Journal of Information Processing Vol.31 78–87 (Feb. 2023)

[DOI: 10.2197/ipsjjip.31.78]

Regular Paper

A Performance Evaluation of Embedded Multi-core Mixed-criticality System Based on PREEMPT_RT Linux

Yixiao Li[1,a)] Yutaka Matsubara[1,b)] Hiroaki Takada[1,c)] Kenji Suzuki[2,d)] Hideaki Murata[2,e)]

Received: May 15, 2022, Accepted: November 8, 2022

Abstract: The use of Linux in the domain of embedded systems is growing very fast. Due to the complexity of hardware and software on Linux-based platforms, it is challenging to meet the performance requirements, especially for the mixed-criticality system with both real-time and best-effort tasks. We propose a reference design of Linux-based mixed-criticality system using PREEMPT_RT patch to improve real-time performance and QEMU/KVM virtual machine to reduce interferences. An evaluation environment of the proposed design is built with recent software and hardware to investigate the performance characteristics via experimental measurements. We measure and analyze the baseline kernel latency, the system throughput and the real-time performance under multiple conditions. In detailed analysis, we reveal novel insights on the real-time capabilities and limitations of Linux-based embedded systems. The results show that our design can meet the 100 μs deadline goal of 1 kHz real-time task with high probability under various extreme interferences, and can deliver high throughput for best-effort workload.

Keywords: embedded system technology, real-time system design, performance evaluation, Linux

1. Introduction

The use of Linux in the domain of embedded systems, as indicated by a recent market survey [1], is growing very fast. Embedded systems (e.g., consumer electronics, home appliances, industrial machinery) usually interact with the physical world, and thus the control tasks must complete its work before a deadline [2]. Although open-source solutions based on Linux enable rapid prototyping with reduced development cost, it is challenging to meet the real-time performance requirements of control tasks on Linux compared to RTOS (real-time operating system) [3], [4].

A mixed-criticality system is a system including tasks with different criticality levels running at the same time [5]. In this paper, we consider two types of tasks: real-time tasks and best-effort tasks. The real-time tasks are activated periodically and must be properly scheduled to meet the deadline of each task. Best-effort tasks do not have deadlines and can be preempted by real-time tasks. An ideal mixed-criticality system shall guarantee the real-time performance of real-time tasks while delivering high throughput to the best-effort tasks. Many of today's embedded systems have become mixed-criticality systems as the software complexity increases. For instance, real-time tasks for control application and best-effort tasks for GUI application often coexist in a single appliance.

In order to meet the real-time requirement, the kernel service latency must be small and highly deterministic. However, the standard Linux kernel was initially designed for servers and desktops only requiring relaxed timing determinism [6]. PREEMPT_RT is a set of patches for Linux kernel able to improve the real-time capabilities [7]. The major feature of PREEMPT_RT is to convert the standard Linux into a fully preemptible kernel with increased predictability and reduced latency. However, since system throughput and latency are two metrics in tradeoff, PREEMPT_RT also has a negative impact on the performance of best-effort tasks [8].

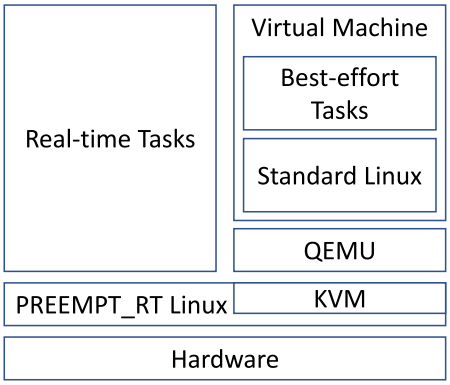


Fig. 1 A reference design of mixed-criticality system based on PREEMPT_RT Linux.

The interference from best-effort tasks is another factor that can influence the determinism of real-time tasks. Using a hypervisor to execute best-effort tasks in a guest virtual machine is an effective method to limit the performance impact [9]. KVM [10] and Xen [11] are two open-source hypervisors supporting embedded Linux. KVM is a hosted hypervisor running on top of a host OS. It requires an emulator (usually QEMU [12]) to emulate

[1] Nagoya University, Nagoya, Aichi 464–8601, Japan
[2] Mitsubishi Heavy Industries, Ltd., Kobe, Hyogo 652–8585, Japan
[a)] liyixiao@ertl.jp
[b)] yutaka@ertl.jp
[c)] hiro@ertl.jp
[d)] kenji.suzuki.wx@mhi.com
[e)] hideaki.murata.57@mhi.com

hardware devices in virtual machines. Xen, on the other hand, is a bare-metal hypervisor running directly on the hardware, which means real-time tasks must also execute in a virtual machine. Previous evaluation results on an Intel x86 platform have indicated that the usability of Xen in the real-time domain is restricted due to high kernel latency [13].

Understanding the real-time performance characteristics (capabilities and limitations) is vital to the engineering of embedded systems based on Linux. In this paper, we evaluate a reference design of mixed-criticality system based on PREEMPT_RT Linux as shown in **Fig. 1**. All real-time tasks run natively on the host OS (PREEMPT_RT Linux) to achieve best timing determinism. All best-effort tasks run inside a KVM/QEMU virtual machine to minimize the interference. In addition, the guest OS is a standard Linux rather than PREEMPT_RT Linux, in order to deliver better throughput. The goal of our system is to keep 100 μs deadline of 1 kHz real-time task, which can potentially satisfy the requirement of many real-world robot systems [14], [15], [16].

In summary, this paper makes the following main contributions:

- A mixed-criticality system design based on PREEMPT_RT Linux is presented. Unlike existing designs depending on hardware features (e.g., ARM TrustZone, big.LITTLE architecture) only available on a few platforms, our design mitigates the interference of best-effort tasks with a software approach (KVM/QEMU guest) supported by most embedded boards. To our knowledge, it is the first mixed-criticality system design evaluated on the popular COTS (Commercial-Off-The-Shelf) embedded platform Raspberry Pi 4.
- An evaluation methodology for investigating the performance characteristics of Linux-based mixed-criticality system under extreme pressure is proposed. The obtained results are synthesized to provide valuable insights for system practitioners.
- The advantages of our design are verified by the quantitative evaluation. The baseline kernel latency of our design can easily meet the 100 μs deadline goal on the evaluation environment while the Xen hypervisor cannot. The usage of dual OS (PREEMPT_RT host and standard Linux guest) in our design can reduce the deadline miss of real-time tasks and improve the throughput of best-effort tasks at the same time. Even under various extreme interferences, our design still shows a very high probability to keep the deadline.

The rest of the paper is organized as follows. In Section 2, the environment used to evaluate our design is described. We measure and analyze the baseline kernel latency in Section 3, the system throughput in Section 4, and the real-time performance in Section 5. In Section 6, we summarize and discuss previous studies. Section 7 concludes this paper and lists future work based on the evaluation results.

2. Evaluation Environment

Figure 2 summarizes the specifications of our evaluation environment for mixed-criticality system based on PREEMPT_RT Linux.

Raspberry Pi 4 is the latest generation of Raspberry Pi, which is a COTS multi-core single-board computer series having been widely used in many embedded products, due to its low cost, modularity, and open design [17]. The model with the largest memory size (8 GB) is chosen to avoid possible out-of-memory problems in the stress testing.

