Real-time virtualization with Xvisor

Real-time virtualization with Xvisor

Abstract

Embedded virtualization has gained attention in recent years due to increasing usage of embedded systems in cyber-physical systems and the Industry 4.0 revolution. Especially in combination with multi-core embedded systems, virtualization reduces the number of embedded systems and simultaneously delivers a secure and separated environment in each virtualized system. Applications in such cyber-physical systems often require real-time guarantees with hard deadlines. To guarantee those real-time constraints in virtualization, both hypervisor and guest operating system must support real-time scheduling. Selecting the optimal scheduling algorithm on both scheduling levels is hard and is only optimal for the analysed application. Due to the multiple scheduling levels, a set of scheduling algorithm combinations must be analysed which is too costly without analysis on higher abstraction levels. By using an analysis methodology to find this optimal combination using higher abstraction levels analysis, we reduce the set at every abstraction level. In this paper, we present a real-time hypervisor, based on Xvisor, for multi-core embedded systems. We modified the hypervisor to support real-time scheduling and the compositional schedulability analysis and validated the analysis methodology using this embedded hypervisor.

1. Introduction

Embedded systems are widely used in today's mechatronic systems, largely due to decreasing cost of resources such as computing power and communication bandwidth. Trends indicate an evolution towards more powerful embedded systems in the mechatronic context. This evolution has two main drivers:

1. The shift from mechanical control towards more powerful electronic control;
2. The merger of cyber-physical systems (CPS) and the Internet-of-Things (IoT).

This evolution fits with what is referred to as the next industrial revolution: Industry 4.0 [1]. In the present-day environment, everything is being connected to the internet to collect and share data. This connection to other devices and/or clouds will result in improved efficiency, accuracy and economic benefits. Smart cities, smart grids, autonomous vehicles are just a few examples of the high potential of this connected world. However, CPSs are often designed as a closed system, particularly when real-time and deterministic behaviour must be

guaranteed. By closed system we mean that every event scenario is taken into account in the design and every component must be engineered to guarantee a pre-defined behaviour. IoT, on the other hand, is an open world where the environment changes dynamically and components are added and removed at run-time. Combining both platforms in a safe and reliable manner is difficult and challenging with current technologies.

In this context, complexity becomes a dominant issue in designing embedded systems. In an attempt to handle increased complexity, techniques for general purpose computing are adopted: (1) multi-core processing increases the computing power on a single chip; (2) virtualization enables multiple operating systems to run on the same hardware simultaneously and independently.

The requirements of a personal computer or cloud infrastructures differ from those of embedded systems. The former are optimized towards the average computing performance while for the latter (hard) real-time behaviour is the key concern.[1] Real-time behaviour requires that the execution of a task must not only be functionally correct; the output has to be produced within a predefined amount of time, i.e. before a deadline. The determinism of the system results in a controllable execution of the task.

Real-time scheduling on an embedded system with a multi-core processor is supported by real-time operating systems (RTOS). However, real-time scheduling in virtualization software for embedded systems has very limited support. Especially for virtualization software with multiple virtual machines (VMs) running applications with real-time constraints on an embedded system.

In this paper we discuss the development of a real-time embedded hypervisor Xvisor-RT that supports multiple VMs with real-time tasks. The focus of the new hypervisor is on scheduling a set virtual processors (VCPUs) with real-time constraints on a multi-core embedded system. The hypervisor is based on Xvisor hypervisor [3]. The key modifications relate to the scheduling mechanism to support real-time scheduling. We have also added scheduling algorithms for the virtual processors. To evaluate the hypervisor, we have measured the overhead of the scheduler and have analysed applications with real-time tasks scheduled on multiple real-time VMs.

The remainder of the paper will be structured as follows: first, the related work on real-time virtualization and embedded hypervisor is summarized in Section 2. The schedulability analysis for real-time scheduling in virtualization together with the architecture of Xvisor are explained in detail in Section 3. We explain our analysis methodology in Section 4. The implementation of the real-time scheduling structure of Xvisor-RT is described in Section 5. The evaluation of analysis methodology, including Xvisor-RT, is performed in Section 6.

[1] Some cloud applications, such as edge computing [2], on-line gaming, multimedia applications, offer highly responsive cloud services with low end-to-end latencies.

2. Related Work

Real-time and embedded virtualization are both active research topics. However, most research on hypervisors focusses on only one of those topics. Few papers consider the link between both.

Oikawa et al. [4] presented the Gandalf hypervisor for the x86 CPU architecture. It supports para-virtualization and can host multiple RTOSes together with Linux. However, no VCPU or VM scheduling is discussed in their work.

Kanda et al. [5] presented SPUMONE, an embedded hypervisor, and implemented it on the SH-4A CPU. SPUMONE is designed for single-core architectures and supports an RTOS and a general purpose OS. The guest uses μITRON as RTOS and Linux as is the general purpose OS [6]. However, only one RTOS can be executed with one VCPU. The VMs are scheduled with a fixed priority algorithm in which the VCPU of the RTOS has the highest priority. Whenever the RTOS is idle, the general purpose OS executes.

The Proteus hypervisor of Baldwin and Kerstan [7] for the PowerPC CPU architecture is the first embedded hypervisor that supports full virtualization, and para-virtualization and the combination of both. It provides deterministic VCPU scheduling for real-time applications. The hypervisor is limited to one PCPU. Gilles et al. [8] extended the Proteus hypervisor towards multi-core processors (PowerPC) with support for global VCPU scheduling. When a PCPU is shared between multiple VCPUs, the fixed time slice approach is applied.

SParK is an hypervisor also for the PowerPC architecture [9]. However, it uses para-virtualization and is limited to single-core platforms. It supports multiple real-time VMs and provides a schedulability analysis for the fixed priority-based scheduling algorithm in an hierarchical scheduling structure.

The NOVA hypervisor [10] is designed via a microkernel-like hypervisor with the focus on virtualization on a multi-core system with the x86 architecture. The small and simple design minimizes the code size of the hypervisor in root mode. However, parts of the hypervisor are replicated on each core in the user mode which increases overhead. The VCPUs, and threads, are scheduled with a priority-based round robin scheduling algorithm.

