Scheduling Tasks with Hard Deadlines in Virtualized Software Systems

Sogand Shirinbab

School of Computing
Blekinge Institute of Technology
SE-379 71 Karlskrona, Sweden

Sogand.Shirinbab@bth.se

Lars Lundberg

School of Computing
Blekinge Institute of Technology
SE-379 71 Karlskrona, Sweden

Lars.Lundberg@bth.se

ABSTRACT. There is scheduling on two levels in real-time applications executing in a virtualized environment: traditional real-time scheduling of the tasks in the real-time application, and scheduling of different Virtual Machines (VMs) on the hypervisor level. In this paper, we describe a technique for calculating a period and an execution time for a VM containing a real-time application with hard deadlines. This result makes it possible to apply existing real-time scheduling theory when scheduling VMs on the hypervisor level, thus making it possible to guarantee that the real-time tasks in a VM meet their deadlines. If overhead for switching from one VM to another is ignored, it turns out that (infinitely) short VM periods minimize the utilization that each VM needs to guarantee that all real-time tasks in that VM will meet their deadlines. Having infinitely short VM periods is clearly not realistic, and in order to provide more useful results we have considered a fixed overhead at the beginning of each execution of a VM. Considering this overhead, a set of real-time tasks, the speed of the each processor core, and a certain processor utilization of the VM containing the real-time tasks, we present a simulation study and some performance bounds that make it possible to determine if it is possible to schedule the real-time tasks in the VM, and in that case for which periods of the VM that this is possible.

Keywords: Cloud, Virtualization, Real-Time Scheduling, Hard Deadlines, Virtual Machine.

1. INTRODUCTION

Most real-time services were originally designed for physical (un-virtualized) computer systems. However, the trend towards virtualization pushes, for cost reasons, more and more systems onto virtualized machines, and at some point one would also like to run real-time systems with hard deadlines in a virtualized environment. Moving a real-time system with hard deadlines to a virtualized environment where a number of Virtual Machines (VMs) share the same physical computer is a challenging task. The original real-time application was designed so that all tasks were guaranteed to meet their deadlines provided that the physical computer was fast enough. In a system with faster processors, and more cores, one would like to put several VMs on the same hardware and some (or all) of these VMs may contain real-time tasks with hard deadlines. In such a system there will be scheduling at two levels [6]: traditional real-time scheduling of the tasks within a VM, and hypervisor controlled scheduling of several VMs on the same physical server. In [25] and [32] the authors refer to this technique as Component-based design. This technique is also known as Hierarchical scheduling [21] [22] [31] [32] [34].

Traditional scheduling of tasks on a physical uni-processor computer is well understood, and a number of useful results exist [9], e.g., it is well known that Earliest Deadline First (EDF) is optimal when we allow dynamic task priorities. Similarly, it is well-known that Rate-Monotonic Scheduling (RMS) where tasks are assigned priorities based on their deadlines is optimal for the case when we use static task priorities. These priority scheduling algorithms are based on a number of parameters for each task τ_i. These parameters are typically, the period T_i the worst-case execution time C_i, and the deadline D_i, for task τ_i. Often, we assume that $D_i = T_i$, and in that case we only need two parameters for each task, namely, T_i and C_i. Priority assignment schemes such as EDF and RMS are typically used in the original real-time scheduling applications, i.e., in the applications that will be running in a VM.

If we ignore the overhead for context switching from one VM to another and if we use (infinitely) small time slots, we could let a VM get a certain percentage of the physical computer, e.g., two VMs where each VM uses every second time slot. This kind of situation could be seen as two VMs running in parallel with 50% of full speed each. In that case, the real-time application would meet all deadlines if the processor on the physical computer is (at least) two times as fast as the processor for which the original real-time application was designed for. However, the overhead for switching from one VM to another cannot be ignored and the time slot lengths for this kind of switching can obviously not be infinitely small. In order to minimize the overhead due to switching between VMs we would like to have relatively long time periods between switching from one VM to another VM. In order to share the physical hardware between as many VMs as possible we would also like to allocate a minimum percentage of the physical CPU to a VM, i.e., we would only like to allocate enough CPU resources to a VM so that we know that the real-time application that runs in that VM meets all its deadlines.

In order to use EDF, RMS or similar scheduling algorithms also on the hypervisor level, i.e., when scheduling the different VMs to the physical hardware, we need to calculate a period $\mathrm{T_{VM}}$ and a (worst-case) execution time $\mathrm{C_{VM}}$ for each VM that share a physical computer. It can be noted that also most real-time multiprocessor scheduling algorithms are based on the period and the worst-case execution time [8] [20]. This is important since most modern hardware platforms, i.e., most platforms on which the VMs will run, are multiprocessors.

VMs with one virtual processor will, for several reasons, be a very important case. Many existing real-time applications with hard deadlines have been developed for uniprocessor hardware. Moreover, even when using state-of-the-art multiprocessor real-time scheduling algorithms, one may miss deadlines for task sets with processor utilization less than 40% [8]. For the uni-processor case it is well known that when using RMS we will always meet all deadlines as long as the processor utilization is less than $\ln(2) = 69.3\%$ [9]. This indicates that, compared to having a small number of VMs with many virtual cores each, it is better to use a larger number of VMs with one virtual core each on a multicore processor (we will discuss this in Section 2). We will present our results in the context of VMs with one virtual core. However, the results could easily be extended to VMs with multiple virtual cores as long as each real time task is allocated to a core (we will discuss this in Section 9). Systems that use global multiprocessor scheduling of real-time tasks, i.e., systems that that allow tasks to migrate freely between processors, are not considered here.

In this paper we will, based on an existing real-time application and the processor speed of the physical hardware, calculate a period $\mathrm{T_{VM}}$ and an execution time $\mathrm{C_{VM}}$ such that the existing real-time application will meet all deadlines when it is executed in a VM, provided that the VM executes (at least) $\mathrm{C_{VM}}$ time unites every period of length $\mathrm{T_{VM}}$. We will show, and it is also well known from previous studies, that if overhead for switching from one VM to another is ignored, it turns out that (infinitely) short VM periods minimizes the utilization that each VM needs to guarantee that all real-time tasks in that VM will meet their deadlines. Having infinitely short VM periods is clearly not realistic, and in order to provide more useful results we consider a fixed overhead at the beginning of each execution of a VM. Considering this overhead, a set of real-time tasks, the speed of the each processor core, and a certain processor utilization of the VM containing the real-time tasks, we present a simulation study and some performance bounds that make it possible to determine if it is possible to schedule the real-time tasks in the VM, and in that case for which periods of the VM that this is possible. We will base our calculations on the case when we use static priorities, and thus RMS, in the original real-time applications. However, we expect that our approach can easily be generalized to cases when other scheduling policies, such as EDF, are used in the original real-time applications (we will discuss this in Section 2).

2. RELATED WORK

Today, most physical servers will contain multiple processor cores. Modern virtualization systems, such as KVM, VMware and Xen, make it possible to define VMs with a number of (virtual) cores, thus allowing parallel execution within a VM. This means that one can use the physical hardware in different ways: one can have a large number of VMs with one (virtual) core each on a physical (multi-core) server, or a smaller number of VMs with multiple (virtual) cores each (or a combination of these two alternatives). It is also possible to make different design decisions in the time domain, e.g., allowing a VM with one virtual core to execute for relatively long time periods, or restricting a VM with multiple cores to relatively short execution periods. Real-time scheduling theory (for non-virtualized systems) shows that the minimum processor utilization for which a real-time system can miss a deadline, using fixed priority scheduling, decreases as the number of processors increases, e.g., 69.3% for one processor systems [7] (using RMS) and 53.2% for two processor systems [8] and then down to as little as 37.5% for systems with (infinitely) many processors [8].

Consequently, compared to multiprocessor systems, the processor utilization is in general higher for systems with one processor. This is one reason why we have assumed that the VM containing the original real-time application only has one (virtual) processor. Also, most existing real-time applications are developed for systems with one processor.

In this paper we have assumed that the real-time application in the VM uses RMS. If we assume some other scheduling policy, e.g., EDF we can use the same technique. The only difference is that the formula $R_i = C_i + \sum_{j=1}^{i-1} \lceil R_i / T_j \rceil C_j$ (see Section 3), needs to be replaced with the corresponding analysis for EDF.

Very little has been done in the area of scheduling real-time tasks with hard deadlines in virtualized systems. Some results on real-time tasks with soft deadlines exist [1] [16].

There are a number of results concerning so called proportional-share schedulers [2][3][4][18]. These results look at a real-time application that runs inside an operating system process. The proportional-share schedulers aim at dividing the processor resource in predefined proportions to different processes.

In [10] the authors look at a model for deciding which real-time tasks to discard when the cloud system's resources cannot satisfy the needs of all tasks. This model does, however, not address the problems associated with hard deadlines.

In [11] the authors ran an experiment using a real-time e-learning distributed application for the purpose of validating the IRMOS approach. The IRMOS uses a variation of the Constant Bandwidth Server (CBS) algorithm based on EDF. Furthermore in [17] the authors developed their particular strategy in the context of IRMOS project. They tried to consider isolation and CPU scheduling effects on I/O performance. However, in IRMOS they do not consider hard real-time tasks scheduled using the RMS.

Reservation-based schedulers are used as hypervisor level schedulers. In [12] and [19] the authors used CPU reservation algorithm called, Constant Bandwidth Server (CBS) in order to prove that the real time performance of the VMs running on the hypervisor is affected by both the scheduling algorithm (CBS) and VM technology (in this case KVM). However, the authors do not present a method for how to schedule different VMs running on the hypervisor.

In [13] the authors presented two algorithms for real-time scheduling. One is the hypervisor scheduling algorithm and the other is the processor selection algorithm. However they only consider scheduling VMs on the hypervisor level, they do not investigate scheduling of the hard real-time tasks that run inside the VMs.

Eucalyptus is open-source software for building private and hybrid clouds. There are several algorithms already available in Eucalyptus for scheduling VMs with some advantages and disadvantages. In [14] the authors proposed a new algorithm for scheduling VMs based on their priority value, which varies dynamically based on their load factors. However they consider dynamic priority based scheduling not static priority.

In [15], a priority based algorithm for scheduling VMs is proposed. The scheduler is first distinguishing the best matches between VMs and empty places and then deploying the VMs onto the corresponding hosts. The authors did a comparison between their priority algorithm and First Come First Serve (FCFS) algorithm, they concluded that the resource performance of their algorithm is not higher than the FCFS algorithm all the time but it has higher average resource performance. Nevertheless, they do not consider periodic tasks and static priority assignment.

The VSched system, which runs on top of Linux, provides soft real-time scheduling of VMs on physical servers [5]. However, the problems with hard deadlines are not addressed in that system.