Component	Specification
Hardware	Raspberry Pi 4 Model B - CPU: ARM Cortex-A72 x 4 Cores @ 1.5GHz - RAM: 8GB LPDDR4-3200 SDRAM - Storage: 128GB SanDisk Ultra A1 microSD
OS	Ubuntu 20.04 LTS
Linux Kernel	5.4.0-1028-raspi (Mainline version: 5.4.78)
PREEMPT_RT	patch-5.4.78-rt44
QEMU	qemu-system-aarch64 4.2.1
cyclictest	rt-tests-1.5
UnixBench	byte-unixbench 5.1.3
stress-ng	0.11.07

Fig. 2 The specification summary of the evaluation environment for mixed-criticality system.

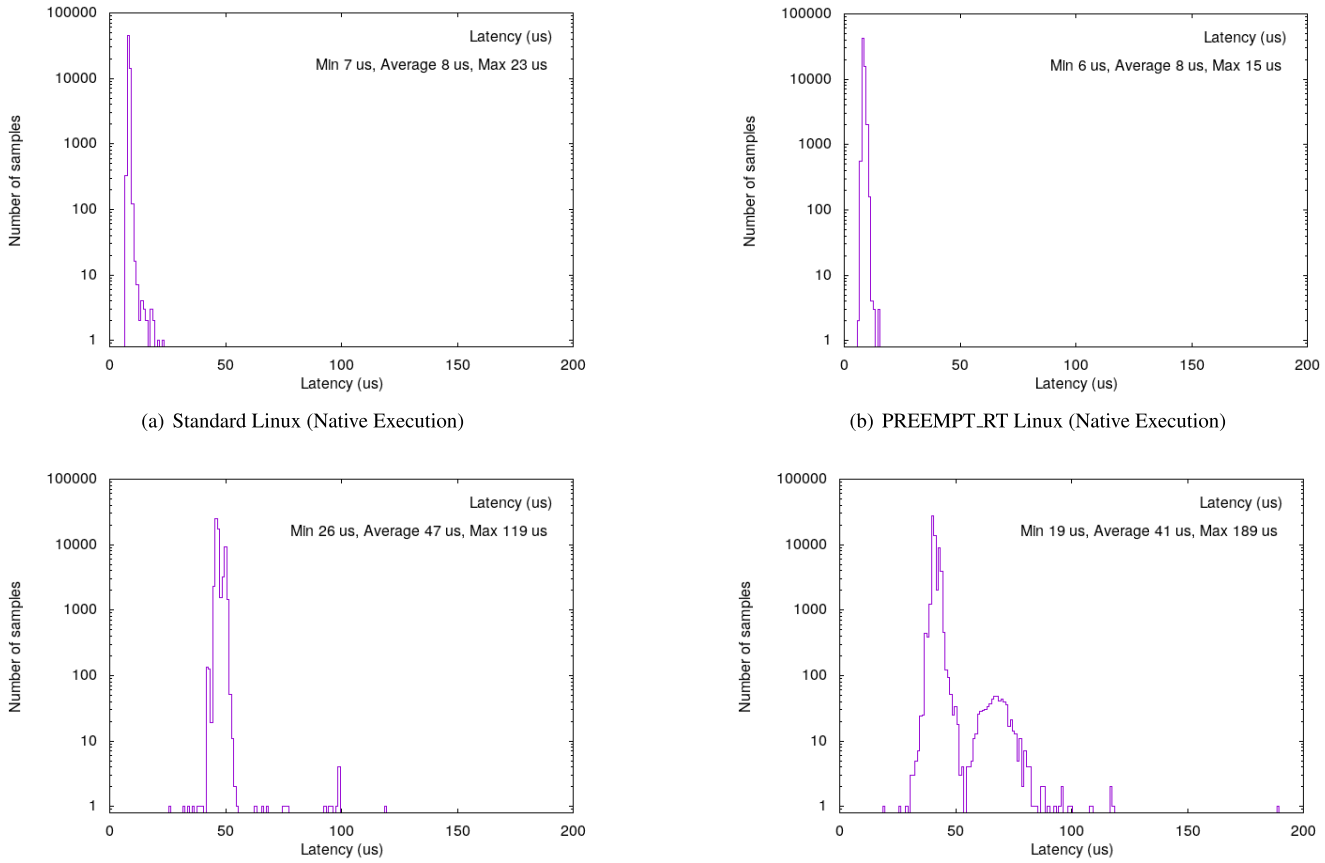
Ubuntu 20.04 LTS is the latest long term support release of the Ubuntu Linux distribution extensively used throughout education, research, and industry. By default, the cpufreq scaling governor of Ubuntu on Raspberry Pi 4 is set to "ondemand" mode, which will dynamically change the CPU frequency depending on the current system load. However, it is difficult to achieve timing determinism with the "ondemand" governor [18]. Therefore, our environment sets the governor to "performance" mode, which always uses maximum CPU frequency. To prevent CPU overheating under the stress testing, a case with active cooling fan is used.

Linux kernel 5.4.0-1028-raspi is an official Ubuntu kernel package based on the longterm mainline 5.4.78 kernel, with necessary configurations and patches for Raspberry Pi series. It supports recent PREEMPT_RT (patch-5.4.78-rt44, released on Nov. 2020) although a manual kernel building from the source code is required.

QEMU [12] is an emulator able to create and run guest virtual machine on KVM hypervisor. The guest of our environment for best-effort tasks is allocated with 4 GB memory, and uses virtio [19] paravirtualized drivers for storage (virtio-blk-device) and networking (virtio-net-device).

cyclictest [20] is the standard benchmark tool provided by the Linux Foundation to accurately measure the kernel latency of a periodic task. It can generate histogram data to summarize all latency samples during the measurement. Since a real-time task must not be swapped out from the main memory, the "mlockall" option provided by the cyclictest command line is used. In the real-time task evaluation, we use cyclictest to examine the latency of a task running at highest priority (99 of SCHED_FIFO scheduling policy) with 1 ms execution interval. All best-effort tasks run at default priority of SCHED_OTHER scheduling policy. Linux will migrate SCHED_OTHER tasks to other cores in order to balance the CPU load, but will never migrate SCHED_FIFO tasks.

UnixBench [21] is a benchmark suite designed to extract a basic performance indicator of a Unix-like operating system. Multiple tests are included to measure various aspects of the system,



(a) Standard Linux (Native Execution)

(b) PREEMPT_RT Linux (Native Execution)

(c) Standard Linux (Xen Dom0)

(d) Standard Linux (Xen DomU)

Fig. 3 Histograms of the baseline kernel latency measured under different conditions.

and each test will produce an index value by comparing to the performance of a baseline system. The entire set of index values is then combined to make an overall index for the system. For multi-core platforms, UnixBench can run in parallel a test program on each core and generate results for both single and multi-core configurations. In our evaluation, UnixBench results are used to reflect the system's overall throughput for supporting best-effort tasks.

stress-ng [22] is a tool suite to stress test a computer system in various selectable ways. It provides a wide range of different stress mechanisms (known as "stressors") covering aspects like CPU, cache, device drivers, I/O, interrupts, file system, memory, networking, security and core kernel services. Due to compatibility problems or resource restrictions, some stressors may fail to execute on a specific system. On our evaluation environment, 214 of 234 stressors can run flawlessly. By using these stressors as best-effort tasks, we can verify whether the real-time performance can still be satisfied under extreme pressure.

