The Real-Time Linux Kernel: A Survey on PREEMPT_RT

FEDERICO REGHENZANI, GIUSEPPE MASSARI, and WILLIAM FORNACIARI,
Politecnico di Milano

The increasing functional and nonfunctional requirements of real-time applications, the advent of mixed criticality computing, and the necessity of reducing costs are leading to an increase in the interest for employing COTS hardware in real-time domains. In this scenario, the Linux kernel is emerging as a valuable solution on the software side, thanks to the rich support for hardware devices and peripherals, along with a well-established programming environment. However, Linux has been developed as a general-purpose operating system, followed by several approaches to introduce actual real-time capabilities in the kernel. Among these, the PREEMPT_RT patch, developed by the kernel maintainers, has the goal to increase the predictability and reduce the latencies of the kernel directly modifying the existent kernel code. This article aims at providing a survey of the state-of-the-art approaches for building real-time Linux-based systems, with a focus on PREEMPT_RT, its evolution, and the challenges that should be addressed in order to move PREEMPT_RT one step ahead. Finally, we present some applications and use cases that have already benefited from the introduction of this patch.

CCS Concepts: • **Computer systems organization** → **Real-time operating systems**; *Embedded software*; • **Software and its engineering** → **Real-time systems software**; • **Hardware** → Safety critical systems;

Additional Key Words and Phrases: Linux, commercial-off-the-shelf

1 INTRODUCTION

1.1 Real-Time Systems

In real-time systems, the correct execution of a real-time program depends not only on the logical correctness of the output but also on whether such results are carried out within a given time frame or deadline (Stankovic and Ramamritham 1990). The *temporal determinism* of the system is therefore the primary requirement to guarantee time-predictable task execution. Rather,

Authors' addresses: F. Reghenzani, G. Massari, and W. Fornaciari, Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, via Ponzio 34/5, 20133 Milano, Italy; emails: {federico.reghenzani, giuseppe.massari, william.fornaciari}@polimi.it.

throughput and *low latencies* are desirable but secondary properties. For this reason, "real time" does not mean *computing as fast as possible*, but rather *computing as fast as required.*

From a deployment standpoint, real-time programs can in fact run *bare metal*, i.e., with no operating system, or atop a *Real-Time Operating System (RTOS).* The second option is preferable when we need to handle the concurrent execution of multiple tasks in the system. In this regard, the system memory can be managed as a single address space, shared among the tasks, or as a set of virtual address spaces, to guarantee the task isolation. In the latter case, the applications usually require services to operating systems via *system calls.* This triggers a switch from user space to kernel space that consists of an execution mode change that enables the software to run *privileged instructions* and to unrestrictedly access the whole memory space. As a consequence, time-predictable executions of real-time applications based on RTOSs need to have temporal determinism also at the kernel-space level.

Generally speaking, the goals of RTOSs and *General-Purpose Operating Systems (GPOSs)* are conflicting, since the former aims at upper-bounding the execution time, while the latter target maximizing the average performance (Yodaiken 1999a). The *Worst-Case Execution Time (WCET)* is the typical metric of a real-time task, since most of the real-time scenarios require upper-bounded response times. The operating system has a significant impact on task WCET due to scheduling decisions, interferences, and system call execution.

Real-time application requirements are usually expressed as timing constraints, often as upper-bound limits. One can identify different *levels* or *classes* of constraints (Buttazzo et al. 2005; Kopetz 2011):

- **Soft real time:** the application goal is to maintain a Quality of Service (QoS), e.g., a video frame rate. Missing some deadlines typically implies a degradation of the QoS without any safety-critical consequences on people or things. This, however, needs to ensure an acceptable *user experience.*
- **Firm real time:** similar to the previous class, but in this case a missed deadline invalidates the output. Results carried out after the deadline have no value and must be discarded.
- **Hard real time:** the strictest class, where the deadline cannot be missed for any reason, since it can lead to undesirable consequences. Guaranteeing hard real-time constraints usually requires formal proofs of the entire system and classical code analyses. This class is typical of safety-critical scenarios.

For example, the autopilot computer in airliners is a clear example of hard real-time class applications. If a deadline is missed, it could potentially cause loss of human lives. Multimedia applications, such as video players or audio software, are usually instances of the soft real-time class. In this case, a missed deadline may cause a quality degradation of the output, without the occurrence of any safety-critical issue. Therefore, a real-time class does not depend on system characteristics, but on the consequences of missed deadlines.

Several other classifications can be found in the literature, e.g., weakly hard real time (Bernat et al. 2001) or (m, k)–firm real time (Hamdaoui and Ramanathan 1995). In recent years, increased interest has been shown in the concept of *probabilistic hard real time* (Bernat et al. 2002); i.e., the deadline is guaranteed to be met with a certain probability. This class has the advantage of being able to describe hard-to-analyze systems with classical worst-case execution time analyses. This occurs by maintaining the hard real-time requirements with a probability of failure that should be taken into account in the overall system fault analysis. However, methods currently used to guarantee and estimate probabilistic worst-case execution time are still immature and not widely accepted (Abella et al. 2015; Gil et al. 2017).

1.2 Challenges in Recent Embedded Systems

The End of Dennard's scaling (Esmaeilzadeh et al. 2011), the increasing computational power requirements, and the outbreak of nonfunctional requirements—e.g., thermal and energy constraints—along with the need to reduce development costs and time to market led to the introduction of features traditionally related to general-purpose systems to real-time embedded systems. In recent years, the spread of *Cyber-Physical Systems* increased the necessity of devices featuring computational capabilities comparable to the ones provided by general-purpose systems, but with safety and reliability requirements typical of an embedded system (Lee 2008). On the contrary, nonfunctional requirements, such as battery-powered devices that have low energy consumption constraints or systems in hard environmental conditions that require strong temperature control, led to the introduction of dedicated techniques that may clash with the necessity of temporal determinism.

Introducing multicore processors in real-time systems can help address the aforementioned issues and scale the performance. The drawback is that the presence of multiple computing units introduces nontrivial challenges to RTOSs. Task scheduling on multicore processors is in fact more complex and may introduce higher latencies (Anderson et al. 2006). Moreover, the execution of multiple concurrent tasks determines *resource contention* (Fedorova et al. 2010), which directly affects the predictability and increases the complexity of WCET evaluation (Chattopadhyay et al. 2014).

The use of *Commercial-off-the-Shelf (COTS)*, opposite to the development of specialized hardware, considerably reduces costs and engineering efforts, but bounding guaranteeing predictability and maintaining tight latencies become complex problems. With the introduction of multicore processors, bounding latencies is even more difficult due to temporal nondeterminism originating when multiple applications are running in different cores trying to access shared resources (Pellizzoni and Caccamo 2007; Nowotsch and Paulitsch 2012). As stated in Dasari et al. (2013), the main unpredictability issues of COTS hardware are caused by the following factors: simultaneous cache accesses from different cores, controllers of shared IO and memory buses, possible interconnection networks (e.g., Nonuniform Memory Access Architecture (NUMA) (Song et al. 2016)), memory devices and controllers, hardware power-saving strategies, System Management Mode (SMM), virtual memory management, and hardware prefetching. Such mechanisms and hardware units in COTS platforms usually contain intrinsic nondeterminism and, in general, are too complex to justify a worst-case timing analysis. In fact, COTS platforms are built for GPOSs and thus tend to improve the average performance; companies producing COTS hardware are usually not interested in investing their time to improve or simply analyze real-time metrics. In addition, SMM is a special operating mode of some architectures and it is specifically built to leave to the firmware of the motherboard some sort of authority over the system, allowing it to arbitrarily interrupt the OS and execute specific low-level code. These interrupts are called System Management Interrupts (SMIs) and they typically involve low-level management of hardware devices, e.g., the control of fan speed. Even if some of these tasks are also performed by the operating system, the firmware must guarantee that a bug in the OS could not damage the hardware. SMIs are generally not predictable and may considerably increase system latencies. In 2000, these interrupt latencies were in the order of milliseconds (Kau et al. 2000) but have decreased to microseconds in recent architectures. However, these metrics have been rising again lately due to security, virtualization, and many-core features of modern processors (Macarenco et al. 2016), and in case of memory failures, SMIs may take hundreds of milliseconds to run the failure recovery routines (Gottscho et al. 2017).

For these reasons, COTS platforms contain not only hardly analyzable cross-core interferences but also nontemporal deterministic hardware mechanisms. The specification data (e.g., maximum memory accessing latency) are not always provided by the manufacturer, making the upper-bounding of the WCET impossible. Furthermore, even if we assume a full-deterministic hardware, calculating the WCET may be too pessimistic; i.e., the computed WCET value is not suitable for any application.

The need to reduce time to market and the increasing computational power requirements are pushing toward COTS and multicore, but this collides with the aforementioned challenges. The high degree of safety required by regulations in certain applications, e.g., avionics, limits the possibility to exploit the COTS and multicore advantages due to hard real-time constraints and challenging environmental conditions. Nevertheless, companies like Boeing and Airbus are currently evaluating the possibility of using COTS components in their aerospace products (Hunter and Basciano 2015; Agrawal et al. 2017). Also, specific technical committees—e.g., IEC Technical Committee 107 for avionics—are developing standards to ensure the product quality and to allow their certification.

In this scenario, moving toward COTS multicore platforms makes the porting of non-time-predictable GPOSs to real-time environments an interesting alternative. The benefits are more evident in Linux OSs, where tons of scientific and industrial research solutions are available for a wide number of problems, from hardware to software aspects. Thanks to the high number of available libraries, often open source, the adoption of Linux as the operating system can considerably reduce costs and development effort (Raghavan et al. 2005). Complex applications, like, for instance, Graphical User Interfaces or image analysis algorithms, require immense work to be reimplemented for RTOSs, due to the common absence of state-of-the-art libraries.

1.3 Current Industrial and Scientific Trends in Linux Community

In 2007, the Linux Foundation[1] was established from the merging of different Linux development associations. It is currently supported by the majority of the main stakeholders in Information Technology, including historical competitors like Microsoft Corporation. Moreover, the member list[2] includes stakeholders of hardware manufacturers but also from automotive, telecommunication, aerospace, and financial sectors.

The advantages and limitations of using Linux for nondesktop markets are presented in this article, specifically focusing on real-time applications. The source code accessibility and portability, the years of industrial and scientific research, and the amount of implemented algorithms and libraries have made Linux a strong alternative to commercial and specialized approaches, also in embedded environments (Henkel 2006).

Furthermore, real-time Linux has been considered by both the European Space Agency (ESA) and National Aeronautics and Space Administration (NASA) for space and ground applications, including mission-critical software (Stakem 2001; Braga et al. 2008; Leppinen 2017).

The interest in using real-time Linux is not confined to classical real-time scenarios, e.g., automotive and aerospace. On the contrary, it is employed in several other fields such as robotics and networking, but also nonembedded fields like simulation frameworks, multimedia applications, and High-Performance Computing (HPC). Section 5 presents an overview of use cases for real-time Linux. Almost all HPC infrastructures are today based on the Linux operating system. Recently, the time sensitiveness of some HPC applications emerged, increasing the interest for real-time guarantees on Linux (Flich et al. 2017; Fornaciari et al. 2018).

In 2016, the Linux Foundation started the *Real-Time Linux collaborative project.*[3] This project aimed at coordinating the development of kernels for the real-time environment and, in particular,

[1]Linux Foundation website: https://www.linuxfoundation.org/.
[2]https://www.linuxfoundation.org/membership/members/.
[3]Real-Time Linux project website: https://wiki.linuxfoundation.org/realtime.

the development of the PREEMPT_RT patch, which started in 2005 with the goal to increase the determinism and to reduce latencies of the Linux kernel. The project aimed also at creating a common knowledge base, starting from the already existent *rt-wiki*.[4] Another stakeholder of real-time Linux is the *Open-Source Automation Development Lab (OSADL)*,[5] an organization intended to promote and support the use of open-source software in embedded environments.

1.4 Survey Structure

The purpose of this article is to survey and review the existent literature about introducing real-time capabilities in the Linux kernel. Section 2 presents the state-of-the-art approaches focusing on alternatives to PREEMPT_RT. In Section 3, the PREEMPT_RT patch is carefully described with reference to several works in the literature. Then, Section 4 presents the experimental evaluations on the real-time performance of the patch and Section 5 the scientific and industrial use cases and derivative works of PREEMPT_RT. Previous surveys are presented in Section 6, while future works and conclusions are discussed in Section 7.

The literature on Linux real time is composed of articles from several heterogeneous sources, including a significant number of white papers and industrial technical reports. Some papers available in the literature were published in conferences without a well-defined peer-review method or as online unpublished resources. However, several of these articles—especially those written by kernel developers—have been cited in more important scientific works. Except for papers published in well-known academic proceedings, the scientific validity and the experimental conditions have to be properly evaluated before drawing any conclusion. In this survey, we adopted the following policy: articles from sources with a well-established peer review process are included; articles from untrustworthy sources are excluded, if they provide incoherent and unrealistic results or perform experiments with insufficient details in the adopted methodology and experimental setup; articles describing internals of the kernel or use cases are instead included, regardless of the publishing status, to allow the reader to explore technical insights in case of interest.

Given the previous policy, we cited, for example, articles from `lwn.net`—the most known website focusing on Linux internals—only if they describe a functionality of the Linux kernel and exclude any claim in terms of performance results, since no form of peer review is adopted.

Where not otherwise specified, we assume the current latest version of the Linux kernel: 4.14 (November 2017).

2 REAL-TIME LINUX KERNELS

The advantages described in the previous section made Linux attractive also for the embedded world. Unfortunately, the kernel real-time performance does not completely fit the requirements for all real-time classes. The Linux kernel in fact has been developed to mainly target general-purpose systems. Before the introduction of the PREEMPT_RT patch, the re-engineering of the kernel was considered to require unsustainable effort, and therefore, alternative solutions were born.

This section mainly presents the state-of-the-art Linux real-time approaches alternative to the kernel based on the PREEMPT_RT patch. Since they are frequently compared against this patch, it is important to understand the differences and the pros and cons of each solution. The PREEMPT_RT patch itself is then briefly introduced.

The following naming convention is assumed: a *process* is composed of one or more threads sharing the same address space; a *thread* is a sequentially ordered set of instructions; a *job* is one

[4]Real-Time Wiki website: https://rt.wiki.kernel.org/.
[5]OSADL website: https://www.osadl.org/.

unit of work carried out by a single thread. Please note that *task* is synonymous with *thread* in Linux environments.

