†The author is now with Gandi.
‡The author is now with Euler, Huawei German Research Center.

EWiLi, October 2017, Seoul, South Korea
© 2017 Copyright held by the owner/author(s).
ACM ISBN 123-4567-24-567/08/06…$15.00
https://doi.org/10.475/123_4

In this paper, we focus on virtualizing real-time systems whose time constraints include deadlines and whose timing optimization criterion is probabilistically specified. A motivating example are multimedia systems, e.g., video encoding/decoding. Occasionally failing to satisfy the deadline of an encoding or decoding task will result in a lost frame. This is acceptable, as long as not "too many" frames are lost. Similar to [21, 24], such a probabilistic timing requirement can be described as follows: each task T_i must satisfy at least ρ_i percentage of its deadlines. For example, if $\rho_i = 0.96$, then T_i must satisfy no less than 96% of its deadlines.

We present PrVM, a framework for scheduling VMs with probabilistic timing assurances. The framework considers a virtualization model, where a set of guest VMs with periodic real-time tasks are consolidated on a hypervisor that supports full virtualization. PrVM uses a hierarchical VM scheduling model where VMs are scheduled such as independent schedulable entities. Each guest VM encapsulates a guest real-time operating system (OS) and a set of (guest) application real-time tasks, and is modeled as a server with a fixed budget and period. PrVM targets modern multicore hardware.

Most of the past efforts on VM scheduling in a real-time setting (e.g. [22]) focus on the timing objective of deterministically meeting all deadlines. In contrast, while probabilistic real-time scheduling has been studied in the past (e.g. [16]), studies mostly focused at the abstraction of task scheduling in a (real-time) OS setting. To the best of our knowledge, this is the first scheduling work that lies at the intersection of three spaces: probabilistic real-time scheduling, VM scheduling, and full virtualization. Similarly to past works in the space of probabilistic real-time systems (e.g. [16, 21, 24]), we consider a probabilistic task model, wherein task execution time is described using a probability density function. This has multiple advantages. For example, many real-time applications (e.g., multimedia) exhibit large variation in their actual execution time demands. Thus, the statistical estimation of task execution time demand is more stable and hence more predictable than the actual execution time.

To obtain the collective probabilistic timing assurance, CPU time must be allocated for each guest task T_i's execution so that its ρ_i is satisfied. There is an inherent tradeoff in this time allocation: longer the time allocation, the "stronger" will be the resulting probabilistic assurance (i.e., more than ρ_i percentage of deadlines will be met), the greater the total number of required CPUs will be. Toward this, we present two probabilistic methods, called *CH* and *DI*. *DI* is more "optimistic" than *CH* in the sense that, for a given density function and ρ_i, *DI* allocates less CPU time than *CH*. Once task time allocations are computed, they are aggregated to determine the respective VM budgets using the compositional scheduling theory [19]. To understand PrVM's effectiveness, we evaluated the framework through simulations using synthetic benchmarks, as well as by implementation-based experiments using the x264 production code.

2 RELATED WORK

In [19], Shin and Lee introduced the Compositional Scheduling Framework (CSF). CSF uses servers which provide a method of representing an application or VM with its own real-time tasks and scheduler as a single entity to be scheduled by the host OS. CSF provides an offline method to compute the budget of a periodic server given a set of guest tasks, the guest scheduler, and the period of the server on the host. CSF is useful for hierarchical scheduling because it is fully composable – i.e., a server can schedule servers which schedules its own servers, and so on. Moreover, CSF supports many scheduling policies and it is easy to implement. PrVM extends the same ideas from Shin and Lee by using a probabilistic approach to reduce the amount of resources needed.

RT-Xen [22] is the most well-known real-time virtualization solution exploiting hierarchical scheduling using servers and CSF. Based on the Xen hypervisor, it schedules servers (i.e., virtual CPUs or vCPUs) according to a fixed-priority algorithm. The same authors extended this work with overhead, cache awareness, multiprocessor support and IO devices ([15, 18, 23]). Several other solutions have been proposed to support hierarchical scheduling (e.g. [9, 11, 14]). However, none of these target soft real-time systems (e.g. [13]), and none of these use a probabilistic approach. Thus, they have to be pessimistic offline while allocating CPU time for each task to avoid deadline misses. Therefore, they allocate resources that may not be used during the execution when tasks terminate their execution earlier than expected. On the other hand, the more aggressive probabilistic approach used by PrVM allows reducing the number of resources even further than just the benefits gained from consolidating multiple physical computers' software stacks into a virtualized environment.