3. Baseline Kernel Latency

In this section, we measure the baseline kernel latency for our native execution environment and a Xen-based environment (Xen version 4.15). The results are obtained by running cyclictest for 60 seconds without any workload of best-effort tasks (except default system services on Ubuntu).

On our environment, both standard and PREEMPT_RT Linux kernels are measured. On Xen environment, both Dom0 (privileged domain) and DomU (unprivileged domain) are measured. Due to the lack of official support, we use xen-rpi4-builder [23] provided by DornerWorks to run Xen+Ubuntu on Raspberry Pi 4. However, this Xen environment does not provide a PREEMPT_RT kernel, and thus we only measure the standard Linux kernel. In the evaluation of the native environment, no QEMU guest is running. In the evaluation of Dom0 guest, no DomU guest is running. Due to the dependency, Dom0 is running in the background when evaluating DomU.

Figure 3 are the histograms plotted from measured samples.

Observation 1: Our environment can easily meet the real-time goal with either standard or PREEMPT_RT Linux kernel if no best-effort task is running. While PREEMPT_RT shows a slight improvement in the maximum latency, both kernels have average and maximum latencies much smaller than the 100 μs goal. The distributions and average latencies of two kernels are very close, which indicates that the results could be too optimistic without stress best-effort workload.

Observation 2: Our environment has a significant advantage in kernel latency compared to the Xen-based environment. Due to the overhead of Xen hypervisor, neither Dom0 nor DomU guest can always guarantee the 100 μs deadline. The minimum and average latencies are also much higher than native execution. It is interesting that DomU shows a lower average latency than Dom0, possibly because the number of devices and system services is less in DomU. Further, the strong deviations in the histogram of DomU indicate low timing determinism. The

Table 1 The summary of UnixBench benchmark results. The score of UnixBench indicates the throughput of best-effort tasks running under 3 different conditions (Baseline, Host and Guest).

UnixBench Benchmark Results	1 Core					4 Cores				
	Baseline	Host	Host%	Guest	Guest%	Baseline	Host	Host%	Guest	Guest%
Dhrystone 2 using register variables	1294.8	1291.5	100%	1281.0	99%	5167.1	5157.8	100%	5121.9	99%
Double-Precision Whetstone	488.3	487.3	100%	483.7	99%	1953.2	1949.2	100%	1934.2	99%
Execl Throughput	140.2	135.5	97%	185.8	133%	480.7	396	82%	528.7	110%
File Copy 1024 bufsize 2000 maxblocks	238.5	216.3	91%	250.6	105%	468.4	148.9	32%	466.1	100%
File Copy 256 bufsize 500 maxblocks	152.3	141.9	93%	163.6	107%	295.8	90.1	30%	301.5	102%
File Copy 4096 bufsize 8000 maxblocks	469.3	423.2	90%	480.3	102%	803.9	355.6	44%	781.3	97%
Pipe Throughput	112.6	125.6	112%	121.6	108%	444.7	501	113%	488.1	110%
Pipe-based Context Switching	89.2	94.9	106%	55.9	63%	319.3	387.7	121%	330.6	104%
Process Creation	278.7	217.9	78%	230.6	83%	556.6	448.4	81%	414.7	75%
Shell Scripts (1 concurrent)	599.4	565.4	94%	594.6	99%	1373.2	1087.6	79%	1332.6	97%
Shell Scripts (8 concurrent)	1210.7	920.5	76%	987.6	82%	1317.8	772	59%	1278.0	97%
System Call Overhead	115.6	111.9	97%	133.1	115%	450.6	436.9	97%	516.8	115%
System Overall Index Score	290.1	272.5	94%	284.2	98%	755.9	539.4	71%	754.2	100%

- Baseline: Standard Linux (Native Execution), Host: PREEMPT_RT Linux (Native Execution), Guest: Standard Linux (QEMU/KVM).
- Host% and Guest% are the percentage ratios relative to baseline score of the same core number.

latency characteristics could restrict the usability of current Xen in many real-time embedded systems.

4. System Throughput

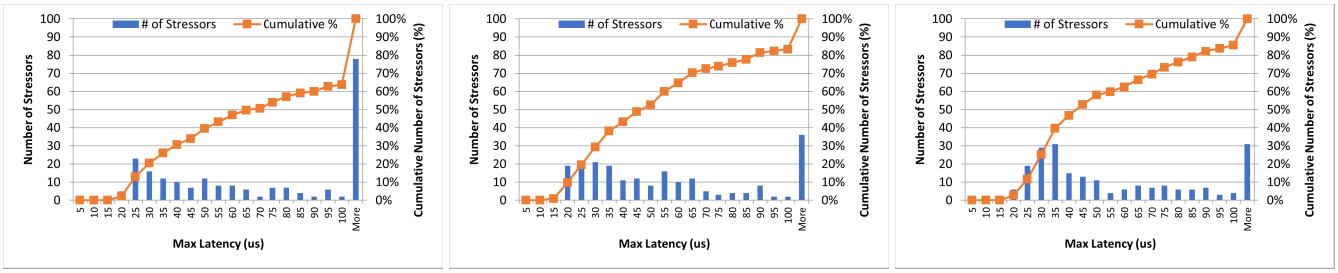
The throughput of best-effort tasks is another important performance metric for a mixed-critical system. Previous study has shown that the use of PREEMPT_RT patched kernel, while the real-time performance can be improved, will cause a degradation in throughput [8]. Moreover, the overhead of QEMU/KVM virtual machine is also a possible performance bottleneck that must be considered.

In this section, we measure the single and 4-core system throughput of the host (natively executed PREEMPT_RT Linux) and the guest (standard Linux on QEMU/KVM virtual machine with 4-core virtual CPU allocated) with UnixBench benchmark suite. To clearly reflect the performance differences, the scores of natively executed standard Linux are used as baselines. When measuring the baseline and host, no guests are running in the background. The results are shown in **Table 1**.

Observation 3: PREEMPT_RT Linux can lead to unignorable throughput degradation and thus it is undesirable for best-effort tasks to run natively. The system overall index scores of PREEMPT_RT host are 6% down compared to single core baseline and 29% down compared to 4-core baseline. For the standard Linux baseline, "File Copy" benchmarks show scalable results (i.e. absolute scores are higher in 4-core than in single core). However, in PREEMPT_RT Linux, the absolute scores of "File Copy" benchmarks become even worse as the core number increases. Since both standard and PREEMPT_RT Linux use the same file system, the poor scalability in PREEMPT_RT should be caused by the increased inter-core synchronization (e.g., spinlocks, mutexes) overhead. PREEMPT_RT optimizes the kernel latencies by increasing the kernel preemptibility, which will also introduce more synchronization points in the kernel. It indicates that the developer is likely to encounter a scalability problem in PREEMPT_RT when concurrently accessing a single device (e.g., the file system) from multiple cores. If best-effort tasks run natively on the PREEMPT_RT Linux host, they will suffer from the throughput degradation. If best-effort tasks run on standard Linux guest, although the overhead of virtual machine is introduced, the throughput degradation of PREEMPT_RT can be mitigated.