2.1 Cokernel Approaches

The most common alternative approaches to the PREEMPT_RT patch are usually based on the presence of an additional OS "pico-kernel," in charge of managing the real-time threads. They are equivalently called *cokernel*, *pico-kernel*, *nano-kernel,* or *dual kernel (dk)* based approaches. The baseline idea here is to have the pico-kernel working as a layer between the hardware and the general-purpose Linux kernel. All the various implementations are similar in nature, mainly acting as an interrupt dispatcher and scheduler. This layer catches interrupts coming from the hardware and forwards it to real-time tasks or to Linux tasks. The scheduler is then responsible for guaranteeing that real-time tasks meet the deadline, by properly assigning the CPU. Residual CPU time is eventually allocated to general-purpose Linux.

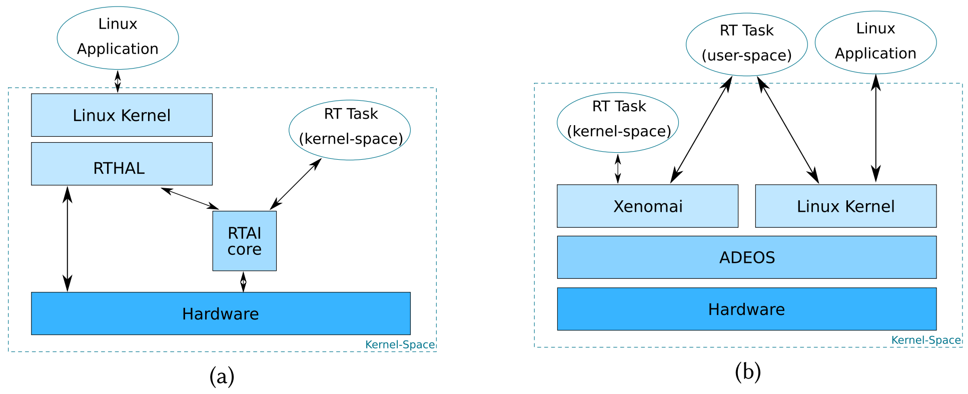
The most common open-source cokernel approaches are RTLinux, Xenomai, and RTAI. The last two are based on the *Adaptive Domain Environment for Operating System (ADEOS)* layer (Yaghmour 2001) that basically acts as a broker of hardware interrupts propagating them to Xenomai or RTAI. ADEOS in fact creates separated OS domains, each one having a private address space and resources. Resource sharing (e.g., devices) among domains is possible only via ADEOS. Interrupts are managed by propagating them via a prioritized pipeline of domains, allowing the real-time kernel—and consequently, the real-time tasks—to immediately receive the interrupts. In this way, the micro-kernel (Liedtke 1995) ensures the execution of real-time tasks independently of the status and the decisions taken by the Linux kernel. In the subsequent paragraphs, the most widespread cokernel approaches are briefly described, with their framework structure depicted in Figure 1.

2.1.1 RTAI. Developed by Paolo Mantegazza at Politecnico di Milano starting in 2000, RTAI (Mantegazza et al. 2000) is a cokernel approach aimed at ensuring Linux real-time capabilities at the *kernel-space* level. A limited set of such capabilities is also provided in *user space* via the LXRT interface. The development path follows the idea that a small number of architectures (x86, x86_64, PowerPC, StrongARM, ARM7, m68k) need to be supported to maintain the lowest possible latencies. Later on RTAI was ported to the ADEOS subsystem. In this case, differently from Xenomai, the propagation of interrupts is mainly managed by RTAI itself.

2.1.2 Xenomai. Released in 2002 by Philippe Gerum, Xenomai (Gerum 2004) is a layer that enables real time in user space. It currently supports more recent architectures than RTAI: x86, x86_64, ARM, PowerPC, and ia64. Xenomai initially relied on the ADEOS layer, but in recent versions only a simplified part of it has been maintained, in particular the interrupt-delivery subsystem called *I-pipe* (interrupt pipeline).

The last version of Xenomai (version 3, released in 2015) also allows the user to select a single-kernel version, named *Mercury*, in which the PREEMPT_RT is applied. In this version, there is no real-time kernel running and all the determinism improvement is delegated to the PREEMPT_RT patch. From the application development standpoint, a provided set of non-POSIX APIs must be used. At the time of writing, we are not aware of any advantage of using the non-POSIX API on a *Mercury* Xenomai system with respect to a PREEMPT_RT-patched kernel.

2.1.3 RTLinux. RTLinux was developed in 1997 by Victor Yodaiken, with the first stable release occurring around the year 2000 (Yodaiken 1999b); then the patent (Yodaiken 1999a) was sold to Wind River Systems in 2007. RTLinux runs the Linux kernel as a fully preemptible process together



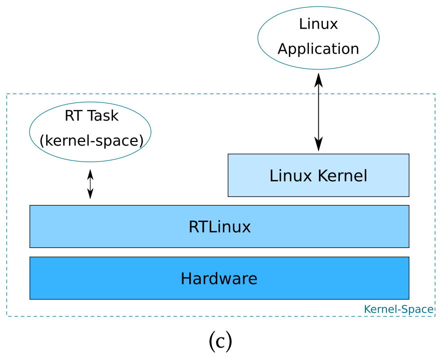


Fig. 1. The three different implementations of the cokernel approach: (a) RTAI, (b) Xenomai, and (c) RTLinux.

with the real-time applications. Basically, RTLinux implements a scheduler that decides when to run the general-purpose kernel and when to run real-time tasks according to their hard deadlines. Obviously, it acts as a filter for the interrupts in order to properly deliver them to the RT task or Linux kernel. All RT tasks share the same address space of the Linux kernel and they consequently can only be kernel-space threads.

2.2 The Preemptible Kernel History

The first traces of a real-time Linux kernel can be found in the literature in the late 1990s. From the very beginning, the hardest problem to solve consisted of the impossibility of preempting the Linux kernel (Barabanov 1997). In fact, some kernel routines were characterized by long execution times, with a dramatic impact in terms of latency. In 2001, a paper from Finney (2001) proposed to use a vanilla Linux kernel for real-time tasks in a single-core system if the deadlines remain in the order of milliseconds. However, the advent of multicore and the explosion of interactive desktop applications—not necessarily real-time ones—pushed the developers to improve the system latencies to make the kernel itself preemptible since version 2.6.0 (December 2003). This version requires additional protections of critical sections also in single-processor machines, since the kernel threads themselves can be preempted.

Without any preemption at the kernel level, it is indeed impossible to consider real-time Linux at any level. This is mainly dictated by the unpredictability of the kernel execution time that is reflected to tasks, as previously explained.

Therefore, the changes introduced in the kernel enable the selection between several degrees of preemptibility, in order to satisfy the requirements of different scenarios. Currently, the vanilla kernel Linux 4.14 (November 2017) contains three levels of preemption selectable at compile time:

- `PREEMPT_NONE`: no forced preemption. The kernel is allowed to run without user-space preemption and the system is able to deliver the maximum throughput.
- `PREEMPT_VOLUNTARY`: the kernel provides explicit preemption points in strategic locations to reduce the latency. Part of the throughput is lost because of the tradeoff with the goal of decreasing latency.
- `PREEMPT` (formerly `PREEMPT_DESKTOP`): the preemption is allowed in any part of the kernel, with the exception of spinlocks and other critical regions.

2.3 The PREEMPT_RT Patch

The PREEMPT_RT patch of the Linux kernel is actually a set of patches developed by a group of kernel developers. The project was started by Ingo Molnár and the first release was based on kernel version 2.6.11 (March 2005). The goal of this project is to trade the throughput of the system with low latencies and predictability, while maintaining the single-kernel approach, to allow the developers to easily write (user-space) real-time applications.

Among the changes introduced, it is worth mentioning the introduction of two additional preemption levels (a fourth and fifth one): *Preemptible Kernel* or Basic RT (`PREEMPT_RTB`) and *Fully Preemptible Kernel* (`PREEMPT_RT_FULL`). In more detail, the latter allows the real-time tasks to preempt the kernel everywhere, even in critical sections. However, some regions can still be made nonpreemptible, like the top half of interrupt handlers and the regions protected by *raw spinlocks*.

During the development of the PREEMPT_RT patch, besides the new features introduced, several positive side effects have been observed. Systems where this patch is applied are more sensitive to bugs due to latency constraints (Gonçalves and de Melo 2008), allowing the kernel developers to discover more easily the bottlenecks of the kernel itself. Moreover, the patch introduced several scheduler improvements and analysis tools in the kernel mainline.

2.4 Cokernel versus PREEMPT_RT

The most evident difference between the state-of-the-art approaches to real time in Linux and the PREEMPT_RT patch is the absence of a second kernel dedicated to the management of the real-time applications. This makes the implementation of real-time processes in user space similar to non-real-time ones, apart from having a special scheduling class and priority, as subsequently described in Section 3.1.6. In practice, a few further aspects must be actually considered in the development of real-time Linux applications. Simultaneously running tasks may in fact interfere from a timing perspective (Reghenzani et al. 2017). Even worse, memory swapping may lead to page faults, heavily impacting latency and predictability. This issue can be prevented by exploiting the classical protection mechanism consisting of locking the memory pages allocated to a real-time process via the `mlockall` system call.

Generally speaking, the PREEMPT_RT software development is completely different with respect to cokernels: an application can be executed in real-time mode without being rewritten. The programming model of cokernel approaches uses specialized system calls. For example, the API of Xenomai is divided into *modules* such as `rt_task_*(...)` for task management, `rt_shm_*(...)` for IPC, `rt_time_*(...)` for timer usage, and so forth. The API of cokernel approaches usually provides functions to deal with cyclic period tasks, typical of real-time applications. In Xenomai, it is possible to set the period of a task via the `rt_task_set_periodic` call, and then the task can easily wait for the next period via the `rt_task_wait_period(...)`. This approach is easier for the developer compared to the `clock_nanosleep(...)` POSIX call, because the latter requires an implementation in the application of the logic to compute the amount of sleep time needed to match the period. Moreover, the `clock_nanosleep(...)` is a guarantee of *minimum sleep time*; in fact, the kernel can, theoretically, delay the execution at infinite time. Even considering the advantages of specialized programming models, they remove one of the most important advantages of using Linux in real-time environments: exploiting the already available huge set of drivers and libraries to speed up the application development process. It is worth mentioning that even if these drivers and libraries are available in Linux, additional effort may be required to make them real-time compliant. In cokernel approaches, the real-time tasks can still access Linux features using dedicated IPC calls to Linux soft real-time tasks. This part of the execution must not be time critical; otherwise, unbounded latencies may occur due to the presence of a nonhard real-time task running on top of the Linux kernel.

All cokernel approaches require by nature invasive modifications of the kernel code, and their interactions are not trivial to be analyzed from both a functional and timing perspective. This requirement also introduces a strict dependency with respect to the kernel version, which may represent a limitation in terms of maintainability and portability. This issue does not affect PREEMPT_RT, which has the advantage of being developed directly by the Linux community. On the contrary, cokernel approaches are typically based on old Linux kernel versions. In addition, cokernel approaches are generally less stable and safe due to the complex interaction between the two kernels, which requires extra effort for real-time developers (Brown and Martin 2010).

Overall, the PREEMPT_RT patch allows the developers to operate in a real Linux environment in which they can easily reuse most of the existent libraries and tools, including all the sets of functions specified by the POSIX standard. However, as discussed later in the article, PREEMPT_RT provides less tight real-time guarantees.

2.5 Other Extensions

Over the years, other Linux extensions or patches have been developed to address problems related to real-time scenarios. We consider LITMUS^{RT} (Calandrino et al. 2006) as the most relevant one. This is a kernel patch whose main goal is to provide a test bench for real-time scheduling algo-rithms in Linux in order to obtain their proof of concept. In order to accomplish its goal, LITMUS^{RT} provides a user-space interface and a suite of tracing tools. Because of its research nature, it is not suitable for real applications and production environments, especially due to the lack of quality assurance procedures and rigorous testing. In fact, merging to the Linux mainline is not a goal of this project, which remains a research and experimental prototype. Part of the effort of the LITMUS^{RT} community is to create an environment with realistic overheads, i.e., reducing as much as possible the Linux kernel's and other tasks' inteference, to guarantee accurate measurements of performance metrics.

For this reason, the project is also used in comparison with the PREEMPT_RT patch, as we will show in Section 4.

Some commercial extensions were also built for real-time Linux, e.g., MontaVista Linux, Wind River Linux, and TimeSys Linux. However, the IP rights covering this software together with the nearly absent literature on them reduce our possibility to analyze and compare their approaches with PREEMPT_RT and the previous described alternatives.

3 THE FULLY PREEMPTIBLE KERNEL

The major goal of the PREEMPT_RT patch is to increase the preemptibility degree of the kernel code, in order to increase the predictability and reduce the latencies characterizing the system. Over the years, several contributions in this sense have been integrated into the Linux kernel

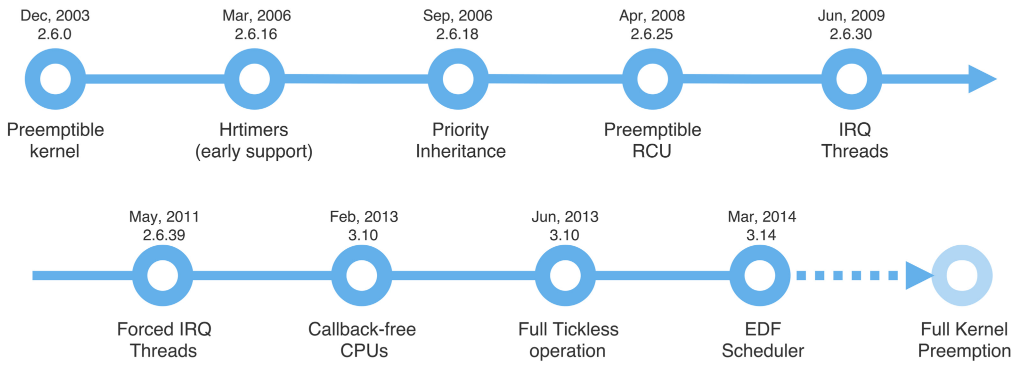


Fig. 2. Timeline of merged real-time features in the mainline Linux kernel, most of them coming from the PREEMPT_RT patch.

mainline to support real time. The remarkable ones are summarized in the timeline shown in Figure 2. It easy to notice how, in the latest years, the developers have mainly focused on introducing incremental improvements and architecture-specific supports, rather than developing new features. At present, the biggest lack in the current mainline kernel version is the full preemptibility—especially in sleeping spinlocks—that is the main feature of the PREEMPT_RT patch.