In [21], the authors proposed a hierarchical bounded-delay resource model that constructs multiple levels resource partitioning. Their approach is designed for the open system environment. Their bounded-delay resource partition model can be used for specifying the real-time guarantees supplied from a parent model to a child model where they have different schedulers, while in [22] and [32], the authors proposed a resource model that can provide a compositional manner such that if the parent scheduling model is schedulable, if and only, its child scheduling models are schedulable. However, none of the proposed resource models consider scheduling in virtualized environment.

In [23] and [31], the authors presented a methodology for computing exact schedulability parameters for two-level framework while in [24] they did an analysis on systems where the fixed priority pre-emptive scheduling policy is used on both level. Further in [26], the authors presented a method for analysis of platform overheads in real-time systems. Similar work by [33] represents that their proposed approach can reduce pre-emption and overhead by modifying the period and execution time of the tasks.

In [25], the authors developed compositional real-time scheduling framework based on the periodic interface, they have also evaluated the overheads that this periodic interface incur in terms of utilization increase. Later in [41], the authors proposed an approach to eliminate abstraction overhead in composition. In their latest study the authors have improved their previous works and proposed a new technique for the cache-related overhead analysis [40].

In [27], the authors implemented and evaluated a scheduling framework that built on Xen virtualization platform. Another similar work has been done by [29]; they represent an implementation of compositional scheduling framework for virtualization using the L4/Fiasco micro kernel which has different system architecture compared to Xen. The authors calculated clock cycle overhead for the L4/Fiasco micro kernel. In [28], the authors proposed and compare the results of overhead of an external scheduler framework called ExSched that is designed for real time systems. In [30], the authors presented and compared several measurements of overheads that their implemented hierarchical scheduling framework imposes through its implementation over VxWorks.

Compositional analysis framework based on the explicit deadline periodic resource model has been proposed by [38]. They have used EDF and Deadline Monotonic (DM) scheduling algorithm and their model supports sporadic tasks. In [39], the authors present the RM schedulability bound in a periodic real time system which is an improvement to the earlier bound that has been proposed by [7]. However none of these works consider the overhead in their models.

3. PROBLEM DEFINITION

We consider a real-time application consisting of n tasks. Task τ_i $(1 \leq i \leq n)$ has a worst-case execution time C_i $(1 \leq i \leq n)$, and a period T_i $(1 \leq i \leq n)$. This means that task τ_i generates a job at each integer multiple of T_i and each such job has an execution requirement of C_i time units that must be completed by the next integer multiple of T_i. We assume that each task is independent and does not interact (e.g., synchronize or share data) with other tasks. We also assume that the first invocation of a task is unrelated to the first invocation of any other task, i.e., we make no assumptions regarding the phasing of tasks with equal or harmonic periods. We assume that the deadline D_i is equal to the period, i.e., $D_i = T_i$ $(1 \leq i \leq n)$. The tasks are executed using static task priorities, and we use RMS scheduling, which means that the priority is inversely proportional to the period of the task (i.e., tasks with short periods get high priority). This static priority assignment scheme is optimal for the uni-processor case [9].

The real-time application is executed by a VM with one virtual processor. The real-time tasks may miss their deadlines if the VM containing the tasks is not scheduled for execution by the hypervisor during a certain period of time. For instance, if some period that the VM is not running exceeds some T_i, it is clear that the corresponding task τ_i will miss a deadline. Also, if the VM gets a too low portion of a physical processor, the tasks may also miss their deadlines since there will not be enough processor time to complete the execution time before the next deadline.

In a traditional real-time application a task τ_i will voluntarily release the processor when it has finished its execution in a period, and C_i denotes the maximum time it may execute before it releases the processor. In the case with real-time scheduling of VMs on the hypervisor level it is more natural to assume that the hypervisor preempts VM_j and puts VM_j in the blocked state when it has executed for C_{VM_j} time units in a period. The hypervisor then moves VM_j to the ready state at the start of the next period. As mentioned before, the length of the period for VM_j is T_{VM_j}.

On the hypervisor level one may use any scheduling policy as long as one can guarantee that each VM is executed C_{VM}, during each period T_{VM}. On multicore processors one could for instance bind each VM to a core and let the VMs that share the same core share it using RMS, or one could let the VMs share a global ready queue, i.e., a VM could be executed on different cores during different time periods.

4. DEFINING T_{VM} AND C_{VM}

Without loss of generality, we order the tasks τ_i $(1 \leq i \leq n)$ such that $T_i \leq T_{i+1}$. This means that τ_1 has the highest priority and τ_n has the lowest priority using RMS. Let R_i denote the worst-case response time for taskτ_i. From previous results we know that

$$R_i = C_i + \sum_{j=1}^{i-1} \lceil R_i / T_j \rceil C_j \qquad (1)$$

on a physical uni-processor server (or when the VM has uninterrupted access to a physical processor). In order to obtain R_i from Equation (1) one needs to use iterative numeric methods [9]. In order to meet all deadlines we must make sure that $R_i \leq T_i$ $(1 \leq i \leq n)$.

Consider a time period of length t, which may extend over several periods T_{VM}. The scenario with minimum execution of the VM during period t, starts with a period of $2(T_{VM} - C_{VM})$ with no execution (see Fig. 1) [37][25], i.e., the period starts exactly when the VM has executed C_{VM} time units as early as possible in one of its periods. Following this line of discussion, it is also clear that for the worst-case scenario $\lfloor (t - 2(T_{VM} - C_{VM}))/T_{VM} \rfloor$ is the number of whole periods of length T_{VM} (each containing a total execution of C_{VM}) that is covered by t.

Let t' denote the minimum amount of time that the VM is running during a time period of length t. From Fig 1 we get the minimum t' as:

$$t' = \left\lfloor \frac{(t-2(T_{VM}-C_{VM}))}{T_{VM}} \right\rfloor C_{VM} + \min\left(\left(t - 2(T_{VM} - C_{VM}) - \left\lfloor \frac{(t-2(T_{VM}-C_{VM}))}{T_{VM}} \right\rfloor T_{VM}\right), C_{VM}\right) \qquad (2)$$

In Equation (2), the first term ($\lfloor (t - 2(T_{VM} - C_{VM}))/T_{VM} \rfloor C_{VM}$) corresponds to the full periods, and the last term to the remaining part. The term $t - 2(T_{VM} - C_{VM}) - \lfloor (t - 2(T_{VM} - C_{VM}))/T_{VM} \rfloor T_{VM}$ is the time that the VM has access to a physical processor during the part of t that exceeds the full periods. The minimum comes from the fact that time that the VM has access to a physical processor during the time interval that exceed the full periods cannot be more than C_{VM}. This means that t' is a function of three parameters, i.e., $t' = f(t, T_{VM}, C_{VM})$. For fixed T_{VM} and C_{VM}, $t' = f(t, T_{VM}, C_{VM})$ is a continuous increasing function in t, consisting of straight line segments from $\left((2(T_{VM} - C_{VM}) + nT_{VM}), nC_{VM}\right)$ to $\left((2(T_{VM} - C_{VM}) + (n+1)T_{VM}), (nC_{VM} + T_{VM})\right)$ for any $n = 0, 1, 2, \ldots$ and horizontal lines connecting them. Fig. 1 displays a general piece of the curve, and the points $P_n = \left((2(T_{VM} - C_{VM}) + nT_{VM}), C_{VM}\right)$ are the lower corners in the graph.

We now define the inverse function

$$t = f^{-1}(f(t, T_{VM}, C_{VM}), T_{VM}, C_{VM}) \qquad (3)$$

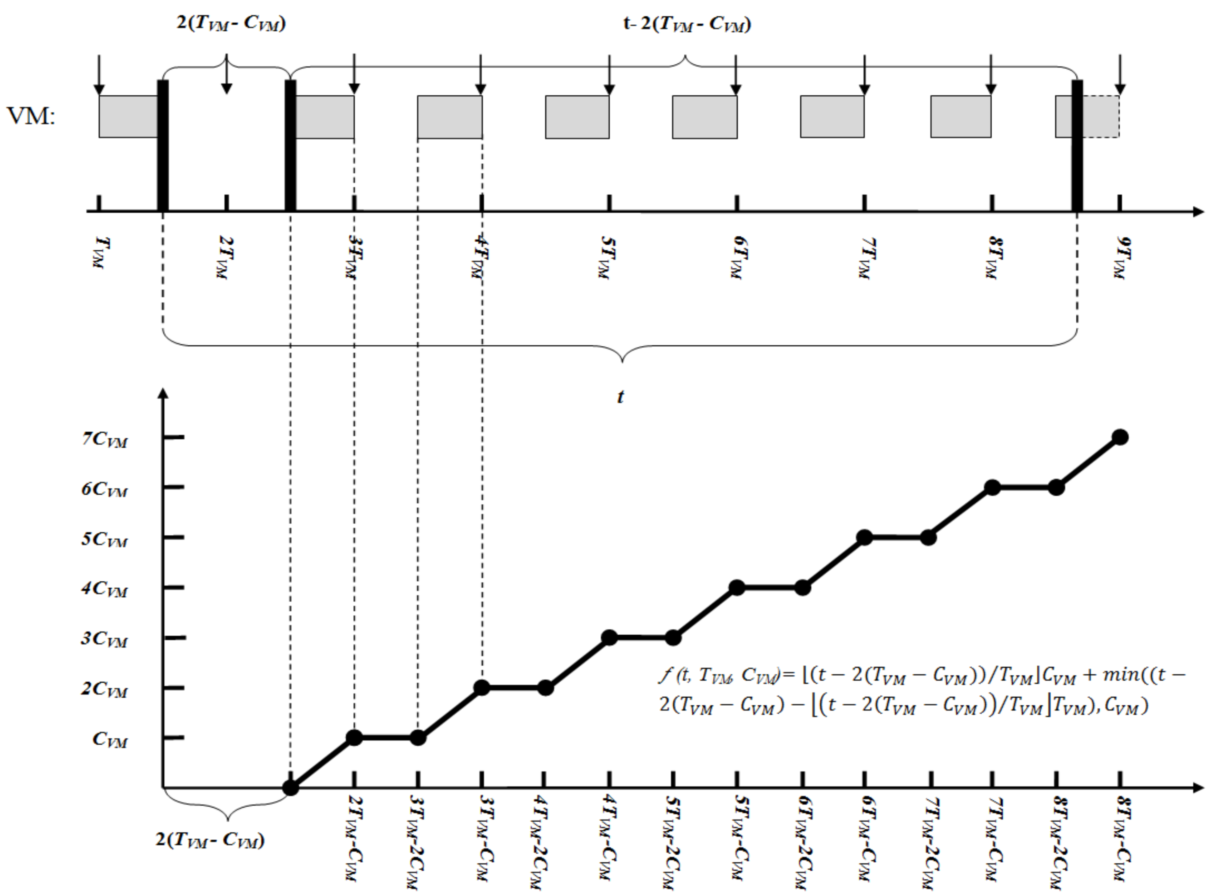


Fig. 1.Worst-case scenario when scheduling a VM with period $\mathbf{T_{VM}}$ and (worst-case) execution time $\mathbf{C_{VM}}$.