Observation 4: The performance cost of QEMU/KVM virtual machine is negligibly small. The system overall index scores of the guest are only 2% down compared to single core baseline and less than 1% down compared to 4-core baseline. In CPU computing benchmarks ("Dhrystone 2 using register variables" and "Double-Precision Whetstone"), the guest also achieves 99% of the baseline performance. The small virtualization overhead is mainly because that recent KVM implementation can leverage the hardware-assisted virtualization feature on ARMv8 platforms (known as ARM Virtualization Extensions) to avoid VM-to-hypervisor transitions when possible [24].

Observation 5: Our design of mixed-criticality system can deliver high throughput to best-effort tasks. The system overall index scores of the guest are 4% higher in single core and 40% higher in 4-core, compared to the host. It indicates that running best-efforts tasks on a standard Linux in QEMU/KVM virtual machine rather than the native PREEMPT_RT Linux host OS, can not only provide a clear boundary for resource and workload isolation, but is also an effective mitigation strategy against the throughput degradation caused by PREEMPT_RT. Moreover, the guest even shows better performance in "System Call Overhead" compared to the native standard Linux baseline. It is possibly because the native OS has many more device drivers and system services running in the background compared to the guest OS. Owing to the smaller system call overhead, the guest also outperforms the native baseline in other kernel service benchmarks such as "Execl Throughput" and "Pipe Throughput". "Pipe-based Context Switching" (single core only), "Shell Scripts (8 concurrent)" (single core only) and "Process Creation" are the only three benchmarks the guest having notable (over 10%) lower scores. As a future work, the guest throughput may be further improved by investigating the cause of performance degradation in these benchmarks.



(a) Stressor on Standard Linux (Native Execution) (b) Stressor on PREEMPT_RT Linux (Native Execution) (c) Stressor on Standard Linux (QEMU/KVM)

Fig. 4 Histograms of the maximum kernel latency observed under each 3-core stressor. Stressors run on 3 different conditions and can represent various workloads of best-effort tasks.

5. Real-time Performance Evaluation

Although Section 3 has shown that our environment can achieve an ultra low baseline kernel latency, the real-time tasks still have the possibility of missing the 100 μs deadline due to the interferences from best-effort tasks. In this section, we further evaluate and analyze the real-time performance characteristics under severe interferences by using various stressors from stress-ng as the workload of best-effort tasks.

5.1 Inter-core Interferences

Running real-time tasks and best-effort tasks on different cores is considered to be a common approach for the interference reduction. However, although the tasks per se can run in parallel, many software (e.g., kernel services, device drivers) and hardware (e.g., cache, memory) components are still shared across cores. To evaluate the effectiveness of this approach, we use cyclictest to measure the kernel latency on a single core while the stressor is running on the remaining 3 cores. The 214 stressors provided by stress-ng are executed one by one, each for 60 seconds to collect the latency samples. **Figure 4** summarizes the maximum kernel latency observed under each stressor with histograms. For comparison, the measurement results are obtained under the following three different conditions.

(a) Stressor on Standard Linux (Native Execution): The host OS uses standard Linux kernel and the system has no guest virtual machine. The stressor is natively running on the host OS.

(b) Stressor on PREEMPT_RT Linux (Native Execution): The host OS uses PREEMPT_RT Linux kernel and the system has no guest virtual machine. The stressor is natively running on the host OS.

(c) Stressor on Standard Linux (QEMU/KVM): The host OS uses PREEMPT_RT Linux kernel. The system has a QEMU/KVM guest virtual machine using a standard Linux for best-effort tasks. The stressor is running on the guest OS. This condition represents our proposed mixed-criticality system design.

Observation 6: A mixed-criticality system based on Linux should be considered as a probabilistic hard real-time system. Probabilistic hard real-time systems are systems where all deadlines must be met for which a low probability of how likely a deadline missed is acceptable [25]. None of the three conditions

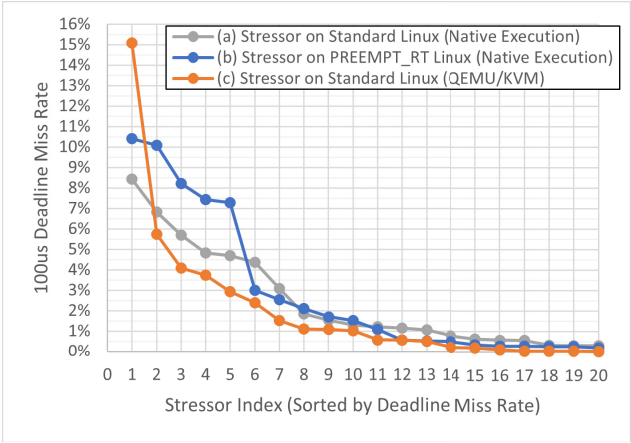
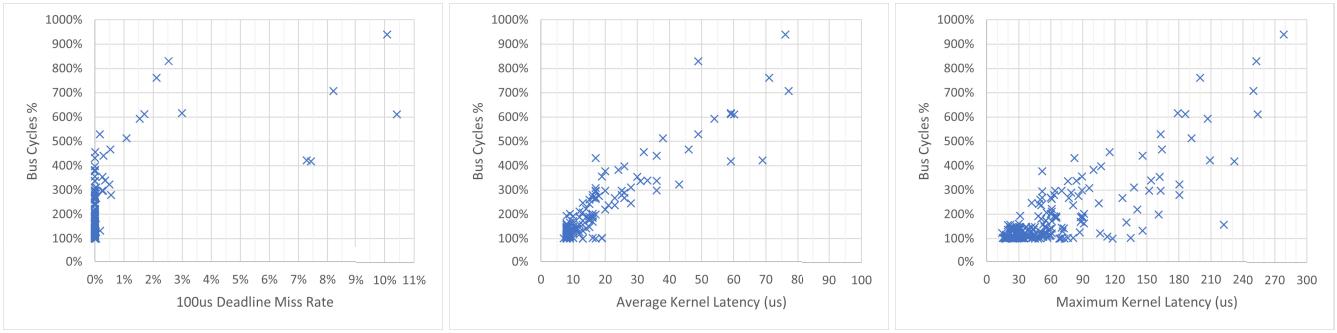


Fig. 5 The index plot of stressors sorted by measured deadline miss rate. (For clarity, only the first 20 stressors are plotted since the remaining ones have near-zero deadline miss rate.)

can meet the 100 μs latency goal under all stressors, which indicates that it is difficult to provide a 100% hard real-time guarantee on Linux. Previous research has also shown that, while Linux kernel can meet the deadline with PREEMPT_RT for 95% of the time under a specific environment, an individual RTOS kernel (e.g., Xenomai) is necessary for satisfying strict hard real-time requirements [26]. Therefore, our mixed-criticality system design should only target those applications that are acceptable to miss the deadline infrequently. How to reduce the probability of deadline miss as much as possible is vital for the reliability of the targeted applications. Although both standard Linux and PREEMPT_RT Linux have probabilistic (i.e., non-deterministic) real-time performance, PREEMPT_RT have an advantage in reducing deadline miss probability.