The OKL4 microvisor [11] is a combination of a microkernel and a hypervisor. It supports both para-virtualization and hardware-assist virtualization for ARM to be able to support a wide range of embedded systems. However, the hypervisor is not open-source.

Xen [12] is an open-source hypervisor developed in 2003 and is deployed in many servers. Because its architecture separates the scheduling mechanism and the scheduling algorithm, it lends itself to the design of new scheduling algorithms. RT-Xen is a real-time patch for the Xen hypervisor [13, 14]. It adds the G-EDF and G-RM algorithms to Xen with support for per VCPU parameter assignment. However, the support for embedded systems of Xen is limited. Its code size is extensive due to the initial target platform (x86 architecture in server environments). Masrur et al. [15] modified the SEDF scheduler of Xen to support real-time VMs by adding priority based scheduling. A real-time VM gets higher priority compared to the general purpose VMs. Between the

real-time VMs, a preemptive fixed priority algorithm is used. However, a real-time VM is limited to one real-time task to guarantee correct timing behaviour. Masrur et al. [16] has lifted that limitation. In this case the VMs are limited to one VCPU and not allowed to migrate between PCPUs.

Kernel-based Virtual Machine (KVM) is a virtualization infrastructure built into in the Linux kernel [17]. It enables Linux to be used as a hypervisor. The combination of KVM and real-time behaviour is an popular topic in research. Multiple papers have been published about real-time scheduling [18, 19, 20] and latency in VMs [21]. However, without adjusting the scheduling algorithms or using real-time extensions, real-time behaviour is hard to realize. Secondly, most research about these topics focusses on general-purpose systems. KairosVM [22] is a latency-bounded real-time extension to the Linux KVM module. It focusses on the real-time scheduling extension. The extension supplies for the communication between the guest-OS and KVM without modifying the guest OS. KairosVM is evaluated on a x86 architecture multi-core platform.

The current embedded hypervisors do not support real-time scheduling on multi-core systems with VCPU sharing between VMs. With our new embedded hypervisor, we try to fill this gap and the gap between the schedulability analysis techniques for virtualization and the implementation of real-time virtualization on embedded systems. Both sides are not compatible and thereby tend to fail to analyse real-time applications from the schedulability analysis until actual deployment on the target platform.

3. Background

In the virtualization technology two scheduling levels exists, together both levels create a hierarchical scheduling structure. In order to guarantee real-time scheduling, both levels must support real-time scheduling. To analyse the schedulability of such a system, compositional schedulability analysis can be used. Before modifying the existing Xvisor hypervisor, we give an overview of its architecture. More specifically this section will focus on: the scheduling structure in virtualization, the compositional scheduling analysis and the architecture of Xvisor.

3.1. Scheduling in virtualization

Virtualization introduces an hierarchical scheduling structure with two levels. In order to schedule real-time tasks, both levels must support real-time scheduling. The physical CPU (PCPU) of the hardware platform is presented to the VMs as a VCPU. These VCPUs are scheduled by the hypervisor to be executed on the PCPUs. Each VM operating system includes a task scheduler that schedules the real-time workload on the VCPUs. The two levels of scheduling, system-level scheduler and task-level scheduler, define an hierarchical scheduling structure, see Figure 1.

In order to analyse such an hierarchical structure for schedulability, various analysis techniques exist. The common one is compositional schedulability

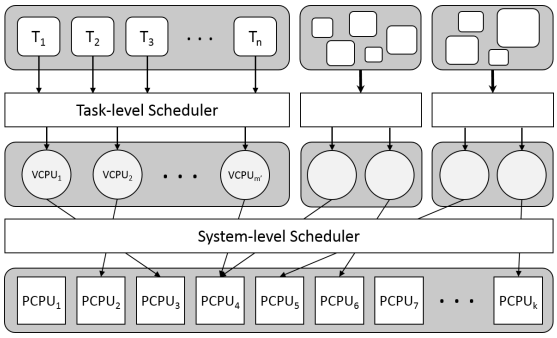

Figure 1: Visual representation of a hierarchical scheduling structure on a multi-core platform.

analysis (CSA). Component-based design decomposes a system into components allowing the reduction of a single complex system design problem into multiple single system designs. Each component is analysed separately and represented by a component interface that abstracts and hides the internal complexity. These interfaces are used for composing the system for the system analysis [23].

Real-time systems can benefit from component-based design, if components are assembled without violating timing properties. This requires that the component interfaces include all the timing information. Component-based real-time systems often involve hierarchical scheduling frameworks that support resource sharing among components under different scheduling algorithms. These frameworks can be represented as a tree of nodes, where each node denotes a component comprising a real-time workload and a scheduling policy [24]. In other words, the component-based design can be used to perform schedulability analysis for hierarchical structures.

3.2. Xvisor

The Xvisor hypervisor [3] has been developed for embedded target platforms via a modular design. It is a monolithic kernel, designed for one purpose: embedded virtualization. All the functionality of the hypervisor runs in root mode, whereas the VMs execute in user mode. Because Xvisor is designed for virtualization, only the necessary functionality has been added to the hypervisor. This results in lightweight hypervisor with little overhead and a small memory footprint. It supports full virtualization (hardware-assist) and para-virtualization guest systems.

