Real-Time Virtualization For Industrial Automation

Conference Paper · September 2020

DOI: 10.1109/ETFA46521.2020.9211890

7 authors, including:

Andrea Tagliavini
Università degli Studi di Modena e Reggio Emilia
4 PUBLICATIONS **13** CITATIONS
SEE PROFILE

Marko Bertogna
Università degli Studi di Modena e Reggio Emilia
140 PUBLICATIONS **3,395** CITATIONS
SEE PROFILE

Marco Solieri
Università degli Studi di Modena e Reggio Emilia
21 PUBLICATIONS **245** CITATIONS
SEE PROFILE

Real-Time Virtualization For Industrial Automation

Claudio Scordino*, Ida Maria Savino*, Luca Cuomo*,
Luca Miccio†, Andrea Tagliavini†, Marko Bertogna†, Marco Solieri†
**Evidence Srl*, Pisa, Italy - {c.scordino, i.savino, l.cuomo}@evidence.eu.com
†*Universitá di Modena e Reggio Emilia*, Italy - {{206497,204105}@studenti., marko.bertogna@, marco.solieri@}unimore.it

***Abstract*—The industry has recently shown a growing interest in running non-critical activities (e.g., design tools) together with real-time control tasks on open-source COTS platforms.**

In this paper, we present a modular platform for industrial automation based on open-source components. Thanks to the isolation provided by a hypervisor, the same platform can run both the real-time control code and the design tools, reducing the overall hardware costs.

Besides illustrating the software architecture and describing the various open-source components, we illustrate extensions needed for convenient applicability, and we address the problem of interference on hardware resources shared by multiple operating systems, which undermines the performance of the real-time control. A set of experimental results show the effectiveness and the performance of the proposed solution.

***Index Terms*—Industrial automation, real-time, multi-core, virtualization**

I. Introduction

During the recent years, the industry has shown a growing interest in running activities with different levels of criticality (i.e. "mixed-criticality") on the same hardware platform. These can consist, for example, of non-critical activities (e.g., monitoring, logging, human-machine interfaces, multimedia, data backup) together with real-time control tasks. Two key factors that have recently contributed in amplifying this interest from the industry are the availability of multi-core processors, and hardware-assisted virtualization even on Commercial Off-the-Shelf (COTS) platforms. This approach can be particularly beneficial for industrial automation [11], where non-critical design tools could be executed along with the real-time control code.

Another topic of high interest relates to the desire of using open-source software for freeing from the traditional commercial solutions, which often impose expensive royalties on system integrators and end-users. Due to its popularity and portability, the industry has often seen Linux as a possible alternative to such commercial systems. Unfortunately, however, the Linux kernel was originally designed to be general-purpose, thus its main design goal has concerned the optimization of the average throughput rather than meeting the timing constraints of the running applications. As a result, the standard Linux kernel does not provide performance suitable for a real-time control with tight timing constraints.

In this paper, we present two main contributions.

1) We introduce a novel *system software platform for industrial automation* composed by a hypervisor and a Real-Time Operating System (RTOS) that aims at tackling both the aforementioned objectives.
    a) It is composed of open-source software and follows a modular approach, that allows substituting or modifying a subset of the system in a flexible way.
    b) Thanks to the isolation provided by an hypervisor, the platform provides mixed-criticality, allowing running concurrently both the real-time control code and the design tools, reducing the overall hardware costs.
2) We extended the two platform components in order to have: fieldbus support in the RTOS, and state-of-the-art isolation technologies in the hypervisor.
3) We provide *extensive evaluation of the platform* and the platform components' key features.

The rest of the paper is organized as follows. Section II provides the background of real-time support on general-purpose open-source operating systems (e.g. Linux). Section III illustrates the overall software architecture and describes the various open-source components and their configuration. Sections IV and V validate the proposed solution through a set of experimental results. Finally, Section VI draws the conclusions.

II. Related work

In the course of the years, several open-source projects have aimed at creating a real-time version of the Linux operating system. These attempts can be classified in the following categories [24].

A. Real-time Linux

These projects have aimed at increasing the predictability of the standard mainline kernel. The first huge boost to kernel predictability was given by Robert Love's *Preemptible Kernel* patch [21]. This mechanism, integrated into the 2.6 kernel series, has made the kernel fully preemptible, allowing a context switch to occur even when a process is executing in kernel context.

Then, the popular PREEMPT_RT project [2] has aimed at shortening the maximum latency experienced by a real-time task by increasing timers' accuracy and kernel concurrency and reducing non-preemptible sections. Several important parts of the project (e.g. high-resolution timers) have been periodically integrated into the mainline kernel. The project has been

This work has been supported by the European Commission through the I-MECH ECSEL project (GA-737453).

recently sponsored by the Linux Foundation and is expected to merge the remaining parts in the very next releases. Very recently, De Oliveira [16] has proposed an automata-based model for describing and validating the behavior of threads in PREEMPT_RT on single-core systems.

In parallel, the Linux kernel community has implemented a real-time CPU scheduler, called SCHED_DEADLINE [23], available by default in the official Linux kernel since release 3.14. Unlike the traditional fixed-priority POSIX-compliant SCHED_FIFO and SCHED_RR policies, SCHED_DEADLINE allows to specify the amount of CPU time reserved to a real-time task with a *per-task* timing granularity.