Observation 7: Using PREEMPT_RT Linux for the host OS can greatly improve the real-time performance under pressure. In the results of "(a) Stressor on Standard Linux (Native Execution)", about 40% stressors have been observed to miss the 100 μs deadline (according to the difference of cumulative % between "100" and "More"). The results of (b) show that the number can be decreased to about 20% by changing the host OS to PREEMPT_RT Linux. The differences of the cumulative curve between (a) and (b) also indicate that PREEMPT_RT Linux tends to have lower latency under most stressors.

Besides the maximum latency, we also evaluate the deadline miss rate under each stressor, since it is important for a proba-



(a) Bus Cycles % vs. 100 μs Deadline Miss Rate (b) Bus Cycles % vs. Average Kernel Latency (c) Bus Cycles % vs. Maximum Kernel Latency

Fig. 6 Scatterplots of stressors natively executed on PREEMPT_RT (normalized bus cycles vs. performance characteristics).

bilistic hard real-time system. **Figure 5** summarizes the results using a sorted index plot.

Observation 8: Running best-effort tasks in virtual machine allows our design to meet the deadline with high probability under most stress workloads. By comparing (a) and (b) in Fig. 5, some natively executed stressors have shown a worse deadline miss rate on PREEMPT_RT Linux than standard Linux, possibly because the performance bottleneck described in Section 4 is triggered by these stressors. The results of (c) indicate that our design, by containing the stressors inside a guest virtual machine, can successfully reduce the deadline miss for all stressors (except the first one). The first stressor (i.e., the one causing the highest deadline miss rate) is "memcpy", which will be further analyzed in Section 5.2. Although a natively executed best-effort task (i.e., a stressor in this evaluation experiment) cannot preempt a real-time task, it can compete kernel resources when invoking system calls and accessing device drivers. When a best-effort task runs in a virtual machine, it acquires the resources from the guest rather than the host OS kernel, and thus the performance of a real-time task can be improved. Under our design, 99% (212 of 214) stressors can meet the deadline with 95% probability, and 97% (207 of 214) stressors can meet with 99% probability. It means that our mixed-criticality design has successfully reduced inter-core interferences by running best-effort tasks in virtual machine, which can keep high real-time performance to support many kinds of real-world application workloads.

5.2 Hardware-related Interferences

In multi-core systems, tasks running on different cores still share many hardware components (e.g., cache, bus, memory), which can lead to performance interferences. In fact, the stressor with the highest (about 15%) deadline miss rate in Fig. 5 (c) is "memcpy". The "memcpy" stressor repetitively calls "memcpy()" and "memmove()" from the standard library and barely invokes kernel services, which indicates that the deadline miss should be mainly related to hardware rather than to operating system factors.

To further examine the possible hardware-related interferences observed in stressors like "memcpy", we use the "perf" tool [27] from Linux kernel to profile the related hardware events. More specifically, we measure the "bus cycles" (i.e., cycles elapsed in the coherent memory bus) of cyclictest under each stressor, and then normalize the number relative to the baseline (i.e., "bus cycles" of cyclictest under no stressor). **Figure 6** uses scatterplots to show the relation between the normalized bus cycles and real-time performance characteristics. It should be noted that "perf" cannot accurately measure hardware event under QEMU/KVM guest, and thus the results are obtained under the condition "(b) Stressor on PREEMPT_RT Linux (Native Execution)" as described in Section 5.1. In Fig. 6 (c), two outliers (the "dev" and "loop" stressors which will be discussed in Section 5.3) with extremely high maximum latency (over 4,000 μs) are omitted.

Observation 9: The bus contention caused by best-effort tasks has a considerable impact on the system's real-time performance. According to Fig. 6 (a), all the stressors with a deadline miss rate over 0.5% are bus-intensive (bus cycles exceed 200%) workload. Moreover, in Fig. 6 (b) and (c), both average and maximum kernel latency show a remarkable correlation with bus cycles. The results indicate that, if best-effort tasks are highly bus-intensive, the interferences may cause performance issues on our mixed-criticality design. In fact, previous research [28] also shows that bus contention due to shared cache on ARM-based hardware can enable potential DoS (Denial-of-Service) attacks to the host OS from a guest OS. The authors have found that the performance impact can vary a lot on different SoCs, and proposed a software-based mitigation (named "MemGuard") by regulating each core's maximum memory bandwidth. However, this previous research only evaluates the slowdown of benchmark execution time with a single attack code, and the actual effectiveness of "MemGuard" to reduce the deadline miss rate is not unknown. On the other hand, our evaluation can show the impact of bus contention from various stress workloads on the actual system latency. We believe that the approach to cope with hardware-related interferences in a mixed-criticality system will be an important direction in the future.

5.3 OS-induced Interferences

The host OS is another possible source of interferences on real-time performance. For example, multiple tasks may compete for mutex locks to access some OS features. Long-lasting critical sections, which should be avoided for real-time tasks, could also exist in the implementation of kernel services and/or device drivers. Therefore, understanding the behavior of OS is important for making design choices in the development of real-time

Stressor	Average Latency (us)	Maximum Latency (us)	Deadline Miss Rate	Bus Cycles%
af-alg	17	161	0.010%	199%
copy-file	10	146	0.180%	132%
dccp	9	131	0.002%	166%
dev	13	4414	0.008%	134%
hdd	9	113	0.008%	108%
idle-page	19	135	0.010%	103%
loop	10	5116	0.003%	102%
tun	13	118	0.027%	99%

Fig. 7 The real-time performance characteristics under natively executed stressors causing OS-induced interferences.

applications.

In this section, we focus on the stressors meeting the following conditions, since they are likely to cause OS-induced interferences.

- The stressor must invoke some OS service(s). For instance, stressors testing CPU and/or memory bandwidth without calling system calls (e.g., the "memcpy" stressor) will be excluded.
- The stressor must have a less than 200% bus cycles, which means it is not a bus-intensive workload.
- The stressor must have a non-zero deadline miss rate to cause notable interferences on real-time performance.

Since those stressors on QEMU/KVM have no direct access to the host OS, only stressors natively executed on the PREEMPT_RT host will be discussed here. The real-time performance characteristics under all eligible stressors are listed in **Fig. 7**.

Observation 10: The storage of Raspberry Pi 4 needs further optimization for a real-time system. The "loop" stressor shows an extremely high maximum latency, which can lead to consecutive deadline misses for a long period. This stressor uses various operations to test loop devices associated with files on the system storage (i.e., microSD card in the case of Raspberry Pi 4). We have modified the source code of "loop" stressor to measure the elapsed time for each operation, and found that the 5,116 μs latency is caused by the LOOP_SET_FD operation used for initializing the loop device. For other operations, the maximum kernel latency is 142 μs at most. Interestingly, all operations have less than 119 μs on a standard Linux environment, which indicates that these operations have triggered some bottlenecks in the PREEMPT_RT kernel. Besides the "loop" stressor, there are another two storage-related stressors, "copy-file" and "hdd", able to cause deadline miss. The results suggest that more analysis and optimization on the internal implementation of storage service would be necessary for improving real-time performance.

