The Journal of Systems and Software 216 (2024) 112147

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

Temporal isolation assessment in virtualized safety-critical mixed-criticality systems: A case study on Xen hypervisor[☆]

Marcello Cinque, Luigi De Simone[*], Daniele Ottaviano

DIETI - Università degli Studi di Napoli Federico II, Via Claudio 21, 80125 Napoli, Italy

ARTICLE INFO

Keywords:
Virtualization
Temporal isolation
Dependability
Xen
ARM
Software certification
Railway
EN50128

ABSTRACT

Today, we are witnessing the increasing use of the cloud and virtualization technologies, which are a prominent way for the industry to develop mixed-criticality systems (MCSs) and reduce *SWaP-C* factors (size, weight, power, and cost) by flexibly consolidating multiple critical and non-critical software on the same System-on-a-Chip (SoC). Unfortunately, using virtualization leads to several issues in assessing isolation aspects, especially temporal behaviors, which must be evaluated due to safety-related standards (e.g., EN50128 in the railway domain). This study proposes a systematic approach for verifying temporal isolation properties in virtualized MCSs to characterize and mitigate timing failures, which is a fundamental aspect of dependability. In particular, as proof of the effectiveness of our proposal, we exploited the real-time flavor of Xen hypervisor used to deploy a virtualized *2 out of 2*-based MCS scenario provided in the framework of an academic-industrial partnership, in the context of the railway domain. The results point out that virtualization overhead must be carefully tuned in a real industrial scenario according to the several features provided by a specific hypervisor solution. Further, we identify a set of directions toward employing virtualization in industry in the context of ARM-based mixed-criticality systems.

1. Introduction

Nowadays, we are witnessing the use of IT technologies as never seen before in several industry domains (e.g., railway, automotive, avionics, etc.) to develop safety- and security-critical systems. This development is recently moving towards an integrated rather than federated approach, in which different services, at different criticality levels, co-exist into common hardware platforms, the so-called *mixed-criticality systems (MCSs)* (Burns and Davis, 2022). Virtualization technologies, combined with the availability of modern embedded platforms, mainly ARM-based (Wolf et al., 2008; Dall et al., 2016), equipped with multiple heterogeneous cores (general-purpose and real-time), GPUs, FPGAs, are becoming a prominent way for the industry to reduce *SWaP-C* factors (size, weight, power, and cost) by consolidating multiple software on the same system-on-a-chip (SoC) in a flexible way (Cinque et al., 2022a; BlackBerry Limited, 2021).

Virtualization serves as a foundational technology across diverse industrial domains, including the Industrial Internet of Things (IIoT) and telecommunications systems, especially with the recent advancements in 5G technology (Shift2Rail, 2024; WindRiver Systems Inc., 2016; De Simone et al., 2024; Hermann et al., 2016; Klingensmith and Banerjee, 2019, 2018; Huang et al., 2018). Furthermore, it plays a crucial role in numerous EU-funded research projects, driving innovation and technological advancements (Hercules, 2020; HERMES2020, 2020; De-RISC, 2019; SELENE, 2019). However, selecting the most suitable virtualization solution presents a significant challenge, as these systems must adhere to stringent safety requirements, ensure fault tolerance, and provide real-time guarantees. Indeed, mixed-criticality systems must satisfy the requirements provided by safety-critical standards (e.g., *EN5012X* series for railway (CENELEC, EN 50128, 2011), *DO-178B* (RTCA, 1992) for avionic, *ISO 26262* (ISO, 2011) for automotive). Such standards refer (directly or indirectly) to *temporal* and *spatial isolation* among software components, which are important properties that these systems have to guarantee. Spatial isolation includes isolating code and data between virtual machines (VMs), and preventing tasks from altering private data belonging to other tasks, including the allocated (memory-mapped) devices. Instead, temporal isolation is about limiting the impact of resource consumption (e.g., tasks running on a VM) on the performance of other software components (e.g., tasks running on the other VMs). Compared to spatial isolation, which is often assured by using hardware-assisted mechanisms (e.g.,MMU, IOMMU),

☆ Editor: Prof. Raffaela Mirandola.
* Corresponding author.
E-mail address: luigi.desimone@unina.it (L. De Simone).

https://doi.org/10.1016/j.jss.2024.112147
Received 11 January 2024; Received in revised form 14 April 2024; Accepted 20 June 2024
Available online 2 July 2024

temporal isolation assessment remains a more critical and open research problem that needs to be addressed to mitigate so-called *timing failures* (Heiser et al., 2019; Ge et al., 2019; Avizienis et al., 2004). Furthermore, the standards mentioned above recommend providing documented evidence of a *fail-safe* (failures that do not harm) and/or *fail-stop* (failures that can be reliably detected) behavior for such systems, with the objective of preventing failures from leading to human and cost losses.

This paper aims to provide a general approach to assessing the temporal isolation guarantees by a virtualized MCS. As a case study, we focused on a proof of concept of a mixed-criticality virtualized *Automatic Train Control (ATC)* system, provided in the framework of an academic-industrial partnership. Such a system is based on *2 out of 2 (2oo2)* schema that is one of the most common fault-tolerant approaches. To deploy the virtualized ATC, we examined the capabilities offered by the real-time *flavor* of the Xen hypervisor running on top of an ARM-based Xilinx Zynq Ultrascale+ board family, which are popular choices as a basis for safety-critical virtualized systems (Schulz and Annighöfer, 2022; Alonso et al., 2020; Sabogal and George, 2018). We focus on Xen since, over the years, the community has provided several features for enabling industrial settings. Indeed, current versions of Xen include support for real-time applications along with built-in cloud computing support (e.g., orchestration), minimal size for ARM architectures (less than 50KSLOC), paravirtual and GPU mediation for rich I/O, Trusted Execution Environment (TEE) virtualization support, and static partitioning (i.e., *Dom0less* architecture), as well as undergoing work for safety certification (FuSa SIG, 2022a,b; The Linux Foundation, 2023). The proposed approach is built on top of the well-known *Design of Experiments (DoE)* method, which allows us to rigorously estimate the *importance* and *significance* of the numerous existing factors in virtualized scenarios (e.g., different real-time schedulers at hypervisor level), against various operational scenarios (e.g., virtual CPU/physical CPU sharing ratios, external disturbance, etc.). Summary of the main findings are in the following:

★ Adjusting configurable knobs, such as schedulers and their parameters, can significantly influence the temporal isolation of the system. Our approach allows us to tailor the configuration according to specific requirements, such as prioritizing high performance for certain VMs over others with soft real-time requirements.
★ When we statically assign each virtual CPU to a physical CPU, we observe no significant difference in predictability by varying the Xen schedulers, even when external CPU-related disturbances are enabled.
★ Despite the static allocation of virtual CPUs to physical ones being the preferred option for real-time scenarios, it still leads to a severe increase in the mean and standard deviation of maximum execution time when the system experiences various types of stress.
★ In a *privileged VM*-based hypervisor (like Xen), the privileged VM (namely *Dom0* in Xen), heavily influences the predictability of the software running on it compared to unprivileged VMs (namely *DomU* in Xen).

These results highlight the challenges that need to be addressed in real industrial scenarios when deploying mixed-criticality software using virtualization. Based on this analysis, we further identify a set of directions toward the real use of virtualization in the context of ARM-based systems. To support future research in this field, we also shared raw data about experiments and scripts to reproduce obtained results (https://github.com/dessertlab/xen_temporal_isolation_data/). The objective is to provide guidelines that allow the development and fine-tuning of virtualized MCSs, to document temporal isolation evidence required for certification purposes. To the best of our knowledge, this is the first study that proposes a DoE-based approach for temporal isolation assessment provided by systematically varying crucial factors of a hypervisor running on an ARM-based platform.

The rest of the paper is organized as follows. In Section 2, we provide the background. Then, a general workflow for the assessment of temporal isolation properties of a virtualized MCS is proposed in Section 3. Section 4 presents a case study, which depicts the application of the proposed approach in the context of the railway domain, with a 2oo2-based virtualized MCS running on Xen. In Section 5, we analyze the results and delineate the findings. A discussion on the limitations and the general applicability of our approach is provided in Section 6. In Sections 7 and 8, we report the related works and conclusions.

2. Background

Virtualization techniques allow running multiple Operating Systems (OSes) on the same hardware. The software component that enables virtualization is the *hypervisor* or *virtual machine monitor (VMM)* which is responsible for creating, managing, and scheduling Virtual Machines (VMs), which represent real machines for the OSes running on it. There are two categories of hypervisors: *type-1* and *type-2*. Hypervisors that run directly on hardware are categorized as type-1, whereas those hosted on top of a host OS are categorized as type-2. Early virtualization techniques initially relied only on *full virtualization*, where the hypervisor emulates all privileged instructions and I/O operations requested by the guest OSes. While this method allows guest OSes (i.e., OSes running as VMs) to run unmodified, it inherently incurs significant overhead. Therefore, to achieve higher performance, the *paravirtualization* technique has been developed. In this approach, guest OSes are modified to directly use the services provided by the hypervisor through the so-called *hypercalls*. However, a drawback of this approach is that guest OSes must be aware of their execution within VMs. Finally, virtualization has been enhanced by leveraging hardware extensions in modern CPUs to improve overall performance.

Over the years, industry and academia have directed their attention towards *real-time hypervisors*. These hypervisors incorporate explicit support (mainly schedulers) to effectively manage time budgets allocated to virtual machines (VMs) to meet real-time constraints. Initially, some of the most widely known and utilized hypervisors, such as Xen and KVM, have undergone modifications to incorporate real-time schedulers tailored for mixed-criticality systems (Abeni and Faggioli, 2020). Subsequently, a variety of new hypervisors have emerged with diverse architectures, including microkernel-based designs and those leveraging ARM TrustZone (Cinque et al., 2022a). Furthermore, *static partitioning hypervisors* have gained considerable traction. These hypervisors function as configuration layers that partition hardware resources, establishing a one-to-one mapping between virtual CPUs and physical CPUs, and directly mapping devices into guest memory areas. Despite this heterogeneity, there remains a notable interest in enhancing traditional hypervisors like Xen for real-time purposes. This stems from the prevalence of existing projects built upon Xen and its widespread recognition.

2.1. ARM virtualization

To improve the performance of unmodified guest OSes, CPU vendors (e.g., Intel, AMD, ARM) have developed hardware virtualization extensions to reduce the overhead caused by the hypervisors. Specifically, ARM introduces, from the ARMv7-A specification, a new privileged execution mode called *EL2* (Exception Level 2) in addition to the kernel (*EL1*) and user (*EL0*) modes. To guarantee the isolation of VMs, the hypervisor runs in EL2 mode, having complete control of the hardware, while the VMs' software (guest OS and user applications) runs in EL1 and EL0 modes. To minimize the overhead, the hypervisor allows the VMs to run without intervention until specific conditions arise, such as the execution of privileged instructions. In the latter scenario, a hardware trap is triggered, prompting the hypervisor to assume control over the hardware. Subsequently, the hypervisor emulates the expected behavior of the VM before eventually returning control to the VM. By doing so, the software running in EL1 transparently works as it would run without virtualization.

2.2. Real-time Xen hypervisor

The Xen hypervisor is an open-source type-1 hypervisor (Barham et al., 2003) and nowadays it is used as the basis for several commercial and open-source applications, involving server virtualization, Infrastructure as a Service (IaaS), desktop virtualization, and embedded appliances. Guest OSes on Xen are typically called *domains*. In particular, the *Domain 0* (or *Dom0*), is a privileged domain by which Xen can manipulate other unprivileged domains called *DomU*.

At the dawn of Xen, only para-virtualized guests (PV guests) were supported. Initially, Xen developers made this choice for increasing performance and simplifying the hypervisor codebase. However, after major processor vendors, like Intel and AMD, started working on CPU virtualization extensions, Xen began to support also full virtualization, even hardware-assisted. As of today, Xen has evolved to accommodate a wide range of virtualization approaches, offering flexibility in determining which system components are fully emulated and which are paravirtualized. To this end, Xen currently provides four virtualization types: *PV* (classical paravirtualization), *HVM* (use virtualization extensions from the host CPU to virtualize guests), *PVHVM* (paravirtualization on HVM), and *PVH* (lightweight HVM-like guests that use CPU virtualization extensions). Among these options, the most high-performing solution is PVH, which can be seen as a PV guest operating within an HVM container, or as a PVHVM guest without any emulated devices. It is worth noting that many interesting design choices have been made during the recent porting of Xen from x86_64 architectures to ARM (Stabellini, 2014). Foremost, an effort was made to make Xen as simple as possible by reducing the lines of code to one-sixth compared with the original implementation (30k LOC). This simplification was achieved by eliminating emulation, meaning no reliance on QEMU in the loop. Additionally, only one type of virtualization mode is permitted, which falls somewhere between PV and HVM modes. Specifically, I/O functionalities are paravirtualized, while all other hypervisor functionalities are supported at the hardware level (HVM).