The already reached achievements are described in Section 3.1, while the features of PREEMPT_RT as of version 4.14 (November 2017) are discussed in Section 3.2, focusing on available works in the literature.

3.1 Preliminary Works

In order to achieve real-time behavior of the system, several preliminary features have been added in the PREEMPT_RT patch. This subsection briefly describes the kernel features no longer part of the PREEMPT_RT patch at the time of writing. A detailed description of the improvements and further insights of the early RT patches can be found in the Rostedt and Hart (2007) article.

3.1.1 High-Resolution Timers. One of the most important achievements has been the possibility to instantiate timers with a resolution in the order of nanoseconds. Previously, the timer interval had the precision of the configured *Tick Rate* (often abbreviated in *HZ*), which can vary between 10Hz and 1,000Hz, reducing the timer precision to the order of milliseconds. Hertz is used to count the number of ticks occurring from the boot (the so-called *jiffies*). In 2006, with the advent of the *High-Resolution Timers (HRTs)* or *Hrtimers* (Gleixner and Niehaus 2006) in kernel version 2.6.16, an application can request a timer with a resolution of nanoseconds. The actual resolution depends on the hardware implementation of the timer. HRTs can be enabled or disabled at compile time via the `CONFIG_HIGH_RES_TIMERS` configuration flag or at runtime. Recently, Patel et al. (2017) analyzed the time unpredictability induced by *hrtimers* and proposed priority-aware high-resolution timers with a subsystem called *TimerShield*. Reducing the interrupt interference on high-priority tasks increases the PREEMPT_RT real-time capabilities but at the same time adds overhead due to the use of complex data structures.

3.1.2 Threaded Interrupt Handlers. In Linux—similarly to most operating systems—each interrupt handler is divided into *hard IRQ* (or *top-half* part), which quickly responds to hardware interrupts and performs the critical operations, and *soft IRQ* (or *bottom-half* part), which performs additional processing. The difference is that *hard IRQ* is executed while keeping interrupts disabled. From version 2.6.30 (June 2009), IRQ threads enable the developers to write the *top-half* part as a

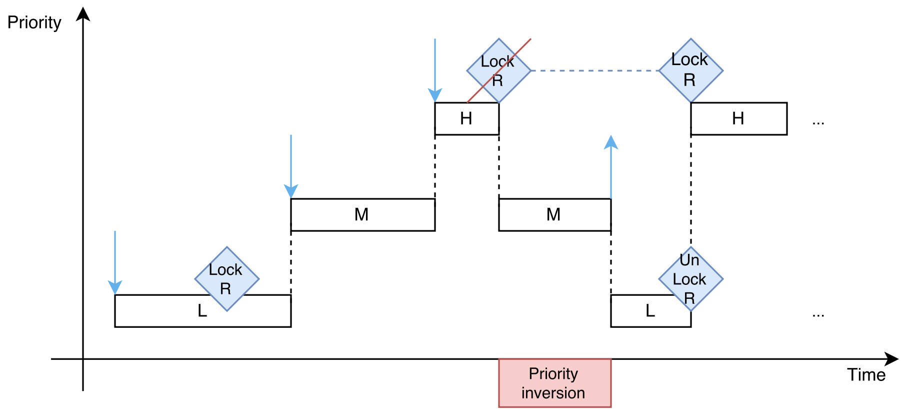


Fig. 3. Priority inversion example (all scheduling latencies are assumed null).

simple function that wakes up the specialized interrupt thread. In this way, the *bottom-half* part is scheduled as a regular kernel thread. A detailed description of the IRQ thread implementation can be found in Rostedt and Hart (2007) and Henriques (2009) and in the `lwn.net` article (Edge 2008). However, since IRQ threads have a static priority, they may incur unbounded or poorly predictable latencies due to the following causes (Kang et al. 2007; Abeni et al. 2009; Elliott and Anderson 2012):

- The presence of the well-known priority inversion problem (the solution implemented by Linux is discussed in 3.1.3).
- The impossibility to run the worker threads at a proper frequency. The delayed execution of kernel utility threads may impact on the overall performance of the system and on the real-time tasks. A couple of examples are the `pdflush` daemon (the thread controlling the writeback mechanism from memory to the disk) and the `kswapd` (swap management).
- The problem of assigning priority to IRQ threads in charge of managing the peripherals shared by tasks of different priority. An IRQ thread providing services to a real-time task should have equal or higher priority with respect to the latter in order to serve it with sufficiently low latency. However, when a low-priority task shares a peripheral with a high-priority task, it may indirectly preempt the high-priority one with the execution of the IRQ thread.

The analysis of Elliott and Anderson (2012) proposed a different solution starting from classical RTOS based on a client/server approach. Nonetheless, it concluded that the Linux kernel is too complex to implement such techniques. The IRQ subsystem complexity also affects the ability to demonstrate the correctness of the kernel. An automata-based approach to model Linux interrupts has been proposed by de Oliveira et al. (2017).

3.1.3 Priority Inheritance. Priority Inversion is a classical synchronization problem affecting real-time systems (Lampson and Redell 1980), in which a lower-priority task may preempt a higher-priority one. In order to clarify this concept[6], let us assume to have three tasks L, M, H, respectively, with low, medium, and high priority, as shown in Figure 3. The task L is the first to arrive and to acquire a resource R, and then the task M arrives and preempts L. Subsequently, the task

[6]For an exhaustive and formal discussion on priority inversion, please refer to the large body of works available in the literature.

H becomes ready, preempts *M*, and tries to acquire *R*. However, since *R* is locked by *L*, the task *H* cannot run and it is therefore put to sleep. Consequently, the task *M* is awakened and put in execution again. In this example, the medium-priority task *M* is indirectly preempting the high-priority *H*, since it does not allow *L* to run and thus to release the resource *R*. Several solutions have been proposed for this. A well-known one is called *priority inheritance* or *priority boosting* (Sha et al. 1990). The basic idea is to boost the priority of the low-level tasks (*L*) that are locking resources required by higher-priority tasks (in our case by *H*). The priority is boosted up to the level of the highest-priority task that is trying to acquire the resource. In our example, *L* would temporarily obtain the priority of *H*.

In Linux, priority inheritance was implemented in the early PREEMPT_RT patch (Corbet 2006), since priority inversion is able to cause unbounded and unpredictable latencies in the system (Kang et al. 2007; Rostedt and Hart 2007). More recently, alternative solutions include the priority ceiling protocol (Carminati 2012), migratory priority inheritance (Brandenburg and Bastoni 2012), and the Probabilistic-Write/Copy-Select mechanism (Zhu et al. 2016).

3.1.4 Read-Copy-Update (RCU). The RCU synchronization mechanism (McKenney et al. 2001) was introduced in Linux kernel version 2.6 and then ported by Desnoyers et al. (2012a) to user space. It has been described in several `lwn.net` articles (Corbet 2003; Corbet 2005; McKenney and Walpole 2007a; McKenney and Walpole 2007b). RCU had impressive success thanks to the performance improvements in synchronization. As a consequence, RCU is currently extensively used in all kernel subsystems (McKenney et al. 2013). The basic idea is to replace the standard read-write locks into RCU primitives to prevent read-side locks from blocking and to maintain multiple versions of the object updated by the writers. The RCU technique is particularly useful in scenarios featuring multiple readers and one writer. Guniguntala et al. (2008) showed how RCU offers deterministic overhead and can be effectively used for real-time applications in PREEMPT_RT-based systems. We omit a detailed description of the mechanism since it is accurately described in the cited papers.

In order to deal with real-time requirements, two features have been implemented initially in the PREEMPT_RT patch and then mainlined: *Preemptible RCU* and *No-Callbacks CPU*. The former allows a real-time kernel to preempt the readers running in a critical section that may represent a source of latency (McKenney and Sarma 2005). The latter refers to RCU callback operations used in the management of the "removal" phase of RCU, in which the callback is called to release memory when all readers exit from the critical section. RCU callbacks can degrade the real-time performance, introducing nonnegligible scheduling latencies especially if they run in an interrupt context. Sarma and McKenney (2004) implemented RCU callbacks deferred to kernel threads that can be moved to non-time-critical CPUs via the `CONFIG_RCU_NOCB_CPU` configuration item and `rcu_nocbs` kernel parameter.

The RCU subsystem is still an active topic in research. Recent works include the modeling and verification of the RCU kernel code (Kokologiannakis and Sagonas 2017; Liang et al. 2018), high-level user-space abstractions (Desnoyers et al. 2012b; Márton et al. 2017), and proposal for possible variants of RCU (McKenney 2015; Arbel and Morrison 2015; Prasad and Gopinath 2018).

3.1.5 Full Tickless Operation. In the Linux kernel, a periodic timer triggers the main event interrupt at a constant rate depending on the HZ value. It updates the internal data structures, such as the scheduler counters for processing time slices. The advent of hrtimers and the RCU improvements previously described make it possible to disable the periodic timer event. This feature is known as *Full Tickless Operation* and typically is not enabled by default. The first steps toward this achievement were described in the article of Siddha et al. (2007) and some years later on `lwn.net` by Corbet (2013). This introduction had extreme importance for the following reasons: (1) it reduces the power consumption of battery-powered devices when inactive, and (2) it decreases the extra latencies due to the rate of preemptions triggered by the periodic timer.

3.1.6 Schedulers. Since kernel version 2.6.23 (October 2007), the *Completely Fair Scheduler (CFS)* (Pabla 2009) is the Linux default scheduler. The system administrator can assign to each process a scheduling policy and a priority. In Linux, the scheduling policies traditionally follow the POSIX standard (Group 1999): `SCHED_OTHER` (sometimes called `SCHED_NORMAL`), `SCHED_RR`, and `SCHED_FIFO`. They respectively refer to the default time-sharing scheduling for best-effort workload and to the Round Robin and First-In First-Out policies for real-time tasks. The real-time classes were tested by Sousa et al. (2012) and the authors showed that the best-effort workload of the system negatively impacted on the performance of real-time workload. They tested a multicore system with over 50% of load and showed that the scheduler was not able to guarantee deadlines. Subsequently, they proposed the scheduling policy *Real-time TAsk Splitting (ReTAS)* inspired by the state-of-the-art *Notional Processor Scheduling-F (NPS-F)* policy (Bletsas and Andersson 2009). However, this policy was never receipted by kernel developers. In 2010, the `SCHED_DEADLINE` policy was presented in Faggioli et al. (2009) and Manica et al. (2010) and subsequently integrated into the kernel mainline (version 3.14, March 2014). This scheduler is based on the *Earliest Deadline First (EDF)* (Lawler and Moore 1969) and the *Constant Bandwidth Server (CBS)* (Abeni and Buttazzo 1998) algorithms. These algorithms are very well known and are not covered in this article.

3.1.7 Memory Allocators. Traditionally, for most of the hard real-time systems, the memory allocation is statically performed. This is because dynamic memory allocator routines may have unpredictable behaviors, making it hard to estimate the WCET upper-bound (Puaut 2002). For example, the regulation of avionics software DO-178B (1992) excludes the possibility to use dynamic memory allocation for safety-critical systems. The more recent version, DO-178C (2011), introduced general guidelines on dynamic memory management verification. The major problems affecting memory allocators w.r.t. real-time constraints are memory fragmentation and exhaustion, dangling references, and possible unbounded worst-case (re)allocation time (Puaut 2002; Shen et al. 2011), which are clearly unwanted effects.

In less critical real-time systems, the operating system usually provides a memory allocator to help the application developers. A noticeable example is the Real-Time Java Specification that provides a sophisticated hierarchy of memory allocators dedicated to real-time software (Higuera-Toledano and Issarny 2002). Another example, implemented in several operating systems, including RTLinux and PREEMPT_RT, is the *Two-Level Segregate Fit (TLSF)* proposed by Masmano et al. (2004). The key point of this algorithm is the capability to allocate memory with $O(1)$ time complexity. It is often called the *O(1) memory allocator* in Linux communities, even if the first $O(1)$ allocator is the Half-fit algorithm (Ogasawara 1995). A constant time complexity entails a bounded worst-case execution time of the memory allocator that is independent from applications and data. Regarding general-purpose memory allocators, Linux has different techniques at the kernel level and user level. At the kernel side, the *slab* allocator (Love and Morton 2004) guarantees the absence of memory fragmentation, while in user space several implementations are available, mostly having linear complexity (Ferreira et al. 2012).

3.2 Patch Anatomy

The kernel preemption is one of the most important features necessary to improve the Linux real-time capabilities (Scordino and Lipari 2006). It is hard, or even impossible, to predict the kernel execution time, due to its complexity and intrinsic nondeterminism. This, in conjunction with the need to have short response times for interactive applications, leads the kernel developers to implement the preemption capability of the Linux kernel-space code to avoid unbounded latencies in application operations. In the vanilla kernel, the preemptibility can be enabled via the CONFIG_PREEMPT since version 2.6. However, the kernel cannot be preempted in interrupt-disabled sections, like during *spinlocks*. Spinlocks represent a locking mechanism where the lock acquisition is iteratively attempted until it is successfully performed. This behavior occurs also if the lock is already held by another thread, causing the current thread to continue "spinning," wasting CPU time and inhibiting the possibility of being preempted. Although this mechanism may not sound very efficient, the overall performance, especially the response times, can benefit if the waiting time is small.

Besides the features described in previous sections, the most important added feature of the PREEMPT_RT patch is the removal of most of the nonpreemptible spinlocks. All the real-time features can be enabled via the CONFIG_PREEMPT_RT_FULL configuration flag. With the real-time patch applied and enabled, the spinlock_t and rwlock_t types become preemptible, while the raw_spinlock_t acts like a normal spinlock. The differentiation between spinlock_t and raw_spinlock_t was introduced with *spinlock annotations* in the mainline in version 2.6.33 (February 2010).

All the sleeping mutexes have been replaced with rt_mutex type that implements priority inheritance, as well as semaphores. The patch significantly increases the preemptibility of the kernel, leaving the top-half part of interrupt handlers and the raw_spinlock_t protected critical regions the only sections still non-preemptible. This increases the response time of the system, reducing latency and improving predictability. However, it increases the number of context switches and resource contention, consequently reducing the throughput, as described in the next section. Also, as a negative effect, it increases the chances of priority inversion scenarios. The priority inheritance and ceiling protocol are also available for user-space-level mutexes, configurable via the PTHREAD_PRIO_INHERIT and PTHREAD_PRIO_PROTECT POSIX options.