Figure 2 displays the architecture of Xvisor. All core components (CPU virtualization, guest IO emulation, Management Terminal) are located in one software layer at the highest privilege mode. In Xvisor, the guest OS configuration is given in a device tree structure (DTS) [25]. The DTS contains a data structure that describes the hardware configuration of the guest system. It contains CPU information, memory banks, buses and other device information. The OS is able to parse the DTS at boot time in order to configure the

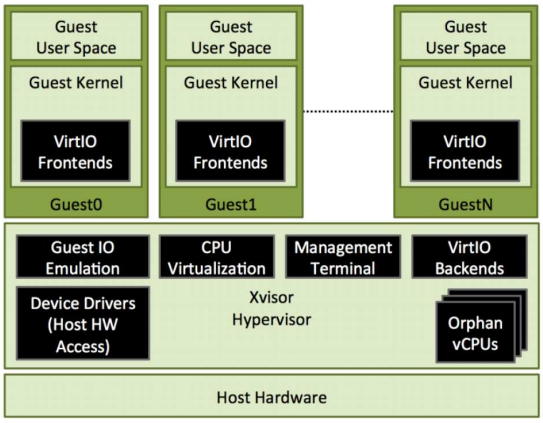

Figure 2: The Xvisor architecture [3]

kernel and load the right device drivers. Figure 3 displays an example of the VCPUs configuration of a guest OS. Xvisor contains two types of VCPUs: normal and orphan VCPUs. The first are the guest OS's VCPUs. The latter are not connected to a guest OS and are used for background processing such as the device drivers and management functions. In Xvisor each VCPU has a priority level; a higher priority will be scheduled before the lower priority. VCPUs at the same priority level are scheduled according to the scheduling algorithm. Those priorities can also be changed at the DTS.

```
vcpus {
    vcpu0 {
        device_type = "vcpu";
        compatible = "armv7a,cortex-a15";
        start_pc = <0x00000000>;
        gentimer_virt_irq = <27>;
        gentimer_phys_irq = <30>;
        time_slice = <15000000>;
        periodicity = <20000000>;
        deadline = <20000000>;
        priority = <3>;
    };
    vcpu1 {
        device_type = "vcpu";
        compatible = "armv7a,cortex-a15";
        start_pc = <0x00000000>;
        gentimer_virt_irq = <27>;
        gentimer_phys_irq = <30>;
        time_slice = <13000000>;
        periodicity = <20000000>;
        deadline = <20000000>;
        priority = <3>;
    };
};
```

Figure 3: DTS example of guest OS with two VCPUs

3.3. Xvisor scheduler

The default scheduler in Xvisor has a partitioned scheduling mechanism with a load balancer to spread the VCPUs over the PCPUs to balance utilization across the PCPUs. However, since version 0.2.9 this load balancer has become optional. When a multi-core hardware platform is used to host Xvisor, the VCPUs (orphans and normal) are assigned to a PCPU. A new VCPU is assigned to PCPU with the lowest utilization. This assignment is static and will not change during run-time unless the user intervenes. For each PCPU, an *idle* VCPU is created. An idle VCPU has the lowest priority and is only scheduled when there is no other candidate VCPU in the run queue of the associated PCPU. The default scheduling algorithm, used on each PCPU, is a priority round-robin [26]. Because of the partitioned scheduling structure, a run queue is created for each PCPU and priority level. The scheduler loops over the run queues of PCPU's. When a VCPU is available in a queue, the scheduler selects this VCPU and schedules the VCPU for 10 ms. When the timer expires or the VCPU changes state, the VCPU is inserted at the back of the appropriate run queue and the same steps are executed again. When a VCPU becomes available and is inserted into a run queue, the scheduler checks if a preemption is necessary: when the new VCPU has a higher priority compared to the running VCPU. Xvisor provides also for each VCPU a PCPU affinity. It indicates on which PCPU the VCPU is allowed to execute.

Xvisor splits the scheduler into two parts: the scheduling mechanism and the functionality related to the scheduling algorithm and run queues. The first part initializes the parameters for the PCPUs, creates the idle VCPUs, switches VCPUs for execution and changes states of VCPUs if required. This part also interacts with other components of Xvisor and is the only part that can access the functionality of the scheduling algorithm. The second part includes the scheduling algorithm and the run queues because these are only accessed in this part and their order is dependent on the scheduling algorithm. Adding or selecting a VCPU from the run queue depends on the scheduling algorithm. This separation makes it possible to switch between scheduling algorithms from the configuration of Xvisor. It also facilitates the introduction of new scheduling algorithms.

4. Analysis methodology

The goal of this analysis methodology is to reduce the candidate scheduling techniques to analyse on the target platform. At the highest levels a formal analysis calculates the schedulability of candidate techniques using an abstract system and application description. This often leads to a strong reduction in the number of candidates. At the second level simulation is used to evaluate deadline behaviour of the candidates. Again a fraction of candidates are eliminated. At the lowest level, at the final stages of the design process, scheduling techniques implemented on the target platform are analysed for the given applications. The analysis methodology selects a scheduling technique based on:

1) whether it meets all the deadlines on the available resources and 2) the minimal PCPU and VCPU usage. If a scheduling technique requires more resources than the available resources on the target platform, this technique is excluded as candidate. Otherwise, if multiple scheduling techniques require less resources than the available resources, we first compare the number PCPUs. When two scheduling techniques need to same minimum number of PCPUs, we compare on the load of the VCPUs. This analysis has to be done for each application context separately. The context includes the task set, scheduling levels, the target platform and other environmental parameters. The real-time hypervisor, Xvisor-RT, fits at the lowest abstraction level to analyse the application at the target platform.

4.1. The Formal Analysis

At the highest abstraction level, the schedulability of a scheduling technique can be formally proven by the mathematical model. The formal proof compares the demand of computing resources for the given task set and a certain scheduling technique, with the supply of computing resources from the available the target platform. There are two major methods to calculate the demand of a task set. The first method is based on schedulability tests. These tests compare the utilization of the task set with the maximum utilization of the scheduling algorithm. These tests are commonly used for uniprocessor scheduling algorithms.

The second method is the parameter-based schedulability test. This test uses all the parameters of the tasks (execution time, period and deadline) to calculate the exact demand of the task set. This test is used if there does not exist a schedulability test for the scheduling algorithm or the result of the schedulability test is not decisive. The parameter-based method compares the worst-case demand of the task set, with the worst-case supply which the system delivers in a specific time interval. This method gives us more specific results, however the method is task-set-depended. For a hierarchical scheduling structure, the schedulability tests are non existing. The analysis methodology uses parameter-based schedulability tests to analyse each scheduling algorithm combination.