Other approaches (e.g. [9]) supporting real-time virtualization use a microhypervisor or micro-kernel, for example NOVA [20]. However, as in PrVM and other previous works (e.g. [11]), using the well-known Linux kernel makes it easier to thorough evaluate the proposed approach. Furthermore, Kiszka [12] showed that Linux using KVM and the PREEMPT_RT patch set has promise as a real-time hypervisor.

Scheduling real-time systems that tolerate deadline misses using probabilistic guarantees have already been studied (e.g. [16, 21]). Such as PrVM, they assume that jobs from the same task can have different execution times modeled using a random variable. Using a different model than PrVM, where jobs are aborted when a deadline is missed, [16] derives tardiness bounds when tasks continue their execution after a deadline miss. Targeting multimedia applications, AIRS [10] is using Linux and an extended version of SCHED_DEADLINE to improve the QoS. HiRes [17] is another system to manage resources and improve the QoS. Contrary to PrVM, these solutions target multimedia applications and use stochastic approaches but do not target virtual environments.

3 MODEL

In PrVM, a guest VM is a real-time system that used to run on a single processor. Each guest VM is thus assigned a single vCPU – future work will consier the case of multiple vCPUs. Guest VMs run on top of the hypervisor and do not have privileged access to the processor. A guest VM has a set of real-time tasks with real-time constraints and schedules tasks using a real-time scheduler that is fully decoupled from the host scheduler – i.e., the host scheduler cannot control the guest scheduler decisions. Guest VMs schedule tasks using the Earliest Deadline First (EDF) algorithm. Although the model do not applies to a specific type of virtualization we target full virtualization, thus guest VMs are oblivious of the virtualization environment. In contrast to paravirtualization, full virtualization does not force a user to adopt a specific OS, hence easily extending the applicability of our approach to any OS.

We consider a real-time periodic task model for each VM; tasks release jobs periodically, and each job must be finished before its deadline. All tasks are independent and may be released simultaneously. The set of all tasks of a guest VM g is called Γ_g. Inside each guest, a task τ_i is characterized by a period T_i and a deadline D_i. We set the deadline of each task equal to its period. The hyperperiod of Γ_g is the Least Common Multiple of T_i for all $\tau_i \in \Gamma_g$. PrVM uses the probabilistic task model defined in [24] and [21] to model the execution time of the tasks: for each task, the distribution of its execution times is known. Based on this distribution, a task is thus additionally characterized by its mean execution time E_i and the variance of its execution time V_i (see Section 4 and Section 5).

As in [24] and [21], a job is immediately aborted as soon as it misses its deadline. Therefore, there is no backlogged demand. A programming model subsuming this abort model, similar to [21, 24], is assumed. Each task has an associated abort handler, which contains programmer-defined logic for transitioning the system to a safe state (e.g., release memory, locks, etc.) in the event of an abort. When a job misses its deadline, the guest OS raises a time constraint violation exception; the job is terminated and its abort handler is executed. Such as [21, 24], we assume that abort handlers have negligible execution time.

The system has multiple identical and independent CPUs, and each guest OS is only executed on one CPU. We use partitioned scheduling, thus migrations are not allowed during the execution. To assign vCPUs to physical CPUs, we use the Best-Fit algorithm. EDF is used to schedule VMs on each physical CPU. Virtual CPUs are assigned to physical CPUs before execution and cannot migrate. We do not take into account preemption costs or cache overheads. However, as our evaluations are performed on real hardware, such latencies are included in the evaluation.