3.3 Performance Metrics and Real Time

As we have seen, the improvements in terms of kernel preemptibility led to a higher rate of context switches. The consequence is that PREEMPT_RT degrades the system performance in terms of throughput. Generally, most of the previously described features improve the real-time capabilities of the system, at the price of negatively impacting on the system throughput. The available literature characterizing this performance degradation will be discussed in Section 4.4.

It is also worth saying that reducing latencies does not necessarily increase the system's ability to meet deadlines, i.e., the real-time performance. In a not fully deterministic system like Linux and COTS platforms, the response time of a task job can be expressed by a random variable X that follows an unknown distribution $X \sim \mathcal{A}$. Given a deadline D, the task is considered hard real time if

$$P(X \leq D) = 1. \tag{1}$$

This result is hard to achieve in practice and, even in that case, hard to formally demonstrate for such systems. Considering then the less tight case of $P(X \leq D) < 1$, let $E[X] = \mu$ and $VAR[X] = \sigma^2$. Assuming an improvement in the kernel latencies that leads to a distribution X' such that $\mu' < \mu$, the researcher cannot draw any conclusion in terms of real-time performance:

$$\mu' < \mu \not\Rightarrow P(X' \leq D) \geq P(X \leq D). \tag{2}$$

This is because reducing the average value does not necessarily entail a better confidence: it may be that $\sigma' > \sigma$ and thus the upper value of the confidence interval of the modified version may be greater than the original one. Even considering an analysis in which $\sigma' \leq \sigma$, the right part of

Equation (2) cannot be stated:

$$\mu' < \mu, \sigma' < \sigma \;\not\Longrightarrow\; P(X' \leq D) \geq P(X \leq D). \tag{3}$$

This is because σ represents the average variation value and not the *jitter*[7] that instead represents the maximum variation. A confidence interval cannot be used to infer the presence and the magnitude of *outliers* (De Haan and Ferreira 2007) needed to evaluate the WCET and consequently the real-time performance. This was the starting point for the still-immature probabilistic hard real-time theory (Bernat et al. 2002), which represents an open research topic.

The difficulty in obtaining the results of Equation (1), the weakness given by Equations (2) and (3), and the immaturity of probabilistic real-time theory are the main causes of missing formal demonstrations of the real-time quality of the Linux kernel. A significant part of available works in the literature are focused on reducing the latency without formal demonstrations of a corresponding increase in predictability. This usually leads the researchers to draw conclusions from the maximum observed execution time, that is, instead, only a lower-bound of WCET. Accurate timing analyses are possible anyway when considering small sections of kernel code or isolated subsystems. Several works presented in this survey follow this approach, increasing the determinism on a particular kernel feature, even if providing an upper-bound to WCET at the application level still requires an unrealistic effort.

4 PERFORMANCE ANALYSIS

The increasing interest in real-time Linux led several researchers to assess the actual real-time capabilities of the Linux kernel. Different approaches are already available in the literature, but it is not straightforward to compare them, in particular in the case of the PREEMPT_RT patch. This is mostly due to nonhomogeneous choices in terms of hardware platforms, metrics, and benchmarks. Furthermore, information about the system configuration is most of the time not provided, undermining the research reproducibility and consequently introducing another source of uncertainty in the state-of-the-art comparison. Correctly configuring both the kernel and user-space applications and services is essential to guarantee the result reliability (Sivanich 2007; Reghenzani et al. 2017). In fact, applying the PREEMPT_RT patch itself is not sufficient to minimize latencies and ensure that a real-time thread could run with suitable guarantees. For example, one of the common actions in embedded Linux systems is to set the CPU cores' affinity of threads and IRQ handlers to reduce as much as possible the intercore interferences.

In this section, we first survey the methodologies adopted in the evaluation of PREEMPT_RT and then we provide an assessment of the real-time capabilities of PREEMPT_RT. A comparison with other RTOSs will follow. Finally, we close with an analysis of how the system throughput is affected by the introduction of this real-time support.

4.1 Methodologies

Since the most remarkable quality of real-time systems is the ability to meet deadlines in the tasks' execution, the system performances are usually evaluated in terms of *latency* and the *response-time jitter*. The throughput, intended as the number of jobs periodically processed, is not a primary concern, unlike the GPOSs, where the throughput is the main performance metric. Since minimizing latencies generally entails a throughput degradation, as already discussed, some works analyzed real-time Linux by observing both metrics, as subsequently presented in Section 4.4. This makes sense because Linux has been proposed also in *mixed time-criticality* systems, i.e., a system

[7]The term "jitter" is a general term describing the maximum variation of a time quantity, e.g., the maximum variation of the system response time. Further details are available in Buttazzo (2011).

Table 1. Different Metrics Used in Cited Articles Sorted by Publication Date

Metric	Feuerer (2007)	Mossige et al. (2007)	Arthur et al. (2007)	Koolwal (2009)	Betz et al. (2009)	Emde (2010)	Brown and Martin (2010)	Marieska et al. (2011)	Litayem and Saoud (2011)	Cerqueira and Brandenburg (2013)	Fayyad-Kazan et al. (2014)	Garre et al. (2014a)	Chalas (2015)	Hajdu and Brassai (2015)	Dantam et al. (2015)	Murikipudi et al. (2015)
Deadlock break time								X								
GPIO[8] latency	X	X	X				X									X
GPIO[8] jitter	X	X	X				X									
Interrupt latency		X		X	X	X		X	X	X	X	X	X	X		X
IPC performance	X											X			X	
Network latency		X		X											X	
Scheduling latency				X	X			X	X	X	X	X	X			
Semaphore A/R time											X					
System throughput			X			X			X		X					
Timer latency[9]						X										
User/kernel latency[10]		X														
User space	X	X	X	X	X	X	X	?	X	X	X	X	X	X	X	X
Kernel space							X	X							X	
Multicore					X			?	X	X		X	X		X	

The use of user and/or kernel-space benchmarks and a SMP system are shown in last three rows. The symbol ? indicates unknown values.

characterized by a workload including tasks with different criticality levels, that run—possibly interacting among them—on the same computational platform. In two common practical approaches, a Linux OS is typically deployed as a

(1) noncritical guest operating system of real-time hypervisor that separates Linux from the hard-real time OS (Burns and Davis 2013) or
(2) full mixed-criticality multicore system, only when the timing requirements are not strictly hard (Sigrist et al. 2015).

Table 1 shows the performance metrics considered in the analyses cited in this survey. As expected, the most recurrent metrics are about scheduling and interrupt latencies, since they are the common overhead affecting all the real-time scenarios. I/O metrics (including GPIO and Networking) may or may not be significant, depending on the specific application. Other secondary metrics include synchronization primitive latencies, IPC communication, user-space from/to kernel-space switch time, and the impact on the overall system throughput.

[8]Toggle a GPIO configured as Digital Output.
[9]Include only specific timer benchmarking (not *cyclictest* or similar approach).
[10]The overhead introduced in kernel-space to user-space context switch and vice versa.

A first important classification of performance analyses is the distinction between *kernel-space* and *user-space* evaluations. The possibility of running real-time processes in user space has important benefits not only from the application development perspective but also in terms of stability, safety, and security (Mehnert et al. 2002). These advantages of Linux led researchers to analyze mostly user-space scenarios. However, it is fundamental to remark that even if a real-time task runs in user space, the kernel-space execution of the system calls may have a nonnegligible impact on the task execution (Reghenzani et al. 2017).

In most of the current approaches, the estimation of the worst-case latencies is obtained through the generation of workload, by using suitable *stresser* programs. These, usually executed with best-effort priority level, increase the overall system load in order to test the real-time "prioritization" and isolation capabilities of the operating system. In this regard, *stress* (Waterland 2013) and its derivative work *stress-ng* (King 2013) are frequently used also in papers whose target is not represented by real-time use cases. This tool allows us to select which portion of the system and of the kernel to stress—i.e., which system call—enabling the possibility of performing fine-grained analyses. Papers that did not use *stress(-ng)* usually implemented custom scripts and applications in order to generate synthetic workloads. Other alternatives for workload generators are represented by *dohell* (typically used in Xenomai-based systems) (Community 2011) and *cpuburn* (Nielsen 2012).

Regarding the benchmark suites, the literature provides us several proposals targeting the worst-case scenario (e.g., Mälardalen WCET Benchmarks (Gustafsson et al. 2010), PapaBench (Nemer et al. 2006)) and the average-case scenario (e.g., MiBench (Guthaus et al. 2001), Parsec (Bienia et al. 2008)). Tests on worst-case response latency instead rely on more specific tools like *cyclictest* (Gleixner 2010). Since it is extensively used in Linux PREEMPT_RT analyses, we dedicated a specific subsection to describe it more in detail. Cyclictest is part of a suite developed by Linux kernel maintainers, called *rt-tests.*[11] It includes both benchmarks and stressers developed specifically to test PREEMPT_RT. Regarding third-party tools, it is necessary to mention the open-source *Linux Test Project*[12] developed by several IT companies: it provides mainly functional tests, but also a condition variable latency measurement and a priority inversion stresser.

4.1.1 cyclictest. This tool cumulates measures from several time samplings, such as interrupt latency, scheduling overhead, and timer precision. The idea behind *cyclictest* is to accumulate all the latencies occurring while the system handles the response to an input event (in this case the firing of a timer interrupt) (Rowand 2013) in a black-box fashion. The simplified pseudo-code of *cyclictest* is shown in Listing 1. All the overheads caused by several factors (IRQ handling, scheduling, etc.) are cumulated through this trivial algorithm. The tool spawns parallel threads that continue to run the algorithm in order to produce statistics, in particular the worst-case latency value. However, since *cyclictest* is periodically invoked (interval = $1ms$ by default), seldom events occurring in the middle of the interval period and causing high overhead may not be visible (Emde 2010). Clearly, this drawback negatively impacts on the validity of the measured worst-case latency. By default, *cyclictest* performs measurements in μs-scale. The researcher who wants to use *cyclictest* for ns timing should carefully evaluate the precision of system clocks, timers, and the overhead added by *cyclictest* itself. The possibility of switching to hardware-based solutions also could be taken into account for accurate timing analysis.

4.1.2 Tracers. The tracing tools—called *tracers*—allow profiling real-time systems by generating traces related to the occurrence of specific events. Their implementation needs to be lightweight and time deterministic to minimize latency and side effects introduced by the tools themselves. The

[11]https://wiki.linuxfoundation.org/realtime/documentation/howto/tools/rt-tests.
[12]http://linux-test-project.github.io/.

```
/* Get the current time */
clock_gettime(clocksource, &now);
next = now;
do {
        /* Calculate the wakeup time */
        next += interval;

        /* Wait until the wakeup time */
        clock_nanosleep(clocksource, TIMER_ABSTIME, &next, 0);

        /* Calculate the latency */
        clock_gettime(clocksource, &now);
        diff = now - next;
        print_stats(diff);
} while(!exit);
```

Listing 1. The simplified algorithm of *cyclictest* benchmark tool.

ftrace tool (Bird 2006) is built into the kernel. It contains basic functions to analyze the function call stacks and to observe kernel latencies (Rostedt 2009). KernelShark (Rostedt 2011) is a similar tool, specific for tracking scheduler events and thus evaluating latencies. LTTng (Desnoyers and Dagenais 2006), instead, with its GUI, is successfully employed to profile real-time applications running on PREEMPT_RT-enabled systems (Beamonte et al. 2012). A comprehensive survey on Linux tracers is available (Gebai and Dagenais 2018).

4.2 Linux PREEMPT_RT Real-Time Classification

The problem of providing a general and quantitative answer to what are the actual real-time capabilities of Linux with the PREEMPT_RT patch is not clearly addressable. The complexity of a GPOS like Linux entails the practical impossibility to execute the static analyses typically performed on RTOS (Arthur et al. 2007). This is caused by the unrealistic effort required to compute the WCET for all the control paths and considering all the possible sources of interference. This, together with the nonpredictability of COTS hardware, makes it impossible to characterize upper bounds for latencies and WCET.

The common trend is not to consider Linux PREEMPT_RT as a hard real-time compliant solution. The work of Brown and Martin (2010) and the technical report of Chalas (2015) led to the conclusion that real-time Linux systems can be considered *95% hard real time*. These studies compared the latencies resulting from the execution of specific applications under certain system configurations. The general idea of a *95% hard real-time* system is that deadline misses are tolerated if they occur with a probability lower than 5%. However, the timespan to be considered is not specified, making this classification not really convincing. A similar classification, but with 99% constraint, was proposed for the Ach IPC RT Library (Dantam et al. 2015). These classes recall somehow the concept of a *probabilistic hard real-time system* (Bernat et al. 2002), in which the WCET of a task can be provided in a probabilistic sense. However, the fact that the constraints get satisfied seems to suffer the same nondeterministic issues of the *hard real-time* class: how can one formally demonstrate that the probability of missing the deadline is not underestimated? Probabilistic real time is based on statistical theories; however, the theoretical results are still preliminary and controversial.

Aware of the conceptual limitations of the hard real timeness of the PREEMPT_RT patch, we now present the experimental evaluations. As previously stated, the following data are extremely dependent on the hardware platform, the benchmarks, and the system configuration. Accordingly, the reader should carefully take into account these factors before making any further conclusions.

4.2.1 Scheduling and Interrupts. All the works in the literature agree on the benefits introduced by the PREEMPT_RT patch in terms of improvements of the scheduling capabilities for real-time tasks.

Betz et al. (2009) argued that these improvements become evident when the best-effort workload exceeds 75%. However, this result is in contrast with that previously cited (Sousa et al. 2012), in which the authors noticed a real-time performance degradation with a system workload greater than 50%. Unfortunately, the two papers make use of very different kernel versions—2.6.25 and 3.2.11, respectively—and both lack details on the actual system configuration, making the experi-ments hard to be replicated and compared. Neglecting the workload effects, in the first approximation, we can state that interrupt and scheduling latencies reported are in the order of micro-seconds (μs) with worst cases under $50\mu s$ (Arthur et al. 2007; Emde 2010; Cerqueira and Brandenburg 2013; Cerqueira and Brandenburg 2013; Fayyad-Kazan et al. 2014; Hajdu and Brassai 2015).

Arthur et al. (2007) noticed a general trend: the faster the processor, the smaller the scheduling jitter. This is naturally expected in single-core processors. In multicore processors instead, this may not be assumed so easily: faster processors mean potentially more resource contention—and consequently locking overhead—that can introduce unpredictable latencies. We think that such a trend could be the object of more accurate evaluations in future works. Since scheduling and interrupt latencies, as well as jitters, are critical metrics, the *Open Source Automation Development Lab (OSADL)* constantly performs a Quality Assurance analysis on Linux PREEMPT_RT in order to measure these parameters by performing suitable tests on different architectures. The details of this continuous monitoring activity can be found in Emde (2010).