To apply the formal analysis for the set of scheduling algorithm combinations, we integrated and adapted the Compositional Analysis of Real-Time Systems (CARTS) [27] tool for calculating the parameters of the PCPUs and VCPUs. CARTS implemented the CSA formulas from Easwaran and Lee [28]. We added the option to select the optimal period for the VCPUs to minimize the gap between the task set utilization of the VM and the utilization of the VCPUs. Depending on the scheduling technique the period can be set per VCPU or per VM. The output of CARTS contains the interfaces and VCPU parameters, calculated with CSA theory, which we use to set the VCPU parameters for each VM in the lower analysis levels.

4.2. Simulation-based analysis

At the second abstraction level, the schedulability is proven based on simulation of the scheduling technique. This requires a system model and task set model to analyse the behaviour of a scheduling algorithm. At every point in time, the state of a job in the task set can be examined. In this manner, timing information about each job of each task reveals jobs which do not meet their deadline. Based on these data, a task set is deemed schedulable if all of its jobs meet their deadline.

The behaviour of the simulation-based analysis is more related with the deployment-based analysis compared to behaviour of the formal analysis. This gives us the opportunity to modify the simulation parameters to be able to create an almost identical setting compared to the deployment-based analysis. This results into a more trustworthy simulation-based analysis and thereby it will be possible to make a more thoughtful choice for the optimal scheduling algorithm at this abstraction level.

For the simulation-based analysis, we modified the hsSim simulation software of [29]. hsSim is an extensible interoperable object-oriented hierarchical scheduling simulator. The simulator has a number of scheduling algorithms, for single and multi-core processors. They can be used at each level to schedule tasks or VCPUs. The major changes that we have introduced are:

- **Compatibility with the CSA theory:** the ability to set a period Π and a budget Θ calculated by the CARTS tool for each VCPU. These parameters can be set for each VCPU individual, which makes it possible to simulate the calculated interfaces from the formal schedulability proof.

- **Partitioned Scheduling Algorithms:** we implemented the partitioned scheduling technique in hsSim to be able to select the partitioned EDF and partitioned deadline monotonic (DM) scheduling algorithm.

- **Compatible output interface:** to evaluate the results of the simulation, the timing information is logged. A set of different logging formats are provided by hsSim; we added a new format to make it possible to analyse the data from each level using the same tools.

Together these modifications make hsSim usable in our analysis methodology.

4.3. Deployment-based analysis

The behaviour of the scheduling algorithms at this level is the most accurate to perform the schedulability experiments. At this level it is possible to analyse the scheduling algorithm on a variety of evaluation criteria. These evaluations,

however, are platform specific. Which means that for every hardware platform 1) a RTOS must exist which support the selected hardware platform, and 2) the scheduling algorithms must be implemented in the RTOS. This makes it hard to evaluate scheduling algorithms on a number of different hardware platforms. At this analysis level, and due to the fact the applications have hard real-time constrains, a scheduling algorithm or a combination of scheduling algorithms is only successful if each job of the task set meet its deadline at all time, and this on the computational resources calculated by the formal analysis level.

For the implementation of this analysis level, we modified Xvisor to our needs. These modifications are elaborated in Section 5

5. Xvisor-RT

In this section we discuss the modifications and enhancements, aimed at support for real-time scheduling and the CSA theory, from Xvisor to Xvisor-RT. We used Xvisor as starting point because of its small code base and wide range of the supported embedded hardware platforms. However (see section 3.2) the default scheduler and scheduling algorithm do not support real-time scheduling. So, new scheduling algorithms were added for real-time scheduling. The partitioned scheduling approach and the round-robin scheduling algorithm make Xvisor incompatible with the CSA theory. In order to be compatible, the Xvisor partitioned scheduler must be replaced by a global scheduler with one set of run queues, one for each priority, for all PCPUs.

5.1. Run queues

Xvisor creates a set of run queues for each PCPU and assigns a VCPU statically to a PCPU. To enable global scheduling algorithms, a global structure is needed. We introduce a single set of run queues for all PCPUs. When a VCPU is added to a run queue, each PCPU can select it to execute. Besides sharing run queues in each PCPU, the PCPUs must share a synchronization object (spinlock) so that only a single PCPU at a time can make changes at the run queues.

Because of the shared run queues, a concurrency problem was created in the scheduling mechanism: the functions of the scheduling algorithm, which are called from the scheduler, are already controlled by spinlocks. However, the sequence of functions of choosing between picking a new VCPU or keeping the current VCPU needed to be altered. In Xvisor, the current VCPU is added again in the run queue. If it is selected again, no context switch is necessary. However, with a global run queue, when the current VCPU is added to the run queue, it could be selected by another PCPU and cause a system failure. In Xvisor-RT, the current VCPU is passed as an argument to the scheduling algorithm. This makes it possible to check if the current VCPU still has the highest priority, without having it in the run queue. If, after the scheduling decision, the current VCPU is not selected to be executed, the VCPU is added to the run queue again after the context switch.

5.2. Scheduling algorithms

Because of the above changes, the implementation of the current scheduling algorithm had to be replaced. We implemented the Earliest Deadline First (EDF) and the Deadline Monotonic (DM) algorithm, because they are rather straightforward to implement. However, we decided to keep the priority levels of the original Xvisor scheduler to separate the normal VCPUs from lower and higher priority orphan VCPUs when needed. Each VCPU has a set of parameters in the DTS that are needed for the scheduling algorithms: a period, a deadline and a time slice (see Figure 3). These parameters are static and can be accessed by both parts of the scheduler. The time slide parameter of a VCPU in the DTS is the budget of the VCPU, calculated in the interface of the VM in the CSA theory. The parameters that change over time, are stored in the additional VCPU information and can only be accessed by the second part of the scheduler. In order to implement the EDF algorithm, we added three parameters as dynamic parameters: the absolute deadline, a timestamp of the VCPU's last start and the remaining budget. The absolute deadline of an instance of a VCPU is the moment in time when the budget must be consumed to complete without a deadline miss. In order to keep track when the VCPU is selected to be scheduled, the timestamp of that moment is saved in the *laststart* parameter of the VCPU. When the VCPU is added back to the run queue the budget gets consumed with the time difference between the current time and the timestamp in the *laststart* parameter. The remaining budget is saved at the respectively parameter of the VCPU.