We use a hierarchical model with one host OS and multiple guest OSes. The host OS manages the guests with a hypervisor or Virtual Machine Monitor (VMM), which in this paper is part of the host OS (type-2, hosted virtualization). The host is responsible for scheduling the guest VMs while each guest VM must schedule

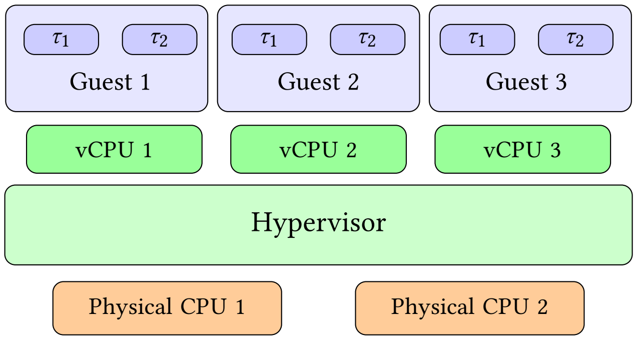

Figure 1: PrVM task, processor and server model

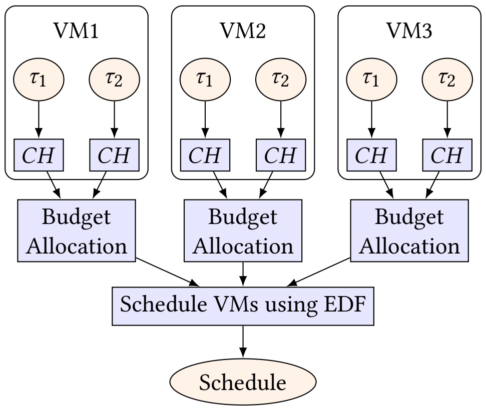

Figure 2: PrVM framework approach

its own periodic real-time tasks using EDF. Servers are used to schedule the VMs on the host. As each guest is only scheduled on one vCPU, each vCPU can be seen as a server, like in RT-Xen [22]. Figure 1 summarizes our model.

4 PRVM FRAMEWORK

The PrVM scheduling framework is offline, thus all steps described in this Section are performed prior to the system execution. PrVM includes two steps: it first estimates the execution time to allocate for each task (Subsection 4.1), then uses CSF to schedule the VMs in the host (Subsection 4.2). Within the CSF theory, a guest can be seen as a server whose time budget and period can be computed using the characteristics of the tasks inside the guest and its scheduling algorithm. Servers provide temporal isolation between guests VMs, hence a malfunctioning or malicious guest cannot disturb other guests. Figure 2 summarizes the approach of PrVM with an example with 3 VMs where each VM contains 2 tasks.

4.1 Task Scheduling

PrVM targets real-time systems for which ensuring that all deadlines are met is not a requirement. PrVM takes advantage of this by allocating less CPU time than what would be required to guarantee that all deadlines are met. This way, the number of processors needed to schedule the system is reduced. To account for the percentage of deadline hits, we introduce ρ, with $\rho \in [0, 1]$. Each task τ_i must satisfy at least ρ_i percentage of its deadlines.

We propose two different ways to compute the allocated execution time of tasks according to the known information about the tasks. If the complete distribution of a task's execution times is not known, we can only rely on the mean and the variance of the execution times. In this case, we use Chebyshev's inequality to compute the allocated execution time for the task. If the distribution is known, it is used to provide a less-pessimistic allocated execution time, i.e., closer to the probability ρ given as input. Such solutions are called *CH* and *DI*, respectively.

For each solution, let c_i be the execution time of task τ_i. c_i is the CPU time allocated to each job inside each guest VM. A job may require more than c_i to terminate.

CH. The probability of each job to terminate its execution before c_i is ρ. c_i can be computed using the following equation, which is based on Chebyshev's inequality [21] (where E_i is the mean of the execution time distribution and V_i its variance):

$$c_i = E_i + \sqrt{\frac{\rho \times V_i}{1 - \rho}} \quad (1)$$

This equation guarantees that the execution time of each job from task τ_i will have an execution time no larger than c_i with a probability no less than ρ.

DI. If the distribution of the execution times is known, using Chebyshev's inequality is too pessimistic. The distribution can instead be used to better estimate the amount of CPU time which must be reserved for each task. Let $e_{i,j}$ be the execution time of the j^{th} job of τ_i, c_i must be such that:

$$\forall j, prob(e_{i,j} > c_i) < \rho \quad (2)$$

The main difference between *CH* and *DI* is that *CH* can be used without knowing the exact distribution of the execution times. Thus, *CH* is more conservative than *DI*. On the other hand, *DI* can guarantee a deadline satisfaction ratio closer to ρ.