Despite these projects have certainly contributed in improving the responsiveness of Linux, they have not been capable of reaching the performance needed for the fast control loops implemented nowadays in industrial automation.

B. Dual kernel

The "dual-kernel" approach consists of creating a Hardware Abstraction Layer (HAL) and modifying the interrupt handling routines for executing a tiny real-time operating system (RTOS) prior to Linux. The Linux kernel and its tasks are thus executed at a lower priority than the real-time tasks. This approach, originally introduced by RT-Linux [4], [35], has been then evolved by the RTAI [3] and Xenomai [7] projects, introducing the concept of "execution domains" for partially overcoming the original limitations of kernel-space programming.

Despite these projects have successfully proven the feasibility and the performance of the proposed approach [10], they have been unable to get enough visibility and momentum to become widespread.

C. Partitioning hypervisor

The most recent research direction in various application domains (e.g. automotive) consists in taking advantage of hardware-assisted virtualization of modern processors for allocating different physical cores to the execution of components with different timing characteristics. This novel approach explains the renewed interest from industry about hypervisor technologies which, originally designed for mainframes and cloud computing, are now being used also on COTS hardware and operating systems.

Xen [6] is a type-1 open-source hypervisor that can run using both plain hardware virtualization or exploiting paravirtualization offered by a Linux VM. Thanks to the device emulation provided by Qemu, it can also run unmodified versions of operating systems (e.g. Windows). RT-Xen [33], [34] was an attempt of creating a real-time version of Xen. More recently, the community developing the Xen hypervisor has implemented the "null" scheduler [8], which statically assigns each virtual CPU to one and only one physical CPU, avoiding any scheduling decision — therefore, reducing the overhead.

Xen is a type-1 hypervisor, therefore running on top of the hardware, without the need of any host OS. However, to simplify the design and avoid the need of device drivers in the hypervisor itself, Xen relies on a special privileged guest called "dom0", usually consisting of a Linux OS. This approach prevents the hypervisor to boot normal guests (called "domU") before the Dom0 has booted. Unfortunately, this constraint represents an issue in the embedded domain, where a short boot time of the guests is often a mandatory requirement. A very recent attempt sponsored by Xilinx, therefore, aims at removing the need of Xen's "dom0" domain at least for ARM processors [5].

KVM [1] is the standard support for adding an hypervisor layer to the Linux kernel. The virtual machines are seen as processes and scheduled by the Linux default scheduler. A recent work has shown that KVM can obtain lower worst-case latencies than Xen [8]. Some kernel contributors have also worked for adapting KVM to real-time workloads [36].

In parallel, Siemens has developed a tiny open-source hypervisor, Jailhouse [30], targeting the embedded safety-critical domains. Very recently, the Bao hypervisor has been proposed [28] to deliver a virtually transparent, and highly secure partitioning layer for the most critical situations.

D. Isolating hypervisor

Contention on shared hardware resources, such as DRAM, bus or, most appreciably, caches, undermines real-time performance in a mixed-criticality context. This has been extensively studied by Cavicchioli et al. [13] and Danielsson et al. [15], and also discussed by Capodieci et al. in the context of automotive applications [12]. Hardware solutions for cache management exist, are either rigid and obsoleted, like cache lockdown support in Arm v7, or virtually not available among the industrial computing platform segment, such as the server-grade Intel Cache Allocation Technology (CAT), part of Resource Director Technology, or Arm's Memory System Resource Partitioning and Monitoring (MPAM).

This has been the motivation for elevating the hypervisor concept up to the hardware isolation role, which has been shown to be effective in restoring memory access determinism in high-performance embedded systems. Software cache partitioning has been demonstrated to be a key feature for real-time systems [22] and military applications [31], and it has been proposed on the Arm implementation of Jailhouse [25], Xen [26], and Bao [28].

E. Industrial automation

Most of the work in the literature for industrial automation has exploited virtualization only for cloud and fog computing [19]. Among the notable exceptions, Mahmud et al. [27] evaluated the feasibility of Xen-based virtualization in a multi-core distributed system running RT-Linux. Their work provided an insightful understanding of the additional overhead introduced by the virtualization in terms of both latency and jitter. However, it did not leverage an RTOS for improving the overall performance. Azarmipour et al. [9], instead, discussed design issues of a virtualized system based on PikeOS, a commercial hypervisor not available as open-source software.

Moreover, both these works did not measure the performance of field-buses like EtherCAT.

III. Software architecture

The software architecture has been designed, and its components selected, in order maximize applicability and relevance in industrial applications. In particular, we followed the following requirements, gathered by dozens of partners in the I-MECH European project [14].

RQ1 Working on COTS x86 platforms
RQ2 Supporting EtherCAT master functionalities
RQ3 Supporting EtherCAT communications at a rate of at least 20 kHz
RQ4 Capable of concurrent execution of multiple OSs
RQ5 Capable of executing unmodified version of the Microsoft Windows OS for running commercial design tools (e.g. Simulink)
RQ6 Capable of executing a RTOS
RQ7 Supporting partitioning of resources (e.g. CPUs, memory)
RQ8 Enforcing freedom-from-interference between the different OSs

The overall software architecture is illustrated in Figure 1.