In the *vmm_schedalgo_rq_dequeue* function, the budget of the current VCPU is consumed. Next, we check whether the deadline information on all VCPUs in the run queue is up to date. If the absolute deadline is in the past, the VCPU is removed from the run queue, a new deadline is calculated and the VCPU is added to the run queue again. For the EDF algorithm, a run queue is sorted based on the absolute deadline of the VCPUs with a common priority. Thus the first VCPU of a run queue will have the shortest deadline. A VCPU is scheduled on a PCPU for a maximum time quantum. When the PCPU's timer expires, the next VCPU is selected. Each VCPU is compared with the current VCPU. Because each priority has its own run queue, we iterate from the highest priority run queue to the lowest priority run queue. Following parameters are checked in the given order (cur_VCPU is the current VCPU and iter_VCPU is the next VCPU in the run queue):

1. **The affinity of iter_VCPU:** if iter_VCPU is not allowed to run on the PCPU, it can not be selected.
2. **The remaining budget of iter_VCPU:** if iter_VCPU has no budget left, it can not be selected.
3. **The priority of iter_VCPU:**
   (a) If the priority of iter_VCPU is higher than the priority of cur_VCPU, iter_VCPU is the next VCPU.
   (b) If the priority of iter_VCPU is equal to the priority of cur_VCPU and the current has no budget left, iter_VCPU is the next VCPU.

(c) If the priority of iter_VCPU is equal to the priority of cur_VCPU and iter_VCPU has a shorter absolute deadline, iter_VCPU is the next VCPU. If not, cur_VCPU is the next VCPU.
(d) If the priority of iter_VCPU is lower than the priority of cur_VCPU and cur_VCPU has no budget left, iter_VCPU is the next VCPU. If cur_VCPU has budget left, it is the next VCPU.

4. When there are no candidate VCPUs, including cur_VCPU, the next VCPU is the idle_VCPU.

If the next VCPU is the current VCPU, no context switch is necessary and the scheduler will schedule the current VCPU again for 1 ms. Otherwise, the context of the current VCPU is saved, the new VCPU is switched with the current VCPU and the new VCPU is scheduled. There is one exception to this: when a PCPU is booted and has no VCPU running. At the first selection of the next VCPU for a PCPU, the scheduler just has to select the highest priority VCPU in the run queue.

We implemented the global scheduler together with the EDF, DM, which automatically includes the Rate-Monotonic (RM) algorithm[2] and the McNaughton algorithm [30, 31, 32].

6. Experiments

With Xvisor-RT we can analyse scheduling algorithms on embedded systems. To evaluate the Xvisor-RT real-time scheduling performance we measured the system for overhead in terms of context switches and PCPU migrations. As a second experiment, we analyse a set of scheduling algorithms or combinations of scheduling algorithms for two applications.

6.1. Experimental setup

We performed the experiments on a Raspberry Pi 2 Model B (RPI2). It is an embedded system with four ARM Cortex-A7 cores (PCPUs), each clocked at a frequency of 900 MHz, and 1 GB of RAM. We installed Xvisor-RT on the SD-card and booted the hypervisor with the U-Boot bootloader [33]. We selected a 1 ms interval between each scheduler decision for each PCPU. A higher quantum would result into higher delays between VCPU events and scheduling decision. A smaller quantum causes a bigger overhead because the scheduler is called at a higher frequency. As guest OS, Xvisor recommends a Linux kernel in combination with a busybox ramdisk as file system to provide a minimal set of executables [34]. We patched the linux kernel, version 4.1, with the LITMUS^{RT} patch [35]. LITMUS^{RT} provides the ability to do real-time scheduling in a Linux OS. Beside the real-time patch, LITMUS^{RT} comes with a set of real-time

[2]The gEDF, gRM and gDM can be easily transformed towards the partitioned variant by setting the affinity of the each VCPU to only one of the PCPUs

scheduling algorithms, a library (liblitmus) to create periodic tasks (without a workload) and tracing tools to trace the periodic tasks.

We used CARTS to calculate for each VM the number of VCPUs and the budget and period of each VCPU [27]. Based on the output of CARTS, the DTS file is generated with the correct number of VCPUs and parameters. The application and scheduling algorithm are simulated by the modified version of hsSim [29].

To analyse an application with Xvisor-RT to find the optimal scheduling algorithms, tasks are needed to create a system load. In both experiments we use periodic tasks. Such a task has a task model τ with three parameters: the worst-case execution time (WCET) C, the period T and the relative deadline D. Each task has a deadline D which is equal to its period T. To generate the tasks and the load of a task, we use a taskset-generator (reference removed for authors). This taskset-generator creates for each task a sequence of benchmark programs from the TACLeBench benchmark suite to generate computational load for the processors [36].

6.2. Scheduling overhead

Each scheduler consumes resources to calculate the next task to run. This translates into an execution period during which the CPU is not available to schedule tasks of the application. This is called the scheduling overhead. Xvisor-RT is a quantum-based scheduler; every quantum the scheduler checks if the current VCPU has not consumed all of its budget and it is still the task with the highest priority. If not, the scheduler selects the next task which is available. Each quantum the scheduler needs some CPU resources and causes an overhead. For Xvisor-RT we measured this overhead by measuring difference between the timestamp when Xvisor-RT enters the scheduler and the timestamp when it exits. This is measured during the execution of a workload to create a realistic situation where multiple VCPUs are in the run queue. The scheduling decisions resulted in an overhead of between $2.5 - 3\%$.

To include this overhead into the analysis of the scheduling algorithm at the formal and simulation-based level, a periodic task is added to the taskset of each VM. The period of this task is equal to the quantum and the execution time is equal to the overhead of the scheduler.

6.3. Experiment

In this experiment, we used the analysis methodology with Xvisor-RT on two applications to determine the optimal scheduling technique for the respective application. Each application has a real-time workload divided over three virtual machines. For each application context we calculated the interfaces via CARTS. We ran CARTS for each combination of scheduling algorithms we have available at all analysis levels:

- **Task-level scheduler**: Global EDF, Partitioned EDF and Partitioned DM.

- **System-level scheduler**: Global EDF, Global DM, Mc- Naughton, Partitioned EDF and Partitioned DM.