We use the same value of ρ among all tasks, among all guests. Future work will address tasks with different values of ρ. The smaller the value of ρ, the higher the chance a job will miss its deadline, because it will have less execution time reserved. However, it will reduce the number of resources needed to schedule the system.

4.2 Server Scheduling

PrVM treats each VM as a server with a respective time budget and period and uses CSF to schedule servers. Let Π be the period of a server and Θ its budget (i.e., its execution time). A server can execute for at most Θ on each period. The CSF theory allows us to compute the budget and period of the servers [19] according to the characteristics of their tasks. For a fixed period, the budget of the server must respect this inequality:

$$\forall 0 < t \leq H, \; dbf_{EDF}(\Gamma, t) \leq sbf(t) \quad (3)$$

The Demand Bound Function dbf [4] of a task set Γ depends on the scheduling algorithm used, here EDF. The Supply Bound Function sbf computes the minimum possible resource supplies during a time interval of length t. First, let k be such that:

$$k = max\left(\left\lceil \frac{t-(\Pi-\Theta)}{\Pi}\right\rceil, 1\right) \tag{4}$$

Then, if $t \in [(k+1)\Pi - 2\Theta, (k+1)\Pi - \Theta]$, we have:

$$sbf(t) = t - (k+1)(\Pi - \Theta) \tag{5}$$

Otherwise, we have:

$$sbf(t) = (k-1)\Theta \tag{6}$$

Equation (3) can be used to compute the budget and the execution time of the servers, and for each task we use the execution time computed with Equation (1) or Equation (2). To use this equation, the period of the server is first fixed arbitrarily. We discuss which values must be used for the period in Section 5. The budget and period of servers are used by the host to schedule VMs. We use partitioned scheduling so this is equivalent to uniprocessor scheduling on each processor, EDF being used to schedule each processor.

The CSF framework guarantees that all the deadlines of the tasks inside each VM are met. However, our model differs from CSF in that it has probabilistic execution times and thus the schedulability analysis does not rely on the WCET of tasks. Each task is allocated a CPU time of c_i to execute. If all jobs terminate before using all their allocated CPU time, CSF guarantees that no deadline is going to be missed. However, when the execution time of a job i exceeds c_i, a deadline miss can occur. The percentage of deadline misses will not be above $1 - \rho$. The *CH* solution being more pessimistic, the number of deadline misses will be even lower while for the *DI* solution, the number of deadline misses is theoretically equal to $1 - \rho$ because it uses the exact distribution of the execution times.

4.3 Example

This subsection illustrates how budgets are assigned to VMs using Madplay, an MP3 decoder. Madplay was patched such that decoding a frame is a real-time job, playing a song is a real-time task. Figure 3 shows the impact of the distribution of the execution times and the probability on the actual execution time reserved for the task during the execution. The two figures represent the probability density function of the execution times for the *CH* and *DI* solutions. A vertical line represents the amount of CPU time reserved for different probability ρ (30%, 50% and 70%) according to Equation 1. For example, for $\rho = 50\%$, *CH* reserves $65\mu s$ for the task to execute according to Eq. 1. This figure illustrates the fact that *DI* is more aggressive because it allocates less CPU time for the task to execute compared to *CH* for the same percentage ρ. We can also see that the values of ρ are the same as the values which can be found on the y axis.

Figure 4 represents the required CPU time vs. the theoretical Deadline Satisfaction Ratio (DSR) for our two solutions, *CH* and *DI*, according to the probability ρ. The first is more conservative and thus the number of deadline misses is lower, but requires more CPU time especially when ρ is close to 100. On the other hand, the relation between ρ and the DSR is linear when using *DI*. Thus, *DI* allows more control over the number of deadline misses and requires less CPUs than *CH*.

4.4 Complexity

The complexity of PrVM can be divided into offline and online complexities. Online, during the execution, the complexity of PrVM on each CPU is the same as EDF. Offline, before execution, the allocated CPU time of tasks and the budget of servers must be computed. The former has a complexity of $O(1)$. However, the complexity of the algorithm for computing the budget of the servers is higher. Equation (3) must be met for t in $]0, LCM_\Gamma]$, and so the complexity thus depends on the hyperperiod of the task set. And for each t, verifying the equation requires computing the dbf and the sbf. The complexity of dbf is $O(n)$ where n is the number of tasks in the task set while the complexity of sbf is $O(1)$. This computation must be done while the budget is not large enough to guarantee the schedulability of the server. However, as PrVM is an offline scheduling framework, the offline complexity does not hurt the performance of the system.