4.2.2 Input/Output. I/O is another typical source of temporal nondeterminism. Besides the previous considerations about comparing experimental data, it is important to remark that acting on the hardware depends not only on the hardware itself but also on the kernel driver implementation.

Classical analyses focused on the performance of GPIO in terms of switching latency and jitter. Feuerer (2007) measured, in his master's thesis, the GPIO jitter under $50\mu s$ in a x86 platform, while Mossige et al. (2007) bounded the timer and jitter under $100\mu s$ on a PowerPC processor. The GPIO toggle frequency is inversely correlated with the GPIO latency. In this regard, Brown and Martin (2010) obtained the highest possible GPIO toggle frequency estimation in kernel 2.6.33 (February 2010) with two probabilistic classes: 95% hard real time and 100% hard real time. In the former, the vanilla kernel enables the reaching of $7.25kHz$, while with PREEMPT_RT this value raises up to $10.64kHz$. In the latter, the vanilla kernel does not go over $0.41kHz$, while with PREEMPT_RT a rate of $2.16kHz$ has been reached. The reader must remember that these numbers depend on hardware, drivers, the kernel version, and experimental conditions. Again, they should not be taken as absolute numbers and conclusions should be drawn with care.

The works of Barbalace et al. (2011) and Murikipudi et al. (2015) indirectly measured the I/O performance of Linux PREEMPT_RT in complex applications. In the former paper, the maximum GPIO toggling frequency achieved was $10kHz$. In the latter, the sustainable frequency for managing a couple of ultrasonic transmitter/receiver sensors was only $2Hz$ for PREEMPT_RT compared to $4Hz$ for an RTOS. These results are an obvious consequence of the application-specific nature of the computation. The former in fact was an experiment on ADC conversions, while the latter was a complex image recognition algorithm.

4.2.3 Interprocess Communication (IPC). Sharing data among processes—i.e., sharing data be-tween threads in different address spaces—can also have an impact on the real-time workload, since we need to employ synchronization mechanisms for that. Concerning the Interprocess Communication (IPC) services, Garre et al. (2014a) showed increased latencies for PREEMPT_RT with respect to the vanilla kernel in the magnitude of tens of μs for operations on semaphores. This is an expected result due to the throughput degradation introduced by PREEMPT_RT. Dantam et al. (2015) presented the Ach IPC library that also tested the *network latency*. The work showed that using TCP/IP protocols in PREEMPT_RT leads to a latency within $200\mu s$, which highlights an improvement with respect to old kernel versions. Previous works indeed showed latency values up to $1ms$ for Ethernet (Mossige et al. 2007) and over $2ms$ for TCP communications (Koolwal 2009).

4.3 Comparison with Other RTOS

A direct comparison between Linux PREEMPT_RT and other real-time operating systems is usually performed considering metrics like system or interrupt latencies. Xenomai, RTAI, and RTLinux are generally considered systems able to meet hard real-time constraints (Barbalace et al. 2008; Buttazzo 2011) and thus frequently used as references. Some works also took into account LITMUS^{RT}.

An extensive comparison between PREEMPT_RT and Xenomai was proposed in 2010 by Brown and Martin (2010). In this work, several experimental evaluations were performed, mainly on GPIO and response time latencies/jitters. This paper classified Linux as a 95% hard real-time system, since it showed the ability to maintain a GPIO switch frequency equivalent to Xenomai with a probability of 95% on the long run. The paper also suggested a decision flow chart to select the best approach to use for real-time in Linux.

In 2013, a recent version of PREEMPT_RT was compared with LITMUS^{RT} (Cerqueira and Brandenburg 2013). Both provided similar scheduling latencies in an idle system (max $10 - 15\mu s$); PREEMPT_RT obtained a similar latency for CPU-bound workload (max $\sim 17\mu s$); instead, LITMUS^{RT} considerably degrades (max $\sim 47\mu s$). As discussed in Section 2.5, LITMUS^{RT} essentially introduced changes on the Linux scheduling part. In fact, the authors proposed the integration of the LITMUS^{RT} scheduling algorithm with the PREEMPT_RT patch as a future work. Linux was also compared to Windows CE and QNX Neutrino RTOS (Fayyad-Kazan et al. 2013a), generally showing lower real-time performance with respect to these commercial solutions. However, Linux outperformed Windows CE in average performance metric values and in synchronization primi-tive latencies.

In 2014, Garre et al. (2014a; Garre et al. 2014b) showed that RTAI and Xenomai can reach task switch times on average less than $1\mu s$, while PREEMPT_RT cannot break the microsecond barrier. The evaluation was performed running two threads with a busy loop containing a *scheduler yield* syscall. For the worst-case scenario, PREEMPT_RT reached similar results compared to RTAI ($\sim 20\mu s$), even if it still cannot compare to Xenomai ($\sim 2\mu s$). Similar results were obtained by the developers of the Ach IPC library (Dantam et al. 2015) when they compared Xenomai and PRE-EMPT_RT in 2015.

It is possible to conclude that, generally, RTOS and cokernel approaches still outperform Linux PREEMPT_RT in terms of real-time performance, while Linux is better on average performance, as expected by its GPOS nature. The next section presents the impact of PREEMPT_RT on system throughput.

4.4 Throughput Impact

As already discussed in Section 1, unlike GPOSs, the performance goal of an RTOS is not to maximize the system throughput. An RTOS must be characterized by predictability, responsiveness, and low latency. In fact, system throughput and latency are two metrics usually in tradeoff. In this section, we aim at summarizing the works that evaluated the negative impact of PREEMPT_RT on system throughput.

When using PREEMPT_RT, we saw that the improved kernel predictability implies a higher number of context switches compared to the vanilla kernel. This is the main cause of throughput degradation. Furthermore, mechanisms like *Priority Inheritance* (see Paragraph 3.1.3) introduce additional complexity in kernel space and a consequent increase of overhead.

Since throughput is not the primary concern of real-time developers and researchers, only a few articles are available in the literature analyzing its degradation when the PREEMPT_RT patch is applied. The Arthur et al. (2007) paper is probably the first work focusing on this aspect. The authors used LMBench (McVoy et al. 1996), a suite of benchmarks, to compare various UNIX system metrics, e.g., memory, IPC latencies, network bandwidth, and system call overheads. They concluded that the PREEMPT_RT patch does not significantly influence the system performance. However, the metrics selected in this paper are not representative of the throughput degradation, since most of the metrics are related to system latencies. Already in the first years of the PREEMPT_RT patch, Jeong et al. (2007) noticed that the actual throughput may be reduced by a factor of 5, but bounding the frequency of context switches may help to maintain an acceptable performance level. Later, the same authors analyzed the throughput degradation in Seo et al. (2009) experiencing a worst-case degradation of 10 times. However, we must consider that these works refer to two very different kernel versions. From 2007 to 2009, Linux and the PREEMPT_RT patch have deeply evolved with the introduction of further mechanisms.

The most valuable analysis of throughput degradation is probably the paper of Litayem and Saoud (2011). The authors tested both latencies and system throughput, using, respectively, *cyclictest* and *unixbench* (Smith et al. 2011). They compared the results for Xenomai, PREEMPT_RT, and the vanilla Linux kernel.[13] Unexpectedly, Xenomai outperformed PREEMPT_RT in terms of throughput. In particular, PREEMPT_RT performed very badly in multicore scenarios, showing a 49% degradation compared to the vanilla Linux kernel. However, we have to consider that, since the kernel version used was older than 2.6.39 (the exact used version is missing in the paper), the *Big Kernel Lock (BKL)* might have had a big impact in multicore performance degradation due to the higher resource contention. In fact, later in 2014, Fayyad-Kazan et al. (2014) showed that there were clear improvements (up to 35%) compared to kernel version 2.6.33, due to the *BKL* removal.

5 APPLICATIONS

In this section, we provide an outline of the most noticeable use cases exploiting the Linux PREEMPT_RT patch. The purpose here is to have an overview of the real-time application fields for which the introduction of PREEMPT_RT has brought benefits. We included industrial works, but only the ones for which a publication is available. Of course, more unpublished industrial applications exist even if not discussed in this survey.

5.1 Use Cases

In this section, we present a list of of both industrial and scientific use cases available in the literature, relying on the exploitation of the PREEMPT_RT patch. The discussion follows the chronological order of publication.

The first use case was carried out in 2007 by IBM Research: a work on a real-time Java virtual machine (RTJVM) (Auerbach et al. 2007). In this scenario, PREEMPT_RT has bound latencies for the Java garbage collector, allowing the execution of a classical soft real-time application: a music synthesizer. The following year, another real-time Java application (Plšek et al. 2008) aimed at exploiting PREEMPT_RT for the kernel preemptibility and the high-resolution timers.

In 2009, Kume et al. (2009) showed how the PREEMPT_RT patch can be effective also for real-time control of a robotic arm, performing inverse kinematic calculations. The maximum

[13]They actually compared three different configurations varying the compile-time settings of the vanilla kernel: standard, low latency, and server.

response-time jitter experienced was in fact on the order of $100\mu s$. They also tested the real-time capability by producing a periodic square wave, obtaining a response-time jitter of just $17ns$. In the same year, two other classes of use cases were proposed: avionic (Schoofs et al. 2009) and networking (Goldoni et al. 2009). In the former, Linux was used as a simulator of the final hardware to allow rapid development of the avionics software. With the use of PREEMPT_RT, they were able to simulate the embedded systems with comparable latencies, and even if the system could not guar-antee hard real-time constraints, it correctly ran the simulations. In the latter case, the authors proposed a tool to estimate the available bandwidth of a network path. They used an active prob-ing strategy, i.e., injecting traffic packet to estimate the bandwidth. In order to reduce the noise in the observations and consequently increase the estimation quality, they applied PREEMPT_RT to obtain accurate timing measurements.

In 2010, a couple of industrial network use cases were introduced: the work of Baumgartner and Schoenegger (2010) and the patent of Khawer and So (2010). The first one implemented the hard real-time *POWERLINK* protocol in a Linux machine and analyzed the maximum achievable transmission rate according to the real-time constraints and variable system load. The patent instead presented a middleware layer for the implementation of device drivers able to meet real-time requirements in Real Link Control (RLC) networks. In the same year, Schoepfer et al. (2010) presented a complex robotic arm application, where Linux played the role of the outmost level of control.

In 2011, another real-time Ethernet protocol, EtherCAT, was used in conjunction with PREEMPT_RT in the work of Mercado et al. (2011). Kacmarik et al. (2011) wrote a Linux device driver communicating with an FPGA PCIe card in order to implement a GNSS (Global Navigation Satellite System) receiver. The hardware of the system was voluntarily kept simple, leaving to the device driver the management of interrupts and of the GNSS protocol. This implied the necessity of meeting timing deadlines, successfully accomplished by the PREEMPT_RT patch. Weisbach et al. (2011) investigated the effects of device drivers on a PREEMPT_RT system, showing that using a user-space driver did not compromise the real-time capabilities. In the same year, we had a couple of other robotic use cases, exploiting the cokernel approach. The first one was by Medina et al. (2011), in which an RTAI system performed the computation for the control part and a PREEMPT_RT system provided real-time speech synthesis and recognition. The second one was by Kaneko et al. (2011), where the PREEMPT_RT system was actually used as the hard real-time system.

In 2012, Bonestroo (2012) published a master's thesis on a tripod robot using a control algorithm implemented on a PREEMPT_RT system. He remarked on the necessity of a ready-to-use system, because RTAI or Xenomai would be too expensive in terms of required setup time. Sakamoto and Kondo (2012) presented a passive robotic medical application in which the Linux system was in charge of reading measurements and carrying out the force-feedback data to motors. A six-legged robot, controlled by several hardware PI[14] controllers that receive the set point from a gait algorithm run on a PREEMPT_RT system with a loop frequency of 100Hz, was presented by Bombled and Verlinden (2012).

In 2013, Yun et al. (2013) evaluated the real-time performance of the PREEMPT_RT patch for a big physics experiment HPC application. Conversely from the previous cases, here the authors experienced no advantages or even worse results compared to the standard Linux. However, as also stated by the authors, the system was not carefully configured and tuned. This may have generated the conditions for unexpected latencies. A very well-received medical surgery framework was presented by Hannaford et al. (2013). This used the PREEMPT_RT patch to ensure real-time capabilities. Since no tight guarantees could be provided, the safety standards limited the use of this framework in research. Still, in 2013, Nguyen et al. (2013) implemented a broadcast/multicast

[14]PID controller variant without derivative effect.

communication system based again on a cokernel system, including RTAI and PREEMPT_RT Linux. Unfortunately, the authors did not provide further details on the configuration and integration of this mixed system.

Moving to 2014, we found four works worth mentioning: (1) Gabrielli et al. (2014) presented an over-LAN transmission system for audio applications using a COTS platform equipped with Linux Debian patched with PREEMPT_RT in order to reduce the cost of introducing specialized hardware; (2) a patent by Khawer and Easwaran (2014) scaled the previous patent to bigger multicell telecommunication networks; (3) a humanoid robotic application by Smith et al. (2014) was programmed in a mixed C++ and Java environment and based on a Linux PREEMPT_RT system in order to guarantee acceptable levels of latency for motor control; and (4) Verschueren et al. (2014) implemented a car-driving visual algorithm, in which the Linux system gets the car position and orientation from a camera (at $100Hz$) and issues control inputs for the car.

In 2015, the PREEMPT_RT patch was again used for a network application by Mao et al. (2015). The work implemented a Radio Access Network over a container cloud system with the usual goal of minimizing the latencies.

In 2016, Zappulla et al. (2016) presented a spacecraft simulator based on the RT patch, providing also an analysis of the operating system latencies, bounded by $100\mu s$. In the same year, Dalbert (2016) presented a master's thesis on the implementation of a real-time field bus system with Linux PREEMPT_RT. One of the most recent works is the application of the patch to the Robot Operating System (ROS) described in the tutorial of Lages (2016).

In 2017, Cheng et al. (2017) tried to port a 3D printer system to PREEMPT_RT Linux, but unfortunately the required $1\mu s$ resolution by stepper motors cannot be achieved. Even if a $1\mu s$ resolution is hard to achieve by PREEMPT_RT or other Linux variants, again the article did not provide any information on how the kernel was configured, preventing further discussions on this. In the same year, Tan et al. (2017) presented a case study where the Linux PREEMPT_RT was used for the control of a synchrotron operating at a $5kHz$ cycle rate. As with other use cases, the selection of PREEMPT_RT was dictated by the need to reduce the development effort.