The first application has a taskset containing 32 tasks and a total utilization of 2.88 ($U = \sum_{i=1}^{n} \frac{C_i}{T_i}$, with n the number of tasks, C the WCET and T the period of the task). The taskset is spread over three VMs with an utilization and number of tasks of respectively 0.95 with 14 tasks, 0.55 with 6 tasks and 1.38 with 12 tasks. The second application has a taskset containing 24 tasks and a total utilization of 3.80. The taskset is spread over three VMs with a utilization and number of tasks of respectively 1.20 with 8 tasks, 1.21 with 9 tasks and 1.39 with 7 tasks. The first application has an unbalanced workload across the VMs which will show to advantage of VCPU sharing to schedule the workload. This will have an impact on the system-level scheduler. In the second application the workload of the VMs is more balanced. However the workload inside a VM is unbalanced. This will challenge the task-level scheduler to handle this type of workload.

Table 1 & 2 shows the load of the VCPUs and number of PCPUs for each combination of scheduling algorithms needed to schedule the applications. If the number PCPUs exceed to maximum number of 4 PCPUs, the application is not schedulable with the selected combination of scheduling algorithms.

Table 1: Load of the VCPUs and PCPUs for application context 1 for each combination of scheduling algorithms

		Task-level Scheduler		
		G-EDF	P-EDF	P-DM
System-level Scheduler	G-EDF	*U = 2.94* *PCPUs = 4*	*U = 2.95* *PCPUs = 4*	*U = 4.41* *PCPUs = 6*
	G-DM	*U = 2.94* *PCPUs = 4*	*U = 2.95* *PCPUs = 4*	*U = 4.41* *PCPUs = 6*
	McNaughton	**U = 2.97** **PCPUs = 3**	n.a.	n.a.
	P-EDF	*U = 2.94* *PCPUs = 4*	*U = 2.95* *PCPUs = 4*	*U = 4.41* *PCPUs = 5*
	P-DM	*U = 2.94* *PCPUs = 4*	*U = 2.95* *PCPUs = 4*	*U = 4.41* *PCPUs = 5*

For the first application the combination of the McNaughton and the global EDF scheduling algorithms is the most promising. The application can be scheduled on 3 PCPUs by using this combination. Most of the other combinations can be scheduled on the platform. However, with the partitioned DM algorithm as system-level scheduler, the application is not schedulable on our hardware platform. We selected the McNaughton & GEDF and the combinations which requires 4 PCPUs to analyse at the next analysis level, the simulation-based analysis.

Table 2: Load of the VCPUs and PCPUs for application context 1 for each combination of scheduling algorithms

System-level Scheduler \ Task-level Scheduler	G-EDF	P-EDF	P-DM
G-EDF	*U = 4.83* *PCPUs = 6*	*U = 3.85* *PCPUs = 6*	*U = 3.84* *PCPUs = 6*
G-DM	*U = 4.83* *PCPUs = 6*	*U = 3.85* *PCPUs = 6*	*U = 3.84* *PCPUs = 6*
McNaughton	*U = 5.1* *PCPUs = 6*	n.a.	n.a.
P-EDF	*U = 4.83* *PCPUs = 6*	**U = 3.85** **PCPUs = 4**	*U = 3.84* *PCPUs = 6*
P-DM	*U = 4.83* *PCPUs = 6*	**U = 3.85** **PCPUs = 5**	*U = 3.84* *PCPUs = 6*

Table 2 shows that for application 2 the combination of the partitioned EDF algorithm and partitioned EDF algorithm is the most promising one. It is the only combination which can schedule the application on 4 PCPUs. The other combinations of scheduling algorithms need 5 or 6 PCPUs to schedule the application. We selected P-EDF & P-EDF combination together with the combinations which require 5 PCPUs to schedule the application to analyse at simulation-based analysis.

Figures 4a and 4b display the results of the simulation-based analysis of the selected combinations of scheduling algorithms for both applications. For application 1 the McNaughton & GEDF combination has no deadline misses on 3 PCPUs as proven in the formal analysis. The other combinations have deadlines misses when scheduled on 3 PCPUs and have no deadline misses on 4 PCPUs. For the second application, we simulated the selected combinations of scheduling algorithms with maximum 4 PCPUs. It is clear that only the P-EDF & P-EDF combination can schedule the application on the available resources. The other selected combinations have deadline misses when using all resources and can not be a candidate for the selection of scheduling algorithm for this application. For both applications, the simulation-based analysis validates the results of the formal analysis.

At the lowest abstraction level, the deployment-based analysis, we deployed both applications on the target platform using Xvisor-RT. For both applications we analysed the most promising combination of scheduling algorithms and the combination of scheduling algorithms which has the lowest number of deadline misses. For the first application we analysed the McNaughton & P-EDF and the P-DM & P-EDF combinations; for the second application the P-EDF & P-EDF and the P-DM & P-EDF combinations.

For each application, we generated the executable tasks to be deployed on the VMs. We configured each VCPU with their period and budget. We executed the tasks for 30 seconds on the target platform. Figures 4c and 4d display the