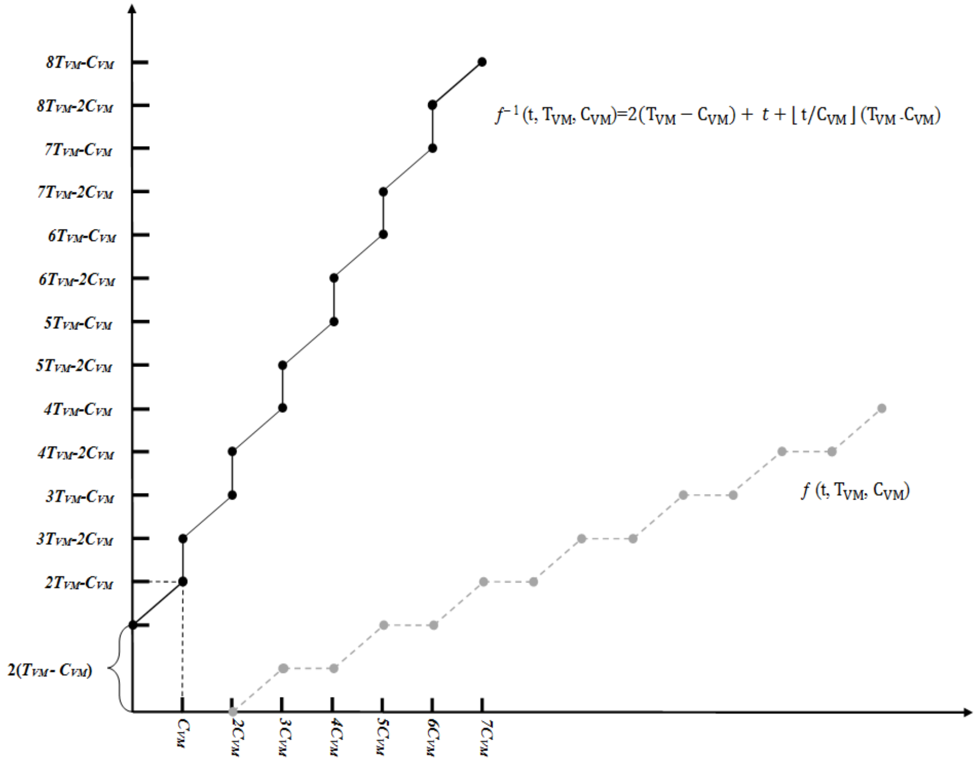


Fig. 2. The function $\mathbf{f^{-1}(t, T_{VM}, C_{VM})}$; $\mathbf{t}$ is the parameter on the x-axis in the graph.

By looking at Fig. 2 we see that

$$f^{-1}(t, T_{VM}, C_{VM}) = 2(T_{VM} - C_{VM}) + t + \lfloor t/C_{VM} \rfloor (T_{VM} - C_{VM}) \tag{4}$$

From previous results on R_i (see [9] and above), and from the definition of f^{-1} we get that the worst-case response time for task τ_i is

$$R_i = f^{-1}\left(\left(C_i + \sum_{j=1}^{i-1} \lceil R_i/T_j \rceil C_j\right), T_{VM}, C_{VM}\right) \tag{5}$$

For example if we have two tasks and $T_1 = 8$, $C_1 = 1$, and $T_2 = 15$, $C_2 = 3$, and $T_{VM} = 6$ and $C_{VM} = 3$ we get $7 = R_1 = f^{-1}(1, 6, 3)$ and $14 = R_2 = f^{-1}\big((3 + \lceil 14/8 \rceil * 1), 6, 3\big)$.

In order to solve Equation (5), one needs to use numeric and iterative methods, i.e., a very similar approach as the well-known method used for obtaining R_i in the non-virtualized case [9] (this approach can easily be implemented in a program that calculates the R_i values). In order to meet all deadlines for all tasks τ_i, we need to select T_{VM} and C_{VM} so that Equation (5) $\leq T_i (1 \leq i \leq n)$.

5. EXAMPLE

Consider the following small real-time application with three tasks.

Table 1. Example of a small real-time application with three tasks.

Task	Period (T_i)	Worst-case execution time (C_i)	Utilization (U_i)
τ_1	16	2	2/16 = 0.125
τ_2	24	1	1/24 = 0.042
τ_3	36	4	4/36 = 0.111
			$\sum$ = 0.278

As discussed above, we use fixed priorities and RMS priority assignment. If we let the VM that executes this application use 40% of a CPU resource, i.e., if $C_{VM}/T_{VM} = 0.4$, we can use Equation (4) to calculate the maximum T_{VM} so that all three tasks will meet their deadlines. When $C_{VM}/T_{VM} = 0.4$ we can replace C_{VM} with $0.4T_{VM}$ in Equation (4), thus obtaining the function $f^{-1}(t, T_{VM}) = 1.2T_{VM} + t + \lfloor t/0.4T_{VM} \rfloor (0.6T_{VM})$.

We start by looking at τ_1. We need to find the maximal T_{VM} so that $R_1 = f^{-1}\left(\left(C_1 + \sum_{j=1}^{0} \lceil R_1/T_j \rceil C_j\right), T_{VM}\right) = f^{-1}(C_1, T_{VM}) = f^{-1}(2, T_{VM}) \leq T_1 = 16$. In general, f^{-1} is solved using a numeric and iterative approach in a similar way as R_i is obtained in the non-virtualized case [9]. However, we will see that for this τ_1 the $\lfloor t/C_{VM} \rfloor (T_{VM} - C_{VM})$ part of f^{-1} can be ignored. In that case, we get the following equation for the maximum T_{VM}: $1.2T_{VM} + 2 = 16$, and from this we get $T_{VM} = 14/1.2 = 11.7$. If we have a period of 11.7 we get a $C_{VM} = 0.4 \times 11.7 = 4.68$, and (as predicted above) since $C_{VM} > C_1$, we know that we do not have to consider the $\lfloor t/C_{VM} \rfloor (T_{VM} - C_{VM})$ part of f^{-1}.

We now look at τ_2. We want to find the maximal T_{VM} so that $R_2 = f^{-1}\left(\left(C_2 + \sum_{j=1}^{1} \lceil R_2/T_j \rceil C_j\right), T_{VM}\right) \leq T_2 = 24$. It is clear that τ_2 will miss its deadline with $T_{VM} = 14/1.2 = 11.7$ (which is the maximal T_{VM} period for which τ_1 will meet its deadlines); if we use $T_{VM} = 14/1.2 = 11.7$, the first execution period will (in the worst-case, see Fig. 1) start at time $2(T_{VM} - C_{VM}) = 1.2T_{VM} = 14$. Since $T_1 = 16$ and $C_1 = 2$ we see that τ_1 will execute two times back-to-back in this interval, i.e., after the first execution of τ_1 it will be released again at time 16. Consequently, τ_2 cannot start executing until time 18, and the first execution period of the VM will end at $2T_{VM} - C_{VM}$ (see Fig 1) $= 1.6T_{VM} = 1.6 \times 11.7 = 18.7$, and since $C_1 = 1, \tau_2$ cannot complete during the first execution period of the VM. The second period of the VM starts at time $3T_{VM} - 2C_{VM}$ (see Fig 1) $= 2.2T_{VM} = 2.2 \times 11.7 = 25.7$, which is after the deadline of $\tau_2 (T_2 = 24)$.

By using our formulas we see that in order for τ_2 to meet its deadlines T_{VM} cannot be larger than $13/1.2 = 10.8$. This means that we now know that the real-time application can at most have $T_{VM} = 10.8$ when $C_{VM}/T_{VM} = 0.4$. For $T_{VM} = 10.8$ and $C_{VM}/T_{VM} = 0.4$, the corresponding C_{VM} is $0.4 \times 10.8 = 4.33$.

We finally look at τ_3. We need to find the maximum T_{VM} so that $R_3 = f^{-1}\left(\left(C_3 + \sum_{j=1}^{2} \lceil R_3/T_j \rceil C_j\right), T_{VM}\right) \leq T_3 = 36$. In this case we see that τ_3 will not meet its deadline when $T_{VM} = 13/1.2 = 10.8$. The reason for this is that both τ_1 and τ_2 will cause interference on τ_3, and τ_3 will as a consequence of this not complete in the first T_{VM} cycle, since $C_1 + C_2 + C_3 = 2 + 1 + 4 = 7 > 4.33$. The second T_{VM} cycle will complete at time $3T_{VM} - C_{VM}$ (see Fig. 1) $= 3 \times 10.8 - 4.33 = 28.07$. Before the end of this cycle both τ_1 and τ_2 will have had one new release each (τ_1 at time 16 and τ_2 at time 24). This means that τ_3 will not complete during the second cycle of T_{VM} since $2C_1 + 2C_2 + C_3 = 4 + 2 + 4 = 10 > 2 \times 4.33 = 8.66$. In the worst-case scenario (see Fig. 1), the third cycle of T_{VM} will start at time $4T_{VM} - 2C_{VM} = 4 \times 10.8 - 2 \times 4.33 = 34.54$. At time 32 there is a new release of task τ_1, and since τ_1 has higher priority than τ_3, task τ_1 will execute for two time units starting at time 34.54.

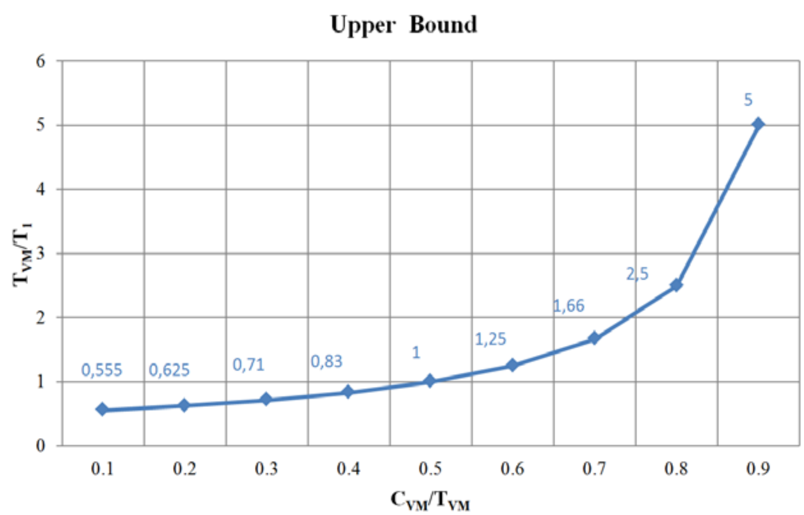


Fig. 3. The upper bound for $\mathbf{T_{VM}}$ / T_1

Since $T_3 = 36$, we see that τ_3 will miss its deadline. This means that we need a shorter period T_{VM} in order to guarantee that also τ_3 will meet its deadlines. When using our formulas, we see that $T_{VM} = 10$ is the maximal period that τ_3 can tolerate in order to meet its deadline when $C_{VM}/T_{VM} = 0.4$, i.e., for $C_{VM}/T_{VM} = 0.4$ we get $T_{VM} = 10$, and τ_3 is the task that requires the shortest period T_{VM}. When $C_{VM}/T_{VM} = 0.5$ we can use our formulas to calculate a T_{VM}. In this case we get a maximal T_{VM} of 14 for task τ_1, and the calculations for tasks τ_2 and τ_3 will result in larger values on the maximal T_{VM}.