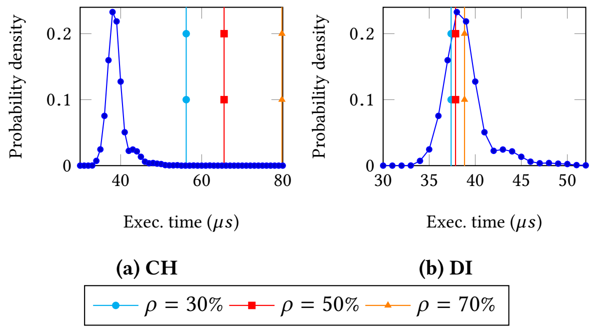

Figure 3: Distribution of the execution times for Madplay

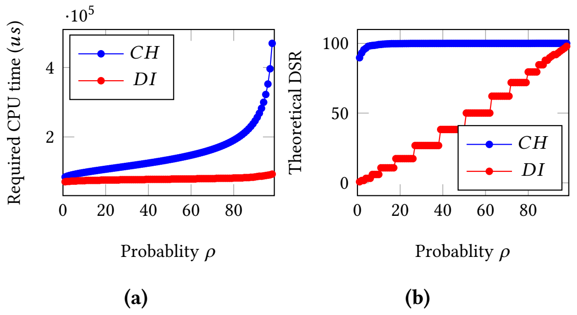

Figure 4: (a) Required CPU time and (b) Theoretical Deadline Satisfaction Ratio (DSR) w.r.t. ρ

5 EVALUATION

In this section the PrVM algorithm is firstly compared through simulations to partitioned-EDF and CSF to answer the questions of how many deadline will be missed, and how many physical processors can be spared. Secondly, a PrVM implementation is benchmarked on hardware using synthetic and production applications to demonstrate that PrVM actually works. We compare the performance of the PrVM implementation against a non real-time solution to demonstrate the need for a real-time approach. Then, we compare the PrVM implementation against the same solution but without probabilistic execution times.

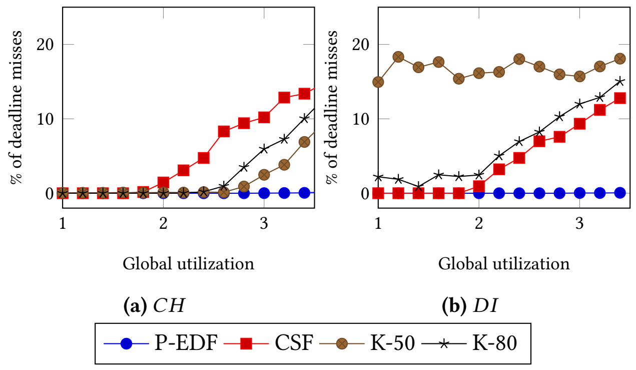

Figure 5: Percentage of deadline misses for *CH* and *DI*

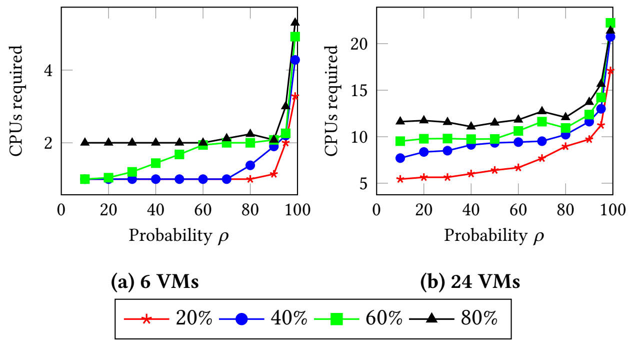

Figure 6: Number of CPUs required according to ρ for various expected execution times (*CH*)

5.1 Implementation and Experimental Setup

PrVM extends KairosVM [5], hence we used KVM/QEMU as the virtualization environment. The current implementation targets the x86 64-bit architecture but PrVM can be easily ported to any other architecture supported by KVM. Due to the choice of KVM we do not compare PrVM to RT-Xen. This is because the evaluation would not just focus on the differences between the scheduling approaches but also on the implementation differences in Xen and KVM.