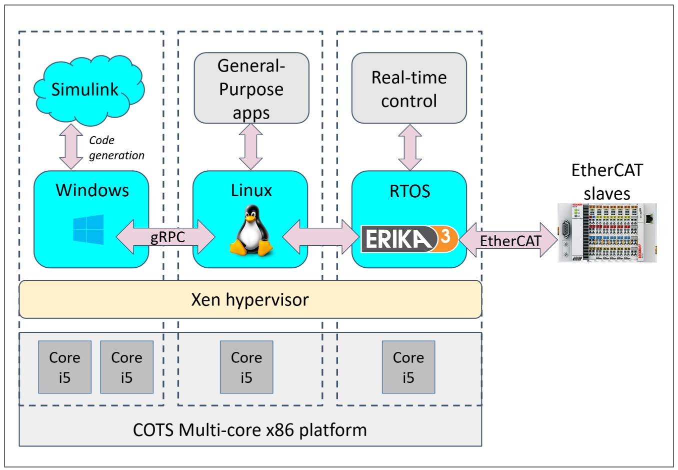


Fig. 1. Overall software architecture.

We will now describe the single software components, also illustrating their specific configuration.

A. Hypervisor

A hypervisor is a key software components for building a complex and modular architecture, because of two main characteristics.

a) Hardware abstraction: the system layer, i.e. the OS or the bare-metal application, can be decoupled from the hardware layer, allowing for example running a legacy component on a supposedly unsupported machine.

b) Multiple OS: large hardware with multi-core and large DRAM availability can host several virtual machines (VMs) enabling consolidation of legacy systems in a cheaper one.

1) Product selection: When selecting the open-source hypervisor technology, we mainly had the following three options: Xen, KVM, Jailhouse and ACRN.

Being oriented to real-time and safety-criticality, the lightweight hypervisor ACRN could be ideal. Nevertheless, the project's youth presents risks of development and maintenance interruption in case of abandonment. Also the limited spread in the market increases the cost of initial adoption, due to the lack of experience.

Thanks to its simplicity, static approach and lack of emulation, Jailhouse could have easily provided the best performance in terms of overhead and latency. However, the need for running unmodified versions of the Windows (RQ5) operating system prevented from selecting such a simple hypervisor.

The selection between Xen and KVM has been more complex, due to their similarities. In the end, the future possibility of running a system without Linux (i.e. "dom0less") has been in favor of the Xen hypervisor. This choice has been also strengthened by the experimental results provided in Section IV-A, aiming at comparing the performance of these three hypervisors when running an RTOS.

2) Configuration: Concerning the Xen configuration, we selected the null scheduler to avoid any scheduling introduced overhead, thus renouncing to the possibility of CPU overprovisioning, i.e. the possibility of having a number of VM greater than cores. Also, we configured the I/O layer for Hardware Virtualization Mode (HVM) and not for paravirtualization (PV), because the latter, although faster on average than the former, relies on an additional Linux VM, which makes it much slower in the worst cases.

3) Cache partitioning extension: To deliver freedom from inter-core interference on the shared Last-Level Cache (LLC), the cache coloring feature has been implemented in the x86 version of Xen. The principle is depicted in Fig. 2: pages are partitioned in a way such that distinct partitions, called colors, map to distinct cache sets. LLC colors are assignable to virtual machines, and a disjoint color assignment thus guarantees cache isolation between any two Virtual Machines (VMs). This, for instance, allows critical domains to be separated between each other and from non-critical domains, while the latter ones can safely share a set of common colors. It is important to remark that the solution is completely transparent to the guest OS or application, being the solution implemented inside the hypervisor, essentially by re-implementing the page allocator. The virtualization extensions indeed provide a second translation stage, after the application-OS level. Section V will specifically evaluate the impact of this feature for the rest of the application.

B. Real-time operating system

ERIKA Enterprise [17] is an open-source RTOS originally designed for the automotive domain, and thus characterized by the simple OSEK/VDX API. The RTOS supports several types of processors (including Infineon AURIX, ARM Cortex-A/-M/-R and x86-64) and has been successfully used also in the household appliance and HVAC domains.

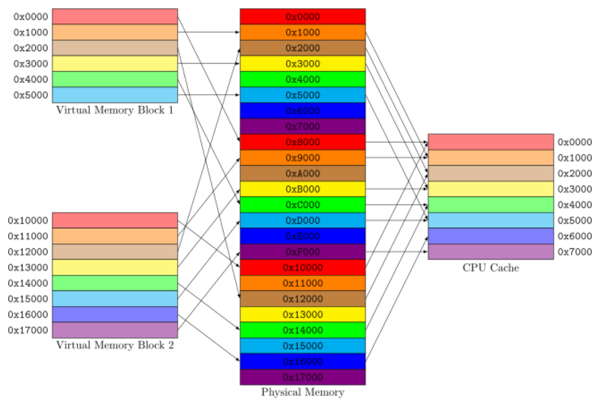


Fig. 2. Cache coloring principle

We started by porting this RTOS as DomU of the Xen hypervisor. We then implemented the device driver for the specific Ethernet network card (i.e. Intel i210 mounted on the PCI Express bus). For best performance, we used the device in polling mode. Finally, we have implemented the X2APIC mode of the interrupt controller in the RTOS for reducing the latency of interrupt dispatching when running on top of the hypervisor.

Note that ERIKA Enterprise is not the only RTOS that we have ported on the Xen hypervisor: aiming at proving the flexibility of the proposed approach, we also ported Windriver VxWorks version 6.9. This flexibility has been specifically used in the I-MECH project for supporting pilots made by different vendors. Although interesting, a performance comparison between ERIKA Enterprise and VxWorks would have been out of the paper scope.