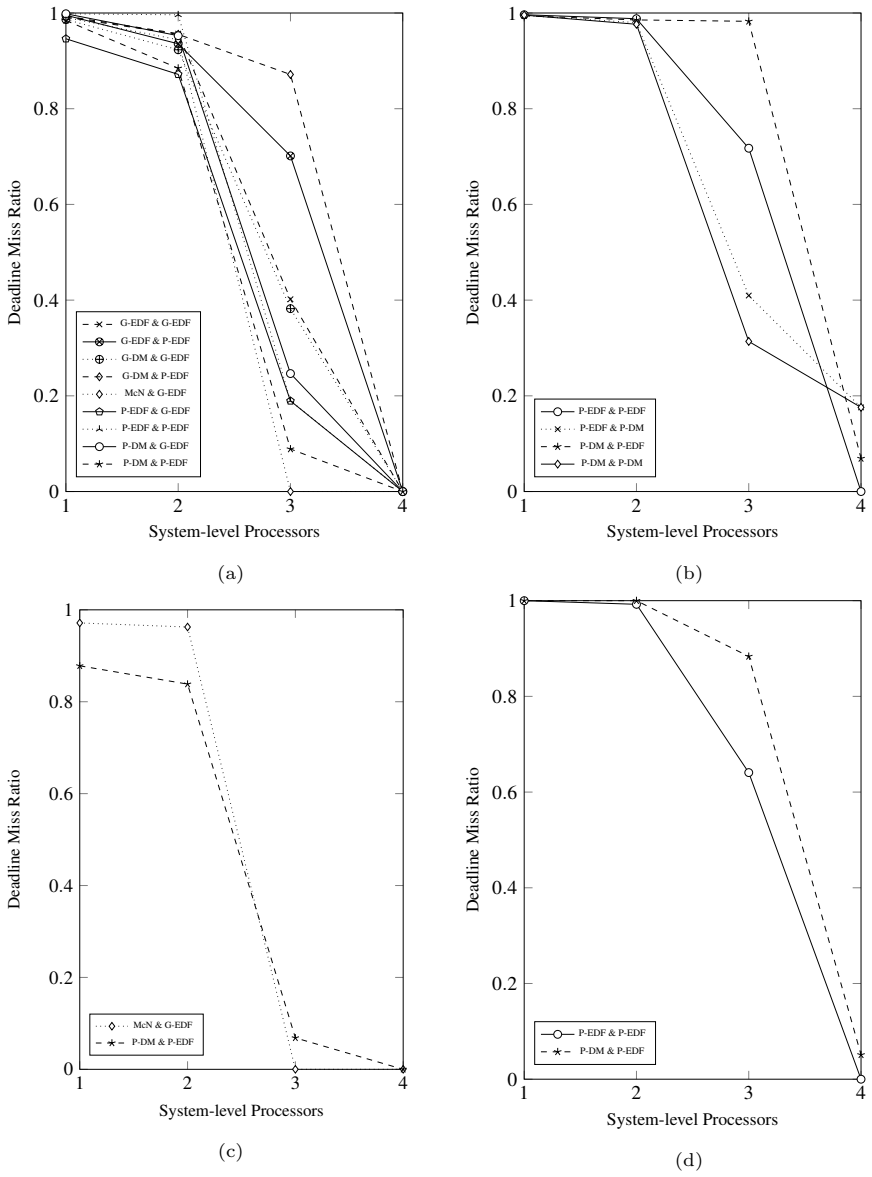

Figure 4: In Figures a & b the results are displayed from the simulation-based analysis for respectively application 1 & 2. Figures c & d display the results of the analysis with Xvisor-RT for the selected scheduling algorithms.

results of the experiments on Xvisor-RT. For both applications, the optimal combination has no deadline misses with the calculated interfaces of the formal analysis. Notice that for all scheduling algorithm combinations the deadline misses are lower compared to the simulation-based analysis. The reason is that the task execution time is shorter than the WCET of the task.

7. Conclusion

In this paper we presented Xvisor-RT, a real-time hypervisor for multi-core embedded systems. We modified the existing embedded hypervisor Xvisor to support the compositional schedulability analysis theory for global and partitioned scheduling algorithms. Xvisor already supports a wide variety of architectures and platforms, which Xvisor-RT inherits. We also added a set of scheduling algorithms in Xvisor-RT. Those algorithms can be selected to schedule applications with real-time tasks. We included Xvisor-RT in our analysis methodology to analyse real-time applications. This makes it possible to analyse an application for embedded systems from the highest abstraction level to the lowest abstraction level, compare the results between each abstraction level and select the combinations of scheduling algorithms for the next abstraction level.

In the experiments we focussed the two objectives of this paper. Firstly, we confirmed that Xvisor-RT can schedule a real-time workload by analysing the schedulability of two applications with the calculated VCPU parameters. Both applications had no deadline misses during the experiments. Secondly, we analysed both applications with the analysis methodology using Xvisor-RT. The consecutive steps in the analysis pare down the set of candidate scheduling algorithms. Consequently, only a few candidate combinations of scheduling algorithms must be analysed at the lowest abstraction level. This reduces the time and cost of the design process.

Acknowledgements

This study was funded by the Agency for Innovation by Science and Technology in Flanders (IWT). (Grant number: 131788)

References

[1] M. Hermann, T. Pentek, B. Otto, Design principles for industrie 4.0 scenarios, in: Proceedings of the Annual Hawaii International Conference on System Sciences, 2016, pp. 3928–3937.

[2] M. Satyanarayanan, The emergence of edge computing, Computer 50 (2017) 30–39. doi:10.1109/MC.2017.9.

[3] A. Patel, M. Daftedar, M. Shalan, M. W. El-Kharashi, Embedded hypervisor xvisor: A comparative analysis, in: Proceedings - 23rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, 2015, pp. 682–691.

[4] S. Oikawa, M. Ito, T. Nakajima, Linux / RTOS Hybrid Operating Environment on Gandalf Virtual Machine Monitor, Lecture Notes in Computer Science 4096 (2006) 287–296.

[5] W. Kanda, Y. Yumura, Y. Kinebuchi, K. Makijima, T. Nakajima, SPUMONE: Lightweight CPU virtualization layer for embedded systems, in: Proceedings of The 5th International Conference on Embedded and Ubiquitous Computing, EUC, volume 1, 2008, pp. 144–151.

[6] K. Sakamura, ITRON, 2013. URL: `http://tronweb.super-nova.co.jp/`.

[7] D. Baldin, T. Kerstan, Proteus, a hybrid virtualization platform for embedded systems, in: International Embedded Systems Symposium, 2009, pp. 185–194.

[8] K. Gilles, S. Groesbrink, D. Baldin, T. Kerstan, Proteus hypervisor: Full virtualization and Paravirtualization for multi-core embedded systems, IFIP Advances in Information and Communication Technology 403 (2013) 293–305.

[9] S. Ghaisas, G. Karmakar, D. Shenai, S. Tirodkar, K. Ramamritham, SParK: Safety Partition Kernel for integrated real-time systems, Lecture Notes in Computer Science 6462 LNCS (2010) 159–174.