Observation 11: The kernel services for network and memory management only have a minor impact on real-time performance. The "af-alg", "dccp" and "tun" are stressors using socket devices to test the networking subsystem of Linux kernel. The "idle-page" stressor tests the memory management subsystem of Linux kernel by accessing the bitmap of idle pages (while the cyclictest benchmark is locked to memory). All of them have tolerable maximum kernel latency (161 μs at most) and a low deadline miss rate (0.027% at most). The results suggest that our design can support real-time applications utilizing network services, such as the nodes in ROS2 (Robot Operating System 2) communicating over the network via the DDS (Data Distribution Service) middleware [29].

Function	Execution Time (us)			
	Median	Average	Maximum	Std. Dev.
kvm_sched_in	5.93	6.26	21.48	1.34
kvm_sched_out	3.00	3.20	15.39	1.11

Fig. 8 The execution time of callback functions related to context switching between the host and the QEMU/KVM guest.

Observation 12: The "dev" stressor is helpful for the rapid identification of problematic device drivers. Device drivers are the single largest contributor to the Linux kernel with million lines of code [30], and are very likely to introduce problematic implementation able to cause deadline miss. Due to the significant software complexity, it is difficult to examine all device drivers manually. The "dev" stressor includes many routines to test various generic device drivers (e.g., storage, video, sound, tty and PTP), which can help us to narrow down the list of related devices more rapidly. In fact, the results of "dev" stressor has shown there is some device in the system leading to an extremely high maximum latency (4,414 μs). We have found that the latency is due to opening the MMC block device file of the microSD card in writable mode, which is again related to "Observation 10". A limitation of the "dev" stressor is the current lack of tests for some common devices in embedded systems, such as GPIO and fieldbuses. Moreover, to mitigate the problematic implementation, it is still necessary to understand the detailed behavior by manually analyzing the source code.

5.4 Intra-core Interferences

Although it is recommended that real-time tasks and best-effort tasks should run on different CPU cores to minimize the interferences, the user may want to run best-effort tasks on all cores to achieve a desired throughput. In this section, we evaluate the following main factors possible to cause intra-core interferences to shed some light on this topic.

- The workload of natively executed tasks sharing the same core.
- The workload of tasks in guest VM sharing the same core.
- The context switching time between the host OS and guest OS on the same core.

For QEMU/KVM virtual machine, each vCPU (virtual CPU) core is actually associated with a host OS thread. Two special functions (called "preempt notifiers" in Linux kernel), "kvm_sched_in" and "kvm_sched_out", are set to the vCPU thread, and will be called back on each context switching of the vCPU thread. More specifically, "kvm_sched_in" is called when switching to the vCPU thread, and "kvm_sched_out" is called when switching from the vCPU thread to another thread. We have measured the execution time of these two functions using the "ftrace" (a kernel function tracer) tool, and the results are shown in **Fig. 8**.

Observation 13: The context switching time between the host and the QEMU/KVM guest is non-crucial for the real-

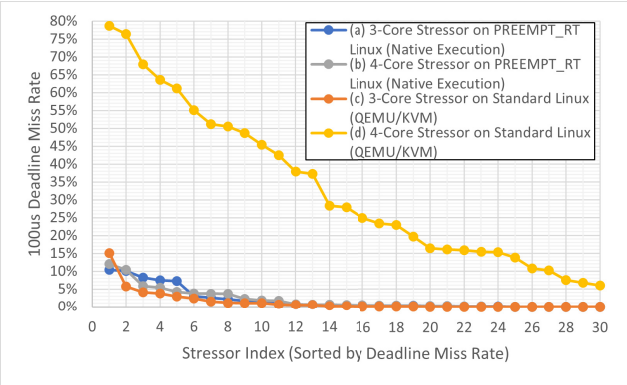


Fig. 9 The index plot of stressors sorted by the measured deadline miss rate to compare 3-core and 4-core interferences (For clarity, only the first 30 stressors are plotted).

time performance on our environment. The execution time results of "kvm_sched_in" and "kvm_sched_out" have shown small average values and low standard deviation. Even if they are added to the baseline latency in Fig. 3 (b), the 100 μs goal can still be satisfied. We believe it is because the QEMU/KVM guest has optimized virtualization overhead on ARM platforms as described in Section 4.

We examine the interferences of intra-core workload, by using a sorted index plot (**Fig. 9**) to compare the deadline miss rate under 3-core and 4-core stressor.

Observation 14: Natively executed best-effort workload can share CPU core with real-time workload without notably increasing the deadline miss rate. For natively executed stressors, the 3-core (Fig. 9 (a)) and 4-core (Fig. 9 (b)) results have shown very close values for each index. Several 4-core stressors (index 3, 4 and 5) even have a lower deadline miss rate than 3-core ones. It indicates that, in most cases, the user can run natively executed best-effort workload on all cores without a problem.

Observation 15: The hypervisor services of QEMU/KVM can severely interfere with the real-time tasks on the same core. The 4-core stressors running on QEMU/KVM guest, as shown in Fig. 9 (d), can lead to very high deadline miss rate compared to the 3-core stressors in Fig. 9 (c). These interferences should be mainly caused by hypervisor services since the overhead of context switching itself is small. For example, when the guest issues a service request via the virtio paravirtualized drivers, many host-side operations involving various kernel components (e.g., KVM kernel module, virtio backend device drivers) are necessary to respond to the request. The interferences are so severe that, at least on our environment, real-time tasks with serious timing requirements should never share the CPU core with the guest of current QEMU/KVM implementation. Further analysis of the QEMU/KVM hypervisor services (e.g., investigation into the detailed hypervisor behaviours around the time of deadline miss) could be a worthwhile future direction for research.

6. Related Work

In this section, we will introduce the related work to support the novelty of our paper.

Performance evaluation of PREEMPT_RT Linux on Embedded System. Raghenzani et al. [5] has also proposed and evaluated a mixed-criticality system based on PREEMPT_RT Linux. This study does not use virtual machine for best-effort tasks, and thus has only considered the interferences from natively executed workload. In the evaluation of OS-induced interferences, the experiment assumes the hardware supports the ARM big.LITTLE architecture which is mainly available on smartphone platforms rather than generic boards like Raspberry Pi 4. Although stress-ng is also used, this study has only measured with two stressors ("proc" and "mmap"). Adam et al. [31] has also evaluated the real-time capabilities of PREEMPT_RT Linux. This study has measured both the kernel latency and the latency to respond to an external interrupt via GPIO. However, it only targets real-time tasks rather than the mixed-criticality system with best-effort tasks, and thus the performance under the stress of best-effort workload is not evaluated. The hardware (Raspberry Pi 3) and software (Linux kernel 4.14) of the evaluation environment are outdated compared to our environment.

Mixed-criticality system design utilizing KVM. Several studies have discussed the capabilities of KVM to serve a mixed-criticality system [13], [32], [33]. They use the host Linux and KVM as a real-time hypervisor, and run real-time and best-effort tasks in separate virtual machines. On the other hand, our design runs real-time tasks natively to deliver best performance, and KVM is used as a method to isolate the interferences caused by best-effort tasks. To our knowledge, there is no previous study evaluating a mixed-criticality system similar to our design.