This means that τ_1 is the task that requires the shortest period T_{VM}, i.e., $T_{VM} = 14$ when $C_{VM}/T_{VM} = 0.5$. In general, the period T_{VM} will increase when the utilization C_{VM}/T_{VM} increases, and the task that is "critical" may change when C_{VM}/T_{VM} changes (e.g., task τ_3 when $C_{VM}/T_{VM} = 0.4$ and task τ_1 when $C_{VM}/T_{VM} = 0.5$).

6. SIMULATION STUDY

In Section 5 we saw that the maximal T_{VM}, for which a task set inside the VM is schedulable increases when C_{VM}/T_{VM} increases. In this section we will quantify the relation between the maximal T_{VM} the utilization C_{VM}/T_{VM}.

We will do a simulation study where we consider two parameters:

- n – the number of tasks in the real-time application
- u – the total utilization of the real-time application

The periods T_i are taken from a rectangular distribution between 1000 and 10000. The worst-case execution time C_i for task τ_i is initially taken from a rectangular distribution between 1000 and T_i.

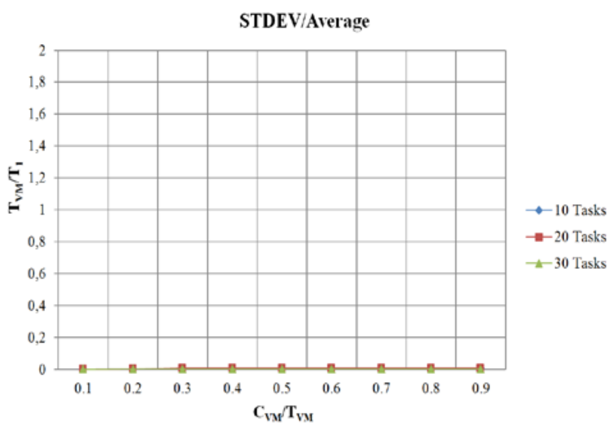


Fig. 4. Standard deviation for $\mathbf{T_{VM}}$ divided with average $\mathbf{T_{VM}}$ when the total utilization u = 0.1.

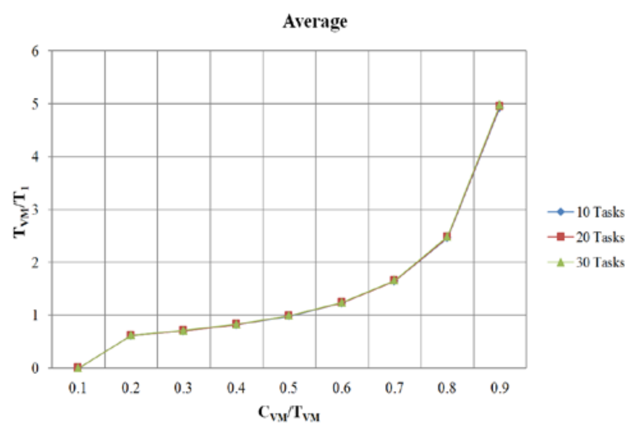


Fig. 5. Average $\mathbf{T_{VM}/T_1}$ when the total utilization u = 0.1.

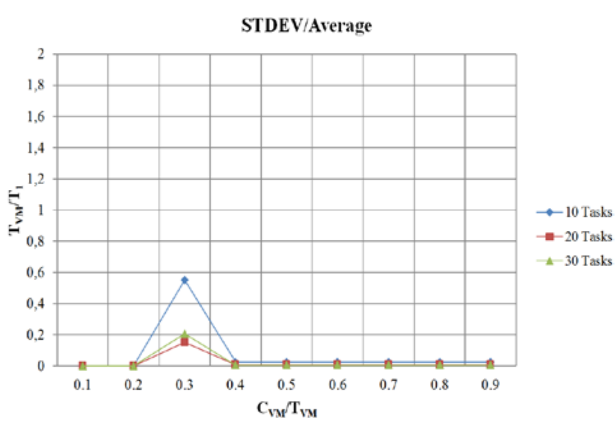


Fig. 6. Standard deviation for $\mathbf{T_{VM}}$ divided with average $\mathbf{T_{VM}}$ when the total utilization u = 0.2.

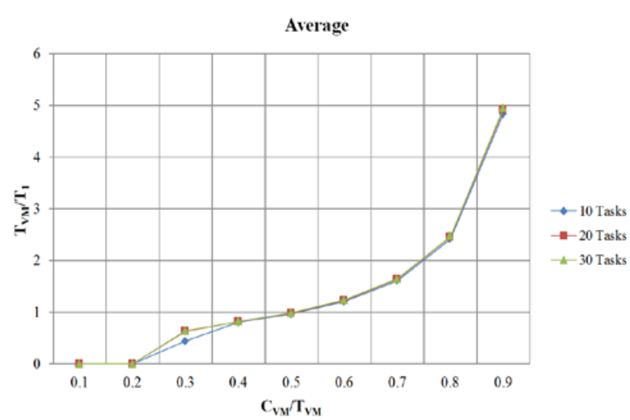


Fig. 7. Average $\mathbf{T_{VM}/T_1}$ when the total utilization u = 0.2.

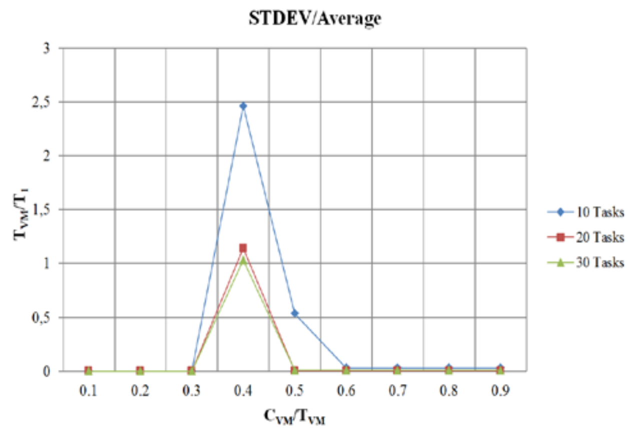


Fig. 8. Standard deviation for $\mathbf{T_{VM}}$ divided with average $\mathbf{T_{VM}}$ when the total utilization u = 0.3.

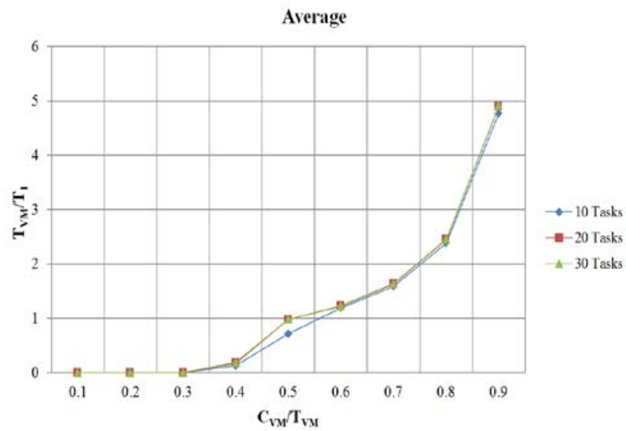


Fig. 9. Average $\mathbf{T_{VM}/T_1}$ when the total utilization u = 0.3.

All worst-case execution times are then scaled by a factor so that we get a total utilization u. For each task, we then find the maximum T_{VM} using Equation (5) so that all tasks meet their deadlines. We refer to this period as T_{max}, and we then select the minimum of the n different T_{max} values (one value for each task). For each pair of n and u, we calculate the minimum T_{max} for $(C_{VM}/T_{VM} = 0.9)$, $(C_{VM}/T_{VM} = 0.8)$, $(C_{VM}/T_{VM} = 0.7)$, $(C_{VM}/T_{VM} = 0.6)$, $(C_{VM}/T_{VM} = 0.5)$, $(C_{VM}/T_{VM} = 0.4)$, $(C_{VM}/T_{VM} = 0.3)$, $(C_{VM}/T_{VM} = 0.2)$, $(C_{VM}/T_{VM} = 0.1)$ for 10 randomly generated programs (we generate the programs using the distribution and the technique described above). We look at n = 10, 20, and 30, and u = 0.1, 0.2 and 0.3. This means that we look at 3*3 = 9 combinations of n and u, and for each of these nine combinations we look at 9 different values on C_{VM}/T_{VM}. This means that we looked at 3*3*9 = 81 different scenarios. For each such scenario we generated 10 programs using a random number generator.

7. RESULTS

In the worst-case scenario (see Fig. 1), the maximum time that a virtual machine may wait before its first execution is $2(T_{VM} - C_{VM})$. In order for the real-time tasks not to miss their deadlines the maximum waiting time, $2(T_{VM} - C_{VM})$ must be less than the shortest period, T_1 (i.e., $2(T_{VM} - C_{VM}) < T_1$). For each value of C_{VM}/T_{VM}, we can replace C_{VM} and calculate the T_{VM}/T_1 using the formula above. For example, if $C_{VM}/T_{VM} = 0.7$ then we can replace C_{VM} by $0.7T_{VM}$ in $2(T_{VM} - C_{VM}) < T_1$ so we have $2(T_{VM} - 0.7T_{VM}) < T_1$, from that we get $0.6T_{VM} < T_1$, this means that $T_{VM}/T_1 < 1/0.6 = 1.66$. By continuing this we can calculate the values of T_{VM}/T_1 for each value of C_{VM}/T_{VM}. The corresponding graph is presented in Fig. 3. These values are clearly the upper bound for all the values of T_{VM}/T_1 , and our simulation study shows that, for all combinations of u, n, and C_{VM}/T_{VM} , T_{VM}/T_1 is less than this upper bound.

7.1 Total Utilization of 0.1

In Fig. 4, we see that the standard deviation is very small for u = 0.1. For n = 10 we get values around 0 and for n = 20, we get 0.006 and for n = 30, we get 0.009.