Each guest VM runs a minimal Ubuntu Server 10.04 with the Linux kernel 3.0.24, patched with ChronOS 3.0. The choice of the guest OS is however not critical and other real-time OSes could have also been chosen. ChronOS Linux [7] is an academic project that implements a pluggable scheduling framework within the Linux scheduler.

ChronOS implements the abort model described in Section 3. In order to force a worst case execution scenario the following experiments were run without exploiting the abort model. However, for the applications used (*sched_test_app*, *x264*, and *Madplay*), potential adverse effects of no-aborts, if any, did not manifest in the experimental results. As a matter of fact, this no-abort experimental setting is actually adversarial to our technique. Thus, with aborts, deadline satisfactions will only improve as the saved cycles will be used for tasks in-flight. We plan to explore this space with more applications in future work.

Linux 3.16 is used as the host OS. Guests are scheduled using SCHED_DEADLINE [8], a Linux scheduling policy implementing the Constant Bandwidth

Linux 3.16 is used as the host OS. The host OS is scheduling Guests using SCHED_DEADLINE [8], a Linux scheduling policy implementing the Constant Bandwidth Server (CBS [2]) to provide bandwidth isolation among tasks (guests). `cpuset` is used to pin tasks (guests) to processors. The host is running Ubuntu Server 10.04 paired with QEMU version 1.6.0. The implementation was deployed on an Intel Xeon E5520 processor with 4 CPUs at 2.27GHz, and 16GB of RAM. We dedicated 2GB of RAM to each guest.

The implementation of PrVM is available online [1].

5.2 Simulation

A first simulation evaluates the percentage of deadline misses of PrVM, CSF, and Best-Fit partitioned-EDF. We use 2 different probabilities for this experiment: 50% (K-50) and 80% (K-80). A simulator was developed to generate random task sets and schedule them for 2 hyperperiods. 5 VMs are scheduled on 2 CPUs. Each VM contains between 1 and 10 real-time tasks and the overall number of real-time tasks in all the VMs is 15. For each task set, the utilization of each task is computed randomly between 0.01 and 0.99 with a uniform distribution [6]. Execution times are randomly generated using the normal distribution with a mean execution time set to half the WCET of the task. The period of each task is chosen randomly with a uniform distribution. The overall global utilization, i.e., the sum of all the utilizations of the VMs, varies between 1 and 3.5.

Figure 5 shows the percentage of deadline misses for all the different scheduling algorithms and for the *CH* and *DI*. While partitioned-EDF always stays close to zero deadline misses, this number quickly increases for our solutions when the global utilization is more than 2, especially when ρ is high. Indeed, as expected, a lower value of ρ leads to more deadline misses. For *DI*, as expected, the number of deadline misses increases faster than with *CH*. Indeed, this second solution is more aggressive and compared to *CH* allocates less CPU time to tasks for the same probability ρ. The percentage of deadline misses for *DI* is even larger for $\rho = 50$ than for $\rho = 80$ because deadline misses appear for a global utilization of 1.

5.3 Number of Processors

A second simulation presents a static analysis of the scalability and effectiveness of PrVM by showing the number of processors required to schedule all the VMs according to the probability ρ. This simulation was done using the *sched_test_app* synthetic benchmark [7]. We generated a set of synthetic tasks for our probabilistic model based on [3]. We chose a uniform distribution because we believe it is representative of many real-world scenarios. We generated 100 task sets for each probability on the x axis and took the average number of CPUs needed to schedule the task sets such that each VM can execute for the execution time computed by CSF.

Figure 6 shows the number of CPUs required varying the probability. Each line represents a different expected value for the execution time, with regard to the WCET. The variance is set to

$(\frac{1}{6} \times \text{WCET})^2$. The number of CPUs increases when the probability increases. Moreover, more CPUs are needed when the expected execution time of tasks increases. This is expected because a larger execution time is reserved for each task. We can thus conclude from these plots that PrVM is able to scale as the number of VMs increases. For the test with 24 VMs, 13 CPUs are on average required to schedule the task sets when using the WCET instead of the probabilistic model. Thus, PrVM requires between 10% and 50% less CPUs when ρ is less than 90% and the expected execution time is 20% and 40% of the WCET. Compared to actual worst-case execution time, our probabilistic model yields even larger gains in terms of the number of CPUs used.