2.2.1. Real-time schedulers

In 2011, *Xi et al.* introduced the *RT-Xen* branch, based on Xen v4.0.1, as an initial attempt to incorporate real-time features into Xen (Xi et al., 2011). This branch implemented new schedulers designed to support soft real-time applications such as *Earliest Deadline First (EDF)* and *Rate Monotonic (RM)* policies. Furthermore, it enables dynamic switching between these policies and allows tuning scheduling parameters at the granularity of virtual CPUs (vCPUs) (Xu, 2013). Notably, after several improvements the *Real-Time Deferrable Server (RTDS)* scheduler remained in the latest version of RT-Xen (v2.2) and was also included in the vanilla Xen starting from v4.5. Conversely, Xen has undergone several changes over the years, with one of the most notable occurring in v4.7, where the RTDS scheduler switched from a *quantum-driven* to an *event-driven* model to achieve better scalability and performance for embedded and real-time workloads (Kurth, 2016).

The RTDS scheduler assigns three main parameters to each vCPU, *budget*, *period*, and *extratime*. The *budget* is the time allocated to a vCPU. If the vCPU is running, its *budget* is burned, otherwise, it is preserved. The *period* is the period after which the *budget* can be refilled. The refill task is called *budget replenishment*. Lastly, the *extratime* is a boolean value: if true, the vCPU can run after running out of time if there is a free physical CPU (pCPU). When a vCPU starts running on a pCPU, its execution is not necessarily continuous. This means that its *budget* can be spent at any time during the *period*. A vCPU can execute if either there is a free pCPU available or a lower-priority vCPU currently running in the system exists. The deadline corresponds with the end of the period. RTDS can group pCPUs in *CPU pools* and assigns each domain (i.e., VM) to one of these pools. For each CPU pool, RTDS keeps two queues: a *global runqueue* that holds all the CPUs with a non-0 residual budget sorted by priority, and a *depleted queue*, which holds all the CPUs that used up their budget (Wiki.Xenproject, 2019).

In the first implementation of RTDS (before Xen v4.7), the scheduler needed to scan the *runqueue* periodically, similarly to polling. In newer versions instead (Xen v4.7 and above), RTDS uses a separate timer for each vCPU for their budget replenishment, considerably reducing the overhead (Wiki.Xenproject, 2019). In RT-Xen, RTDS allows users to choose between two policies to assign priority to the vCPU. These two are *EDF* and *Rate Monotonic (RM)*. *EDF*, assigns dynamic priority to the vCPU, favoring the ones that have a closer deadline. *RM*, conversely, assigns a static priority to the task, precisely, the lower the period of a task (and then the higher its rate), the higher its priority.

Starting from Xen v4.9, developers introduced a way to lower latencies and increase predictability for real-time scenarios drastically. Specifically, they implemented the *null* scheduler to statically assign every vCPU to a single isolated pCPU, reducing at minimum the scheduler overhead. The benchmarks we run on the Xilinx Zynq Ultrascale+ MPSoCs show a maximum interrupt latency of fewer than 2 microseconds.

Finally, another scheduler introduced in Xen (from v4.1) for real-time purposes is *ARINC-653* scheduler. This latter is a cyclic executive scheduler that uses static time configurations to guarantee predictable CPU time allocated for guests according to the ARINC-653 specification (AEEC, 2010). ARINC-653 only supports one CPU core per guest and wastes compute resources for virtual guests that do not fully utilize compute time (The Linux Foundation, 2022a). These limitations reduce the usability of that scheduler in a virtualized MCS, and therefore, it is not explored in the experimental section of our work.

2.3. Real-time Linux

In virtualized applications, Linux is becoming one of the most common guest OSes for different reasons. Linux brings several advantages, like accessible and portable source code, in-depth research in the industrial field, and countless libraries, many of which are open source. All these features make Linux a strong alternative in embedded environments (Henkel, 2006). Besides, even both the European Space Agency (ESA) and the National Aeronautics and Space Administration (NASA), are considering the use of real-time Linux for mission-critical software (Leppinen, 2017). For the aforementioned reasons, over the years, the Linux Foundation supported several collaborative projects in the context of the real-time domain. This led to the development of the `PREEMPT_RT` patch (Reghenzani et al., 2019; The Linux Foundation, 2022b).

2.3.1. The `PREEMPT_RT` patch

Generally, the Linux vanilla kernel provides a scheduler that allows choosing three levels of preemption, the higher the level, the more the kernel is likely to be preempted. These levels include the (i) *no forced preemption*, in which the task with higher priority will not be scheduled until the finish of the kernel code; (ii) *voluntary preemption*, in which the kernel can run a rescheduling, deciding to yield voluntarily the resource to the task with higher priority; (iii) *preemptible kernel*, in which the higher priority task preempts the lower priority task, even if it is running kernel code, but not in critical sections. Besides these types of preemption, the `PREEMPT_RT` patch adds two more levels. In particular, the *preemptible kernel (Basic RT)*, which is used for debugging, and the *fully preemptible kernel*, in which the kernel can always be preempted, even in critical sections, except for the top half of interrupt handlers and regions protected by *raw spinlocks* (Reghenzani et al., 2019).

Experimental results show that Linux with `PREEMPT_RT` patch produces low latencies even in virtualized environments compared to the vanilla version (Hughes and Awad, 2019). Despite that, the `PREEMPT_RT` patch is known to increase the number of context switches, resulting in a degradation of system performance in terms of throughput (Reghenzani et al., 2019).

Table 1
Temporal isolation properties across safety-related standards.

Domain	Standard	Details	Reference
Generic	IEC 61508, Part 3 (International Electrotechnical Commission, 1998)	"*Temporal: one element shall not cause another element to function incorrectly by taking too high a share of the available processor execution time or by blocking the execution of the other element by locking a shared resource of some kind.*"	F.2, F.5 (Annex F)
Avionic	DO-178C (RTCA, 2011)	"*A partitioned software component should be allowed to consume shared processor resources only during its scheduled period of execution.*"	Section 2.4.1.b
Automotive	ISO 26262, Part 6 (ISO, 2011)	"*With respect to timing constraints, the effects of faults such as those listed below can be considered for the software elements executed in each software partition: blocking of execution; deadlocks; livelocks; incorrect allocation of execution time; incorrect synchronization between software elements.*"	D.2.2 (Annex D)
Railway	EN 50128 (CENELEC, EN 50128, 2011)	"*An analysis is performed which will identify the distribution demands under average and worst-case conditions. This analysis requires estimates of the resource usage and elapsed time of each system function. These estimates can be obtained in several ways, for example, comparison with an existing system or the prototyping and benchmarking of time critical systems*"	D.45 (Annex D)

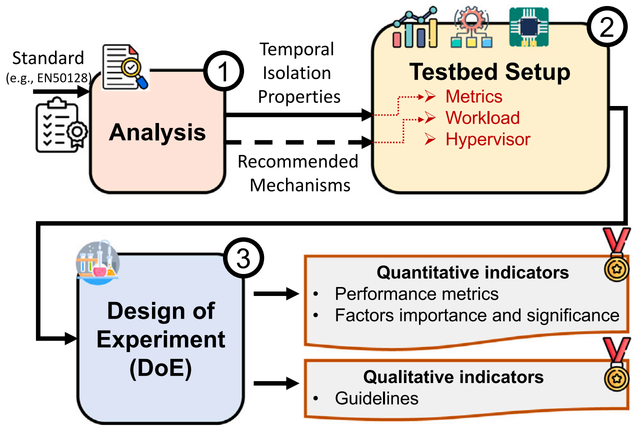


Fig. 1. Proposed temporal isolation assessment workflow.

3. Temporal isolation assessment workflow

The proposed temporal isolation assessment approach follows the steps highlighted in Fig. 1. The proposal includes a systematic approach to unveil potential interference in a virtualized MCS under corner case conditions (e.g., stressful privileged/unprivileged VMs, faults in the hypervisor, bad configurations, etc.). The ultimate objective is twofold: (i) quantifying how strong the temporal isolation is assured by the selected hypervisor, and (ii) providing guidelines for industry practitioners to fine-tune the system parameters in case of weak isolation. The main idea is to leverage the well-known *DoE* to rigorously assess the *impact* and *significance* of virtualized system parameters in real operational scenarios.

3.1. Analysis

In the **first step** (①in Fig. 1), industry practitioners need to analyze the target safety-related standard to extrapolate *temporal isolation properties*, then *recommended mechanisms* (if specified) to assess such properties.

Table 1 shows how four different safety-related standards, i.e., ISO 61508 (International Electrotechnical Commission, 1998), DO-178C (RTCA, 2011), ISO 26262 (ISO, 2011), EN 50128(CENELEC, EN 50128, 2011), treat or mention temporal isolation properties to be met according to specific level of safety. Depending on the standards, the safety level is indicated as *Safety Integrity Level (SIL)*, *Automotive Safety Integrity Level (ASIL)*, *Software Safety Integrity Level (SSIL)*, or *Design Assurance Level (DAL)*. Furthermore, some standards also mention specific techniques to ensure these properties. For instance, DO-178C and IEC 61508 recommend achieving temporal predictability through fixed cyclical scheduling, time-triggered scheduling, fixed priority-based scheduling, CPU execution time monitoring, or Worst-Case Execution Time (WCET) analysis for temporal isolation purposes.

In the context of virtualized MCSs, temporal isolation includes the ability to isolate or limit the impact of resource consumption (e.g., CPU, network, disk) of a VM on the performance degradation of other VMs and even against the host. This means that a task running in a VM must not cause delays to other critical and non-critical tasks running in different VMs, avoiding phenomena such as starvation, reduced throughput, and increased latency. Indeed, by adding this new level of software indirection (i.e., the hypervisor), we need to take into account additional knobs, e.g., the guest OSes schedulers, vCPUs/pCPUs mapping, the hypervisor scheduler, and so on: if not properly managed they could easily break temporal isolation requirements.

3.2. Testbed setup

Once specified what isolation properties should be monitored and verified, the **second step** (②in Fig. 1) includes *testbed setup*. This step consists of (i) selecting metrics that reflect the temporal isolation properties identified in the previous step; (ii) defining a workload that involves average and worst-case conditions; the workload can reflect recommended mechanisms proposed by the target standard to verify temporal isolation properties; (iii) selecting the target hypervisor to deploy the virtualized MCS. In turn, the hypervisor selection includes several knobs to be manipulated, e.g., how many guests need to be used for running the target workload, guest OSes schedulers and real-time patches to be applied (e.g., `PREEMPT_RT`), vCPUs/pCPUs affinity, hypervisor scheduler, hardware/software isolation mechanisms (e.g., cache coloring (Kloda et al., 2019; Suzuki et al., 2013; Ye et al., 2014), memory bandwidth reservations (Suzuki et al., 2013)). Clearly, if these features are not properly tuned, they could easily cause temporal isolation violations.

3.3. Design of experiment (DoE)

The **third step** is the *Design of Experiment (DoE)* (Fisher, 1936) (③in Fig. 1). DoE enables a systematic way to unveil the relationship between the key factors in a virtualized deployment (e.g., vCPU/pCPU affinity) and the variation of information according to the variation of these factors (e.g., the standard deviation of application latencies). In particular, the DoE allows planning experiments so that appropriate data can be analyzed by statistical methods that result in valid, unbiased, and meaningful conclusions. Realizing a DoE involves a *design phase*. This latter consists of choosing a *response variable* of interest and a set of controllable *factors* that we expect to affect it. Each factor is characterized by different values called *levels*, and each experiment has to be replicated to reduce as much as possible the statistical error. By performing enough experiments, it is possible to define which of the factors is more *important* and *significant* than the others and which level of each factor allows for improvement in the response variable. Subsequently, DoE exploits statistical methods such as the ANOVA analysis (St et al., 1989), which includes a set of guidelines to extract statistically meaningful information from data to provide numerical evidence. Eventually, the DoE step produces both *quantitative* and *qualitative indicators*. The former is related to performance metrics, such as throughput and latency, and the statistical importance and significance of chosen factors during *step 2*. The latter, instead, includes guidelines about providing the best combination of testbed configurations to achieve the desired temporal isolation.

4. Case study

Our case study is related to a train's onboard control equipment for an Automatic Train Control (ATC)/Automatic Train Operation (ATO), which corresponds to the partner's real-world deployment. The target application is a 2oo2-based mixed-criticality system that uses a voter mechanism to assess vital functions. The objective is to determine in which conditions industry practitioners can employ virtualization to migrate existing applications. We used such a system as a case study that adheres to the *European Rail Traffic Management System/European Train Control System (ERTMS/ETCS)* standard. This latter specifies the *European Vital Controller (EVC)*, one of the fundamental onboard ERTMS/ETCS equipment that computes the reference speed patterns based on information from the wayside equipment and performs brake control based on the speed reference (see Directorate-General for Mobility and Transport (2024) for further details about ETCS equipment).

Generally, the EVC subsystem is deployed according to the classical 2oo2 schema, in which two replicas are sending monitored vital values (e.g., from train sensors) toward a voter, which compares received data to detect the (potential) difference and notify (if needed) an error. When an error is notified, the train should be brought to a fail-safe state (e.g., stop the train). The further reference we used to real-world deployments is in Amendola et al. (2023).