C. *Fieldbus communication*

In the proposed platform, we decided to rely on the EtherCAT open protocol. For supporting such a fieldbus, we integrated the open-source SOEM EtherCAT master stack [29]. The porting activity has been successful, and the ERIKA support has been merged upstream to the original project. Currently, the stack runs in single-thread mode, while the multi-thread extension will be implemented in the next future.

D. *Inter-guest communications*

As shown in Figure 1, external communications towards the ERIKA Enterprise RTOS go through Linux. This has been a specific design choice to avoid an additional Ethernet peripheral or some unpredictable bridging of the Ethernet cards.

On Linux, a daemon implemented using the C++ programming language is in charge of "bridging" the communications to/from the ERIKA Enterprise RTOS. The API towards the external world (e.g., Windows) is based on Google gRPC and Protocol Buffers cross-platform open-source technologies [20]. Besides handling code generation (see next paragraph), the API also allows run-time monitoring as well as setting of variables and signals of the ERIKA control application.

For the inter-guest communication between Linux and ERIKA Enterprise, we needed a fast mechanism with a simple API allowing exchanging data easily and with a low overhead over a set of channels. We decided to remain aligned with the original nature of the RTOS, implementing an API similar to the AUTOSAR COM standard [1]. Each channel is unidirectional and can handle multiple signals (i.e. messages). The API for sending/receiving is straightforward:

- `uint8 Com_SendSignal(Com_SignalIdType SignalId, const void *SignalDataPtr);`
- `uint8 Com_ReceiveSignal(Com_SignalIdType SignalId, void *SignalDataPtr);`

Similar functions allow to send/receive signals with dynamic length or to block the caller until the operation has been completed.

Under the hood the mechanism works as follows (see Figure 3 for reference). When the ERIKA DomU starts, it allocates in its shared memory space a set of circular buffers (one for each channel). It then saves in the XenStore the specific information about their addresses, so that the Dom0 (Linux) can remap the same memory space and create the file descriptors. Additionally, an ERIKA ISR is bound to the reception event, happening when the Linux OS sends some data to the ERIKA RTOS.

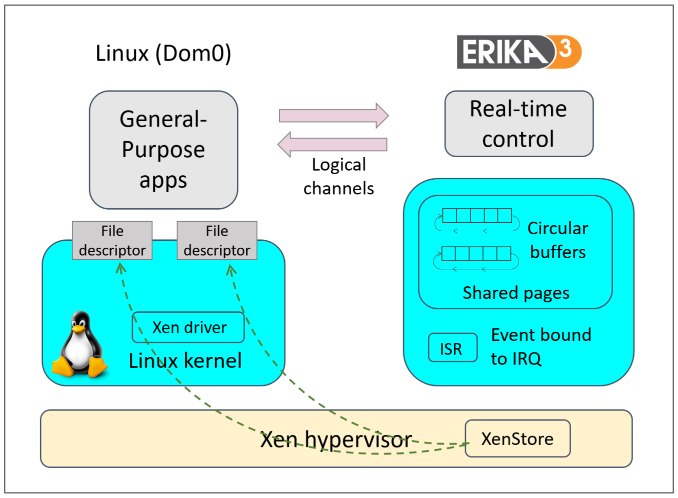


Fig. 3. Inter-guest communication mechanism.

E. *Model-based design tool*

Good software engineering practices suggest the adoption of model-based design methodologies and tools for preventing the typical errors of manually written code. This is also true for the I-MECH project, where model-based engineering has been one of the founding principles.

For the presented framework, we relied on Mathworks Simulink version R2012a, using the *Embedded Coder* plugin

[1]The development has been partly supported by the RETINA Eurostars Project (GA-E10171).

for automatic code generation. A specific toolbox has been designed to handle and generate the code for EtherCAT communications. A set of scripts in Python, based on gRPC and invoked by Simulink, allow to send the generated files to the Linux OS. On the Linux side, the C++ daemon is in charge of receiving the generated files, perform the last step of compilation (through the *gcc* compiler), and finally create and start the guest VM.

IV. General evaluation

In this section, we provide a set of experimental results that show the effectiveness of the proposed design. The hardware platform is an Advantech ARK-3520P, featuring a quad-core Intel i5-6440EQ processor with VT-x, VT-d and EPT instructions for virtualization. The Ethernet peripheral consists of an Intel i210 network card.

A. Performance of the hypervisor

For evaluating the overhead introduced by the hypervisor and the operating system, we have implemented a simple benchmarking suite that measures the time needed by the ERIKA RTOS for performing a set of critical scheduling activities (e.g., task activation time, task exit time, ISR call time, etc.). The test suite allows to benchmark the two types of interrupt service routines available in OSEK/VDX kernels (i.e. ISR1 and ISR2).

The tests have been performed on the mentioned hardware platform, keeping the cores at the maximum performance (i.e. using the `performance` governor). For best accuracy, times have been measured using the high-precision Time Stamp Counter (TSC) available on the x86 architecture (read through the `rdtsc` assembly instruction).