Bare-metal hypervisors for real-time systems. Bare-metal hypervisors, as an alternative to the hosted KVM hypervisor in our design, can also be used to build real-time systems. There are many embedded hypervisors [9], [34] targeting the coexist of RTOS and GPOS, but we focus on the open-source bare-metal hypervisors supporting dual Linux OS configuration here. Xen was the first attempt of bare-metal hypervisor overcoming the performance penalty with paravirtualization technologies [11]. However, Xen was not initially designed for real-time systems and the current version can still lead to high kernel latency as shown in our baseline evaluation. The Jailhouse hypervisor optimizes real-time performance by direct hardware assignment and static partitioning techniques [35]. While it can deliver near-native performance by eliminating device emulation, only passthrough devices are supported in virtual machines. This lack of flexibility limits the usability of Jailhouse in real world applications since the guest cannot communicate with the host via virtual network devices. ACRN is a bare-metal hypervisor aiming to pursue real-time performance and security for embedded systems [36]. It can achieve much lower results compared to existing hypervisor solutions in the latency evaluation, but only Intel platforms are supported.

7. Conclusion & Future Work

As the use of Linux and the adoption of mixed-criticality system keep growing in the embedded domain, the importance of quantitative understanding on the performance characteristics of such systems is increasing. In this paper, we proposed a reference design of mixed-criticality system design based on PREEMPT_RT Linux, which supports two types of tasks (real-time

tasks and best-effort tasks). An evaluation environment of the proposed design was built on a recent Linux kernel (5.4.78) and a representative COTS multi-core embedded board (Raspberry Pi 4). The measurement results of baseline kernel latency have shown that our environment has a significant advantage compared to Xen-based environment. Our design can successfully mitigate the throughput degradation of best-effort tasks caused by the PREEMPT_RT patch, by running these tasks on a standard Linux in QEMU/KVM virtual machine. We used the 214 stressors from stress-ng as best-effort workload to evaluate the real-time performance of our design under various extreme pressures. The results have shown that our design can meet the 100 μs deadline goal of 1 kHz real-time task with 99% probability under over 97% stressors. In the detailed analysis, we investigated the interferences caused by different factors and made many observations not covered by previous studies. These new findings can not only provide useful information about the real-time capabilities on today's hardware and software, but also have given clues to the following important future directions to research.

- Evaluate hardware-related interferences on different platforms and discuss the mitigation approach to reduce the impact of bus contention.
- Analyze the reason of UnixBench score degradation observed under the guest and attempt to further improve the throughput.
- Benchmark the real-time performance of real-world applications such as ROS2 nodes to further test the usability of our design.
- Dive into the implementation of problematic device drivers to identify the root cause of OS-induced interferences.
- Investigate the QEMU/KVM hypervisor services to optimize the performance when the real-time task and the guest coexist on the same CPU core.

References

[1] Embedded.com: 2017 Embedded Markets Study: Integrating IoT and Advanced Technology Designs, Application Development & Processing Environments, AspenCore Electronics Industry Media (online), available from ⟨https://m.eet.com/media/1246048/2017-embedded-market-study.pdf⟩ (accessed 2022-05-08).

[2] Sheikh, S.Z. and Pasha, M.A.: Energy-Efficient Multicore Scheduling for Hard Real-Time Systems, *ACM Trans. Embedded Computing Systems*, Vol.17, No.6, pp.1–26 (online), DOI: 10.1145/3291387 (2019).

[3] Cheng, Z., West, R. and Ye, Y.: Building Real-Time Embedded Applications on QduinoMC: A Web-Connected 3D Printer Case Study, *2017 IEEE Real-Time and Embedded Technology and Applications Symposium* (*RTAS*), IEEE (online), DOI: 10.1109/rtas.2017.39 (2017).

[4] Li, Y., Matsubara, Y. and Takada, H.: EV3RT: A real-time software platform for LEGO mindstorms EV3, *Computer Software*, Vol.34, pp.91–115 (2017).

[5] Reghenzani, F., Massari, G. and Fornaciari, W.: Mixed Time-Criticality Process Interferences Characterization on a Multicore Linux System, *2017 Euromicro Conference on Digital System Design* (*DSD*), IEEE (online), DOI: 10.1109/dsd.2017.18 (2017).

[6] Li, Y., Matsubara, Y. and Takada, H.: A Comparative Analysis of RTOS and Linux Scalability on an Embedded Many-core Processor, *Journal of Information Processing*, Vol.26, pp.225–236 (online), DOI: 10.2197/ipsjjip.26.225 (2018).

[7] Reghenzani, F., Massari, G. and Fornaciari, W.: The Real-Time Linux Kernel, *ACM Computing Surveys*, Vol.52, No.1, pp.1–36 (online), DOI: 10.1145/3297714 (2020).

[8] Litayem, N. and Saoud, S.B.: Impact of the Linux Real-time Enhancements on the System Performances for Multi-core Intel Architectures, *International Journal of Computer Applications*, Vol.17, No.3, pp.17–23 (online), DOI: 10.5120/2202-2796 (2011).

[9] Cinque, M., Cotroneo, D., Simone, L.D. and Rosiello, S.: Virtualizing mixed-criticality systems: A survey on industrial trends and issues, *Future Generation Computer Systems*, Vol.129, pp.315–330 (online), DOI: 10.1016/j.future.2021.12.002 (2022).

[10] Kivity, A., Kamay, Y., Laor, D., Lublin, U. and Liguori, A.: KVM: The Linux Virtual Machine Monitor, *Proc. 2007 Ottawa Linux Symposium* (*OLS 07*), IEEE (2007).

[11] Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I. and Warfield, A.: Xen and the art of virtualization, *ACM SIGOPS Operating Systems Review*, Vol.37, No.5, pp.164–177 (online), DOI: 10.1145/1165389.945462 (2003).

[12] QEMU Website: Homepage of QEMU, QEMU Community (online), available from ⟨https://www.qemu.org/⟩ (accessed 2022-05-08).

[13] Abeni, L. and Faggioli, D.: An Experimental Analysis of the Xen and KVM Latencies, *2019 IEEE 22nd International Symposium on Real-Time Distributed Computing* (*ISORC*), IEEE (online), DOI: 10.1109/isorc.2019.00014 (2019).

[14] Park, J., Delgado, R. and Choi, B.W.: Real-Time Characteristics of ROS 2.0 in Multiagent Robot Systems: An Empirical Study, *IEEE Access*, Vol.8, pp.154637–154651 (online), DOI: 10.1109/access.2020.3018122 (2020).

[15] Muratore, L., Laurenzi, A., Hoffman, E.M., Rocchi, A., Caldwell, D.G. and Tsagarakis, N.G.: XBotCore: A Real-Time Cross-Robot Software Platform, *2017 1st IEEE International Conference on Robotic Computing* (*IRC*), IEEE (online), DOI: 10.1109/irc.2017.45 (2017).

[16] Puck, L., Keller, P., Schnell, T., Plasberg, C., Tanev, A., Heppner, G., Roennau, A. and Dillmann, R.: Performance Evaluation of Real-Time ROS2 Robotic Control in a Time-Synchronized Distributed Network, *2021 IEEE 17th International Conference on Automation Science and Engineering* (*CASE*), IEEE (online), DOI: 10.1109/case49439.2021.9551447 (2021).