Finally, in 2018, it has been used to reduce the kernel latencies that can hinder the PCIe interconnection performance of a distributed real-time heterogeneous system (Erickson et al. 2018).

5.1.1 Discussion. We can classify the aforementioned use cases into four different categories, as summarized in Table 2. As we have seen, PREEMPT_RT can be helpful not just in embedded systems applications. For example, real-time requirements can be defined to build simulators, improving the performance of the underlying models. In Table 3, we made instead a recap of all the possible scenarios for which PREEMPT_RT can introduce benefits, along with the possible drawbacks.

It is worth remarking on how some of the cited papers used PREEMPT_RT-based systems as a black box, without conducting any analysis. However, a suitable configuration of the kernel and related tunable knobs is required in order to get an acceptable level of real time. This should be required for both production systems and research works aiming at providing an analysis of the actual real-time capabilities of Linux. In our opinion, there is a recurrent lack of the research works on real-time Linux. Even if the development effort is much lower than RTAI or Xenomai, the system has to be properly tuned, as also outlined by numerous cited papers. It is important that any researcher performing tests with real-time Linux remains aware of the effects of a wrong or missing configuration of the system, in particular at the kernel level.

5.2 Virtualization

Linux PREEMPT_RT has also found space in virtualization-based systems. Some works used it as a test bench to compare the real-time capabilities of the hypervisor (Fayyad-Kazan et al. 2013b;

Table 2. List of Cited Use Cases Categorized by Operating Scenario and Scientific/Industrial Works

	Scientific Papers	Industrial Works
Embedded systems	Plšek et al. (2008) Kume et al. (2009) Schoepfer et al. (2010) Kacmarik et al. (2011) Medina et al. (2011) Kaneko et al. (2011) Sakamoto and Kondo (2012) Bombled and Verlinden (2012) Smith et al. (2014) Verschueren et al. (2014) Lages (2016) Tan et al. (2017)	Schoofs et al. (2009) Ghosh and Sampath (2011) Sampath and Rao (2011)
General-purpose systems	Weisbach et al. (2011) Gabrielli et al. (2014)	Auerbach et al. (2007)
High-performance computing	Yun et al. (2013)	-
Network applications	Goldoni et al. (2009) Baumgartner and Schoenegger (2010) Mercado et al. (2011) Nguyen et al. (2013)	Khawer and So (2010) Khawer and Easwaran (2014)

In case of conjoined works, scientific was selected.

Table 3. Possible Scenarios and Limitations of the PREEMPT_RT Patch in Different Fields

	Possible Scenarios	Limitations
Embedded systems	• Any non-safety-critical application • Automotive, avionics, medical, robotics, etc. • Infotainment systems • Slow regulators	• No mathematical guarantees • No native fault-tolerance mechanism • Hard to certify
Network applications	• Network devices, e.g., firewalls, routers, etc. • Real-time networks	• Throughput degradation • It usually requires specialized protocols
General-purpose systems	• Audio and video applications • Physical system simulators • Data simulation and analysis	• Throughput degradation • Difficult to control interferences from other applications
High-performance computing	• Fixed or adaptive QoS for HPC applications • Interactive applications that require big computational power	• Custom kernels sometimes not allowed in supercomputers • Complex management of non-real-time internode networks

Aichouch et al. 2013a; Sheridan-Barbian 2015). In 2010, Yu et al. (2010) proposed a scheduler for the Xen hypervisor able to correctly manage real-time guests. This filled the lack shown by the standard Xen scheduler in meeting real-time requirements, even for maximum priority guests.

In other scenarios, Linux PREEMPT_RT has also been deployed as host operating system in conjunction with the *Kernel-based Virtual Machine (KVM)*. KVM substantially consists of a kernel module that allows the user to run a guest operating system. In this regard, the first work available is Schild et al. (2009), who evaluated the overhead in terms of latencies introduced by KVM over a Linux PREEMPT_RT system. They found that the impact of KVM on interrupt latencies is lower than $\leq 3\mu s$. The same year, Kiszka (2009) presented a detailed analysis of the real-time capabilities of Linux PREEMPT_RT when used as both the host and guest OS, in conjunction with the KVM hypervisor. They quantified the overall system overhead in the range from 15% to 25% including scheduling latency obtained in the guest operating system under $< 1ms$. Later approaches for industrial control systems include the work by Ghosh and Sampath (2011) and by Sampath and Rao (2011), the last focused on QEMU.

Aichouch et al. (2013b) instead presented a preliminary evaluation of LITMUS^{RT} in terms of scheduling and interrupt latencies in two scenarios: (1) installed as main RTOS and (2) virtualized on top of a Linux PREEMPT_RT with QEMU/KVM virtualization system. An automotive scenario was presented by Hamayun et al. (2014), where several possible solutions were evaluated, including cokernel approaches. However, at the end they suggested a custom RTOS that, besides the high development cost, allows the system to meet the hard real-time constraints required by automotive environment.

A mixed work was proposed by Zuo et al. (2010), who used a vanilla Linux kernel with KVM as host and Linux PREEMPT_RT as guest. They identified an acceptable average latency ($< 26\mu s$) but a nonacceptable worst-case latency ($> 1ms$). They observed how most of the latency spikes were caused by SMI enabled in the host machine.

The PREEMPT_RT patch showed good results in latency optimization in cloud computing. In 2015, Mao et al. (2015) used Docker (a container-based virtualization system) over a PREEMPT_RT Linux system, noticing an improvement in terms of latency reduction on both the host and containers.

5.3 Derivative Works

In this section, we briefly mention a couple of derivative works/uses of the PREEMPT_RT patch.

The *ChronOS Linux* (Dellinger et al. 2011) is a Linux kernel variant that includes the PREEMPT_RT patch. It is focused on the implementation of different schedulers for multiprocessor systems aimed at improving the scheduling of both real-time and best-effort applications. In fact, it has been used in different scientific works, in particular for the development of scheduling algorithms.

Finally, a specialized derivative work was presented by Wings et al. (2015) as a variant of LinuxCNC (i.e., a Linux distribution for CNC machines). The idea was to implement a real-time Ethernet interface in LinuxCNC using PREEMPT_RT to increase the real-time capabilities at the kernel level, thus removing the legacy RTAI interface, which cannot support the required user-space driver.

6 RELATED WORK

Several surveys (Stankovic and Rajkumar 2004; Baskiyar and Meghanathan 2005; Anh and Tan 2009; Davis and Burns 2011; Hambarde et al. 2014) and books (Laplante et al. 2004; Stoyenko 2012) on real-time operating systems and a massive quantity of related scientific papers have been written since the 2000s. However, none of them has systematically discussed Linux strategies for real time. Vun et al. (2008) presented a very short survey on real-time Linux variants, including

Xenomai and RTAI. In early years, Dietrich and Walker (2005) and Scordino and Lipari (2006) suggested possible paths of Linux development and research for real time.

Concerning real-time Linux in virtualization-based contexts, Gu and Zhao (2012) presented a survey on challenges in embedded system virtualization, focusing on Linux-based solutions. Recently, Sheridan-Barbian (2015) discussed real-time virtualization, providing a brief description of Linux PREEMPT_RT.

To the best of our knowledge, no survey on PREEMPT_RT Linux is available in the literature. The relevant number of articles spread over the last 15 years and the increasing interest in real-time multicore and mixed-criticality systems, combined with the increasing challenges discussed in Section 1, justify the necessity of a methodical survey covering the development of Linux PREEMPT_RT and the application scenarios that can benefit from it.

7 CONCLUSIONS AND FUTURE WORKS

Researchers and engineers are facing new challenges in both industrial and scientific environments for the future real-time systems, due to the growing complexity of architectures. It is necessary to find solutions to satisfy the increasing demand of computational power required by new applications while maintaining the timing constraints. This work surveyed the available literature regarding the use of the Linux kernel for real-time applications, particularly focusing on the PREEMPT_RT patch. This patch has been developed in the last decade with the aim of introducing real-time capabilities in the Linux kernel, avoiding the adoption of dual-kernel (or cokernel) approaches. The dual-kernel approaches currently perform better than PREEMPT_RT in terms of system predictability and absolute latencies. However, they introduce disadvantages by making the application development more complex, which means harder maintainability and lower portability.

The PREEMPT_RT patch and its features have been presented together with the survey of performance analysis and the comparisons with the state-of-the-art RTOSs. Some industrial works available in the literature have been mentioned and briefly described.

We can conclude that the PREEMPT_RT patch seems promising when adopted in industrial environments only if the hard real-time requirements are not dictated by safety-critical constraints. Linux could not in fact provide any formal guarantee of WCET using classical static analysis tools. This is due to both the kernel complexity and intrinsic unpredictability of modern architectures. Besides embedded scenarios, other scenarios can take advantage of the latency reduction and increased predictability provided by PREEMPT_RT, e.g., multimedia, HPC, and network applications. Time predictability has gained interest in the HPC community, where the Linux kernel is extensively used (Flich et al. 2018).

For academic and research purposes, PREEMPT_RT is a potential candidate for the development of both applications and test benches. In the first case, any type of application can be implemented and tested in a real-time environment with less effort rather than using complex RTOSs. In test-bench cases, the experimenters can use Linux to test the performance of scheduling algorithms, IPC calls, and any other operating system mechanisms. It is important that researchers correctly configure the Linux system in order to have realistic latencies. This activity has been sometimes neglected and may lead to erroneous or unreliable conclusions.

As we said, most of the systems that require hard real-time constraints typically have safety requirements. Even though these implications are beyond the scope of this work, they have to be properly evaluated for PREEMPT_RT. Linux cokernel approaches can be certified for SIL2—a detailed description can be found in the reports of Pierce (2002) and Berger (2010)—and this should be a goal also for PREEMPT_RT in order to allow the industry to exploit the advantages of using real-time Linux. In 2015, the Open Source Automation Laboratory started a project denominated

SIL2LinuxMP[15] with the goal of certificating the base components of Linux both in single-core- and in multicore-based systems. Cotroneo et al. (2016) presented a work in 2016 on the certification of open-source operating systems. The safety-related research on open source over the COTS platform may be of great interest in the next few years, especially for embedded applications. From one point of view, all sources of previously described uncertainties have to be addressed in the hardware, operating system, runtime, and application layers. On the other hand, all nonfunctional requirements—which typically characterize the design of embedded systems—have to be taken into account, considering also that they are frequently in contrast with real-time requirements.

These issues are becoming more and more challenging considering the transition from multicore to many-core systems (Cucinotta et al. 2017). User-space and kernel-space processes running simultaneously and/or concurrently on a large set of cores require properly tuned scheduling algorithms with the lowest possible computational complexity. Addressing these challenges is much harder in a real-time operating system, since the contention on kernel resources potentially entails longer waiting time on locks and, as a consequence, increased latencies.

Furthermore, recent state-of-the-art software architectures include a resource management middleware (or runtime) able to make scheduling decisions with a long-term goal. In Linux systems, resource managers use CGroup subsystems (Menage et al. 2008; Bellasi et al. 2015) to interact with the scheduler. Unfortunately, group scheduling is not supported when PREEMPT_RT is applied. An earlier attempt to support the group scheduling with the real-time patch was developed by Watkins et al. (2009). This is one of the most critical limitations of using Linux PREEMPT_RT in complex soft real-time application scenarios. Resource managers are also essential in scheduling decisions in heterogeneous computing systems; the real-time constraints pose several challenges to be addressed for these kinds of architectures.

Concerning the real-time classification of Linux, we can state that Linux cannot be considered strict hard real time, at least not for safety-critical scenarios. As discussed in this survey, it is not currently possible to formally and accurately predict the latencies of the Linux kernel and consequently the WCET of a real-time application. For this reason, the classical definition of a hard real-time operating system cannot be applied. Other classifications have been proposed in the literature, but they are not widely accepted.

In Section 4, we presented the current state-of-the-art performance analysis on the real-time capabilities of Linux, but further systematic investigations on the kernel latencies and jitters is needed to improve both the GPOS-to-RTOS transition pursued by PREEMPT_RT and the timing analysis of such a complex operating system. The ongoing evolution of the kernel and the lack of reproducibility of several works call for more systematic and accurate analysis. Even if still immature, another possible future option is to include the concept of probabilistic hard real time (Bernat et al. 2002; Reghenzani et al. 2018), which has been continuously evolving in the last two decades.

REFERENCES

Jaume Abella, Carles Hernandez, Eduardo Quiñones, Francisco J. Cazorla, Philippa Ryan Conmy, Mikel Azkarate-askasua, Jon Perez, Enrico Mezzetti, and Tullio Vardanega. 2015. WCET analysis methods: Pitfalls and challenges on their trustworthiness. In *10th IEEE International Symposium on Industrial Embedded Systems (SIES'15)*. 1–10. DOI: https://doi.org/10.1109/SIES.2015.7185039

Luca Abeni and Giorgio Buttazzo. 1998. Integrating multimedia applications in hard real-time systems. In *Proceedings of the 19th IEEE Real-Time Systems Symposium*. 4–13. DOI: https://doi.org/10.1109/REAL.1998.739726

Luca Abeni, Nicola Manica, and Luigi Palopoli. 2009. Reservation-based scheduling for irq threads. In *Proceedings of the 11th OSADL Real-Time Linux Workshop*. TU Dresden Faculty of Computer Science, Dresden, Germany, 179–186.

[15]Website of the SIL2LinuxMP project: http://www.osadl.org/SIL2LinuxMP.sil2-linux-project.0.html.

Ankit Agrawal, Gerhard Fohler, Johannes Freitag, Jan Nowotsch, Sascha Uhrig, and Michael Paulitsch. 2017. Contention-aware dynamic memory bandwidth isolation with predictability in COTS multicores: An avionics case study. In *29th Euromicro Conference on Real-Time Systems (ECRTS'17) (Leibniz International Proceedings in Informatics (LIPIcs))*, Marko Bertogna (Ed.), Vol. 76. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 2:1–2:22. DOI: https://doi.org/10.4230/LIPIcs.ECRTS.2017.2

Mehdi Aichouch, Jean-Christophe Prevotet, and Fabienne Nouvel. 2013b. Evaluation of an RTOS on top of a hosted virtual machine system. In *2013 IEEE Conference on Design and Architectures for Signal and Image Processing (DASIP'13).* University of Cagliari, Cagliari, Italy, 290–297.