4.1. Step 1: the EN50128 standard analysis

In the context of the railway domain, the EN50128 standard regulates the development of software for use in railway control and protection applications. The standard states that for the `Software Verification` (ref. EN50128 section 6.2) and `Architecture and Design` (ref EN50128 section 7.3) activities, developers should provide the `Software Verification Plan` and `Software Integration Test Specification` documents, both of them refer to `Performance Testing` requirement (ref. EN50128 table A.18). This latter, proposes to use three techniques/measures, all of them highly recommended (i.e., HR) for SIL3/SIL4 levels. In particular, the `D.45 Response Timing and Memory Constraints` requirement aims to "`ensure that the system will meet its` **`temporal`** `and memory` **`requirements`**", specifying that:

- "`[...] an analysis is performed, which will identify the` **`distribution demands under average`** `and` **`worst case conditions`** `[...]`"
- "`[...] this analysis requires estimates of the` **`resource usage`** `and` **`elapsed time`** `of each system function. [...]`"
- "`[...] These estimates can be obtained in several ways, for example,` **`comparison`** `with an existing system or the prototyping and` **`benchmarking`** `of time critical systems. [...]`"

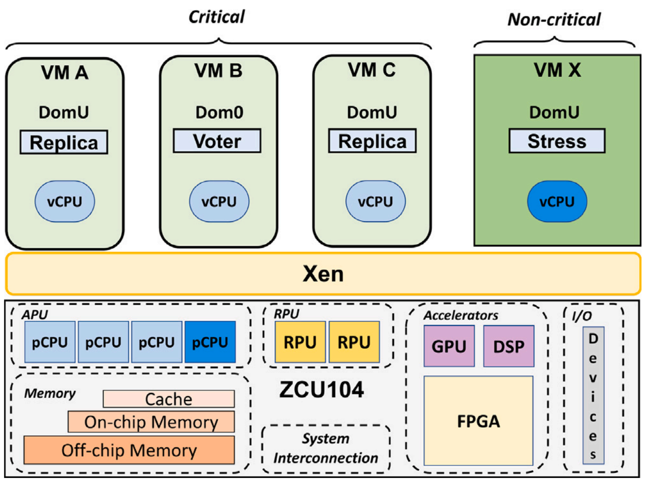


Fig. 2. The Xen-based virtualized 2oo2-based testbed used for experimentation.

This analysis provides the temporal isolation properties to be verified (*resource usage* and *elapsed time*) of target critical system function by *comparing* different configurations, and leveraging *benchmarks* to unveil potential issues.

4.2. Step 2: Xen on ARM-based mpsoc testbed setup

Following the *step 2* in the workflow depicted in the previous section, we deployed a 2oo2-based MCS testbed (see Fig. 2), which mimics an *Automatic Train Control (ATC)* (a general class of train protection systems for railways) according to the needs of our industrial partner. The 2oo2-based MCS is consolidated via Xen v4.16 hypervisor, all on top of the ARM-based Zynq Ultrascale+ ZCU104 board, which provides a quad-core ARM Cortex-A53, a dual-core ARM Cortex-R5F, 16 nm FinFET + Programmable Logic, and an ARM Mali-400MP2. This use case was proposed by our industry partner, which required us to verify the feasibility of shifting their legacy ATC application into a virtualized one.

The implemented 2oo2-based MCS application (see Fig. 2) includes two replicas (i.e., *VM A* and *VM C* running as *DomUs*) that periodically produce measurements to be assessed by a voter (i.e., *VM B* running within *Dom0*) that compares them once received. The period of the replicas and voter is 20 ms; the period was chosen to address the requirements of our industry partner and to speed up the DoE step, which includes several factors and test repetitions. After sending the measurement, each replica sleeps until the next period. As soon as a value from each replica is received, the voter compares them, returning an *ACK* to replicas within the next period. All the VMs leverage UNIX sockets for inter-VM communication. This choice was made due to requirements imposed by our industry partner since their main objective is to reuse (unmodified) its legacy software. This setting is also useful to unveil whether the utilization of the network software stack can lead or not to unpredictability.

The testbed described before is realistic in terms of the replication schema commonly used in the railway domain. However, what is crucial for our industry partner is to understand how to fine-tune the critical knobs mentioned before to build a virtualized MCS properly. To this end, during experimentation, we launch a stress VM, i.e., *Stress DomU* (*VM X* in Fig. 2), which runs side by side with replicas and voter domains to emulate a co-located non-critical workload. The purpose is to cause a high computational load that might result in interference between the existing critical domains. To introduce the desired stress (e.g., CPU, network, I/O), we used *stress-ng* (King, 2017), a versatile stress testing tool suite, also used in industrial domains (Tran et al.,

Table 2
Experimental factors and levels.

Factors	Levels								
Host-level scheduler	*Credit2*				*RTDS*				*null*
Scheduler parameters	*Rate Limit [µs]*				*Budget/Period [µs]*				N/A
	100	500	1000	5000	1k/5k	3k/9k	5k/10k	10k/10k	
CPU affinity	*One-to-One*					*Many-to-Many*			
Stress	*Yes*							*No*	
	CPU	Cache	Device	I/O	Intr	FS	Net		

2021; Jo and Choi, 2022), designed to assess system stability and performance by inducing various types of stress on hardware components. It offers a wide range of stressor classes, including CPU, memory, disk, network, and other subsystems, allowing for comprehensive testing of system resilience under diverse workloads. Clearly, in an in-production scenario, the testbed in Fig. 2 can be implemented by adopting multiple MPSoCs running spare versions of both replicas and the voter. However, we aim to unveil temporal isolation breaks induced by the hypervisor.

As mentioned before, we isolate each replica in different VMs (*DomUs* in Xen jargon) and deploy the voter application on the most critical domain in Xen, namely *Dom0*. It is important to specify that in a realistic scenario, the voter is a very high-critical component that needs to be implemented on dedicated hardware with hard-wired logic for real-time communication. However, we still decided to deploy the voter on *Dom0*, both to facilitate the testing phase and to understand how much this domain affects the execution time of replicas within *DomUs*. This analysis is crucial since by default Xen comes with *Dom0* anyway, and avoiding running applications on top of it would be a waste of resources, which are already scarce in real-world embedded scenarios, especially in the case of strict partitioning, when physical hardware resources are statically assigned to VM and not shared.

We compare different configurations of Xen to verify which is the best one according to the requested real-time requirements. In general, for interlocking systems, there is no standard agreement on performance requirements in Europe, and each railway infrastructure provider can adopt its thresholds. For example, in *Rete Ferroviaria Italiana (RFI)*, the Italian railway infrastructure manager, set the requirement for an acceptable voting response time in between 350 ms and 500 ms (Rete Ferroviaria Italiana (RFI), 2004; Amendola et al., 2023). It is important to note that such a threshold includes the communication overhead between the train's onboard components and software elements needed to run the voting logic. By excluding such overheads, we can assume an acceptable time for the voting logic to be in the order of hundreds of µs. This threshold is consistent with the study in Amendola et al. (2023), in which the authors targeted a similar system (i.e., the *Vital Control Module - VCM*). Specifically, the authors in Amendola et al. (2023) considered 3 real-time tasks for the voting logic, which include an estimated WCET within 20 µs on average for each of the 3 tasks; by including also the WCET switching time between tasks, which they estimate around 35 µs, they assumed an acceptable WCET for voting within ≈100 µs. In general, such thresholds can be considered valid also in the European-wide railway regulation, i.e., ERTMS/ETCS framework.

Regarding the metric selection task in the proposed workflow (see *step 2* in Fig. 1), we choose the replicas *maximum execution time* as the worst-case metric for comparing all the target hypervisor configurations, where each replica runs a periodic task that produces data and sends it to the voter whose check whether the data produced by the two replicas is equivalent. The testbed knobs we decided to manipulate are the host-level schedulers and the related configuration parameters (e.g., budget/period ratio for *RTDS* scheduler), and the affinity between virtual CPU of VMs and physical CPUs, i.e., *CPU affinity*. Therefore, we set a `PREEMPT_RT`-patched Linux, enabling the

`SCHED_FIFO` policy for each guest OS. `PREEMPT_RT` patch is known to provide the best real-time guarantees (lowest kernel-induced latencies) for Linux environments (Hughes and Awad, 2019; Reghenzani et al., 2019); further, we leveraged `SCHED_FIFO` policy since each application (the voter and the replicas) is single-threaded, and `SCHED_FIFO` is the simplest Linux scheduler with real-time capabilities. We discarded any of the new experimental techniques proposed in the literature that improve resource isolation (e.g., *Memguard* (Yun et al., 2013), *cache coloring* (Kloda et al., 2019), *I/OGuard* (Jiang et al., 2021)), due to either no support or still ongoing work for Xen. The ultimate objective is to have a simple and easily reproducible scenario, as well as a precise worst-case scenario.

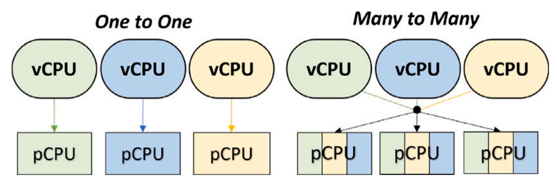


Fig. 3. CPU affinity factor and its levels.

Given this scenario, we can understand if some configurations exhibit more interference than others by precisely estimating the significance of factors via the DoE approach (see *step 3* in Fig. 1), which is described in the following.

4.3. Step 3: Design of experiments

The last step within the proposed workflow is to define the factors and levels to be used in DoE. Table 2 shows their description.

The first factor is the **host-level scheduler** that, as mentioned above, includes all the possible Xen schedulers except for ARINC-653, which shows known issues described in 2.2.1, and the old version of *Credit*. Thus, we consider *RTDS*, *Credit2*, and *null* schedulers as levels for that factor.

We assume the host-level **scheduler parameters** as a second factor. Regarding *RTDS*, the scheduler parameters include the *budget* and the *period* of the scheduler; specifically, we give to the VMs one-fifth, one-third, one-half, and finally the entire budget in each period as levels. For the *Credit2* scheduler, we decided to vary the *rate limit*, i.e., the minimum execution time interval of a vCPU before a context switch. The valid range in Xen is 100 µs to 500000 µs (500 ms), but since the replicas and voter period were set at 20 ms in our experiments, and we have four VMs, we decided to reduce the range from 100 µs to 5000 µs (5 ms) so that at least once in a period all VMs can run. Regarding *Credit2*, Xen allows modifying also the *weight* associated with each VM, where the weight indicates how much physical CPU to reserve for one VM compared to the others. However, this parameter is too application-dependent, so we set the same weight for each VM even though could not be the optimal solution. Lastly, for the *null* scheduler, there are no parameters modifiable.

Therefore, we assume the affinity level of vCPUs over available pCPUs as the third factor, namely, **CPU affinity**. Specifically, we focus on two scenarios depicted in Fig. 3: the first one is where each vCPU (thus, each VM) is pinned on a dedicated pCPU (i.e., the *One-to-One* level); the second case is where more vCPUs (thus, more VMs) are not pinned to a specific pCPUs but run on multiple shared pCPUs (i.e., the *Many-to-Many* level). In our experimental scenario, each critical VM (replicas and voter) has a single vCPU, and we can use 3 out of 4 pCPUs available on the target board (i.e., Zynq Ultrascale+ ZCU104) for these VMs since the fourth pCPU is dedicated to the non-critical stress VM. Finally, the *One-to-One* level implies that each VM has one pCPU at his own disposal, meanwhile the *Many-to-Many* level implies that the three critical VMs share three pCPUs. However, in case the host-lost scheduler is *null*, the only *CPU affinity* level available is *One-to-One*; this is because *null* scheduler by default assigns each vCPU to a single pCPU not allowing any kind of scheduling of vCPUs over several pCPUs.

Finally, the **stress** factor indicates the presence/absence of the stress VM, where 8 possible levels of stress were considered if the stress VM is enabled. In particular, we focused on *CPU* (CPU intensive), *Cache* (stress CPU instruction and/or data caches), *Device* (raw device driver stressors), *I/O* (generic Input/Output operations), *Intr* (high interrupt load generators), *FS* (file system activity), and *Net* (TCP/IP, UDP, and UNIX domain socket stressors).

To assess the temporal isolation provided by Xen, we need to record the time needed by the replicas for sending measurements. In particular, we measure the execution time, of both replicas, $n = 2000$ times for each configuration test, then we take the *maximum execution time* among the n samples. To provide statistical significance in our experimentation, and to remove the noise, we performed 30 repetitions for each experimental configuration, by also resetting the target board, for each repetition, to assure test independence. Considering the DoE described in Table 2 we have 4 factors with respectively 3, 4, 2 and 8 levels. Using a *Full Factorial Design* we would have a total of 192 potential experiments to conduct, repeated 30 times, for a total of 5.760 experiments. However, some of these configurations are not implementable; for example, *null* scheduler has just one possible level for both *scheduler parameters* and *CPU affinity* factors. Furthermore, we decided to not consider all the stress classes (CPU, Cache, Device, etc.) for each possible combination of the other three factors but we decided to test first the CPU stress class fixing the best scheduler parameters factor and varying the Scheduler and the CPU affinity factors and then to test all the eight stress classes fixing the best level for each of the other factors. In this way, the number of final experiments is reduced to only 29 such as to allow us a more targeted analysis of our data.