The test suite is modular and extensible, so that new tests can be easily added or removed. The tests currently implemented are, namely:

- `act`: activates a higher priority task and measures how long it takes to start its execution.
- `actl`: activates a low priority task and measures how long it takes to return to the caller.
- `intdisable`: measures the time needed for disabling all interrupts.
- `intenable`: measures the time needed for enabling all interrupts.
- `isrentry`: measures the time elapsed between the occurrence of an interrupt and the execution of the related ISR1 handler.
- `isr2entry`: measures the time elapsed between the occurrence of an interrupt and the execution of the related ISR2 handler.
- `isrexit` : measures the time elapsed between the end of an interrupt handler and when the task previously running resumes execution.

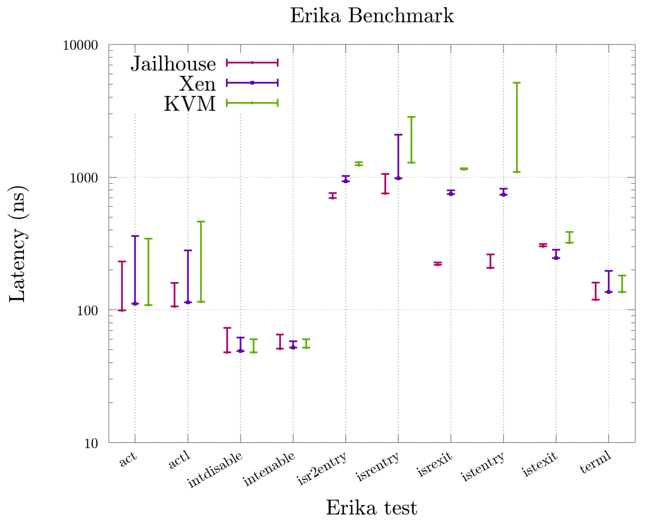


Fig. 4. ERIKA benchmarking experiment results: minimum and maximum latency with different hypervisors. Vertical axis is logarithmic.

- `istentry`: measures the time elapsed between the end of an interrupt handler and the execution of the task activated by such interrupt handler.
- `istexit`: measures the time elapsed between the end of a task handling an interrupt and when the task previously running resumes execution.
- `terml`: measures the time needed for terminating a task and switching to a lower priority one.

Fig. 4 shows the best-case and the worst-case times (expressed in nanoseconds) measured over 100 consecutive runs using Jailhouse, Xen and KVM. The values are absolute, thus they include both the time for the RTOS activity and the overhead introduced by the hypervisor. The versions of Xen, Jailhouse and the Linux kernel are 4.11, 0.9.1 and 4.15, respectively. Xen has been configured for using the null scheduler. In the case of KVM, instead, we isolated a core from the rest of the system (through the `isolcpus` kernel option) and scheduled the guest VM on such core.

The fact that Jailhouse could provide the best performance was largely expected. However, this hypervisor has not been selected due to the requirement RQ5 about the execution of unmodified versions of the Windows operating system. Then, the difference in performance between Xen and KVM has justified the choice of the former hypervisor for our real-time platform. The different result about Xen vs KVM performance with respect to the work by Abeni et al. [8] is given by the fact that they tested the Linux OS (under stress conditions), while we have measured the performance of a RTOS (running in isolation on a different core).

B. Performance of the EtherCAT communication

We have then implemented a set of test for measuring the performance of the system when controlling a Beckhoff EtherCAT slave.

The first test measured the Round-Trip Time (RTT) for the communication between our master platform and the EtherCAT slave. Since there is no other platform where the ERIKA RTOS supports EtherCAT communications, the performance has been compared against the same application and master stack running on a Ubuntu Linux distribution on the same hardware. In the case of Linux, the test has been also repeated for different numbers of threads of the EtherCAT stack. Table I shows the best-case and worst-case times, respectively, when sending 1000 samples.

TABLE I
ROUND-TRIP TIME FOR COMMUNICATION WITH AN ETHERCAT SLAVE (VALUES IN μS).

Operating System	Number of threads	Min RTT	Max RTT
ERIKA	1	45	55
Linux	1	49	246
Linux	2	47	202
Linux	3	47	165

As expected, the worst-case RTT with ERIKA is almost one order of magnitude lower than using a general-purpose operating system like Linux.

A further test aimed at measuring the throughput (in terms of EtherCAT messages) sustainable by the proposed platform. In case of ERIKA connected to the mentioned EtherCAT slave, we have measured a maximum throughput of 22 Kpackets/second, which respected requirement RQ3.

V. ISOLATING PROPERTIES EVALUATION

In this section, we specifically discuss and evaluate the hypervisor capabilities of ensuring cache isolation between CPU cores.

The main risk we want to handle is represented by the contention that any two cores in a multi-core architecture experience on shared caches. The cache hierarchy contributes with up to three orders of magnitude improvement with respect to DRAM.

More explicitly, we have performance of one core often depending on availability of code and/or data in the cache hierarchy. But cached lines in the shared LLC may be evicted by a noisy neighbor core, which can legitimately and benevolently have high memory requirements. Sha et al.i [32] have shown that an application can easily experience a 6x of its WCET as the number of competing tasks on a multi-core platform increases. This either forces the application architect to dramatic hardware underutilization, in order to meet the required determinism, or brings a potentially dangerous degree of non-determinism to the application.

A. Impact of cache contention

To evaluate relevance of cache contention, we return to the experimental setup previously introduced. The target industrial PC platform's CPU, a quad-core Intel i5-6440EQ with Skylake microarchitecture, is equipped with a three-level hierarchy: core-dedicated 32 KiB 8-way set associative L1 instruction/data caches, core-dedicated 256 KiB 4-way set associative L2 caches, shared 6 MiB 12-way set associative L3 cache.