Mehdi Aichouch, Jean-Christophe Prévotet, and Fabienne Nouvel. 2013a. Evaluation of the overheads and latencies of a virtualized RTOS. In *8th IEEE International Symposium on Industrial Embedded Systems (SIES'13).* 81–84. DOI: https://doi.org/10.1109/SIES.2013.6601475

James H. Anderson, John M. Calandrino, and UmaMaheswari C. Devi. 2006. Real-time scheduling on multicore platforms. In *12th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'06).* IEEE, 179–190. DOI: https://doi.org/10.1109/RTAS.2006.35

Tran Nguyen Bao Anh and Su-Lim Tan. 2009. Real-time operating systems for small microcontrollers. *IEEE Micro* 29, 5 (Sept. 2009), 30–45. DOI: https://doi.org/10.1109/MM.2009.86

Maya Arbel and Adam Morrison. 2015. Predicate RCU: An RCU for scalable concurrent updates. In *Proceedings of the 20th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP'15).* ACM, New York, 21–30. DOI: https://doi.org/10.1145/2688500.2688518

Siro Arthur, Carsten Emde, and Nicholas McGuire. 2007. Assessment of the realtime preemption patches (RT-Preempt) and their impact on the general purpose performance of the system. In *Proceedings of the 9th OSADL Real-Time Linux Workshop.* Institute for Measurement Technology, Linz, Austria.

Joshua S. Auerbach, David F. Bacon, Florian Bömers, and Perry Cheng. 2007. Real-time music synthesis in Java using the metronome garbage collector. In *Proceedings of International Computer Music Conference*, Vol. 2007. 103–110. DOI: https://doi.org/2027/spo.bbp2372.2007.132

Michael Barabanov. 1997. *A Linux-Based Real-Time Operating System.* Ph.D. Dissertation. Socorro, NM. Advisor(s) Yodaiken, Victor J.

Antonio Barbalace, A. Luchetta, G. Manduchi, M. Moro, A. Soppelsa, and C. Taliercio. 2008. Performance comparison of VxWorks, Linux, RTAI, and Xenomai in a hard real-time application. *IEEE Transactions on Nuclear Science* 55, 1 (Feb. 2008), 435–439. DOI: https://doi.org/10.1109/TNS.2007.905231

Antonio Barbalace, Gabriele Manduchi, Andre Neto, Gianmaria De Tommasi, Filippo Sartori, and Daniel F. Valcarcel. 2011. Performance comparison of EPICS IOC and MARTe in a hard real-time control application. *IEEE Transactions on Nuclear Science* 58, 6 (Dec. 2011), 3162–3166. DOI: https://doi.org/10.1109/TNS.2011.2167350

Sanjeev Baskiyar and Natarajan Meghanathan. 2005. A survey of contemporary real-time operating systems. *Informatica, Slovenian Society Informatika* 29, 2 (2005), 233–240.

Josef Baumgartner and Stefan Schoenegger. 2010. POWERLINK and real-time Linux: A perfect match for highest performance in real applications. In *Proceedings of the 12th OSADL Real-Time Linux Workshop.* Strathmore University, Nairobi, Kenya.

Raphaël Beamonte, Francis Giraldeau, and Michel Dagenais. 2012. High performance tracing tools for multicore Linux hard real-time systems. In *Proceedings of the 14th OSADL Real-Time Linux Workshop.* Department of Computer Science, University of North Carolina, Chapel Hill, NC.

Patrick Bellasi, Giuseppe Massari, and William Fornaciari. 2015. Effective runtime resource management using Linux control groups with the BarbequeRTRM framework. *ACM Transactions on Embedded Computing Systems (TECS)* 14, 2, Article 39 (March 2015), 17 pages. DOI: https://doi.org/10.1145/2658990

Robert Berger. 2010. Embedded GNU/Linux and real-time an executive summary. In *Proceedings of Embedded World 2010.*

G. Bernat, A. Burns, and A. Liamosi. 2001. Weakly hard real-time systems. *IEEE Transactions on Computing* 50, 4 (Apr. 2001), 308–321. DOI: https://doi.org/10.1109/12.919277

Guillem Bernat, Antoine Colin, and Stefan M. Petters. 2002. WCET analysis of probabilistic hard real-time systems. In *23rd IEEE Real-Time Systems Symposium, 2002 (RTSS'02).* IEEE, 279–288. DOI: https://doi.org/10.1109/REAL.2002.1181582

Wolfgang Betz, Marco Cereia, and Ivan Cibrario Bertolotti. 2009. Experimental evaluation of the Linux RT patch for realtime applications. In *IEEE Conference on Emerging Technologies & Factory Automation (ETFA'09).* IEEE, 1–4. DOI: https://doi.org/10.1109/ETFA.2009.5347056

Christian Bienia, Sanjeev Kumar, Jaswinder Pal Singh, and Kai Li. 2008. The PARSEC benchmark suite: Characterization and architectural implications. In *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques.* ACM, New York, 72–81. DOI: https://doi.org/10.1145/1454115.1454128

Tim Bird. 2006. Measuring function duration with ftrace. In *Proceedings of the Linux Symposium*, Vol. 1. Ottawa, Canada, 47–54.

Konstantinos Bletsas and Björn Andersson. 2009. Notional processors: An approach for multiprocessor scheduling. In *15th IEEE Real-Time and Embedded Technology and Applications Symposium*. 3–12. DOI: https://doi.org/10.1109/RTAS.2009.25

Quentin Bombled and Olivier Verlinden. 2012. Dynamic simulation of six-legged robots with a focus on joint friction. *Multibody System Dynamics* 28, 4 (Nov. 2012), 395–417. DOI: https://doi.org/10.1007/s11044-012-9305-z

Gert Bonestroo. 2012. *A Safe-Guarded Multi-Agent Control System for Tripod.* Master's thesis. University of Twente, Enschede, Netherlands.

Pedro Braga, Luís Henriques, Bruno Carvalho, Philippe Chevalley, and Marco Zulianello. 2008. xLuna-D Emonstrator on ESA Mars Rover. In *2008 Data Systems in Aerospace (DASIA'08)*, Vol. 665.

Björn B. Brandenburg and Andrea Bastoni. 2012. The case for migratory priority inheritance in Linux: Bounded priority inversions on multiprocessors. In *Proceedings of the 14th OSADL Real-Time Linux Workshop.* Department of Computer Science, University of North Carolina, Chapel Hill, NC.

Jeremy H. Brown and Brad Martin. 2010. How fast is fast enough? Choosing between Xenomai and Linux for real-time applications. In *Proceedings of the 12th OSADL Real-Time Linux Workshop.* Strathmore University, Nairobi, Kenya.

Alan Burns and Robert Davis. 2013. *Mixed Criticality Systems-A Review.* Technical Report. Department of Computer Science, University of York, York, UK.

Giorgio Buttazzo. 2011. *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications* (3rd ed.). Real-Time Systems Series, Vol. 24. Springer US. DOI: https://doi.org/10.1007/978-1-4614-0676-1

Giorgio Buttazzo, Giuseppe Lipari, Luca Abeni, and Marco Caccamo. 2005. *Soft Real-Time Systems: Predictability vs. Efficiency.* Springer US. DOI: https://doi.org/10.1007/0-387-28147-9

John M. Calandrino, Hennadiy Leontyev, Aaron Block, UmaMaheswari C. Devi, and James H. Anderson. 2006. LITMUSˆ RT: A testbed for empirically comparing real-time multiprocessor schedulers. In *2006 27th IEEE International Real-Time Systems Symposium (RTSS'06).* IEEE, 111–126. DOI: https://doi.org/10.1109/RTSS.2006.27

Andreu Carminati. 2012. Implementation and evaluation of the synchronization protocol immediate priority ceiling in PREEMPT-RT Linux. *Journal of Software* 7, 3 (Mar. 2012), 516–525. DOI: https://doi.org/10.4304/jsw.7.3.516-525

Felipe Cerqueira and Björn Brandenburg. 2013. A comparison of scheduling latency in Linux, PREEMPT-RT, and LITMUS RT. In *Proceedings of International Workshop on Operating Systems Platforms for Embedded Real-Time Applications (OSPERT'13).* SYSGO AG, 19–29.

Konstantinos Chalas. 2015. *Evaluation of Real-time Operating Systems for FGC Controls.* Technical Report CERN-STUDENTS-Note-2015-201.

Sudipta Chattopadhyay, Lee Kee Chong, Abhik Roychoudhury, Timon Kelter, Peter Marwedel, and Heiko Falk. 2014. A unified WCET analysis framework for multicore platforms. *ACM Transactions on Embedded Computing Systems (TECS)* 13, 4s, Article 124 (April 2014), 29 pages. DOI: https://doi.org/10.1145/2584654

Zhuoqun Cheng, Richard West, and Ying Ye. 2017. Building real-time embedded applications on QduinoMC: A web-connected 3D printer case study. In *Proceedings of IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'17).* 13–24. DOI: https://doi.org/10.1109/RTAS.2017.39

Xenomai Community. 2011. dohell - Manual Page. Retrieved January 9, 2018, from https://xenomai.org/documentation/xenomai-2.6/html/dohell/index.html.

Jonathan Corbet. 2003. Using read-copy-update. *LWN.net* (Jun. 2003). Retrieved from https://lwn.net/Articles/37889/

Jonathan Corbet. 2005. Realtime preemption and read-copy-update. *LWN.net* (Mar. 2005). Retrieved from https://lwn.net/Articles/129511/

Jonathan Corbet. 2006. Priority inheritance in the kernel. *LWN.net* (Apr. 2006). Retrieved from https://lwn.net/Articles/178253/

Jonathan Corbet. 2013. Full tickless operation in 3.10. *LWN.net* (May 2013). Retrieved from https://lwn.net/Articles/549580/

Domenico Cotroneo, Domenico Di Leo, Roberto Natella, and Roberto Pietrantuono. 2016. Prediction of the testing effort for the safety certification of open-source software: A case study on a real-time operating system. In *2016 12th European Dependable Computing Conference (EDCC'16).* 141–152. DOI: https://doi.org/10.1109/EDCC.2016.22

Tommaso Cucinotta, Giuseppe Lipari, and Lutz Schubert. 2017. Operating system and scheduling for future multicore and many-core platforms. *Programming Multicore and Many-core Computing Systems* 86 (Jan. 2017). DOI: https://doi.org/10.1002/9781119332015.ch22

Fabian Dalbert. 2016. *Real-Time Field Bus Systems with Linux.* Master's thesis. ETH Zürich, Zurich, CHE. Advisor(s) Pengcheng Huang, Georgia Giannopoulou, Tonio Gsell, Gert Brettlecker, Lothar Thiele. SA-2015-21.

Neil T. Dantam, Daniel M. Lofaro, Ayonga Hereid, Paul Y. Oh, Aaron D. Ames, and Mike Stilman. 2015. The Ach library: A new framework for real-time communication. *IEEE Robotics Automation Magazine* 22, 1 (March 2015), 76–85. DOI: https://doi.org/10.1109/MRA.2014.2356937

Dakshina Dasari, Benny Akesson, Vincent Nelis, Muhammad Ali Awan, and Stefan M. Petters. 2013. Identifying the sources of unpredictability in COTS-based multicore systems. In *2013 8th IEEE International Symposium on Industrial Embedded Systems (SIES'13).* 39–48. DOI: https://doi.org/10.1109/SIES.2013.6601469

Robert I. Davis and Alan Burns. 2011. A survey of hard real-time scheduling for multiprocessor systems. *ACM Computing Surveys (CSUR)* 43, 4, Article 35 (Oct. 2011), 44 pages. DOI:https://doi.org/10.1145/1978802.1978814

Laurens De Haan and Ana Ferreira. 2007. *Extreme Value Theory: An Introduction.* Springer-Verlag New York. DOI:https://doi.org/10.1007/0-387-34471-3

Daniel B. de Oliveira, Rômulo S. de Oliveira, Tommaso Cucinotta, and Luca Abeni. 2017. Automata-based modeling of interrupts in the Linux PREEMPT RT kernel. In *2017 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA'17).* 1–8. DOI:https://doi.org/10.1109/ETFA.2017.8247611

Matthew Dellinger, Piyush Garyali, and Binoy Ravindran. 2011. ChronOS Linux: A best-effort real-time multiprocessor Linux kernel. In *2011 48th ACM/EDAC/IEEE Design Automation Conference (DAC'11).* IEEE, 474–479.

Mathieu Desnoyers and Michel R. Dagenais. 2006. The lttng tracer: A low impact performance and behavior monitor for GNU/Linux. In *Proceedings of the Linux Symposium*, Vol. 1. Ottawa, Canada, 209–224.

Mathieu Desnoyers, Paul E. McKenney, Alan S. Stern, Michel R. Dagenais, and Jonathan Walpole. 2012a. User-level implementations of read-copy update. *IEEE Transactions on Parallel and Distributed Systems* 23, 2 (Feb. 2012), 375–382. DOI:https://doi.org/10.1109/TPDS.2011.159

Mathieu Desnoyers, Paul E. McKenney, Alan S. Stern, Michel R. Dagenais, and Jonathan Walpole. 2012b. User-level implementations of read-copy update. *IEEE Transactions on Parallel and Distributed Systems* 23, 2 (Feb. 2012), 375–382. DOI:https://doi.org/10.1109/TPDS.2011.159

Sven-Thorsten Dietrich and Daniel Walker. 2005. The evolution of real-time Linux. In *Proceedings of the 7th Open Source Automation Development Lab Real-Time Linux Workshop.* University for Science and Technology of Lille, Lille, France.

DO-178B. 1992. DO-178B, Software Considerations in Airborne Systems and Equipment Certification.

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

Jake Edge. 2008. Moving interrupts to threads. *LWN.net* (Oct. 2008). Retrieved from https://lwn.net/Articles/302043/Glenn A. Elliott and James H. Anderson. 2012. The limitations of fixed-priority interrupt handling in PREEMPT RT and alternative approaches. In *Proceedings of the 14th OSADL Real-Time Linux Workshop.* Department of Computer Science, University of North Carolina, Chapel Hill, NC, 149–155.