5.4 Deadline Misses

Two different kinds of benchmarks are used in this subsection. First, the *sched_test_app* synthetic benchmark is used to generate random task sets. While generating random task sets, we set their global utilization between 20% and 30% so that the 6 VMs cannot be scheduled on one CPU only. To compute the budget for each VM using CSF, we set the period to 100μs. 2 physical CPUs and 6 VMs are used for all the evaluations, unless stated otherwise. Then, we used two production applications: x264 (video streams encoder), and Madplay (audio decoder).

We first evaluated the execution times of these applications. The distribution of the execution times of Madplay follows a normal distribution while this is not true for x264. However, the larger the standard deviation of the application, the more pessimistic the allocation of the execution times will be with *CH*. We set a different period for each application according to its mean execution time.

Secondly, we experiment with the default scheduling policy of Linux, the Completely Fair Scheduler (CFS), which does not aim to have good real-time performance. 6 VMs running the x264 application were scheduled on 2 CPUs, and CFS was responsible for assigning these VMs to the CPUs and for scheduling them. Results showed that the average percentage of deadline misses amongst all the real-time tasks in the VMs was 16%. A real-time policy is thus needed for acceptable performance.

Using *sched_test_app*, we first scheduled random real-time tasks inside different VMs. Execution times are generated using the normal distribution. The standard deviation is set to $\frac{1}{6}$ of the WCET and the expected value is set to: $0.3 \times WCET$, $0.5 \times WCET$ and $0.7 \times WCET$. Figure 7 represents the mean DSR of all VMs when the probability given to Chebyshev's inequality increases, and for task sets with three different mean execution times compared to their WCET. On the figure on the left, 4 VMs are scheduled on 2 CPUs while on the right, 6 VMs are scheduled, still on 2 CPUs. For Figure 7, the higher the better because a higher DSR means less deadline misses. As expected, the DSR is reduced when the mean execution time is larger. We also see that the DSR is always higher than the probability in the x axis, due to the use of Chebyshev's inequality which guarantees a DSR given the probability. For the *CH* solution, the DSRs in Figure 7 are significantly larger than the associated probabilities in the x axis, especially for the lowest probabilities. This is due to Chebyshev's inequality, which needs to be conservative because it only knows the expected execution time and the standard deviation of the execution times and nothing else about the distribution. The same experiments were also performed using x264 with similar results. Overall, PrVM provides provable probabilistic guarantees regarding the number of deadline misses. The evaluation shows that, as expected, it never goes lower than ρ and requires less resources than traditional solutions. In comparison, solutions using the WCET require a much higher number of CPUs while only increasing the DSR by a few percentage.

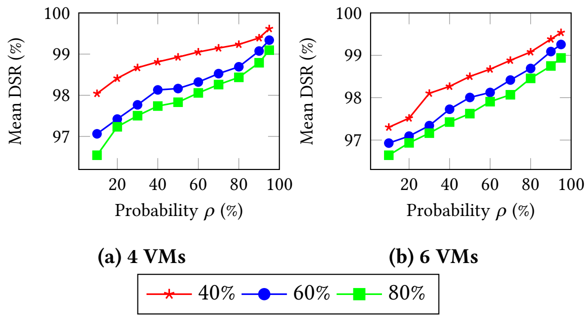

Figure 7: DSR vs expected execution time with *sched_test_app* (*CH*)

6 CONCLUSION

Inspired by the practice of consolidating multiple real-time software stacks (guests VMs) on a single multicore embedded platform this paper introduced PrVM, a hierarchical real-time scheduling framework for virtualized environments which reduces the number of physical processor cores required – or increases the number of guest VMs that can be consolidated on a set of processor cores. PrVM adopts two different probabilistic approaches to reduce the CPU time allocated to each task according to the amount of information known about the task sets. We show that even with partial information probabilistic time guarantees can be met. Evaluations show that PrVM allows a greater number of VMs to run on a single embedded platform than KVM/Linux while providing VM's applications temporal guarantees. PrVM always meets probabilistic guarantees on the number of deadline misses for each VM, which we demonstrated to be important for certain classes of applications, such as multimedia, and cannot be guaranteed just with SCHED_DEADLINE.