5. Experimental results

This section presents the experimental results from the tests conducted on the established testbed setup and experimental design outlined earlier.

To ensure clarity, we initially present the results by isolating each scheduler and analyzing its performance under various scheduler parameters configurations and CPU affinity. This approach allows us to discern the optimal configuration for each scheduler and gain insights into their respective behaviors.

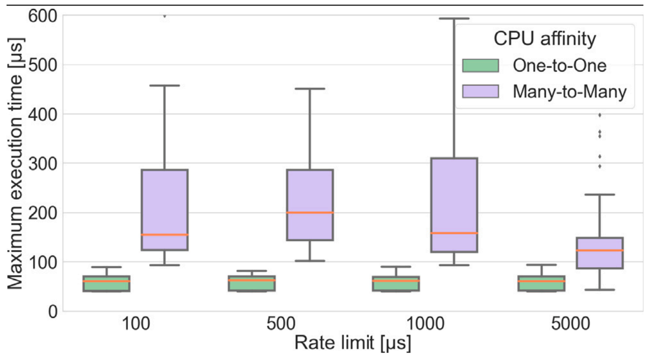
Subsequently, we fix the best configuration found for each scheduler and compare the schedulers with each other. To perform a fair comparison, we first compare the schedulers when the physical CPUs are fully shared (*Many-to-Many* scenario), and then we compare the schedulers when the physical CPUs are not shared (*One-to-One* scenario), thus including *null* scheduler in the comparison. Having identified the most optimal scheduler and configuration based on the mean and standard deviation of the response variable (i.e., maximum execution time), we proceed to evaluate its performance under various stress conditions introduced by executing stress tests on an isolated stress VM.

5.1. Credit2 *analysis*

The *Credit2* scheduler is the Xen default non-real-time scheduler. As mentioned, we compute the maximum execution time, performing 30 repetitions w/ and w/o the *stress DomU*, configured with a CPU stressor, to reveal potential isolation breaks. We could use a two-way ANOVA with interactions and repetitions to verify if both factors (*CPU affinity* and *rate limit*) induce a significant difference in the response variable (i.e., the *maximum execution time*). To apply it we need independence of observations, the homoscedasticity, and the normality of the residuals (St et al., 1989). In order to ensure as much as possible independence between observations, we reset the target hardware for each experiment. Hence, we have to check for homoscedasticity and normality of the residuals. The residuals are normal according to the performed Shapiro–Wilk test (i.e., $W = 0.83$ and p-value < 0.0001); however, the homoscedasticity assumption is rejected by the Levene test with high confidence from each type of test. In such conditions, we must proceed with a *heteroscedastic ANOVA* test (Jan and Shieh, 2014), and we decided to use Welch's ANOVA test. The results indicate statistical significance of both factors with a very low p-value (< 0.0001), i.e., variation in the levels of both *CPU affinity* and *Rate Limit* factors implies significant variation in the response variable.

(a) Means (M) and standard deviations (SD) of maximum execution time for replica VMs . *One-to-One* is highlighted with green (🟩), while *Many-to-Many* with magenta (🟪).

Rate Limit [μs]	*100*	*500*	*1000*	*5000*
M [μs]	56.62	56.81	56.19	57.33
	211.50	227.07	217.87	134.72
SD [μs]	15.42	14.07	14.58	15.96
	116.22	107.00	131.48	76.04



(b) Boxplots **depicting the** maximum execution times **of** replica VMs .

Fig. 4. *Credit2* host-level scheduler analysis, by varying *rate limit* i.e., the minimum execution time interval of a vCPU before a context switch.

Table 4(a) and Fig. 4(b) show the variability of the *maximum execution time* in one of the replicas used in the 2oo2-based application, according to the variation of *CPU affinity* and *rate limit* factors. The sharing of physical CPUs implies a large increase in average maximum execution time as well as greater standard deviation. Furthermore, in the case of non-sharing of pCPU (i.e., the *One-to-One* scenario), the *rate limit* factor, for each level, does not lead to any noticeable change, neither in the average nor in the standard deviation of the maximum execution time. On the other hand, in the *Many-to-Many* case, the configuration with the highest *rate limit* (i.e., $5000us$) has both a lower average of maximum execution time and lower standard deviation, thus is the best configuration for this type of scheduler in this scenario. This is because this configuration causes as few context switches as possible.

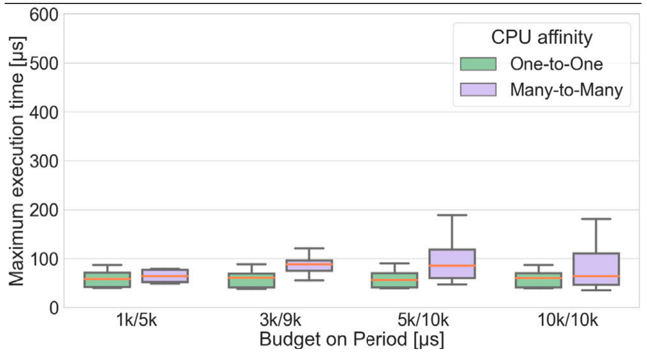
5.2. RTDS *analysis*

Same as with *Credit2*, we leveraged two-way ANOVA with interactions and repetition for *RTDS* level for host-level scheduler factor. Accordingly, the *scheduler parameters* factor is the ratio between *budget* and *period*. Again, the Shapiro–Wilk test demonstrates the normality of the residuals ($W = 0.9$ and p-value < 0.0001) and the Levene test proves the heteroscedasticity. Therefore, in this case, as well, we leveraged Welch's ANOVA test to prove the significance of our factors with high confidence (p-value < 0.0001 for both factors).

Table 5(a) and Fig. 5(b) show the difference between maximum execution time by varying the CPU affinity factor. In contrast with *Credit2*,

(a) Means (M) and standard deviations (SD) of maximum execution time for replica VMs . *One-to-One* is highlighted with green (🟩), while *Many-to-Many* is highlighted with magenta (🟪).

Budget/Period [μs]	*1k/5k*	*3k/9k*	*5k/10k*	*10k/10k*
M [μs]	57.78	56.54	55.82	56.87
	64.28	85.68	92.32	80.88
SD [μs]	15.06	14.38	15.41	15.76
	12.60	15.59	37.34	45.01

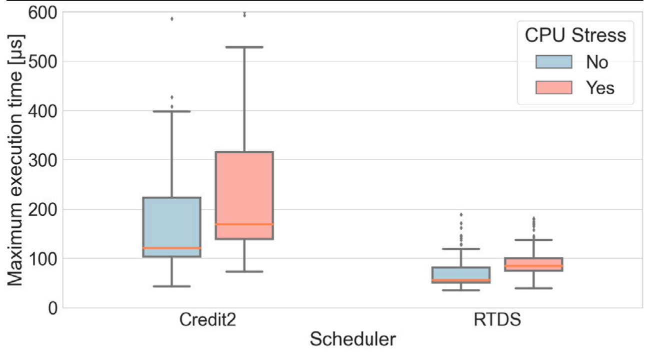


(b) Boxplots **depicting the** maximum execution times **of** replica VMs

Fig. 5. Analysis of *"RTDS"* host-level scheduler factor, while varying the *budget/period* in microseconds.

(a) Means (M) and standard deviations (SD) of maximum execution time replica VMs . *Stress* factor disabled is highlighted with blue (🟦), while *stress* factor enabled (CPU level) is highlighted with red (🟥).

Scheduler	*Credit2*	*RTDS*
M [μs]	163.13	70.01
	232.46	91.62
SD [μs]	94.84	30.74
	123.01	30.63



(b) Boxplots **depicting the** maximum execution times **of** replica VMs

Fig. 6. Analysis of *"Credit2"* and *"RTDS"* host-level scheduler factors, fixing the *CPU affinity* factor to the *Many-to-Many* level, and by enabling/disabling *stress* factor, CPU level. Parameters for both schedulers are configured based on the optimal results attained in prior experiments.

RTDS is much more robust to variations, as we expected. Anyway, as for *Credit2*, in the case of the *One-to-One* scenario, the variation of factor levels does not affect significantly the maximum execution time; however, in the *Many-to-Many* case, it is clear that the configuration set with one-fifth of the bandwidth, i.e. the *1k/5k* level, guarantees very low variation both in the mean and standard deviation of the response variable. Thus, we can state that the *1k/5k* configuration seems to be the best configuration for *RTDS* in our scenario.

Credit2 and RTDS Analysis

★ ***Credit2* and *RTDS* scheduler parameters** (rate limit and budget/period, respectively) **heavily impact temporal isolation**. Fixing the best scenario, in which no pCPUs are shared between VMs, comparing to vCPU/pCPU sharing enabled:

- *Credit2* leads to a noticeable deterioration in the standard deviation of maximum execution time, i.e., +801% and +376%, in the worst (rate limit equal to 1000 µs) and the best (rate limit equal to 5000 µs) cases, respectively;
- *RTDS* leads to a smaller increase in the standard deviation of maximum execution time, i.e., +185% in the worst case (budget/period equal to $5k/10k$). The best case (budget/period equal to $1k/5k$) includes no statistically significant difference by enabling or disabling pCPUs sharing between VMs.

5.3. Credit2 *vs* RTDS*: Many-to-many analysis*

In this section, we take into account scenarios in which there is a strict requirement for using more virtual CPUs than the available physical CPUs. This is the common case for high consolidation degrees in virtualization environments. However, these kinds of scenarios, especially in the presence of sources of disturbance, can heavily affect real-time guarantees. To evaluate the robustness of Xen in the worst case, against these scenarios, we performed a comparative analysis of *Credit2* and *RTDS* in the case of *Many-to-Many* level for *CPU affinity* factor. Precisely, we fixed the best-case configurations of both*Credit2* (rate limit equal to *5000*) and *RTDS* (budget/period equal to *1k/5k*) in the *Many-to-Many* case (see Section 5.2). However, in this case, instead of joining the repetitions w/ and w/o the stress DomU, we provide two different isolated data samples for each scheduler; one with the stress DomU (configured with a CPU stressor), and one without.

In this scenario, the stress VM runs on a dedicated physical CPU, and can be configurable according to the desired stress workload: we leveraged *CPU intensive stress* class workloads provided by *stress-ng* (e.g. Fibonacci, factorial, etc.). This is to understand how robust are *Credit2* and *RTDS* against such kind of disturbance. From a DoE point of view, we consider two factors with two levels, respectively, i.e., the *stress* factor (specifically the *CPU* stress level) with $\{yes, no\}$ levels, and *host-level scheduler* factor with $RTDS, Credit2$ levels. We still used the ANOVA test in this scenario, so we need to verify the normality and homoscedasticity of the residuals. The results show that residuals are normally distributed according to the Shapiro–Wilk test ($W = 0.88$ and p-value < 0.0001), and heteroscedastic according to the Levene test. Also, in this case, Welch's ANOVA test suggests that both factors are significant with a high level of confidence (p-value < 0.0001).

As we expected, Table 6(a) and Fig. 6(b) show that *RTDS* is the most suitable scheduler for real-time setting. Indeed, the standard deviation of the maximum execution time is much lower than *Credit2*, and in addition, the mean variation w/ and w/o stress enabled is also lower.

Credit2 vs RTDS: Many-to-Many Analysis

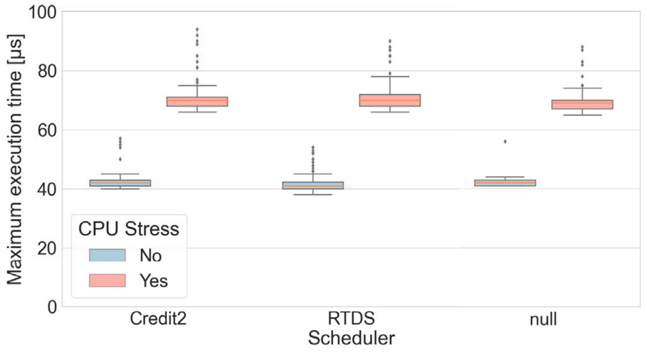
★ ***RTDS* is the most suitable scheduler for real-time setting when vCPU/pCPU sharing is enabled**, even enabling external CPU-related disturbances. Indeed, the standard deviation (+30% vs +0%) and means (+42% vs +31%) of the maximum execution time are much lower comparing *RTDS* against *Credit2*.

5.4. Credit2 *vs* RTDS *vs* `null`*: One-to-one analysis*

So far, we left alone *null* scheduler since it does not imply any scheduling actions but simply runs each vCPU on an isolated pCPU. Notably, factors considered for Credit2 and RTDS schedulers, such as *CPU affinity* and *scheduler parameters*, cannot be applied for *null* scheduler. In this section, we take into account the *One-to-One* level of *CPU affinity* factor to make a fair comparison between *null*, *Credit2*, and *RTDS* schedulers. As in the previous section, we also considered the *stress* factor configured with the *CPU* level. In this case, we cannot rely on the ANOVA Welch test since the analysis of the residuals, via the Shapiro–Wilk test, shows that they are not normally distributed

(a) Means (M) and standard deviations (SD) of maximum execution time replica VMs . *Stress* factor disabled is highlighted with blue (🟦), while *stress* factor enabled (CPU level) is highlighted with red (🟥).