1) Interference and microbenchmark: We implemented an *interference* application for VxWorks v6.9 and Linux 4.19 that stresses both the read and write memory channels with continuous and contiguous activities on a 20 MiB contiguous array. A `memcpy`, essentially. We also implemented a microbenchmark application called *membench*, that computes the average time needed to perform a sequence of read operations, following a sequential or pseudo-random access pattern, on a variable-depth working set size (from 1 KiB to 96 MiB). We measured membench results under two circumstances: isolated as the sole VM hosted on Xen ('base' for short) and in presence of three instances (2x VxWorks and 1x Linux) of the interference app ('3int'). Results are plotted in the blue and magenta curves in Fig. 5.

In both base curves we observe three latency plateaux, corresponding to the three cache levels. Comparing the sequential/random curves, we appreciate how, at the highest memory depth, the continuous increase in the bus width allows for greater parallelism, which greatly reduces the apparent latency in the contiguous case. More importantly, we measure how significant the performance degradation caused by interference can be — worst cases happen between 1 MiB and 6 MiB, where the slowdown reaches 7x and 11x in the sequential and random case, respectively.

2) ERIKA RTOS: We repeated the ERIKA benchmark introduced in Sec. IV-A with the same triple trouble setup '3int'. To ease comparison with the baseline isolated setup, relative performance data is reported.

The three most exposed routines are `act`, `isr2entry` and `act1` — the only ones over 1.5x, and ranging between 4x and 6.5x slowdown. The cache hit ratio for code and data belonging to the ERIKA test and to the underlying Xen routines (e.g. interrupt dispatching for the appropriate VM) degraded because of the continuously evicting interference applications. The system responsiveness would thus become unacceptable for many hard-real time applications.

B. Isolating hypervisor

Thanks to the cache coloring support, we can confine the three interference applications in a cache partition taking 12 out of 18 available cache colors, i.e. in a 4 MiB cache region. This leaves us 6 colors, that are 2 MiB, for the application under test.

Results for both membench and ERIKA benchmark in the isolated colored case 'base+col' show that coloring does not come at any unexpected price. Comparing green curves with the base ones in Fig. 5, or looking at the green bars in Fig. 6, the only visible drawback is represented by the smaller cache availability that can be observed in membench plots, where the application start experiencing capacity misses after 2 MiB, expectedly.

The most interesting case, instead, is when coloring comes into play, i.e. when cache partitioning is crucial to determine isolation. In membench results we observe that, when the coloring configuration is in place, the presence of interference is completely irrelevant. Latency is unaffected.

Also in ERIKA benchmark we record a successful validation. Violet bars in Fig. 6 are now considerably shorter than the '3int' blue counterpart, and enable many use cases to be designed where it seemed impossible. We cannot be as optimistic (yet) for ERIKA benchmark, though, since we still did not reached the ideal unitary relative-distance from the distance. Worst cases are again: `actls`, `act` and `isr2entry` (respectively).

The reason of this behavior is that the coloring support is able to isolate VMs from another, but is not (yet) capable to isolate the hypervisor itself. This leaves some critical portion of the system exposed, such as the interrupt injector. The cache coloring support for Xen on Arm [26] and for Bao [28] already implemented and experimented hypervisor coloring.

VI. Conclusions

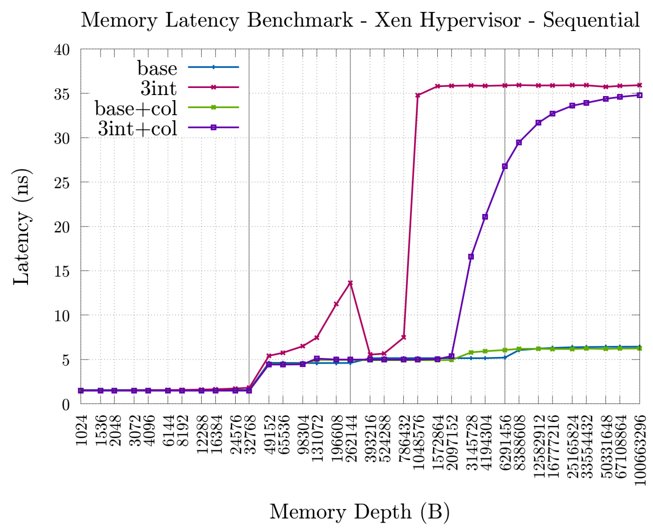
In this paper, we have shown how hypervisor-based virtualization can be effectively used for creating an innovative controller for industrial automation. With respect to other previous works, the major blocks of our framework consist exclusively of open-source software which is available online with the presented extensions.

- ERIKA Enterprise RTOS (GPLv3)
  https://github.com/evidence/erika3
- SOEM EtherCAT master stack (GPLv2)
  https://github.com/OpenEtherCATsociety/SOEM
- Xen with cache coloring (GPLv2)
  https://git.hipert.unimore.it/rtes/xen

Thanks to the high modularity, the overall system offers a degree of flexibility not to be found in commercial solutions.

We have illustrated the issues about interference on shared hardware resources, providing mechanisms for limiting or even preventing such interference. The experimental results have shown that with this approach it is possible to reach real-time performance suitable for the target domain. A video on Youtube [18] has been created for showing the full workflow, from code generation to the controller execution.