ACKNOWLEDGMENTS

This work is supported in part by US NSWCDD under grant N00178-13-D-1031/0005 and US NAVSEA/NEEC under grant 3003279297.

REFERENCES

[1] 2017. KairosVM. http://www.ssrg.ece.vt.edu/kairos. (2017).
[2] Luca Abeni and Giorgio Buttazzo. 2004. Resource Reservation in Dynamic Real-Time Systems. *Real-Time Systems* 27, 2 (2004).
[3] Theodore P. Baker. 2005. *Comparison of empirical success rates of global vs. partitioned fixed-priority and edf scheduling for hard real time.* Technical Report.
[4] Sanjoy K. Baruah *et al.* 1990. Algorithms and Complexity Concerning the Preemptive Scheduling of Periodic, Real-time Tasks on One Processor. *Real-Time Syst.* 2, 4 (Oct. 1990).
[5] Kevin Burns, Antonio Barbalace, Vincent Legout, and Binoy Ravindran. 2014. KairosVM: Deterministic Introspection for Real-time Virtual Machine Hierarchical Scheduling *(VtRES 2014).*

[6] R.I. Davis *et al.* 2011. Improved priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems. *Real-Time Syst.* 47, 1 (Jan. 2011).

[7] Matthew Dellinger *et al.* 2011. ChronOS Linux: A Best-effort Real-time Multiprocessor Linux Kernel *(DAC 2011)*.

[8] Dario Faggioli *et al.* 2009. An EDF scheduling class for the Linux kernel *(RTLW 2009)*.

[9] Stefan Groesbrink *et al.* 2014. Towards Certifiable Adaptive Reservations for Hypervisor-based Virtualization *(RTAS 2014)*.

[10] Shinpei Kato, Ragunathan Rajkumar, and Yutaka Ishikawa. 2010. AIRS: Supporting Interactive Real-Time Applications on Multicore Platforms *(ECRTS 2010)*.

[11] N.M. Khalilzad, M. Behnam, and T. Nolte. 2013. Multi-level adaptive hierarchical scheduling framework for composing real-time systems *(RTCSA 2013)*.

[12] Jan Kiszka. 2011. Towards Linux as a Real-Time Hypervisor *(RTLW 2011)*.

[13] YoungWoong Ko *et al.* 2012. Soft Realtime Xen Virtual Machine Scheduling Using Compositional Model.

[14] H. Leontyev and J.H. Anderson. 2008. A Hierarchical Multiprocessor Bandwidth Reservation Scheme with Timing Guarantees *(ECRTS 2008)*.

[15] Xu Meng *et al.* 2013. Cache-Aware Compositional Analysis of Real-Time Multicore Virtualization Platforms *(RTSS 2013)*.

[16] A.F. Mills and J.H. Anderson. 2011. A Multiprocessor Server-Based Scheduler for Soft Real-Time Tasks with Stochastic Execution Demand *(RTCSA 2011)*.

[17] Gabriel Parmer and Richard West. 2010. HiRes: A System for Predictable Hierarchical Resource Management *(ECRTS 2010)*.

[18] L.T.X. Phan *et al.* 2013. Overhead-aware compositional analysis of real-time systems *(RTAS 2013)*.

[19] Insik Shin and Insup Lee. 2008. Compositional Real-time Scheduling Framework with Periodic Model. *ACM Trans. Embed. Comput. Syst.* 7, 3, Article 30 (May 2008).

[20] Udo Steinberg and Bernhard Kauer. 2010. NOVA: A Microhypervisor-based Secure Virtualization Architecture *(EuroSys 2010)*.

[21] H. Wu, B. Ravindran, E.D. Jensen, and Peng Li. 2004. CPU scheduling for statistically-assured real-time performance and improved energy efficiency *(CODES + ISSS 2004)*.

[22] Sisu Xi, Justin Wilson, Chenyang Lu, and Christopher Gill. 2011. RT-Xen: Towards Real-time Hypervisor Scheduling in Xen *(EMSOFT 2011)*.

[23] Sisu Xi *et al.* 2014. Real-time Multi-core Virtual Machine Scheduling in Xen. In *EMSOFT 2014*.

[24] Wanghong Yuan *et al.* 2003. Energy-efficient Soft Real-time CPU Scheduling for Mobile Multimedia Systems *(SOSP 2003)*.