Scheduler	*Credit2*	*RTDS*	*null*
M [μs]	🟦 42.45	🟦 42.29	🟦 42.60
	🟥 71.03	🟥 71.22	🟥 69.03
SD [μs]	🟦 3.31	🟦 3.33	🟦 2.77
	🟥 5.04	🟥 4.89	🟥 3.85

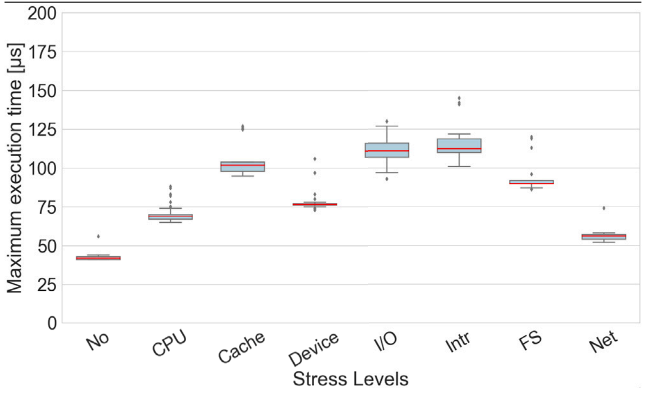


(b) Boxplots **depicting the** maximum execution times **of** replica VMs .

Fig. 7. Analysis of *"Credit2"* and *"RTDS"* host-level scheduler factors, fixing the *CPU affinity* factor to the *One-to-One* level, and by enabling/disabling *stress* factor, CPU level.

(a) Means (M) and standard deviations (SD) of maximum execution time for replica VM (DomUs) .

Stress	*No*	*CPU*	*Cache*	*Device*	*I/O*	*Intr*	*FS*	*Net*
M [μs]	42.60	69.03	104.56	78.13	111.66	115.57	92.80	56.07
SD [μs]	2.77	3.86	9.88	6.71	8.37	10.28	8.60	3.66



(b) Boxplots **depicting the** maximum execution times **of** replica VMs .

Fig. 8. Analysis of *"null"* host-level scheduler factor on DomU, fixing the *CPU affinity* factor to the *One-to-One* level, and varying all levels within *stress* factor (CPU, Cache, Device, I/O, Intr, FS, Net).

($W = 0.69$ and p-value < 0.0001). According to the Levene test, the residuals are heteroscedastic for the *stress* factor and homoscedastic for the *host-level scheduler* factor. Thus, to verify whether the distribution of data, by varying schedulers, belongs to the same population or not, we leverage the Kruskal–Wallis test. The test proves, with a high level of confidence (p-value < 0.0001), that the data distributions, when the disturbance factor varies, belong to different populations, meanwhile, when the scheduler factor varies, they belong to the same population.

Table 7(a) and Fig. 7(b) confirm that all the schedulers behave similarly when we avoid any sharing of physical CPUs among virtual CPUs (i.e., the *One-to-One* scenario). However, what stands out is that even for *One-to-One* scenario, the *stress DomU*, configured with a CPU stressor, leads to statistically significant delays in maximum execution times for replicas.

Credit2 vs RTDS vs `null`: One-to-One Analysis

★ Considering the ***null* scheduler**, which is designed to statically assign every vCPU to a pCPU, and **no pCPUs sharing** between VMs, **compared to *Credit2* and *RTDS*, we obtain no significant difference** on both means and standard deviations of maximum execution time, even enabling external CPU-related disturbances.

5.5. null *scheduler stress analysis: DomU vs Dom0*

To further analyze the stress impact on our testbed, we decided to fix the *host-level scheduler* to *null* scheduler, as the best scheduler for real-time by design, and perform several stress tests by varying levels (i.e., *stress-ng* stressors) of the *stress* factor, as depicted in Table 2. For each stress level, we use all the stressors provided by *stress-ng* application belonging to it. For example, to stress the CPU, *stress-ng* computes the Fibonacci sequence, finds factorials from 1 to 150 using Stirling's and Ramanujan's approximations, solves a 21 disc Towers of Hanoi stack using the recursive solution and so on. In the following, we compare both replicas and voter behaviors.

5.5.1. *Replicas (DomU)*

Also in these experiments, the residuals are not normally distributed according to the Shapiro–Wilk test ($W = 0.60$ and p-value < 0.0001), and they are heteroscedastic according to the Levene test. Therefore, we use the Kruskal–Wallis test, which confirms with a high level of confidence that all stress levels lead to distributions of data that are statistically different, i.e., the type of stress is a significant factor for our response variable.

Table 8(a) and Fig. 8(b) show that the presence of a stress VM, running on a separated pCPU, cause non-negligible delays on the execution time of high-critical real-time VMs. However, we must also consider that in the case of not-shared physical CPUs, we can observe that all execution times even in the worst case are always under 150us, a time that according to the railway performance requirements discussed in 4.2 is largely within the required specifications.

`null` Scheduler Stress Analysis: DomU

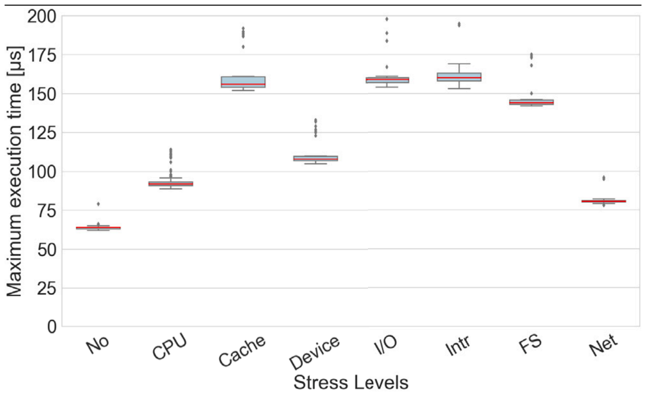
★ Despite the ***null* scheduler** was designed to be the best choice for real-time scenarios, it still **leads to severe increase against a different kind of stress** in the means and standard deviation of maximum execution time, i.e., +31.62% and +32.13% in the best case (socket stressors), and +162% and +271% in the worst case (high interrupt load).

5.5.2. *Voter (Dom0)*

The results provided previously focused on replicas and their maximum execution time distributions. As specified in Section 4.2, the voter runs on *Dom0* for easy testing and inducing a realistic disturbance on the target testbed. However, we also analyzed the maximum execution time distribution for voter VM (Dom0) by fixing the host-level scheduler to *null* and by varying the stress factor across all the target levels. As for experiments in 5.5.1, we can state that applying all stress levels leads to distributions of data that are statistically different with a high level of confidence. Further, Table 9(a) and Fig. 9(b) show the results obtained. Compared to what was obtained for replicas, it is clear that running applications on *Dom0* heavily influences predictability, with higher standard deviation variation compared to no stress. Furthermore, looking at Fig. 10, the tail distribution of the maximum execution times in *domU* is significantly smaller than in *dom0*, making it more suitable for real-time applications. This is due to critical operations that *Dom0* performs in tandem with the Xen hypervisor. As mentioned in previous sections, the voter should be implemented in an ad-hoc component, leaving *Dom0* with no applications running on top. Actually, *Dom0* is well-known to be one of the weakest points

(a) Means (M) and standard deviations (SD) of maximum execution time for voter VM (Dom0)

Stress	No	CPU	Cache	Device	I/O	Intr	FS	Net
M [μs]	64.32	94.19	162.60	112.20	161.42	162.61	147.90	81.46
SD [μs]	2.96	5.93	14.17	9.08	10.38	9.64	9.87	3.89



(b) Boxplots **depicting the** maximum execution times **of** voter VM (Dom0)

Fig. 9. Analysis of *"null"* host-level scheduler factor on Dom0, fixing the *CPU affinity* factor to the *One-to-One* level, and varying all levels within *stress* factor (CPU, Cache, Device, I/O, Intr, FS, Net).

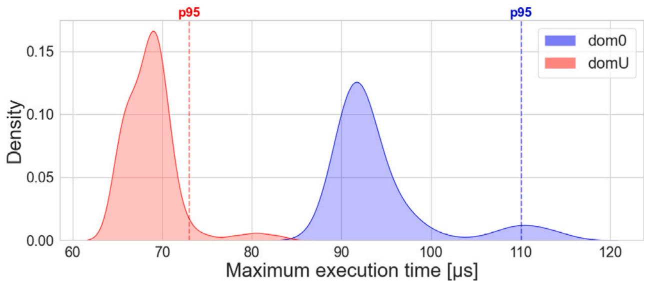


Fig. 10. Distribution of maximum execution times for voter(dom0) and replica(domU) by fixing *host-level scheduler* factor to *null* level, the *CPU affinity* factor to the *One-to-One* level, and the *stress* factor to *CPU*.

in terms of predictability of the Xen hypervisor, as it includes all the features for handling paravirtualized devices (i.e., PV drivers) and VMs lifecycle. For this reason, current works by *Xen FuSa Special Interest Group* (FuSa SIG, 2022a,b) are focusing on making Xen a fully static partitioned hypervisor for resource-constrained embedded systems and mixed-criticality systems. involves the complete removal of *Dom0* to speed up VMs' boot, simplify the certification process, and reduce the overall complexity of Xen-based systems.

`null` Scheduler Stress Analysis: Dom0

★ ***Dom0* heavily influences the predictability of the software running on it compared to *DomUs***. This is due to the critical operations *Dom0* performs in tandem with Xen hypervisor (i.e., PV drivers). *CPU instruction and/or data caches* stress the most impactful, with an increase in the standard deviation of the maximum execution time of +379%. Additionally, *Dom0* exhibits a larger tail distribution compared to *DomU*.

5.5.3. Aggregated response time analysis

Up to this point, our analysis has focused on examining the temporal isolation of individual replicas and the voter in isolation. However, the most crucial metric for the application under analysis is the aggregated response time. This encompasses the duration from when the replicas initiate computation to when the voter renders its final evaluation.

Table 3
Means (M) and standard deviations (SD) percentage increase, comparing *One-to-One* level to *Many-to-Many* level. SD best case and worst case are highlighted in green and red, respectively.

Rate limit [µs] (**Credit2**)	*100*	*500*	*1000*	*5000*
M [µs]	+274%	+300%	+288%	+134%
SD [µs]	+654%	+660%	+801%	+376%
Budget/Period [µs] (**RTDS**)	*1k/5k*	*3k/9k*	*5k/10k*	*10k/10k*
M [µs]	+11%	+51%	+65%	+42%
SD [µs]	-16%	+8%	+142%	+185%

Table 4
Means (M) and standard deviations (SD) percentage increase, comparing *No Stress* level to *CPU Stress* level. SD best case and worst case are highlighted in green and red, respectively.

Scheduler [Many-to-Many]	*Credit2*	*RTDS*	
M [µs]	+42%	+31%	
SD [µs]	+30%	+0%	
Scheduler [One-to-One]	*Credit2*	*RTDS*	*null*
M [µs]	+67%	+68%	+62%
SD [µs]	+52%	+47%	+39%

To conclude our analysis, we present in Fig. 11 the distributions of WCET for the aggregated response times. These are computed as the sum of the voter's response time and the maximum execution time among the two replicas. Each distribution corresponds to a specific type of disturbance applied using stress-ng in the other DomU. Our results reveal that despite the utilization of a real-time scheduler, the inherent complexities of a hypervisor such as Xen introduce a certain level of interference when the system is under stress. However, this interference can be thoroughly considered and analyzed to provide a realistic estimation of WCET for mixed-criticality applications.

5.6. Results and lessons learned

Tables 3–5 summarize results obtained, and highlight the percentage increases caused by factor variations in the analysis shown so far. The experimental analysis shows us a clear and statistically significant spectrum of Xen's behaviors across several configurations for building mixed-criticality systems on ARM-based architectures. First, we can state that the virtualized 2oo2-based MCS use case provided by our industry partner complies with the maximum execution times requested. The aggregate maximum execution time for the replica and voter components remains consistently below 350 microseconds, even under varying stress conditions. However, the virtualized software operations heavily depend on whether the hypervisor allows or not the sharing of physical CPUs. Further, the fewer physical CPUs shared, the better the time isolation (i.e., lower means and standard deviations). However, this does not mean that temporal isolation is assured if a partitioning approach is used, i.e., running virtual CPUs on dedicated physical CPUs. This is because sharing resources, such as I/O and memory (Last Level Cache - LLC), still causes uncontrolled latency under the hood.

A partitioning approach (see One-to-One and *null* scheduler experiments in Section 5) might be a good solution for high-criticality VMs. However, this approach obliges knowing in advance vCPU/pCPU affinity for all VMs, which is not the case if many less critical VMs need to be handled. An optimal solution would still use less stringent scheduling algorithms if possible (e.g., *Credit2*, *RTDS*) to optimize resource utilization across high and low-critical VMs, especially in embedded and resource-constrained environments.