In the future, we will improve performance by implementing the multi-thread support for the EtherCAT stack, and by extending coloring support to the hypervisor. Furthermore, we will extend applicability by porting the solution on heterogeneous SoCs Arm-based platform.



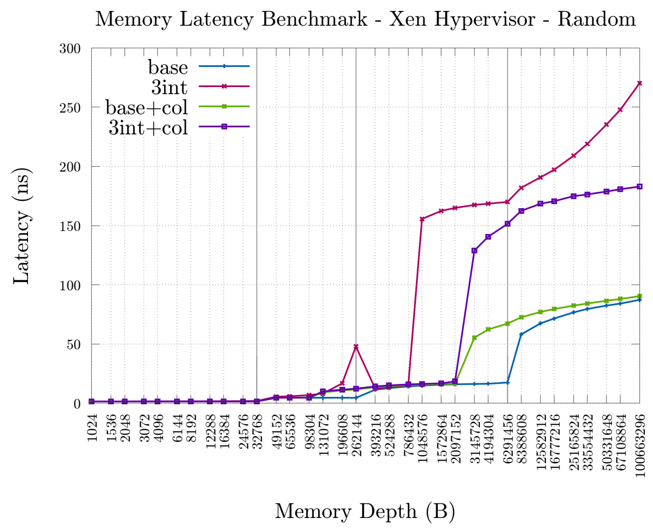


Fig. 5. Memory benchmarking experiment results. L1 data, L2 and L3 sizes are highlighted. X scale is logarithmic.

References

[1] Kernel-based Virtual Machine (KVM). https://www.linux-kvm.org/.
[2] PREEMPT_RT. https://wiki.linuxfoundation.org/realtime.
[3] RTAI. https://www.rtai.org.
[4] RTLinux. https://en.wikipedia.org/wiki/RTLinux.
[5] Xen dom0less. https://xenproject.org/2019/04/02/whats-new-in-xen-4-12/.
[6] Xen project. https://xenproject.org/.
[7] Xenomai. https://xenomai.org.
[8] L. Abeni and D. Faggioli. An experimental analysis of the xen and kvm latencies. In *2019 IEEE 22nd International Symposium on Real-Time Distributed Computing (ISORC)*, pages 18–26. IEEE, 2019.
[9] M. Azarmipour, H. Elfaham, J. Grothoff, C. von Trotha, C. Gries, and U. Epple. Dynamic resource management for virtualization in industrial automation. In *IECON 2018-44th Annual Conference of the IEEE Industrial Electronics Society*, pages 2878–2883. IEEE, 2018.
[10] A. Barbalace, A. Luchetta, G. Manduchi, M. Moro, A. Soppelsa, and C. Taliercio. Performance comparison of VxWorks, Linux, RTAI, and Xenomai in a hard real-time application. *IEEE Transactions on Nuclear Science*, 55(1):435–439, 2008.
[11] H. P. Breivold, A. Jansen, K. Sandström, and I. Crnkovic. Virtualize for architecture sustainability in industrial automation. In *2013 IEEE 16th International Conference on Computational Science and Engineering*, pages 409–415. IEEE, 2013.
[12] N. Capodieci, P. Burgio, R. Cavicchioli, I. Sanudo Olmedo, M. Solieri, and M. Bertogna. Real-time requirements for adas platforms featuring shared memory hierarchies. *IEEE Design and Test*, 2020. To appear.
[13] R. Cavicchioli, N. Capodieci, and M. Bertogna. Memory interference characterization between CPU cores and integrated GPUs in mixed-criticality platforms. In *2017 22nd IEEE International Conference on*

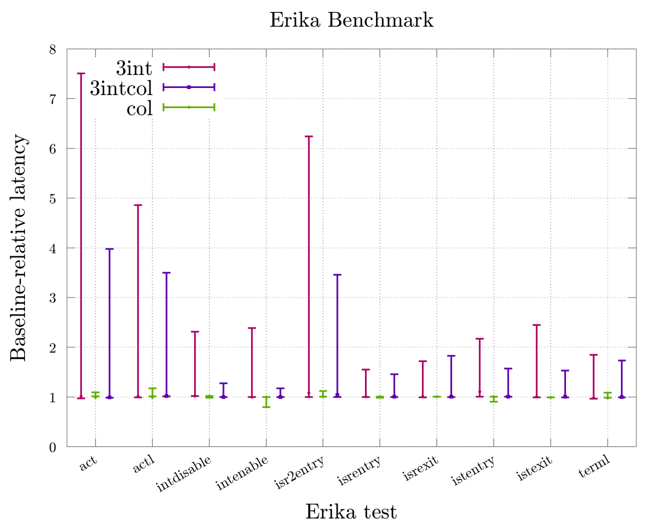


Fig. 6. ERIKA benchmarking experiment results: minimum, average and maximum. Values are relative with respect to 'base' average values.

Emerging Technologies and Factory Automation (ETFA), pages 1–10, Sept. 2017. https://doi.org/10.1109/ETFA.2017.8247615.

[14] M. Cech, A.-J. Beltman, and K. Ozols. I-MECH - Smart System Integration for Mechatronic Applications. In *IEEE International Conference on Emerging Technologies and Factory Automation, ETFA*, pages 843–850. IEEE, Sept. 2019.