As shown in Fig. 5, for different number of the tasks (n = 10, 20 and 30) T_{VM} / T_1 increases when C_{VM}/T_{VM} increases. The first observation is thus that the maximum T_{VM} for which the task set inside the VM is schedulable increases when the VM gets a larger share of the physical processor, i.e., when C_{VM}/T_{VM} increases. The second observation is that the curves in Fig. 5 are below, but very close to, the upper bound in Fig. 3. Also, when total utilization (u) is 0.1, we observe that T_{VM} / T_1 is zero when $C_{VM}/T_{VM} = 0.1$. This means that the task set inside the VM is not schedulable when $C_{VM}/T_{VM} = 0.1$.

7.2 Total Utilization of 0.2

Fig. 6 shows that the standard deviation divided with the average is almost zero except when $C_{VM}/T_{VM} = 0.3$. This means that for $C_{VM}/T_{VM} = 0.3$ (and n = 10, 20, and 30) some task sets are schedulable with T_{VM} / T_1 close to the upper bound (0.71, see Fig. **3**), but other task sets need a much shorter T_{VM} **.** When is C_{VM}/T_{VM} larger than 0.3, all task sets are schedulable with T_{VM} / T_1 close to the upper bound.

Fig. 7 shows that the maximum T_{VM}, for which the task set inside the VM is schedulable, increases when C_{VM}/T_{VM} increases. When C_{VM}/T_{VM} is larger than 0.3 the curves in Fig. 7 are very close to the upper bound in Fig. 3. When $C_{VM}/T_{VM} = 0.1$, and 0.2, Fig. 7 shows that the task set inside the VM is not schedulable (i.e., T_{VM} / T_1 = 0 for these values). When $C_{VM}/T_{VM} = 0.3$, Fig. 7 shows that the average T_{VM} / T_1 are significantly below the upper bound. As discussed above, the reason for this is that when $C_{VM}/T_{VM} = 0.3$ some task sets are schedulable with T_{VM} / T_1 close to the upper bound, but other task sets need a much shorter T_{VM}.

7.3 Total Utilization of 0.3

Fig. 8 shows that the standard deviation divided with the average is almost zero except when $\mathbf{C_{VM}/T_{VM} = 0.4}$ (for n = 10, 20, and 30), and when $\mathbf{C_{VM}/T_{VM} = 0.5}$ (for n = 10). This means that for $\mathbf{C_{VM}/T_{VM} = 0.4}$ (and n = 10, 20, and 30) some task sets are schedulable with $\mathbf{T_{VM}}$ / T1 close to the upper bound (0.83, see Fig. 3), but other task sets need a much shorter $\mathbf{T_{VM}}$. For n = 10, we have a similar situation when $\mathbf{C_{VM}/T_{VM} = 0.5}$. In general, the standard deviation decreases when n increases.

8. CONSIDERING OVERHEAD

In our previous model presented in Section 3, we neglected the overhead induced by switching from one VM to another. However, in reality this is not the case. So in this section we consider overhead at the beginning of execution of each VM.

8.1 DEFINING OVERHEAD

By considering the overhead we can rewrite the Equation (4) as

$$f^{-1}(t, T_{VM}, C_{VM}, X) = (T_{VM} - C_{VM}) + t + \left\lceil \frac{t}{(C_{VM} - X)} \right\rceil (T_{VM} - C_{VM} + X) \qquad (6)$$

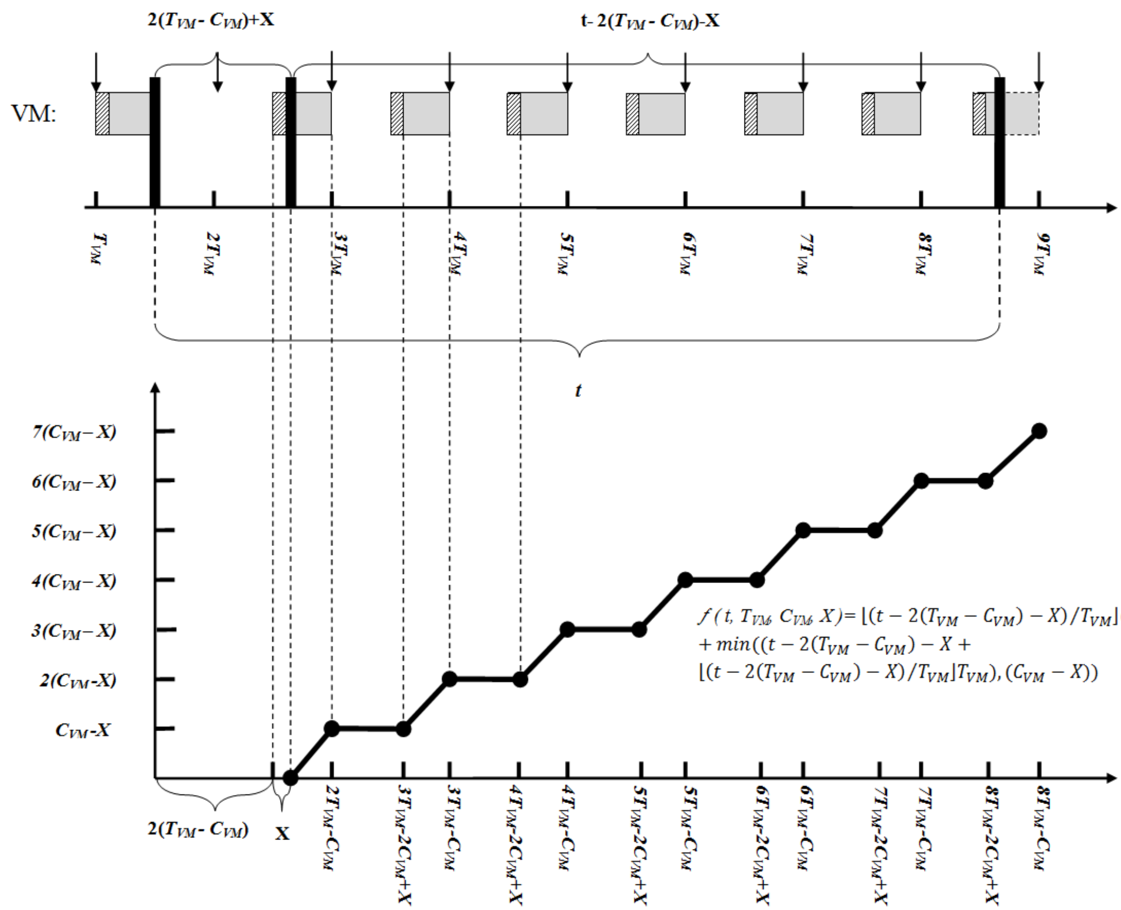


Fig 10. Worst-case scenario when considering context switches overheads.

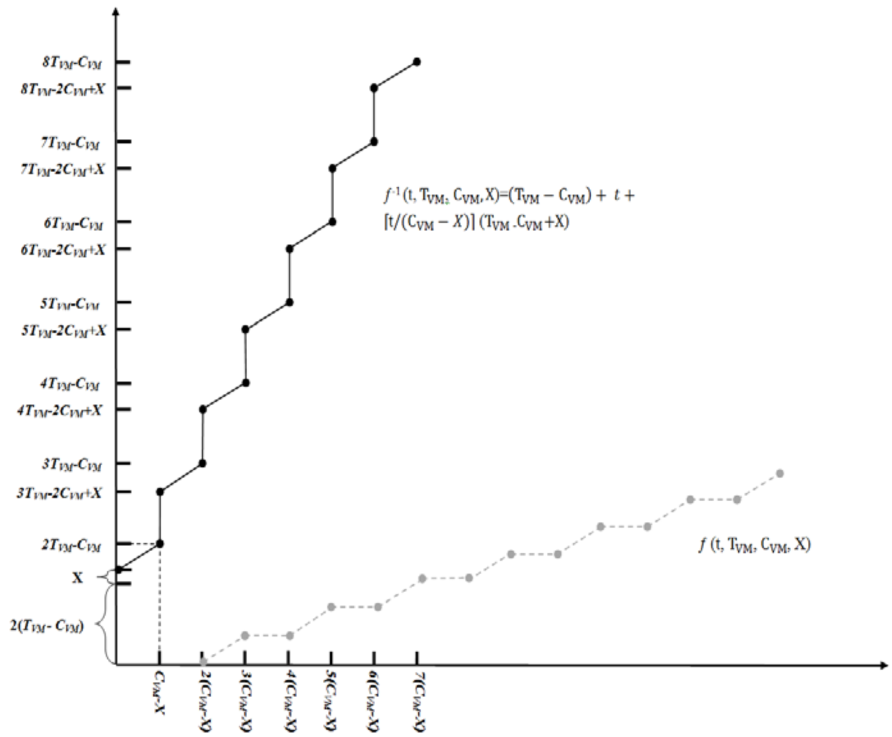


Fig 11. The function $\mathbf{f^{-1}(t, T_{VM}, C_{VM}, X)}$; $\mathbf{t}$ is the parameter on the x-axis in the graph.

where X denotes the overhead (see figures 10 and 11). So in the worst-case scenario considering this overhead model, the first execution of task τ_1 is after $2(T_{VM} - C_{VM}) + X$ time units (see Fig. 10). Our model is obviously valid for non-preemptive scheduling since we have considered overhead at the beginning of the execution of each task [35]. The overhead model can also be used in systems with preemptive scheduling, since one can put a bound on the number of preemptions in RM and EDF schedulers [36]. By multiplying the maximum number of preemptions with the overhead for a preemption, and then making the safe (but pessimistic) assumption that all this overhead occurs at the start of a period, we arrive at the model considered here. In [25] and [41] the authors calculated a different kind of overhead for periodic tasks; using our notation they calculated the ratio $((C_{VM}/T_{VM})$ – u)/u (which for fixed C_{VM}/T_{VM} and u is an increasing function of T_{VM}). In [26] and [40] the authors considered different overhead models (including overhead due to cache misses) on the task level in compositional analysis of real-time systems. Our model considers overhead on the hypervisor level, and our overhead analysis is in thus orthogonal to the overhead analysis on the task level (i.e., both models could be applied independently).

8.2 PREDICTION MODEL

For any task set, if C_{VM}/T_{VM} is given, it is possible to predict a range where we can search for values of T_{VM} that can make the task set inside the VM schedulable (i.e., a value of T_{VM} such that all tasks meet their deadlines). For values of T_{VM} that are outside of this interval, we know that there is at least one task that will not meet its deadline. In Fig 12(a) and (b), the solid line represents Maximum R_i/T_i $(1 \leq i \leq n)$ versus different values of T_{VM}, for given values of C_{VM}/T_{VM} and overhead X. As long as Maximum $R_i/T_i \leq 1$ we know that the task set is schedulable. If we can find two values of T_{VM} such that Maximum R_i/T_i = 1, Fig. 12(a) indicates that the interval between these two values is the range of values of T_{VM} for which the task set is schedulable.