[17] Saari, M., bin Baharudin, A.M. and Hyrynsalmi, S.: Survey of prototyping solutions utilizing Raspberry Pi, *2017 40th International Convention on Information and Communication Technology, Electronics and Microelectronics* (*MIPRO*), IEEE (online), DOI: 10.23919/mipro.2017.7973568 (2017).

[18] Qin, Y., Zeng, G., Kurachi, R., Li, Y., Matsubara, Y. and Takada, H.: Energy-Efficient Intra-task DVFS Scheduling Using Linear Programming Formulation, *IEEE Access*, pp.30536–305471 (online), DOI: 10.1109/access.2019.2902353 (2019).

[19] Russell, R.: virtio: Towards a de-facto standard for virtual I/O devices, *ACM SIGOPS Operating Systems Review*, Vol.42, No.5, pp.95–103 (online), DOI: 10.1145/1400097.1400108 (2008).

[20] cyclictest Website: Wiki, The Linux Foundation (online), available from ⟨https://wiki.linuxfoundation.org/realtime/documentation/howto/tools/cyclictest/start⟩ (accessed 2022-05-08).

[21] Website, U.: GitHub, Kelly Lucas (online), available from ⟨https://github.com/kdlucas/byte-unixbench⟩ (accessed 2022-05-08).

[22] stress-ng Website: GitHub, Colin Ian King (online), available from ⟨https://github.com/ColinIanKing/stress-ng⟩ (accessed 2022-05-08).

[23] xen-rpi4-builder Website: Build Xen for Raspberry Pi 4, DornerWorks (online), available from ⟨https://github.com/dornerworks/xen-rpi4-builder⟩ (accessed 2022-05-08).

[24] Dall, C., Li, S.-W., Lim, J.T. and Nieh, J.: ARM Virtualization: Performance and Architectural Implications, *ACM SIGOPS Operating Systems Review*, Vol.52, No.1, pp.45–56 (online), DOI: 10.1145/3273982.3273987 (2018).

[25] Bernat, G., Colin, A. and Petters, S.: WCET analysis of probabilistic hard real-time systems, *23rd IEEE Real-Time Systems Symposium, RTSS 2002, IEEE Comput. Soc.* (online), DOI: 10.1109/real.2002.1181582 (2002).

[26] Brown, J.H. and Martin, B.: How fast is fast enough? Choosing between Xenomai and Linux for real-time applications, *Proc. 12th OSADL Real-Time Linux Workshop* (2010).

[27] perf tool: Main Page, The Linux Foundation (online), available from ⟨https://perf.wiki.kernel.org/index.php/Main_Page⟩ (accessed 2022-05-08).

[28] Bechtel, M. and Yun, H.: Denial-of-Service Attacks on Shared Cache in Multicore: Analysis and Prevention, *2019 IEEE Real-Time and Embedded Technology and Applications Symposium* (*RTAS*), IEEE (online), DOI: 10.1109/rtas.2019.00037 (2019).

[29] Kronauer, T., Pohlmann, J., Matthe, M., Smejkal, T. and Fettweis, G.: Latency Analysis of ROS2 Multi-Node Systems, *2021 IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems* (*MFI*), IEEE (online), DOI: 10.1109/mfi52462.2021.9591166 (2021).

[30] Kadav, A. and Swift, M.M.: Understanding modern device drivers, *ACM SIGPLAN Notices*, Vol.47, No.4, pp.87–98 (online), DOI: 10.1145/2248487.2150987 (2012).

[31] Adam, G.K., Petrellis, N. and Doulos, L.T.: Performance Assessment

of Linux Kernels with PREEMPT_RT on ARM-Based Embedded Devices, *Electronics*, Vol.10, No.11, p.1331 (online), DOI: 10.3390/electronics10111331 (2021).
[32] Zhang, J., Chen, K., Zuo, B., Ma, R., Dong, Y. and Guan, H.: Performance analysis towards a KVM-Based embedded real-time virtualization architecture, *5th International Conference on Computer Sciences and Convergence Information Technology*, IEEE (online), DOI: 10.1109/iccit.2010.5711095 (2010).
[33] Cucinotta, T., Giani, D., Faggioli, D. and Checconi, F.: Providing Performance Guarantees to Virtual Machines Using Real-Time Scheduling, *Euro-Par 2010 Parallel Processing Workshops*, pp.657–664, Springer Berlin Heidelberg (online), DOI: 10.1007/978-3-642-21878-1_81 (2011).
[34] Li, Y. and Takada, H.: iSotEE: A Hypervisor Middleware for IoT-Enabled Resource-Constrained Reliable Systems, *IEEE Access*, Vol.10, pp.8566–8576 (online), DOI: 10.1109/access.2022.3144044 (2022).
[35] Ramsauer, R., Kiszka, J., Lohmann, D. and Mauerer, W.: Look Mum, no VM Exits! (Almost) (online), DOI: 10.48550/ARXIV.1705.06932 (2017).
[36] Li, H., Xu, X., Ren, J. and Dong, Y.: ACRN: a big little hypervisor for IoT development, *Proc. 15th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments - VEE 2019*, ACM Press (online), DOI: 10.1145/3313808.3313816 (2019).

Yixiao Li is a Designated Assistant Professor at the Center for Embedded Computing Systems, Graduate School of Informatics, Nagoya University. He received his B.E. degree in software engineering from East China Normal University in 2012, and the Master and Ph.D. degrees of Information Science from Nagoya University in 2015 and 2019. His research interests include real-time operating systems, embedded multi/many-core systems, and software platforms for embedded systems.

Yutaka Matsubara is an Associate Professor at the Graduate School of Informatics, Nagoya University. He received his Ph.D. degree in Information Science from Nagoya University in 2011. From 2009 to 2018, he was a Researcher, and then an Assistant Professor at the Center of Embedded Computing Systems (NCES), Nagoya university. His research interests include real-time operating systems, real-time scheduling theory, and system safety and security for embedded systems. He is a member of IEEE, IEICE and JSAE.

Hiroaki Takada is a professor at Institutes of Innovation for Future Society, Nagoya University. He is also a professor and the Executive Director of the Center for Embedded Computing Systems (NCES), the Graduate School of Informatics, Nagoya University. He received his Ph.D. degree in Information Science from University of Tokyo in 1996. He was a Research Associate at University of Tokyo from 1989 to 1997, and was a Lecturer and then an Associate Professor at Toyohashi University of Technology from 1997 to 2003. His research interests include real-time operating systems, real-time scheduling theory, and embedded system design. He is a fellow of IPSJ and JSSST, and is a member of ACM, IEEE, IEICE, and JSAE.

Kenji Suzuki is an Engineering Manager of CIS Department, Digital Innovation Headquarters at Mitsubishi Heavy Industries, Ltd. His research interests include real-time operating systems and software platforms for embedded systems.

Hideaki Murata is a Program Execution General Manager of CIS Department, Digital Innovation Headquarters at Mitsubishi Heavy Industries, Ltd. His research interests include real-time operating systems and software platforms for embedded systems.