[10] U. Steinberg, B. Kauer, NOVA: A Microhypervisor-Based Secure Virtualization Architecture, in: Proceedings of the 5th European conference on Computer systems, ACM, 2010, p. 209.

[11] G. Heiser, The OKL4 Microvisor : Convergence Point of Microkernels and Hypervisors, 2010.

[12] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, A. Warfield, Xen and the art of virtualization, ACM SIGOPS Operating Systems Review 37 (2003) 164.

[13] S. Xi, J. Wilson, C. Lu, C. Gill, RT-Xen: Towards real-time hypervisor scheduling in Xen, in: 2011 Proceedings of the Ninth ACM International Conference on Embedded Software (EMSOFT), Department of Computer Science and Engineering, Washington University in St. Louis, USA, Ieee, 2011, pp. 39–48.

[14] S. Xi, C. Lu, C. Gill, M. Xu, L. T. X. Phan, I. Lee, O. Sokolsky, Global Real-Time Multi-Core Virtual Machine Scheduling in Xen, in: Technical Report, Washington University Technical Report, 2013.

[15] A. Masrur, S. Drössler, T. Pfeuffer, S. Chakraborty, VM-Based real-time services for automotive control applications, in: Proceedings - 16th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, RTCSA, 2010, pp. 218–223.

[16] A. Masrur, T. Pfeuffer, M. Geier, S. Drossler, S. Chakraborty, S. Drössler, S. Chakraborty, S. Drossler, S. Chakraborty, Designing VM schedulers for embedded real-time applications, in: Proceedings on Hardware/Software Codesign and System Synthesis (CODES+ISSS), 2011, pp. 29–38.

[17] A. Kivity, U. Lublin, A. Liguori, Y. Kamay, D. Laor, kvm: the Linux virtual machine monitor, in: Proceedings of the Linux Symposium, volume 1, 2007, pp. 225–230.

[18] F. Checconi, T. Cucinotta, D. Faggioli, G. Lipari, Hierarchical multiprocessor CPU reservations for the linux kernel, Ospert 2009 (2009) 1–8. URL: http://www.artist-embedded.org/docs/Events/2009/OSPERT/Proceedings-PreWorkshop.pdf{#}page=15.

[19] T. Cucinotta, G. Anastasi, L. Abeni, Respecting Temporal Constraints in Virtualised Services., COMPSAC (2) (2009). URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.149.4623{&}rep=rep1{&}type=pdf.

[20] T. Cucinotta, D. Giani, D. Faggioli, F. Checconi, Providing performance guarantees to virtual machines using real-time scheduling, Euro-Par 2010 Parallel Processing Workshops (2011) 657—-664.

[21] J. Zhang, K. Chen, B. Zuo, R. Ma, Y. Dong, H. Guan, Performance Analysis Towards a KVM-Based Embedded Real-Time Virtualization Architecture, in: Computer Sciences and Convergence Information Technology (ICCIT), 2010, pp. 421–426.

[22] K. Burns, A. Barbalace, V. Legout, B. Ravindran, KairosVM: Deterministic introspection for real-time virtual machine hierarchical scheduling, in: 19th IEEE International Conference on Emerging Technologies and Factory Automation, ETFA, 2014.

[23] J. L. Lorente, G. Lipari, E. Bini, A hierarchical scheduling model for component-based real-time systems, 20th International Parallel and Distributed Processing Symposium, IPDPS 2006 (2006).

[24] A. Easwaran, I. Lee, I. Shin, O. Sokolsky, Compositional Schedulability Analysis of Hierarchical Real-Time Systems, in: 10th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC), Ieee, 2007, pp. 274–281.

[25] G. Likely, J. Boyer, A Symphony of Flavours: Using the device tree to describe embedded hardware, in: Embedded Linux Conference, volume 2, 2008, pp. 27–37.

[26] C. Faisstnauer, D. Schmalstieg, W. Purgathofer, Priority round-robin scheduling for very large virtual environments, Virtual Reality, 2000. Proceedings. IEEE (2000) 135–142.

[27] L. T. X. Phan, J. Lee, A. Easwaran, V. Ramaswamy, S. Chen, I. Lee, O. Sokolsky, CARTS: a tool for compositional analysis of real-time systems, ACM SIGBED Review 8 (2011) 62–63.

[28] A. Easwaran, I. Lee, Compositional schedulability analysis for cyber-physical systems, ACM SIGBED Review 5 (2008) 1–2.

[29] J. P. Craveiro, R. O. Silveira, J. Rufino, hsSim: an Extensible Interoperable Object-Oriented n-Level Hierarchical Scheduling Simulator, in: 3rd International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems, 2012.

[30] C. L. Liu, J. W. Layland, Scheduling Algorithms for Multiprogramming in a Hard- Real-Time Environment, Journal of the Association for Computing Machinery 20 (1973) 46–61.

[31] N. C. Audsley, A. Burns, M. Richardson, A. Wellings, Deadline-Monotonic Scheduling, University of York, Department of Computer Science, York, 1990.

[32] R. McNaughton, Scheduling with Deadlines and Loss Functions, Management Science 6 (1959) 1–12.

[33] W. Denk, U-Boot: the universal boot loader, 2009. URL: `https://www.denx.de/wiki/U-Boot`.

[34] E. Andersen, BusyBox: The Swiss Army Knife of Embedded Linux, 2008. URL: `https://busybox.net/`.

[35] J. M. Calandrino, H. Leontyev, A. Block, U. C. Devi, J. H. Anderson, LITMUS-RT : A Testbed for Empirically Comparing Real-Time Multiprocessor Schedulers, in: Real-Time Systems Symposium (RTSS), 2006, pp. 111–126.

[36] H. Falk, S. Altmeyer, P. Hellinckx, B. Lisper, W. Puffitsch, C. Rochange, R. B. S. Schoeberl, P. Waegemann, S. Wegener, TACLeBench: A Benchmark Collection to Support Worst-Case Execution Time Research, in: 16th International Workshop on Worst-Case Execution Time Analysis (WCET 2016), volume i, 2016, p. 10.