The experimental results highlight that industry practitioners can identify optimal combinations of scheduler parameters for both *Credit2* and *RTDS* schedulers through the systematic assessment approach proposed in this study. This allows for effective management of Xen overhead, making it manageable across various industrial applications.

Table 5
Means (*M*) and standard deviations (*SD*) percentage increase, comparing *No Stress* level by fixing the *host-level scheduler* factor to *null* scheduler. SD best case and worst case are highlighted in green and red, respectively.

Stress Levels: [DomU]	*CPU*	*Cache*	*Device*	*I/O*	*Intr*	*FS*	*Net*
M [μs]	+62%	+145%	+83%	+162%	+171%	+118%	+32%
SD [μs]	+39%	+257%	+142%	+202%	+271%	+210%	+32%
Stress Levels: [Dom0]	*CPU*	*Cache*	*Device*	*I/O*	*Intr*	*FS*	*Net*
M [μs]	+46%	+152%	+74%	+151%	+152%	+130%	+27%
SD [μs]	+100%	+379%	+207%	+251%	+226%	+233%	+31%

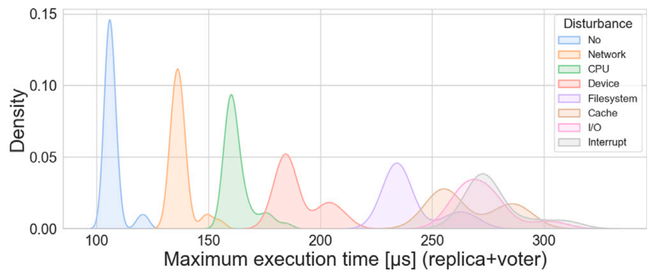


Fig. 11. Distribution of maximum execution times for the entire execution (replica + voter) by fixing *host-level scheduler* factor to *null* level, the *CPU affinity* factor to the *One-to-One* level, and varying all levels within *stress* factor.

Additional mechanisms are currently developed for the next versions of ARM-Xen, which promise to greatly reduce the hypervisor latency and overhead revealed in this analysis, at the cost of increasing the complexity of configuring Xen and reducing resource sharing. In particular:

- **Cache Coloring:** a static partitioning of LLC for VMs to avoid cache misses induced by running many VMs.
- **Dom0less:** Xen can be configured statically without the need for a privileged VM (i.e., Dom0), which has full control over the hardware. This completely removes dependence on Linux to run Dom0, and speeds up the boot of guests (DomUs);
- **Static memory communication:** It allows (i) static allocation of shared memory between VMs, (ii) static event channels without the need for complex communication drivers (e.g., xenbus (The Linux Foundation, 2015));
- **CPU-pools:** Static group division of pCPUs that enables the usage of different host-level schedulers (e.g., *null*, *RTDS*) simultaneously for different groups of VMs.

In addition, other studies in the literature proposed novel mechanisms to increase the temporal isolation in hypervisor-based environments (e.g., Mempol (Zuepke et al., 2023), Memguard (Yun et al., 2013), I/OGuard (Jiang et al., 2021), RPUGuard (Cinque et al., 2022b)), but unfortunately they are still under development and not currently integrated into the mainline Xen.

6. Threats to validity

Our experimental setup employs the Xilinx Ultrascale+ board (ZCU104) and the Xen hypervisor. While the proposed approach is broadly applicable to other boards and hypervisors, it is important to emphasize that the results are specific to our chosen configuration. Furthermore, the DoE is tailored to our railway industrial scenario. The level of interference in a mixed-criticality system can vary significantly based on factors such as application load (e.g., memory or CPU intensity) and the specific code executed by VMs within the configuration. However, note that our findings remain consistent with intuitive expectations and highlight the significance of our analysis in a real-world industrial scenario. Our results demonstrate the practical versatility of the proposed approach in analyzing interference within virtualized mixed-criticality scenarios, offering valuable insights applicable across a wide range of industrial applications.

The definition of levels and factors related to the studied scenario is strictly bound to the referenced hypervisor (i.e., Xen). When considering general-purpose hypervisors like Xen, KVM, and others, the factors chosen for assessment on Xen may be reused as-is, but the associated levels may vary. This variation arises because each hypervisor offers a different set of features, such as scheduler types and CPU affinity management. For example, in KVM the scheduler levels cannot be *Credit2*, *RTDS*, and *null*. Since KVM is a virtualization solution based on Linux it uses kernel's native CPU schedulers such as *SCHED_OTHER* (CFS), *SCHED_FIFO*, *SCHED_RR* and *SCHED_DEADLINE*. The same applies to any other general-purpose hypervisor chosen for assessment. On the other hand, if a *static partitioning hypervisor*, like Jailhouse (Siemens AG, 2024) or Bao (Martins et al., 2020), is chosen as the hypervisor to be selected, differences in features and levels become more evident. This is because those types of hypervisors do not implement scheduling algorithms, since they perform a static partitioning (allocation) of CPUs by design. Therefore, the only factor that can be reused *as-is* from our assessment in the case study is the *Stress* factor. However, other factors can be added to the analysis, such as the *cache isolation* factor that may consider the following levels: *no isolation without superpages*, *no isolation with superpages*, and *cache isolation with coloring*. This is because the use of *superpages* decreases the number of *Translation Lookaside Buffer (TLB)* misses, providing very good performance, but it is conflicting with the use of cache coloring approaches (Martins and Pinto, 2023). These considerations are fundamental for further analysis in future works.

Finally, the experiments performed on a specific complex board, such as the Xilinx Ultrascale+, reveal that interference from shared resources (e.g., the bus and the memory hierarchy), can significantly impact performance. It is important to note that the interference observed on the target board may vary compared to that experienced on a different platform, highlighting the need for tailored assessments in diverse hardware environments.

7. Related work

In this section, we survey representative virtualization solutions used for MCS and empirical analysis performed against Xen to unveil temporal isolation issues. This overview is not intended to provide an exhaustive list of current virtualization solutions adopted in the industry. The aim is to highlight impactful ones regarding the virtualization technique adopted and disruption potential, both for commercial products and by research initiatives and experiments with open-source artifacts.

7.1. Virtualization solutions in industrial domain

PikeOS (PikeOS, 2024) is a commercial hypervisor from SYSGO, used in the avionic domain. PikeOS architecture is based on the L4 microkernel (Klein et al., 2009; Elphinstone and Heiser, 2013) and can run on Intel x86, ARM, PowerPC, SPARC v8/LEON, or MIPS processors. PikeOS supports multicore platforms natively. That solution adopts three different kinds of scheduling algorithms, that is, priority-based, time-driven, and proportional share. For each real-time VM (critical

VM), PikeOS statically assigns a time slice; whenever a critical VM does not have any task to be executed, it donates CPU time to non-critical VMs. PikeOS architecture is ARINC653-compliant, in the sense that the PikeOS microkernel is the only privileged software, and it is in full control of the virtual partitions. PikeOS supports several guest OSes like Android, RT-POSIX, ARINC653-based, Java, and RTEMS. PikeOS has been the target and the basis of several academic and industrial evaluations (August, 2014; Heron, 2009) and certification and formal verification (Verbeek et al., 2015; Baumann and Bormer, 2009).

VOSYSMonitor (Lucas et al., 2018, 2017) is a low-level closed-source software layer that runs in the monitor mode of the ARM TrustZone architecture. It was conceived for the automotive industry and it is compliant with the ASIL-C requirements of the ISO 26262 standard (ISO, 2011). VOSYSmonitor enforces the RTOS, or safety-critical OS, to run on the secure world, while multiple non-critical guests can run on the normal world, managed by a non-real-time hypervisor (e.g., Xen or KVM). Non-critical guests can run only when the critical OS releases the permission to run on the assigned core in normal mode. Context switches are efficiently managed, through interrupt handling, in the monitor mode.

Jailhouse (Siemens AG, 2024) is a Linux-based partitioning hypervisor developed within a research project by Siemens and publicly available in 2013. In particular, Jailhouse enables asymmetric multiprocessing (AMP) cooperating with the Linux kernel to run bare-metal applications or guest OSes properly configured. Given the Jailhouse objective is more related to isolation than virtualization, the hypervisor splits physical resources (CPUs, memory, I/O ports, PCI devices, etc.) into strongly isolated compartments called *cells*. Each *cell* is exclusively assigned to one guest OS and its applications called *inmates*. Jailhouse includes a cell, called *root cell*, which runs the Linux kernel and will execute the Jailhouse hypervisor itself and the other cells. Despite the main objective is to partition resources, Jailhouse allows inter-cell communication through the *ivshmem* device model (QEMU, 2024a) from the QEMU project (QEMU, 2024b), which is based on an abstraction of PCI devices.

Quest-V (West et al., 2016) is a virtualization solution that fits the requirement of a mixed-criticality system. Quest-V introduces the concept of sandbox kernels, in which each sandbox contains a subset of memory, I/O, and CPU resources, following the partitioning approach. In particular, partitioning of the resource (e.g., CPUs and memory) is static as it takes place at boot-time, with no need for a global scheduler. Each sandbox is allocated exclusively to physical memory regions, but certain driver data structures and communication channels are shared. Inter-sandbox communication takes place through message passing based on shared memory and asynchronous event notification mechanisms using Inter-processor Interrupts. Quest-V also provides a fault recovery mechanism, which includes a system reboot if faulty software compromises the kernel. *DriveOS* (Sinha and West, 2021) is a recent real-time separation kernel based on Quest-V with several improvements for predictability.

Xvisor is a hypervisor developed for embedded platforms. It executes the functionality of the hypervisor in root mode, while VMs are executed in user mode. One of the principal characteristics that distinguish XVisor from other hypervisors is that only a few necessary functionalities have been added to the hypervisor. Thus, we have a very thin hypervisor with little overhead and a small memory footprint. It supports both full virtualization and para-virtualization guest systems (Patel et al., 2015). The core components are executed at the highest privilege mode. In Xvisor, the guest OS configuration is described through a device tree structure (DTS) (Likely and Boyer, 2008). It is a data structure that describes the hardware configuration of the guest system. Xvisor parses the DTS at boot time to set up the right configuration (De Bock et al., 2020). Xvisor divided the vCPUs into two types: The CPU of guest OS is called *normal*. Furthermore, there is another type of vCPU called *Orphan*: this kind of CPU runs a background process and is not used by guest OS (De Bock et al., 2020). Besides, it is possible to assign a priority to each vCPU through the DTS. vCPU with the same priority will be scheduled according to the policy implemented.

XtratuM (Masmano et al., 2009; Crespo et al., 2010) is a type-1 hypervisor that uses para-virtualization. It was designed to execute concurrently several OSes, including RTOS. It has been designed to achieve temporal and spatial requirements of safety-critical systems (Crespo et al., 2010). XtratuM provides virtualization services to partitions. A partition is a virtual environment managed by the hypervisor which executes tasks using virtualization services. Each partition can run one or more concurrent processes. A partition can be a bare-metal application, an RTOS that runs its application, or a general-purpose OS (Masmano et al., 2009). XtratuM defines two types of partitions: normal and system. System partitions are allowed to manage and monitor the state of the system and other partitions (Crespo et al., 2010). The system partition is scheduled as a normal partition, and cannot access the hardware, neither has the privilege to break the isolation. XtratuM implements a mechanism of *Fault Management*. It is provided with a *Health Monitor* (HM) that detects and reacts to anomalous events or states. If the HM discovers errors, it will try to solve them or at least confine the faulting subsystem to avoid or reduce the possible consequences.

7.2. Xen-based empirical analysis

In the literature, some studies provided a qualitative analysis for virtualized MCS (Cinque et al., 2022a), with an emphasis on guidelines on how to select hypervisor in industrial scenarios (Hamelin et al., 2020).

In Toumassian et al. (2016), the performance overhead caused by Xen deployed on an ARM architecture is computed and compared to the overhead induced by a partitioning hypervisor such as Jailhouse and a native Linux OS on the same architecture. Although the Xen overhead is still relatively high compared to Jailhouse, the authors show the improvement of predictability using a real-time VM scheduler such as RTDS instead of Credit (default). They emphasize the importance of the Xen features, stating that the selection of the most suitable hypervisor strictly depends on the requirements (e.g., flexibility over static assignment). For example, Xen provides several mechanisms for sharing resources and orchestration compared to Jailhouse, at the cost of increased overhead.

In Jiménez et al. (2022), the authors focused on automated testing facilities for hypervisors in embedded systems. They decided to leverage FPGA technology to compute the latency of the interrupt service during workload stress in Xen and Jailhouse, running on top of a Zynq Ultrascale+ family board. They show that despite Jailhouse turning out to have a better baseline, it presents many more latency spikes than Xen. These temporal spikes are caused by transitions in the hypervisor, which are less frequent in Jailhouse but cause higher latency spikes. This implies that although Jailhouse response times are closer to a non-virtualized system on average, the standard deviation of latencies remains lower in Xen.