We now define a prediction model consisting of three lines that will help us to identify the range of values that will result in a schedulable task set. We will refer to these three lines as: the left bound, the lower bound and the schedulability limit. These three lines will produce a triangle. The intersection between the schedulability limit line and the left bound will give us the first point and the intersection between the schedulability limit line and the lower bound will give us the second point. We consider the corresponding T_{VM} values of these two points on the x-axis and the interval between these two values (see Fig 12(a) and (b)). If the intersection between the lower bound and the schedulability limit is left of the intersection of the left bound and the schedulability limit (see Fig. 12(b)), then it is not possible to find a T_{VM} that will make the task set schedulable.

In order to find the left bound we use the equation below:

$$\left(\frac{(C_{VM}-X)}{T_{VM}}\right) \geq \text{Total Utilization} \quad (7)$$

According to Equation (7) a T_{VM} value must be selected so that $\left(\frac{(C_{VM}-X)}{T_{VM}}\right) \geq$ Total Utilization. Here X represents the amount of overhead and $(C_{VM}-X)$ represents the effective execution of the VM in one period. Obviously, $\left(\frac{(C_{VM}-X)}{T_{VM}}\right)$ should be higher than or equal to total utilization since in order to successfully schedule all tasks in the VM, the utilization of the VM should be a value that is the same as the total utilization or higher.

In order to calculate the lower bound we consider the Maximum R_i/T_i $(1 \leq i \leq n)$ value. Our previous experiments showed that it is often (but not always) the task with the shortest period that restricts the length of T_{VM}. τ_1 is the task in the task set which has the shortest period T_1, and obviously $R_1/T_1 \leq$ Maximum R_i/T_i.

We have $R_1/T_1 = (2(T_{VM}-C_{VM})+X+C_1)/T_1$, and if we rewrite this equation and consider T_{VM} as a variable then we can calculate the lower bound using the equation below:

$$f(T_{VM}) = \frac{2(T_{VM}-C_{VM})}{T_1} + \frac{(C_1+X)}{T_1} \quad (8)$$

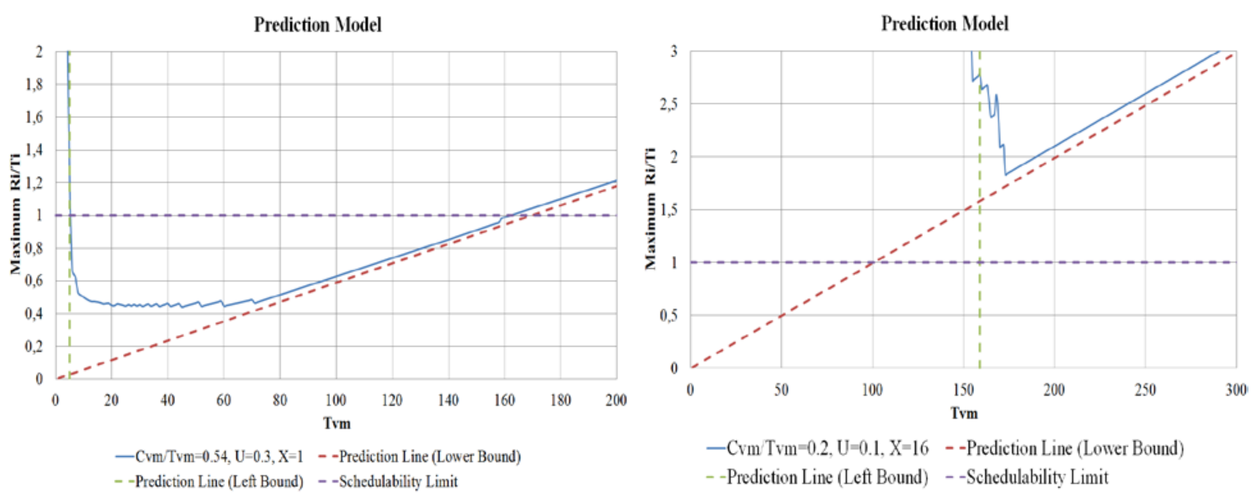
By considering the common form of a linear equation $f(x) = mx + b$ where m and b are constant values and m represents the slope of the line and b represents the offset, we can rewrite the Equation (8) if the value of C_{VM}/T_{VM} is given, e.g., $C_{VM}/T_{VM} = 0.54$, we can rewrite the Equation (8) as $f(T_{VM}) = \frac{2(1-0.54)}{T_1}(T_{VM}) + \frac{(C_1+X)}{T_1}$,. Thus the slope of the line becomes m=$\frac{2(1-0.54)}{T_1}$ and the offset is b=$\frac{(C_1+X)}{T_1}$.

The schedulability limit is represented by a horizontal line at Maximum $R_i/T_i = 1$. R_i/T_i $(1 \leq i \leq n)$.

In order to calculate the value Maximum R_i/T_i, we first calculate R_i/T_i for each task τ_i $(1 \leq i \leq n)$, and then Maximum R_i/T_i $(1 \leq i \leq n)$ is selected for the entire task set.

For instance in Fig 12(a), total utilization $U = 0.3$, $C_{VM}/T_{VM} = 0.54$ and $X = 1$, so to calculate the left bound using Equation (7), for $T_{VM} = 1$, we will get $(0.54T_{VM} - 1)/T_{VM} = (0.54(1) - 1)/1 = -0.46$ for $T_{VM} = 2$ we get 0.04, for $T_{VM} = 3$ we get 0.206 and for $T_{VM} = 4$ we will have 0.29 while for $T_{VM} = 5$ we will get 0.34. So in this case for $T_{VM} = 5$ we get the value of 0.34 which is higher than total utilization (0.3) so we know that for T_{VM} values that are higher than 5 we have a chance to find the suitable T_{VM} for the entire task set.

However for different values of X the same T_{VM} value may not be valid anymore, e.g., if we consider X = 2, then for $T_{VM} = 5$ we will get $\frac{(C_{VM}-X)}{T_{VM}} = 0.14$. For $T_{VM} = 9$ we get $\frac{(C_{VM}-X)}{T_{VM}}$= 0.31 which is higher than total utilization (0.3).



(a) (b)

Fig 12. (a)Prediction Model for $C_{VM}/T_{VM} = 0.54, u = 0.3$ and $X = 1$ and (b) Prediction Model for $C_{VM}/T_{VM} = 0.2, u = 0.1$ and $X = 16$

In Fig 12(b), for total utilization $u = 0.1$, $C_{VM}/T_{VM} = 0.2$ and $X = 16$, if we calculate the left bound using Equation (7), then for $T_{VM} = 160$ we get $(0.2T_{VM} - 1)/T_{VM} = (0.2(160) - 16)/160 = 0.1$ which is equal to total utilization (0.1) (i.e., $T_{VM} = 160$ is the left bound).

For the lower bound, if we consider the first task in the task set has the shortest period $T_1 = 157$, and its execution time is $C_1 = 2$. Using Equation (8), we can get the slope of the line $\frac{2(1-0.54)}{T_1} = \frac{2(0.46)}{157} \approx 0.006$ and the offset will be $\frac{(C_1+X)}{T_1} = \frac{(2+1)}{157} \approx 0.02$ so we can plot a line which changes in proportional to T_{VM} values $f(T_{VM}) = 0.006(T_{VM}) + 0.02$ (see Fig 12 (a)).

In Fig. 12 (b), we can consider a task in the task set with the shortest period, $T = 161$ and execution time $C = 2$, so we will have the linear equation $f(T_{VM}) = 0.009(T_{VM}) + 0.11$ which corresponds to the lower bound and changes in proportional to T_{VM} values. As it can be observed from Fig. 12 (b), if the intersection between the left bound and the lower bound become above the schedulability limit line there is no chance of finding any T_{VM} value that can schedule this task set when the overhead is $X = 16$.

In order to validate our model, in the next section we have presented different experiments with different number of task sets and various values of total utilizations and C_{VM}/T_{VM}. We have also considered different amounts of overhead (X).

8.3 OVERHEAD SIMULATION STUDY

During overhead simulation we had 10 task sets of 10 tasks each, n = 10 (we saw no major difference when increasing the number of tasks). Each task τ_i $(1 \leq i \leq n)$ has a period T_i $(1 \leq i \leq n)$ and execution time C_i $(1 \leq i \leq n)$. Each task‘s period T_i is randomly generated in the range of [100, 1000] and the execution time C_i is randomly generated in the range of [100, T_i]. The C_i values are then multiplied with a factor to get the desired utilization u. Different values have been considered for overheads $X = 1, 2, 4, 8, 16$. Different values for total utilization u and C_{VM}/T_{VM} are also considered (see Table 2).

Table 2. Overhead simulation, different values for **Total utilization (u)** and $\mathbf{C_{VM}/T_{VM}}$.

		$\mathbf{C_{VM}/T_{VM}}$			
Total Utilization (u)	**0.1**	0.14	0.16	0.18	0.2
	0.2	0.28	0.32	0.36	0.4
	0.3	0.42	0.48	0.54	0.6

Given X, u and C_{VM}/T_{VM}, and using Equation (6) we calculate R_i for all different values of T_{VM} in the range of [0, 500]. We then calculate R_i/T_i $(1 \leq i \leq n)$ for each task for each value of T_{VM}, and then we obtain the Maximum R_i/T_i $(1 \leq i \leq n)$ for each value of T_{VM} (see Figures 13-15). The figures show the average value of the 10 task sets.

8.3.1 Total Utilization of 0.1

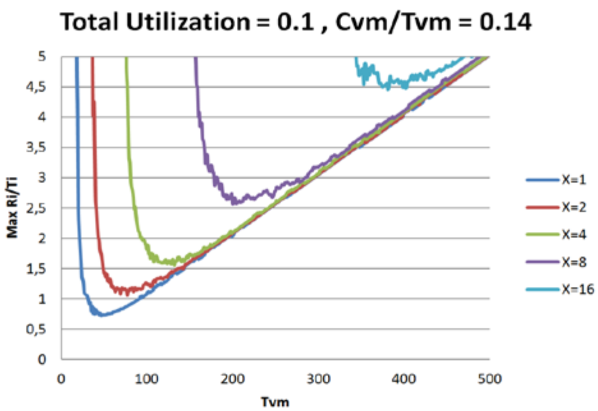
Fig. 13 shows that for total utilization of 0.1 and overhead values are more than 4 ($X = 8$ and $X = 16$), the task sets are not schedulable. However, for small value of overhead $X = 1$, the task sets are always schedulable for the values of C_{VM}/T_{VM} considered here. When the overhead values are $X = 2$ and $X = 4$, the task sets are schedulable only when C_{VM}/T_{VM} = 0.2.