[15] J. Danielsson, T. Seceleanu, M. Jägemar, M. Behnam, and M. Sjödin. Testing performance-isolation in multi-core systems. In *2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC)*, volume 1, pages 604–609. IEEE, 2019.

[16] D. B. De Oliveira, R. S. De Oliveira, and T. Cucinotta. Untangling the Intricacies of Thread Synchronization in the PREEMPT_RT Linux Kernel. In *2019 IEEE 22nd International Symposium on Real-Time Distributed Computing (ISORC)*, pages 1–9. IEEE, 2019.

[17] Evidence Srl. ERIKA Enterprise. http://www.erika-enterprise.com.

[18] Evidence Srl. ERIKA Enterprise with SOEM EtherCAT under Xen Hypervisor and Simulink code generation integration. http://y2u.be/jtVdTYgxZU4.

[19] O. Givehchi, J. Imtiaz, H. Trsek, and J. Jasperneite. Control-as-a-service from the cloud: A case study for using virtualized plcs. In *2014 10th IEEE Workshop on Factory Communication Systems (WFCS 2014)*, pages 1–4. IEEE, 2014.

[20] Google Inc. gRPC. https://grpc.io.

[21] A. C. Heursch, D. Grambow, A. Horstkotte, and H. Rzehak. Steps towards a fully preemptable linux kernel. *memory*, 48:31, 2003.

[22] T. Kloda, M. Solieri, R. Mancuso, N. Capodieci, P. Valente, and M. Bertogna. Deterministic memory hierarchy and virtualization for modern multi-core embedded systems. In *2019 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 1–14. IEEE, 2019.

[23] J. Lelli, C. Scordino, L. Abeni, and D. Faggioli. Deadline scheduling in the Linux kernel. *Software: Practice and Experience*, 46(6):821–839, 2016.

[24] G. Lipari and C. Scordino. Linux and real-time: Current approaches and future opportunities. In *IEEE International Congress ANIPLA '06*, Rome, Italy, Nov. 2006.

[25] Luca Miccio and Marco Solieri. Cache colouring for Jailhouse on Armv8, June 2019. https://git.hipert.unimore.it/rtes/jailhouse.

[26] Luca Miccio and Marco Solieri. Cache colouring for Xen on Xilinx, Dec. 2019. https://github.com/Xilinx/xen/tree/xilinx/release-2019.2-coloring, https://git.hipert.unimore.it/rtes/xen.

[27] N. Mahmud, K. Sandström, and A. Vulgarakis. Evaluating industrial applicability of virtualization on a distributed multicore platform. In *Proceedings of the 2014 IEEE Emerging Technology and Factory Automation (ETFA)*, pages 1–8. IEEE, 2014.

[28] J. Martins, A. Tavares, M. Solieri, M. Bertogna, and S. Pinto. Bao: A Lightweight Static Partitioning Hypervisor for Modern Multi-Core Embedded Systems. In M. Bertogna and F. Terraneo, editors, *Workshop on Next Generation Real-Time Embedded Systems (NG-RES 2020)*, volume 77 of *OpenAccess Series in Informatics (OASIcs)*, pages 3:1–3:14, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. https://doi.org/10.4230/OASIcs.NG-RES.2020.3.

[29] Open EtherCAT Society. Simple Open EtherCAT Master (SOEM). https://github.com/OpenEtherCATsociety/SOEM.

[30] R. Ramsauer, J. Kiszka, D. Lohmann, and W. Mauerer. Look mum, no VM exits!(almost). *arXiv preprint arXiv:1705.06932*, 2017.

[31] I. Sañudo, P. Cortimiglia, L. Miccio, M. Solieri, P. Burgio, C. Di Biagio, F. Felici, G. Nuzzo, and M. Bertogna. The Key Role of Memory in Next-Generation Embedded Systems for Military Applications. In P. Ciancarini, M. Mazzara, A. Messina, A. Sillitti, and G. Succi, editors, *Proceedings of 6th International Conference in Software Engineering for Defence Applications*, Advances in Intelligent Systems and Computing, pages 275–287, Cham, 2020. Springer International Publishing.

[32] L. Sha, M. Caccamo, R. Mancuso, J.-E. Kim, M.-K. Yoon, R. Pellizzoni, H. Yun, R. Kegley, D. Perlman, G. Arundale, and R. Bradford. Single core equivalent virtual machines for hard real-time computing on multicore processors, 2014.

[33] S. Xi, J. Wilson, C. Lu, and C. Gill. RT-Xen: Towards real-time hypervisor scheduling in Xen. In *Proceedings of the ninth ACM international conference on Embedded software*, pages 39–48. ACM, 2011.

[34] S. Xi, M. Xu, C. Lu, L. T. Phan, C. Gill, O. Sokolsky, and I. Lee. Real-time multi-core virtual machine scheduling in xen. In *2014 International Conference on Embedded Software (EMSOFT)*, pages 1–10. IEEE, 2014.

[35] V. Yodaiken et al. The rtlinux manifesto. In *Proc. of the 5th Linux Expo*, 1999.

[36] J. Zhang, K. Chen, B. Zuo, R. Ma, Y. Dong, and H. Guan. Performance analysis towards a kvm-based embedded real-time virtualization architecture. In *5th International Conference on Computer Sciences and Convergence Information Technology*, pages 421–426. IEEE, 2011.