The suitability of the Xen hypervisor for real-time safety-critical applications is analyzed in Schulz and Annighöfer (2022). In this paper, several empirical results are presented using the cyclic test benchmark (The Linux Foundation, 2024) running on top of an Intel-based hardware architecture with virtualization support, by varying the hypervisor scheduler and the VM configurations. According to the authors, Xen is potentially suitable for real-time scenarios, but it is still immature for two reasons. First, there is the problem of cache contention between VMs that could cause problems in temporal isolation. Fortunately, the problem could be handled using the cache coloring function (Ye et al., 2014), which is under development in Xen. This mechanism represents a very promising solution that enables spatial partitioning of cache memory to provide better temporal isolation of Xen VMs in the case of a sufficient number of CPU cores. Second, the

Xen size, in terms of line of code, would heavily affect the certification process, especially for the Intel x86 flavor.

In McFarland and Awad (2022) instead, the authors propose a real-time alternative Xen scheduler, called *Traspose-Xen*, capable of satisfying hard real-time constraints by using profiling and multiple resource grouping. The main idea is to consider the variation of WCET caused by resource sharing (pCPU, LLC, and memory) between VMs. Furthermore, they can avoid the starvation of non-real-time VMs, by using a mechanism based on a virtual deadline and EDF scheduler. They ran tests on an Intel-based architecture and demonstrated that the latency for different benchmarks is higher using classical Xen schedulers (RTDS, Credit) compared to *Traspose-Xen*.

In Zhang et al. (2021), the authors present the architectural differences of various hypervisors, i.e., Xen, Jailhouse, KVM, and Xvisor, in an ARM-based environment. The aim is to perform empirical analysis to verify latency growth and overhead caused by target hypervisors compared to a vanilla Linux system. The authors considered CPU, memory, I/O, and system calls as factors in the analysis. The empirical tests are carried out only on Xen and Jailhouse (KVM and Xvisor have been left for future work) and show that Jailhouse has slightly higher latencies compared to a Linux bare-metal system, while Xen exhibits higher latencies than Jailhouse. Despite that, the authors highlight that Xen is one of the most used hypervisors, and it is more flexible compared to Jailhouse.

7.3. Discussion

In this work, we focused our experimentation using Xen as a target since we are assisting a proliferation of projects that use this hypervisor as a basis for safety-critical virtualized systems (FuSa SIG, 2022a). Currently, Xen provides real-time support for scheduling (ARINC, RTDS, and *null* schedulers), a minimal size (less than 50KSLOC) for ARM-based hardware environments, paravirtual and GPU mediation for rich I/O, TEE virtualization support, *Dom0less* architecture, and undergoing efforts for safety certifiability (FuSa SIG, 2022a,b; Cesarano et al., 2022). Xen is also gaining popularity in the automotive domain, thanks to vertical initiatives like the *Automotive Grade Linux (AGL)* project, which is creating a safety-critical execution environment for workloads in software-defined vehicle architecture according to ISO 26262 (The Linux Foundation, 2018), and a path for certifying Xen as *Safety Element out of Context (SEooC)* (FuSa SIG, 2022b). Finally, Xen fully supports cloud computing infrastructures, providing mechanisms that include migration, balancing, and high availability. Thus, it can easily support potential solutions for orchestrating mixed-criticality scenarios, by running RTOSes and GPOSes consolidated on the same hardware platform (Barletta et al., 2023, 2022, 2024; Cinque et al., 2023). In that direction, Xilinx is currently developing a lightweight solution called *RunX*, which exploits Xen to both run containers as VMs in the context of the industrial IoT and edge devices, which often require assuring real-time behavior (The Linux Foundation, 2024).

As a direct consequence, there are several surveyed studies focused on measuring Xen performance for real-time and mixed-criticality purposes. The main objective was to give an idea of using Xen in safety-critical environments, especially in embedded industrial systems. To the best of our knowledge, this is the first study that proposes a DoE-based approach for estimating temporal isolation guarantees systematically. We estimated the importance and significance of factors available in the Xen hypervisor used for deploying, on top of an ARM-based MPSoC, a virtualized fault-tolerant MCS in the context of a real case study in the railway domain. We conducted a thorough empirical analysis by varying several real-time features provided by Xen applying DoE to systematically point out quantitative indicators about temporal guarantees.

8. Conclusion

In this work, we conducted a temporal assessment against a virtualized 2oo2-based MCS via the Xen hypervisor, which depicts a use case provided in the framework of an academic-industrial partnership, in the context of the railway domain. First, we proposed a general approach that, from the temporal isolation properties and the recommended mechanisms inferred by the safety-related standard of interest (e.g., EN50128), allows us to set up a testbed and perform a Design of Experiments (DoE). Those experiments (raw data and replication scripts publicly available) produce quantitative (e.g., overhead induced by the hypervisor to critical VMs) and qualitative (guidelines for the configuration of Xen) indicators, which can guide industry practitioners in using virtualization technologies in a real industry setting.

The experimental analysis confirms that temporal isolation is easily breakable, and can lead to timing failures that undermine the overall system's dependability. Therefore, our approach systematically reveals the knobs that should be carefully tuned for a hypervisor (Xen, in the presented case study), to lower the induced overhead and make it bearable in most industrial applications. In summary, we show how much a real-time scheduler of Xen (e.g., RTDS), can enhance temporal isolation compared to non-real-time schedulers like Credit2, reducing the increase of maximum execution time from +134% to +11% compared to the case without virtualization. Furthermore, our exploration of the Xen "null" scheduler, which includes no CPU sharing, showcases even further latency reduction potential. However, it is crucial to note that even with the best configuration ("null" scheduler), our experiments reveal that VMs are still susceptible to execution delays ranging from +31% to +271% under specific and intensive stress conditions. This underscores the importance of our comprehensive statistical analysis, which provides valuable insights for determining the suitability of virtualization for specific applications. Looking ahead, several new techniques and methodologies proposed in the literature are promising to enforce temporal isolation provided by Xen and other hypervisors strongly. This will open new rooms for future works, including applying the proposed approach to assess temporal isolation against alternative solutions such as static partitioning hypervisors (e.g., Jailhouse).

CRediT authorship contribution statement

Marcello Cinque: Writing – review & editing, Supervision, Project administration, Investigation, Funding acquisition, Conceptualization. **Luigi De Simone:** Writing – review & editing, Writing – original draft, Supervision, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **Daniele Ottaviano:** Writing – original draft, Software, Resources, Methodology, Data curation.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgments

This work has been carried out within the framework of the EUROfusion Consortium, funded by the European Union via the Euratom Research and Training Programme (Grant Agreement No 101052200 - EUROfusion) and within the MOST – Sustainable Mobility National Research Center and received funding from the European Union NextGenerationEU (PIANO NAZIONALE DI RIPRESA E RESILIENZA (PNRR) – MISSIONE 4 COMPONENTE 2, INVESTIMENTO 1.4 – D.D. 1033

17/06/2022, CN00000023). The work by L. De Simone was supported by European Union - FSE-REACT-EU, PON Research and Innovation 2014–2020 DM1062/2021 contract number 18-I-15350a-6. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Commission. Neither the European Union nor the European Commission can be held responsible for them.

References

Abeni, L., Faggioli, D., 2020. Using xen and kvm as real-time hypervisors. J. Syst. Archit. 106, 101709.

AEEC, 2010. ARINC-653: Avionics application software standard interface part 1.

Alonso, S., Lázaro, J., Jiménez, J., Muguira, L., Largacha, A., 2020. Analysing the interference of xen hypervisor in the network speed. In: Conference on Design of Circuits and Integrated Systems. IEEE, pp. 1–6.

Amendola, A., Barbareschi, M., De Simone, S., Mezzina, G., Moriconi, A., Saragaglia, C.L., Serra, D., De Venuto, D., 2023. A real-time vital control module to increase capabilities of railway control systems in highly automated train operations. Real-Time Syst. 59 (4), 636–661.

August, M., 2014. IDP: An Analysis of a Cache-Based Timing Side Channel Attack and a Countermeasure on PikeOS.

Avizienis, A., Laprie, J.-C., Randell, B., Landwehr, C., 2004. Basic concepts and taxonomy of dependable and secure computing. IEEE Trans. Depend. Secure Comput. 1 (1), 11–33.

Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., Warfield, A., 2003. Xen and the art of virtualization. SIGOPS Operating Systems Review 37 (5), 164–177.

Barletta, M., Cinque, M., De Simone, L., Corte, R.D., 2024. Criticality-aware monitoring and orchestration for containerized industry 4.0 environments. ACM Trans. Embedd. Comput. Syst. 23 (1), 1–28.

Barletta, M., Cinque, M., De Simone, L., Della Corte, R., Farina, G., Ottaviano, D., 2022. Runphi: Enabling mixed-criticality containers via partitioning hypervisors in industry 4.0. In: 2022 IEEE International Symposium on Software Reliability Engineering Workshops. ISSREW, IEEE, pp. 134–135.

Barletta, M., Cinque, M., De Simone, L., Della Corte, R., Farina, G., Ottaviano, D., 2023. Partitioned containers: Towards safe clouds for industrial applications. In: 2023 53rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks-Supplemental Volume (DSN-S). IEEE, pp. 84–88.

Baumann, C., Bormer, T., 2009. Verifying the pikeos microkernel: first results in the verisoft xt avionics project. In: Doctoral Symposium on Systems Software Verification. p. 20.

BlackBerry Limited, 2021. Blackberry limited., are hypervisors the answer to the coming silicon shortages?, https://blackberry.qnx.com/content/dam/blackberry-com/Documents/pdf/BlackBerry_QNX_Hypervisor_WhitePaper_22April2021_FINAL.pdf.

Burns, A., Davis, R.I., 2022. Mixed Criticality Systems-a Review:(February 2022). York.

CENELEC, EN 50128, 2011. Railway Applications-Communication, Signaling and Processing Systems-Software for Railway Control and Protection Systems.

Cesarano, C., Cotroneo, D., De Simone, L., 2022. Towards assessing isolation properties in partitioning hypervisors. In: 2022 IEEE International Symposium on Software Reliability Engineering Workshops. ISSREW, IEEE, pp. 193–200.

Cinque, M., Cotroneo, D., De Simone, L., Rosiello, S., 2022a. Virtualizing mixed-criticality systems: A survey on industrial trends and issues. Future Gener. Comput. Syst. 129, 315–330.

Cinque, M., De Simone, L., Mazzocca, N., Ottaviano, D., Vitale, F., 2023. Evaluating virtualization for fog monitoring of real-time applications in mixed-criticality systems. Real-Time Syst. 59 (4), 534–567.

Cinque, M., De Tommasi, G., Dubbioso, S., Ottaviano, D., 2022b. Rpuguard: Real-time processing unit virtualization for mixed-criticality applications. In: European Dependable Computing Conference. IEEE, http://dx.doi.org/10.1109/EDCC57035.2022.00025.

Crespo, A., Ripoll, I., Masmano, M., 2010. Partitioned embedded architecture based on hypervisor: The xtratum approach. In: European Dependable Computing Conference. IEEE, pp. 67–72.

Dall, C., Li, S., Lim, J.T., Nieh, J., Koloventzos, G., 2016. Arm virtualization: performance and architectural implications. In: Annual International Symposium on Computer Architecture. IEEE, pp. 304–316.

De Bock, Y., Mercelis, S., Broeckhove, J., Hellinckx, P., 2020. Real-time virtualization with xvisor. Int. Things 11, 100238.

De-RISC, 2019. De-RISC: Dependable Real-time Infrastructure for Safety-critical Computer. https://cordis.europa.eu/project/id/869945.

De Simone, Luigi, Di Mauro, Mario, Natella, Roberto, Postiglione, Fabio, 2024. Performance and availability challenges in designing resilient 5g architectures. IEEE Trans. Netw. Serv. Manag. 1–1. http://dx.doi.org/10.1109/TNSM.2024.3404560.

Directorate-General for Mobility and Transport, 2024. Subsystems and Constituents of the ERTMS. https://transport.ec.europa.eu/transport-modes/rail/ertms/what-ertms-and-how-does-it-work/subsystems-and-constituents-ertms_en.

Elphinstone, K., Heiser, G., 2013. From l3 to sel4 what have we learnt in 20 years of l4 microkernels? In: Symposium on Operating Systems Principles. pp. 133–150.

Fisher, R.A., 1936. Design of experiments. Br. Med. J. 1 (3923), 554.

FuSa SIG, 2022. FuSa SIG Charted, https://wiki.xen.org/wiki/FuSa_SIG/Charter.

FuSa SIG, 2022. FuSa SIG/Presentations, https://wiki.xenproject.org/wiki/FuSa_SIG/Presentations.

Ge, Q., Yarom, Y., Chothia, T., Heiser, G., 2019. Time protection: the missing os abstraction. In: European Conference on Computer Systems. pp. 1–17.

Hamelin, E., Hmid, M.A., Naji, A., Mouafo-Tchinda, Y., 2020. Selection and evaluation of an embedded hypervisor: Application to an automotive platform. In: European Congress of Embedded Real Time Software and Systems.