8.3.2 Total Utilization of 0.2

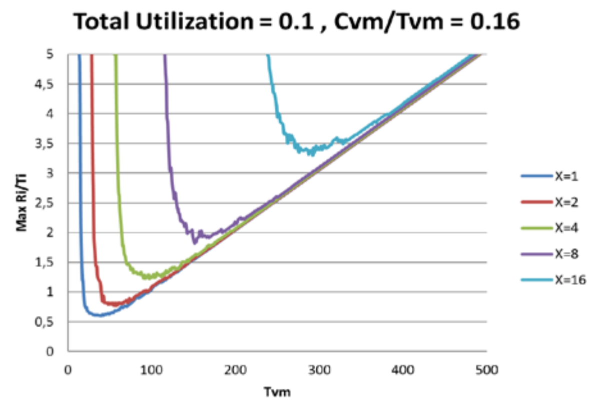
Fig. 14 shows the Maximum R_i/T_i values for different T_{VM} and overhead values when total utilization is 0.2. As we can observe in the figures, the task sets are not schedulable when $X = 16$ (even for C_{VM}/T_{VM} = 0.4). For other overhead values ($X = 1, 2, 4$), we see that in most of the cases the task sets are schedulable. When the value of C_{VM}/T_{VM} = 0.4, even task sets with overhead value of $X = 8$ are schedulable.

8.3.1 *Total Utilization of 0.3*

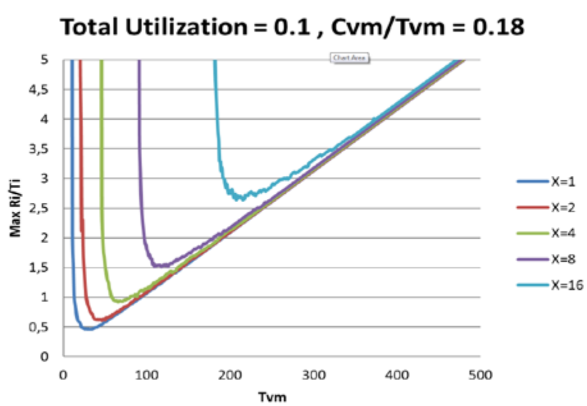
For total utilization of 0.3, Fig. 15 shows that when C_{VM}/T_{VM} is 0.54 and 0.6, all tasks in the task set will meet their deadlines for at least some T_{VM} value.



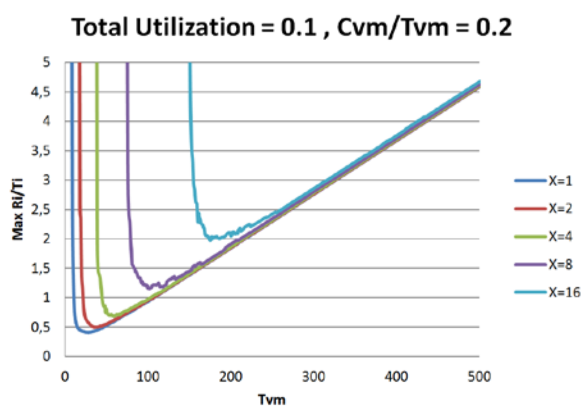
(a)



(b)

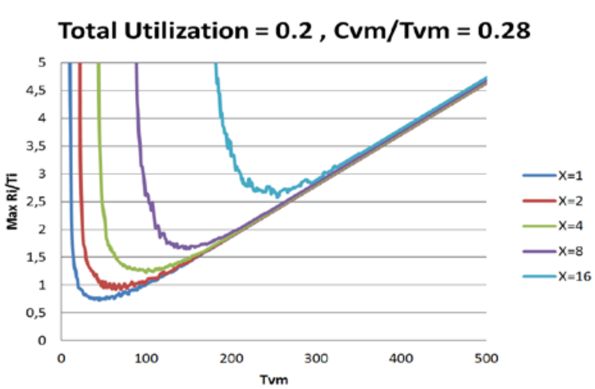


(c)

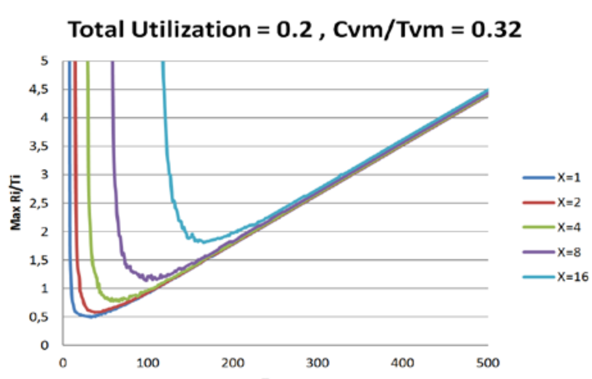


(d)

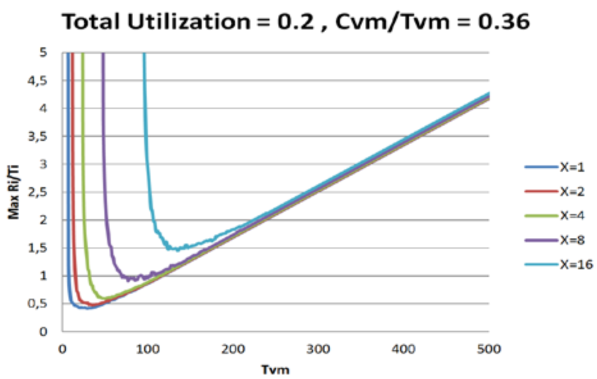
Fig. 13. Overhead simulation results for $\mathbf{C_{VM}/T_{VM} = 0.14, 0.16, 0.18, 0.2}$ when total utilization $\mathbf{u = 0.1}$



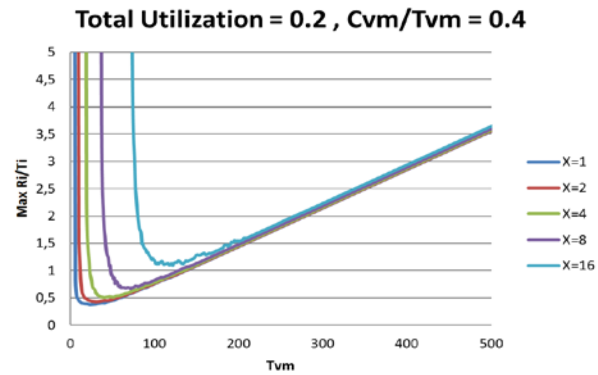
(a)



(b)

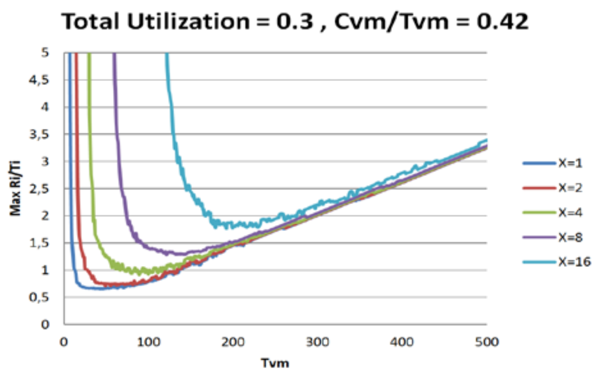


(c)

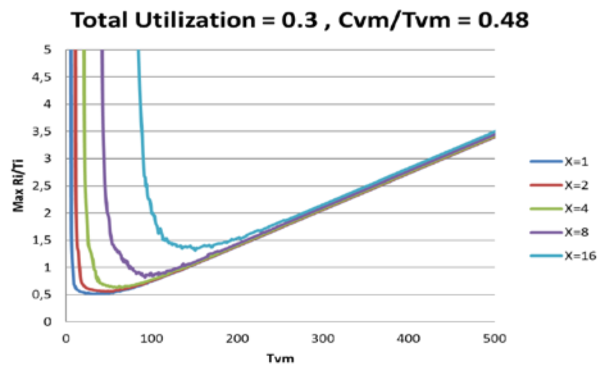


(d)

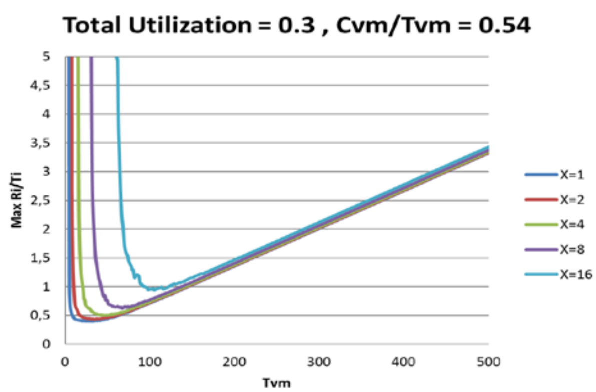
Fig 14. Overhead simulation results for $\mathbf{C_{VM}/T_{VM} = 0.28, 0.32, 0.36, 0.4}$ when total utilization is $\mathbf{u = 0.2}$



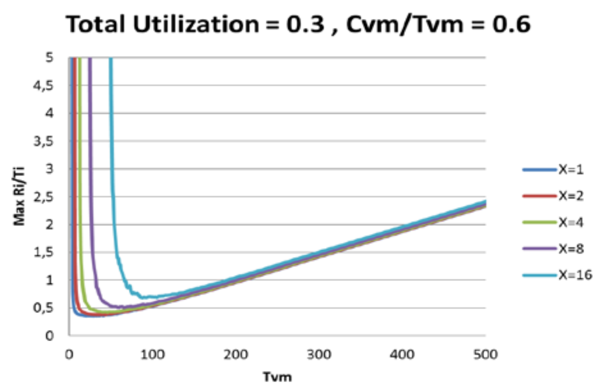
(a)



(b)



(c)



(d)

Fig 15. Overhead simulation results for $\mathbf{C_{VM}/T_{VM} = 0.42, 0.48, 0.54, 0.6}$ when total utilization is $\mathbf{u = 0.3}$

9. CONCLUSIONS

We consider a real time application consisting of a set of n real-time tasks $\tau_i (1 \leq i \leq n)$ that are executed in a VM; the tasks are sorted based on their periods, and τ_1 has the shortest period. We have defined a function $f^{-1}(t, T_{VM}, C_{VM})$ such that a real-time application that uses fixed priorities and RMS priority assignment will meet all deadlines if we use a VM execution time C_{VM} and a VM period T_{VM} such that $R_i = f^{-1}\left(\left(C_i + \sum_{j=1}^{i-1} \lceil R_i/T_j \rceil C_j\right), T_{VM}, C_{VM}\right) \leq T_i (1 \leq i \leq n)$. This makes it possible to use existing real-time scheduling theory also when scheduling VMs containing real-time applications on a physical server.

The example that we looked at in Section 5 shows that there is a trade-off between on the one hand a long T_{VM} period (which reduces the overhead for switching between VMs), and low processor utilization (i.e., low C_{VM}/T_{VM}). The example also shows that the "critical" task, i.e., the task which puts the toughest restriction on the maximal length of T_{VM}, may be different for different values on C_{VM}/T_{VM} .

From the simulation results shown in Section 6, we see that increasing the number of the tasks (n) does not affect the maximum T_{VM} for which the task set inside the VM is schedulable (see Fig. 5, Fig. 7 and Fig. 9). The simulation results also show that the standard deviation of the maximum T_{VM} is almost zero except when C_{VM}/T_{VM} is slightly above the total utilization (u) of the task set (see Fig. 4, Fig. 6 and Fig. 8).

We have also presented an upper bound on the maximum T_{VM} for which the task set inside the VM is schedulable (see Fig. 3). The simulation results show that the maximum T_{VM} is very close to this bound when C_{VM}/T_{VM} is (significantly) larger than the total utilization (u) of the task set inside the VM.

If overhead from switching from one VM to another is ignored, the simulation study in Section 6 shows those infinitely small periods (T_{VM}) are the best, since they minimize processor utilization. In order to provide more realistic results, we included and evaluated an overhead model that makes it possible to consider the overhead due to context switches between VMs. Along with the model we also defined two performance bounds and a schedulability line, each representing a straight line in a figure that plots the Maximum R_i/T_i as a function of the period of the VM (T_{VM}). These three lines form a triangle and we show that the intersection between the performance bounds and the schedulability lines defines an interval where valid periods (i.e., periods that could result in all tasks meeting their deadlines) can be found. This performance model also makes it possible to easily identify cases when no valid T_{VM} can be found.

We have also done a simulation study that shows how the overhead for switching from one VM to another affects the schedulability of task set running in the VM.

Our method is presented in the context of VMs with one virtual core. However, it is easily extendable to VMs with multiple cores as long as each real-time task is allocated to one of the (virtual) cores. In that case we need to repeat the analysis for each of the virtual cores and make sure that all real-time tasks on each core meet their deadlines.

REFERENCES

[1] Lee M., Krishnakumar A.S., Krishnan P., Singh N., and Yajnik S. 2010. Supporting Soft Real-Time Tasks in the Xen Hypervisor. The 2010 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution (Pittsburg, Mar. 2010).

[2] Duda K. and Cheriton D. 1999, Borrowed-Virtual-Time (BVT) scheduling: supporting latency-sensitive threads in a general-purpose scheduler. ACM SIGOPS Operating Systems Review, 33 (5), December 1999.

[3] Stoica I., Abdel-Wahab H., Jeffay K., Brauha S., Gehrke J., and Plaxton G., 1996. A Proportional Share Resource Allocation Algorithm for Real-Time, Time-Shared Systems. 17^{th} IEEE Real Time Systems Symposium, December 1996.

[4] Nieh J. and M. Lam. 2003. A SMART scheduler for multimedia applications. ACM Transactions on Computer Systems, vol. 21, No. 2, May 2003.

[5] Lin B. and DindaP.A. 2005. VSched: Mixing Batch and Interactive Virtual Machines Using Periodic Real-Time Scheduling. The 2005 ACM/IEEE SC05 Conference (Seattle, Nov. 2005).

[6] Salimi H, Najafzadeh M., and Sharifi M. 2012. Advantages, Challenges and Optimization of Virtual Machine Scheduling in Cloud Computing Environments, International Journal of Computer Theory and Engineering, vol. 4, no. 2, April 2012.

[7] Liu C. and Leyland J. 1973. Scheduling algorithms for multiprogramming in a hard real-time environment, Journal of the ACM, 20(1), 1973.

[8] Lundberg L. 2002. Analyzing Fixed-Priority Global Multiprocessor Scheduling. IEEE Real Time Technology and Applications Symposium, San Jose, USA, September 2002.

[9] Burns A. and Wellings A. 2009. Real-Time Systems and Programming Languages, Addison Wesley, ISBN 978-0-321-41745-9, 2009.

[10] Liu S., Quan G., and Ren S. 2010. On-line Scheduling of Real-time Services for Cloud Computing. IEEE 6^{th} World Congress on Services, Miami, USA, July, 2011.

[11] Cucinotta T., Checconi F., Kousiouris G., Kyriazis D., Varvatigou T., Mazzetti A., Zlatev Z., Papay J., Boniface M., Berger S., Lamp D., Voith T., Stein M. 2010. Virtulised e-Learning with Real-Time Guarantees on the IRMOS Platform. IEEE International Conference on Service-Oriented Computing and Applications (SOCA).December 2010.

[12] Luca A. and Tommaso C. 2011, Efficient Virtualization of Real-Time Activities. IEEE International Conference on Service-Oriented Computing and Applications. USA, 2011, pp 1-4.

[13] Yunfa L., Xianghua X., Jian W., Wanqing L., Youwei Y. 2010. A Real-Time Scheduling Mechanism of Resource for Multiple Virtual Machine System. The ChinaGride Conference. Guangzhou, China, 2010, pp 137-143.

[14] Subramanian S., Nitish K., Kiran K. M., Sreesh P., Karpagam G. R. 2012. An Adaptive Algorithm for Dynamic Priority Based Virtual Machine Scheduling in Cloud. The IJCSI International Journal of Computer Science, November 2012, pp 397-383.

[15] Xiao J., Wang Z. 2012. A Priority Based Scheduling Strategy for Virtual Machine Allocations in Cloud Computing Environment. The International Conference on Cloud Computing and Service Computing, Shanghai, China, 2012, pp 50-55.

[16] Sisu X., Justin W., Chenyang L., Christopher G. 2011. RT-Xen: Towards Real-Time Hypervisor Scheduling in Xen. The International Conference on Embedded Software, Taipei, Taiwan, 2011, pp 39-48.

[17] Tommaso C., Dhaval G., Dario F., Fabio C. 2010. Providing Performance Guarantees to Virtual Machines. Proceedings of The 5th Workshop On Virtualization And Cloud Computing, Italy, 2010.

[18] Tommaso C., Gaetano A., Luca A. 2008. Real-Time Virtual Machines. The 29th Real Time Systems Symposium, Barcelona, Spain, December 2008.

[19] Tommaso C., Gaetano A., Luca A. 2009. Respecting temporal constraints in virtualized services. The Computer Software and Applications Conference, Seattle, U.S., July 2009, pp 73-78.

[20] Davis R., Burns A. A survey of Hard Real-Time Scheduling for Multiprocessor Systems, ACM Computing Surveys, Vol. 43, No. 4, October, 2011.

[21] Feng X. and Mok A. K. A Model of Hierarchical Real-Time Virtual Resources. In Proceedings of the 23rd IEEE Real-Time Systems Symposium, Austin, TX USA, Dec. 2002, pp 26-35.

[22] Shih I. and Lee I. Periodic Resource Model for Compositional Real-Time Guarantees. In Proceedings of the 24th Real-Time Systems Symposium, Cancun, Mexico, Dec. 2003, pp 2-13.

[23] Lipari G. and Bini E. A methodology for designing hierarchical scheduling systems. J. Embedded Computing, 2005, pp 257-269.

[24] Davis R. and Burns A. Hierarchical fixed priority pre-emptive scheduling. In 26th IEEE International Real-Time Systems Symposium. RTSS 2005.

[25] Shin I. and Lee I. Compositional real-time scheduling framework with periodic model. ACM Transactions on Embedded Computing Systems (TECS), 7(3):30, 2008.

[26] Phan L. T.X., Xu M., Lee J., Lee I., Sokolsky O. Overhead-Aware Compositional Analysis of Real-Time Systems. In 19th IEEE Real-Time and Embedded Technology and Applications Symposium, Philadelphia, PA, 2013, pp 237-246.

[27] Lee J., Xi S., Chen S., Phan L. T. X., Gill C., Lee I., Lu C., Sokolsky O. Realizing Compositional Scheduling through Virtualization. Proceedings of the IEEE 18th Real-Time and Embedded Technology and Applications Symposium, USA, 2012, pp 13-22.

[28] Asberg M., Nolte T., Kato S., Rajkumar R. ExSched: An External CPU Scheduler Framework for Real-Time Systems. 18th IEEE International conference on Embedded and Real-Time Computing Systems and Applications (RTCSA), Seoul, 2012, pp 240-249.

[29] Yang J., Kim H., Park S., Hong C., Shin I. Implementation of Compositional Scheduling Framework on Virtualization. Published in Newsletter ACM SIGBED, Vol 8 Issue 1, 2011, pp 30-37.

[30] Behnam M., Nolte T., Shin I., Asberg M., Bril R. Towards Hierarchical Scheduling in VxWorks. 4th International Workshop on Operating Systems Platforms for Embedded Real-Time Applications, Prague, Czech Republic, 2008, pp 63-72.

[31] Lipari G., Bini E. Resource Partitioning among Real-Time Applications. Proceedings of the 15th Euromicro conference on Real-Time Systems, 2003, pp 151-158.

[32] Shin I., Lee I. Compositional Real-Time Scheduling Framework. 25th IEEE International Real-Time Systems Symposium, 2004, pp 57-67.

[33] Zmaranda D., Gabor G., Popescu D.E., Vancea C., Vancea F. Using Fixed Priority Pre-emptive Scheduling in Real-Time Systems. Published in International Journal of Computers Communications and Control, 2011, pp 187-195.

[34] Saewong S., Rajkumar R., Lehoczky J., Klein M. Analysis of hierarchical fixed-priority scheduling. Proceedings of the 14th Euromicro Conference on Real-Time systems, CA, 2002, pp 173-181.

[35] Baruah S. The Non-preemptive scheduling of periodic tasks upon multiprocessors. Published in journal of real-time systems, USA, 2006, pp 9-20.

[36] Easwaran A., Shin I., Lee I., Sokolsky O. Bounding Preemptions under EDF and RM Schedulers. MS-CIS-06-07, Department of Computer and Information Science, University of Pennsylvania.

[37] Lundberg L., Shirinbab S. Real-time scheduling in cloud-based virtualized software systems. In proceedings of the Second Nordic Symposium on Cloud Computing, Oslo, Norway:ACM, 2013, pp 54-58.

[38] Easwaren A., Anand M., Insup L. Compositional analysis framework using EDP resource models. Published in Real-time systems symposium, 2007, pp 129-138.

[39] Lu W., Li K., Wei H., Shih W. Rate monotonic schedulability tests using period dependent conditions. Published in Journal Real-Time systems, 2007, pp 123-138.

[40] Meng X., Phan L., Lee I., Sokolsky O. Cache-aware compositional analysis of real-time multicore virtualization platforms. Published in Real-Time systems symposium, 2013, pp 1-10.

[41] Chen S., Phan L., Lee J., Lee I., Sokolsky O. Removing abstraction overhead in the composition of hierarchical real-time systems. Proceedings of the 17th IEEE Real-time and embedded technology and applications, 2011, pp 81-90.