Heiser, G., Klein, G., Murray, T., 2019. Can we prove time protection? In: Workshop on Hot Topics in Operating Systems. pp. 23–29.

Henkel, J., 2006. Selective revealing in open innovation processes: The case of embedded linux. Res. Policy 35 (7), 953–969.

Hercules, 2020. High-performance real-time architectures for low-power embedded systems. https://cordis.europa.eu/project/id/688860.

Hermann, M., Pentek, T., Otto, B., 2016. Design principles for industrie 4.0 scenarios. In: Hawaii International Conference on System Sciences. IEEE, pp. 3928–3937.

HERMES2020, 2020. Qualification of High Performance Programmable Microprocessor and Development of Software Ecosystem. https://cordis.europa.eu/project/id/101004203.

Heron, S., 2009. Advanced encryption standard (AES). Netw. Secur. 2009 (12), 8–12.

Huang, X.-L., Ma, X., Hu, F., 2018. Machine learning and intelligent communications. Mob. Netw. Appl. 23, 68–70.

Hughes, A., Awad, A., 2019. Quantifying performance determinism in virtualized mixed-criticality systems. In: International Symposium on Real-Time Distributed Computing. IEEE, pp. 181–184.

International Electrotechnical Commission, 1998. Software Requirements, IEC 61508-3.

ISO, 2011. Product Development: Software Level, ISO 26262: Road vehicles – Functional safety, 6.

Jan, S., Shieh, G., 2014. Sample size determinations for welch's test in one-way heteroscedastic anova. Br. J. Math. Stat. Psychol. 67 (1), 72–93.

Jiang, Z., Yang, K., Ma, Y., Fisher, N., Audsley, N., Dong, Z., 2021. I/o-guard: Hardware/software co-design for i/o virtualization with guaranteed real-time performance. In: Design Automation Conference. IEEE, pp. 1159–1164.

Jiménez, J., Muguira, L., Bidarte, U., Largacha, A., Lázaro, J., 2022. Specific electronic platform to test the influence of hypervisors on the performance of embedded systems. Technologies 10 (3), http://dx.doi.org/10.3390/technologies10030065, https://www.mdpi.com/2227-7080/10/3/65.

Jo, Y.H., Choi, B.W., 2022. Performance evaluation of real-time linux for an industrial real-time platform. Int. J. Adv. Smart Convergence 11 (1), 28–35.

King, C.I., 2017. Stress-ng. http://kernel.ubuntu.com/git/cking/stressng.git/ (visited on 28/03/2018).

Klein, G., Elphinstone, K., Heiser, G., Andronick, J., Cock, D., Derrin, P., Elkaduwe, D., Engelhardt, K., Kolanski, R., Norrish, M., et al., 2009. sel4: Formal verification of an os kernel. In: Symposium on Operating Systems Principles. pp. 207–220.

Klingensmith, N., Banerjee, S., 2018. Hermes: A real time hypervisor for mobile and iot systems. In: International Workshop on Mobile Computing Systems & Applications. pp. 101–106.

Klingensmith, N., Banerjee, S., 2019. Using virtualized task isolation to improve responsiveness in mobile and iot software. In: International Conference on Internet of Things Design and Implementation. pp. 160–171.

Kloda, T., Solieri, M., Mancuso, R., Capodieci, N., Valente, P., Bertogna, M., 2019. Deterministic memory hierarchy and virtualization for modern multi-core embedded systems. In: ìReal-Time and Embedded Technology and Applications Symposium. IEEE, pp. 1–14.

Kurth, L., 2016. Xen project 4.7 released. https://lwn.net/Articles/692542/.

Leppinen, H., 2017. Current use of linux in spacecraft flight software. Aerosp. Electron. Syst. Mag. 32 (10), 4–13.

Likely, G., Boyer, J., 2008. A symphony of flavours: Using the device tree to describe embedded hardware. In: The Linux Symposium. vol. 2, pp. 27–37.

Lucas, P., Chappuis, K., Boutin, B., Vetter, J., Raho, D., 2018. Vosysmonitor, a trustzone-based hypervisor for iso 26262 mixed-critical system. In: Conference of Open Innovations Association. IEEE, pp. 231–238.

Lucas, P., Chappuis, K., Paolino, M., Dagieu, N., Raho, D., 2017. Vosysmonitor, a low latency monitor layer for mixed-criticality systems on ARMv8-a. In: Euromicro Conference on Real-Time Systems. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

Martins, J., Pinto, S., 2023. Shedding light on static partitioning hypervisors for arm-based mixed-criticality systems. arXiv preprint arXiv:2303.11186.

Martins, J., Tavares, A., Solieri, M., Bertogna, M., Pinto, S., 2020. Bao: A lightweight static partitioning hypervisor for modern multi-core embedded systems. In: Workshop on Next Generation Real-Time Embedded Systems (NG-RES 2020). Schloss Dagstuhl-Leibniz-Zentrum für Informatik.

Masmano, M., Ripoll, I., Crespo, A., Metge, J., 2009. Xtratum: a hypervisor for safety critical embedded systems. In: Real-Time Linux Workshop. Citeseer, pp. 263–272.

McFarland, J., Awad, A., 2022. Transpose-xen: virtualized mixed-criticality through dynamic allocation. In: SIGAPP Symposium on Applied Computing. ACM, pp. 3–12.

Patel, A., Daftedar, M., Shalan, M., El-Kharashi, M.W., 2015. Embedded hypervisor xvisor: A comparative analysis. In: Euromicro International Conference on Parallel, Distributed, and Network-Based Processing. IEEE, pp. 682–691.

PikeOS, 2024. PikeOS product overview. https://www.sysgo.com/fileadmin/user_upload/data/flyers_brochures/SYSGO_PikeOS_Product_Overview.pdf.

QEMU, 2024. IVSHMEM Documentation page, https://www.qemu.org/docs/master/system/devices/ivshmem.html.

QEMU, 2024. [Homepage of QEMU]. https://www.qemu.org/.

Reghenzani, F., Massari, G., Fornaciari, W., 2019. The real-time linux kernel: A survey on preempt_rt. Comput. Surv. 52 (1), 1–36.

Rete Ferroviaria Italiana (RFI), 2004. Schema di riferimento per lo sviluppo delle logiche acc. Tech. rep. Rete Ferroviaria Italiana (RFI).

RTCA, 1992. DO-178B Software Considerations in Airborne Systems and Equipment Certification. Requirements and Technical Concepts for Aviation.

RTCA, 2011. DO-178C - Software Considerations in Airborne Systems and Equipment Certification.

Sabogal, D., George, A.D., 2018. Towards resilient spaceflight systems with virtualization. In: Aerospace Conference. IEEE, pp. 1–8.

Schulz, B., Annighöfer, B., 2022. Evaluation of adaptive partitioning and real-time capability for virtualization with xen hypervisor. Trans. Aerosp. Electron. Syst. 58 (1), 206–217. http://dx.doi.org/10.1109/TAES.2021.3104941.

SELENE, 2019. SELENE: Self-monitored Dependable platform for High-Performance Safety-Critical Systems. https://cordis.europa.eu/project/id/871467.

Shift2Rail, 2024. Home page of Shit2Rail projects, https://projects.shift2rail.org/s2r_projects.aspx.

Siemens AG, 2024. Jailhouse hypervisor source code. https://github.com/siemens/jailhouse.

Sinha, S., West, R., 2021. Towards an integrated vehicle management system in driveos. Trans. Embedd. Comput. Syst. 20 (5s), 1–24.

St, L., Wold, S., et al., 1989. Analysis of variance (anova). Chemometr. Intell. Laboratory Syst. 6 (4), 259–272.

Stabellini, S., 2014. Xen arm with virtualization extensions white paper.

Suzuki, N., Kim, H., De Niz, D., Andersson, B., Wrage, L., Klein, M., Rajkumar, R., 2013. Coordinated bank and cache coloring for temporal protection of memory accesses. In: International Conference on Computational Science and Engineering. IEEE, pp. 685–692.

The Linux Foundation, 2015. XenBus. https://wiki.xenproject.org/wiki/XenBus.

The Linux Foundation, 2018. The automotive grade linux software defined connected car architecture. White Paper, https://www.automotivelinux.org/wp-content/uploads/sites/4/2018/06/agl_software_defined_car_jun18.pdf.

The Linux Foundation, 2022a. Arinc 653 scheduler - xen. https://wiki.xenproject.org/wiki/ARINC653_Scheduler.

The Linux Foundation, 2022b. Technical details of preempt_Rt patch. https://wiki.linuxfoundation.org/realtime/documentation/technical_details/start.

The Linux Foundation, 2023. Xen Project 4.18 Feature List. https://wiki.xenproject.org/wiki/Xen_Project_4.18_Feature_List.

The Linux Foundation, 2024. Cyclictest. https://wiki.linuxfoundation.org/realtime/documentation/howto/tools/cyclictest/start.

The Linux Foundation, 2024. Homepage of LF Edge Foundation, https://elisa.tech/.

Toumassian, S., Werner, R., Sikora, A., 2016. Performance measurements for hypervisors on embedded arm processors. http://dx.doi.org/10.1109/ICACCI.2016.7732152.

Tran, L., Radcliffe, P.J., Wang, L., 2021. Simulation is essential for embedded control systems with task jitter. Des. Autom. Embedded Syst. 25, 177–191.

Verbeek, F., Havle, O., Schmaltz, J., Tverdyshev, S., Blasum, H., Langenstein, B., Stephan, W., Wolff, B., Nemouchi, Y., 2015. Formal api specification of the pikeos separation kernel. In: NASA Formal Methods Symposium. Springer, pp. 375–389.

West, R., Li, Y., Missimer, E., Danish, M., 2016. A virtualized separation kernel for mixed-criticality systems. ACM Trans. Comput. Syst. (TOCS) 34 (3), 1–41.

Wiki.Xenproject, 2019. Xen wiki - rtds-based-scheduler. https://wiki.xenproject.org/wiki/RTDS-Based-Scheduler.

WindRiver Systems Inc., 2016. Virtualization and the internet of things. WindRiver White Paper, p. 4, https://www.windriver.com/whitepapers/iot-virtualization/1436-IoT-Virtualization-White-Paper.pdf.

Wolf, W., Jerraya, A.A., Martin, G., 2008. Multiprocessor system-on-chip (MPSoC) technology. IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 27 (10), 1701–1713.

Xi, S., Wilson, J., Lu, C., Gill, C., 2011. Rt-xen: Towards real-time hypervisor scheduling in xen. In: International Conference on Embedded Software. IEEE, pp. 39–48.

Xu, M., 2013. Rt-xen: Real-time virtualization based on xen. https://sites.google.com/site/realtimexen/home/rt-xen-2-2-is-released.

Ye, Y., West, R., Cheng, Z., Li, Y., 2014. Coloris: a dynamic cache partitioning system using page coloring. In: International Conference on Parallel Architecture and Compilation Techniques. IEEE, pp. 381–392.

Yun, H., Yao, G., Pellizzoni, R., Caccamo, M., Sha, L., 2013. Memguard: Memory bandwidth reservation system for efficient performance isolation in multi-core platforms. In: Real-Time and Embedded Technology and Applications Symposium. IEEE, pp. 55–64.

Zhang, Z., Liu, Y., Chen, J., Qi, Z., Zhang, Y., Liu, H., 2021. Performance analysis of open-source hypervisors for automotive systems. In: International Conference on Parallel and Distributed Systems. pp. 530–537. http://dx.doi.org/10.1109/ICPADS53394.2021.00072.

Zuepke, A., Bastoni, A., Chen, W., Caccamo, M., Mancuso, R., 2023. Mempol: Policing core memory bandwidth from outside of the cores. In: 2023 IEEE 29th Real-Time and Embedded Technology and Applications Symposium. RTAS, IEEE, pp. 235–248.

Marcello Cinque is an Associate Professor at the University of Naples Federico II, Italy, where he teaches operating systems and real-time systems. He earned his master with honors and his PhD in Computer Engineering at the University of Naples in 2003 and 2006, respectively. His research interests are in virtualized mixed-criticality systems and dependability analysis. He is actively involved in the dependability community, chairing conferences, such as EDCC, and PRDC. He is or has been PI of several private or public-funded research projects on dependable industrial systems.

Luigi De Simone (Ph.D.) is an assistant professor at the University of Naples Federico II, Italy. His research interests include dependability benchmarking, fault injection testing, virtualization reliability and its application on safety- and secure-critical systems. He contributed, as author and reviewer, to several top journals and conferences on dependable computing and software engineering, and he has been organizing multiple editions of the international workshop on software certification (WoSoCer) within the IEEE ISSRE conference.

Daniele Ottaviano is a PhD candidate enrolled in the Fusion Science and Engineering (FSE) Doctoral Programme jointly offered by the University of Padova and the University of Naples Federico II. His research centers on virtualization technologies tailored for real-time and embedded systems. He is especially interested in hypervisor-level resource management to create reliable mixed-criticality systems on advanced Multiprocessor Systems-on-Chip (MPSoCs) leveraging heterogeneous resources and reprogrammable hardware such as FPGAs.