Carsten Emde. 2010. Long-term monitoring of apparent latency in PREEMPT RT Linux realtime systems. In *Proceedings of the 12th Real-Time Linux Workshop.*

Keith G. Erickson, M. Dan Boyer, and D. Higgins. 2018. NSTX-U advances in real-time deterministic PCIe-based internode communication. *Fusion Engineering and Design* 133 (2018), 104–109. DOI:https://doi.org/10.1016/j.fusengdes.2018.02.055

Hadi Esmaeilzadeh, Emily Blem, Renee St. Amant, Karthikeyan Sankaralingam, and Doug Burger. 2011. Dark silicon and the end of multicore scaling. In *ACM SIGARCH Computer Architecture News*, Vol. 39. ACM, New York, 365–376. DOI:https://doi.org/10.1145/2024723.2000108

Dario Faggioli, Fabio Checconi, Michael Trimarchi, and Claudio Scordino. 2009. An EDF scheduling class for the Linux kernel. In *Proceedings of the 11th OSADL Real-Time Linux Workshop.* Technische Universität Dresden, Dresden, Germany, 197–204.

Hasan Fayyad-Kazan, Luc Perneel, and Martin Timmerman. 2013b. Full and para-virtualization with Xen: A performance comparison. *Journal of Emerging Trends in Computing and Information Sciences* 4, 9 (2013) 719–727.

Hasan Fayyad-Kazan, Luc Perneel, and Martin Timmerman. 2013a. Linux PREEMPT-RT vs. commercial RTOSs: How big is the performance gap? *GSTF Journal on Computing (JoC)* 3, 1 (2013), 136.

Hasan Fayyad-Kazan, Luc Perneel, and Martin Timmerman. 2014. Linux PREEMPT-RT v2. 6.33 versus v3. 6.6: Better or worse for real-time applications? *ACM SIGBED Review* 11, 1 (Feb. 2014), 26–31. DOI:https://doi.org/10.1145/2597457.2597460

Alexandra Fedorova, Sergey Blagodurov, and Sergey Zhuravlev. 2010. Managing contention for shared resources on multicore processors. *Communications of the ACM* 53, 2 (Feb. 2010), 49–57. DOI:https://doi.org/10.1145/1646353.1646371

Taís Borges Ferreira, Márcia Aparecida Fernandes, and Rivalino Matias Jr. 2012. A comprehensive complexity analysis of user-level memory allocator algorithms. In *2012 Brazilian Symposium on Computing System Engineering.* 99–104. DOI:https://doi.org/10.1109/SBESC.2012.27

Peter Feuerer. 2007. *Benchmark and Comparison of Real-Time Solutions Based on Embedded Linux.* Master's thesis. Hochschule Ulm, Germany.

Steven A. Finney. 2001. Real-time data collection in Linux: A case study. *Behavior Research Methods, Instruments, & Computers* 33, 2 (2001), 167–173. DOI:https://doi.org/10.3758/BF03195362

José Flich, Giovanni Agosta, Philipp Ampletzer, David Atienza Alonso, Carlo Brandolese, Etienne Cappe, Alessandro Cilardo, Leon Dragić, Alexandre Dray, Alen Duspara, William Fornaciari, Gerald Guillaume, Ynse Hoornenborg, Arman Iranfar, Mario Kovač, Simone Libutti, Bruno Maitre, José Maria Martínez, Giuseppe Massari, Hrvoje Mlinarić, Ermis Pa-pastefanakis, Tomás Picornell, Igor Piljić, Anna Pupykina, Federico Reghenzani, Isabelle Staub, Rafael Tornero, Marina Zapater, and Davide Zoni. 2017. MANGO: Exploring manycore architectures for next-generation HPC systems. In *2017 Euromicro Conference on Digital System Design (DSD'17).* 478–485. DOI:https://doi.org/10.1109/DSD.2017.51

José Flich, Giovanni Agosta, Philipp Ampletzer, David Atienza Alonso, Carlo Brandolese, Etienne Cappe, Alessandro Cilardo, Leon Dragić, Alexandre Dray, Alen Duspara, William Fornaciari, Edoardo Fusella, Mirko Gagliardi, Gerald Guillaume, Daniel Hofman, Ynse Hoornenborg, Arman Iranfar, Mario Kovač, Simone Libutti, Bruno Maitre, José Maria Martínez, Giuseppe Massari, Koen Meinds, and Hrvoje Mlinarić et al. 2018. Exploring manycore architectures for next-generation HPC systems through the MANGO approach. *Microprocessors and Microsystems* 61 (2018), 154–170. DOI: https://doi.org/10.1016/j.micpro.2018.05.011

William Fornaciari, Giovanni Agosta, David Atienza, Carlo Brandolese, Leila Cammoun, Luca Cremona, Alessandro Cilardo, Albert Farres, José Flich, Carles Hernandez, Michal Kulchewski, Simone Libutti, José Maria Martínez, Giuseppe Massari, Ariel Oleksiak, Anna Pupykina, Federico Reghenzani, Rafael Tornero, Michele Zanella, Marina Zapater, and Davide Zoni. 2018. Reliable power and time-constraints-aware predictive management of heterogeneous exascale systems. In *2018 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS'18).*

Leonardo Gabrielli, Michele Bussolotto, and Stefano Squartini. 2014. Reducing the latency in live music transmission with the BeagleBoard xM through resampling. In *2014 6th European Embedded Design in Education and Research Conference (EDERC'14).* IEEE, 302–306. DOI: https://doi.org/10.1109/EDERC.2014.6924409

Carlos Garre, Domenico Mundo, Marco Gubitosa, and Alessandro Toso. 2014b. *Performance Comparison of Real-Time and General-Purpose Operating Systems in Parallel Physical Simulation with High Computational Cost.* Technical Report. SAE Technical Paper.

Carlos Garre, Domenico Mundo, Marco Gubitosa, and Alessandro Toso. 2014a. Real-time and real-fast performance of general-purpose and real-time operating systems in multithreaded physical simulation of complex mechanical systems. *Mathematical Problems in Engineering* 2014 (2014). DOI: https://doi.org/10.1155/2014/945850

Mohamad Gebai and Michel R. Dagenais. 2018. Survey and analysis of kernel and userspace tracers on Linux: Design, implementation, and overhead. *ACM Computing Surveys* 51, 2, Article 26 (March 2018), 33 pages. DOI: https://doi.org/10.1145/3158644

Philippe Gerum. 2004. Xenomai-implementing a RTOS emulation framework on GNU/Linux. *White Paper*, 81.

Sanjay Ghosh and Pradyumna Sampath. 2011. Evaluation of embedded virtualization on real-time Linux for industrial control system. In *Proceedings of the 13th OSADL Real-Time Linux Workshop.* Faculty of Electrical Engineering, Czech Technical University, Prague, Czechia, 173–180.

Samuel Jiménez Gil, Iain Bate, George Lima, Luca Santinelli, Adriana Gogonel, and Liliana Cucu-Grosjean. 2017. Open challenges for probabilistic measurement-based worst-case execution time. *IEEE Embedded Systems Letters* 9, 3 (Sept. 2017), 69–72. DOI: https://doi.org/10.1109/LES.2017.2712858

Thomas Gleixner. 2010. Cyclictest. Retrieved January 9, 2018, from https://wiki.linuxfoundation.org/realtime/documentation/howto/tools/cyclictest.

Thomas Gleixner and Douglas Niehaus. 2006. Hrtimers and beyond: Transforming the Linux time subsystems. In *Proceedings of the Linux Symposium*, Vol. 1. Ottawa, Canada, 333–346.

Emanuele Goldoni, Giuseppe Rossi, and Alberto Torelli. 2009. Assolo, a new method for available bandwidth estimation. In *2009 4th International Conference on Internet Monitoring and Protection.* IEEE, 130–136. DOI: https://doi.org/10.1109/ICIMP.2009.28

Luis Claudio R. Gonçalves and Arnaldo Carvalho de Melo. 2008. Application testing under realtime Linux. In *Proceedings of the Linux Symposium.* Ottawa, Canada, 143–150.

Mark Gottscho, Mohammed Shoaib, Sriram Govindan, Bikash Sharma, Di Wang, and Puneet Gupta. 2017. Measuring the impact of memory errors on application performance. *IEEE Computer Architecture Letters* 16, 1 (Jan. 2017), 51–55. DOI: https://doi.org/10.1109/LCA.2016.2599513

Austin Joint Working Group. 1999. IEEE standard for information technology-portable operating system interface (POSIX)-Part 1: System application program interface (API)-amendment D: Additional real time extensions [C Language]. *IEEE Std 1003.1d-1999* (1999), 1–114. DOI: https://doi.org/10.1109/IEEESTD.1999.91515

Zonghua Gu and Qingling Zhao. 2012. A state-of-the-art survey on real-time issues in embedded systems virtualization. *Journal of Software Engineering and Applications* 5, 4 (2012), 277. DOI: https://doi.org/10.4236/jsea.2012.54033 Dinakar Guniguntala, Paul E. McKenney, Josh Triplett, and Jonathan Walpole. 2008. The read-copy-update mechanism for supporting real-time applications on shared-memory multiprocessor systems with Linux. *IBM Systems Journal* 47, 2 (2008), 221–236. DOI: https://doi.org/10.1147/sj.472.0221

Jan Gustafsson, Adam Betts, Andreas Ermedahl, and Björn Lisper. 2010. The Mälardalen WCET benchmarks: Past, present and future. In *10th International Workshop on Worst-Case Execution Time Analysis (WCET'10) (OpenAccess Series in Infor-matics (OASIcs))*, Vol. 15. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 136–146. DOI: https://doi.org/10.4230/OASIcs.WCET.2010.136

Matthew R. Guthaus, Jeffrey S. Ringenberg, Dan Ernst, Todd M. Austin, Trevor Mudge, and Richard B. Brown. 2001. MiBench: A free, commercially representative embedded benchmark suite. In *IEEE International Workshop on Work-load Characterization (WWC-4' 01).* IEEE, 3–14. DOI: https://doi.org/10.1109/WWC.2001.990739

Szabolcs Hajdu and Sándor-Tihamér Brassai. 2015. Implementation of embedded Linux systems on FPGA based circuits for real time process control. In *Proceedings of the 5th International Conference on Recent Achievements in Mechatronics, Automation, Computer Sciences and Robotics (MACRo'15)* 1, 1 (2015), 145–154.

Mian M. Hamayun, Alexander Spyridakis, and Daniel S. Raho. 2014. Towards hard real-time control and infotainment applications in automotive platforms. In *Proceedings of International Workshop on Operating Systems Platforms for Embedded Real-Time Applications (OSPERT'14)*. Euromicro, Madrid, Spain, 39–44.

Prasanna Hambarde, Rachit Varma, and Shivani Jha. 2014. The survey of real time operating system: RTOS. In *2014 International Conference on Electronic Systems, Signal Processing and Computing Technologies*. IEEE, 34–39. DOI: https://doi.org/10.1109/ICESC.2014.15

Parameswaran Ramanathan and Moncef Hamdaoui. 1995. A dynamic priority assignment technique for streams with (m, k)-firm deadlines. *IEEE Transactions on Computers* 44, 12 (Dec. 1995), 1443–1451. DOI: https://doi.org/10.1109/12.477249

Blake Hannaford, Jacob Rosen, Diana W. Friedman, Hawkeye King, Phillip Roan, Lei Cheng, Daniel Glozman, Ji Ma, Sina Nia Kosari, and Lee White. 2013. Raven-II: An open platform for surgical robotics research. *IEEE Transactions on Biomedical Engineering* 60, 4 (Apr. 2013), 954–959. DOI: https://doi.org/10.1109/TBME.2012.2228858

Joachim Henkel. 2006. Selective revealing in open innovation processes: The case of embedded Linux. *Research Policy* 35, 7 (2006), 953–969. DOI: https://doi.org/10.1016/j.respol.2006.04.010

Luis Henriques. 2009. Threaded IRQs on Linux PREEMPT-RT. In *Proceedings of International Workshop on Operating Systems Platforms for Embedded Real-Time Applications (OSPERT'09)*. Euromicro, Dublin, Ireland, 23–32.

M. Teresa Higuera-Toledano and Valérie Issarny. 2002. Analyzing the performance of memory management in RTSJ. In *Proceedings 5th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISIRC'02)*. 26–33. DOI: https://doi.org/10.1109/ISORC.2002.1003657

Stephen Hunter and Tom Basciano. 2015. Advantages and challenges of using COTS industrial networking technology on the international space station. In *The Automation Conference 2015*.

Jinkyu Jeong, Euiseong Seo, Dongsung Kim, Jin-Soo Kim, Joonwon Lee, Yung-Joon Jung, Donghwan Kim, and Kanghee Kim. 2007. Transparent and selective real-time interrupt services for performance improvement. In *IFIP International Workshop on Software Technologies for Embedded and Ubiquitous Systems*. Springer, Berlin, 283–292. DOI: https://doi.org/10.1007/978-3-540-75664-4_28

Petr Kacmarik, Pavel Kovar, Ondrej Jakubov, and Frantisek Vejrazka. 2011. The witch navigator–A software GNSS receiver built on real-time Linux. In *Proceedings of the 13th OSADL Real-Time Linux Workshop*. Faculty of Electrical Engineering, Czech Technical University, Prague, Czechia, 17–28.

Kenji Kaneko, Fumio Kanehiro, Mitsuharu Morisawa, Kazuhiko Akachi, Go Miyamori, Atsushi Hayashi, and Noriyuki Kanehira. 2011. Humanoid robot hrp-4-humanoid robotics platform with lightweight and slim body. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 4400–4407. DOI: https://doi.org/10.1109/IROS.2011.6094465

Dongwook Kang, Woojoong Lee, and Chanik Park. 2007. Kernel thread scheduling in real-time Linux for wearable computers. *ETRI Journal* 29, 3 (2007), 270–280. DOI: https://doi.org/10.4218/etrij.07.0506.0019

Weiyuen Kau, John H. Cornish, Qadeer A. Qureshi, and Shannon A. Wichman. 2000. Method and apparatus for handling system management interrupts (SMI) as well as, ordinary interrupts of peripherals such as PCMCIA cards. Holder Texas Instruments Inc., US Patent 6,112,273.

Mohammad R. Khawer and Shriram K. Easwaran. 2014. Apparatus for multi-cell support in a network. Holder Alcatel Lucent, US Patent 8,634,302.

Mohammad R. Khawer and Lina So. 2010. Core abstraction layer for telecommunication network applications. Alcatel Lucent, US Patent WO2012050939 A1.

Colin Ian King. 2013. Stress-ng. Retrieved January 9, 2018, from http://kernel.ubuntu.com/cking/stress-ng/.

Jan Kiszka. 2009. Towards Linux as a real-time hypervisor. In *Proceedings of the 11th OSADL Real-Time Linux Workshop*. TU Dresden Faculty of Computer Science, Dresden, Germany, 205–214.

Michalis Kokologiannakis and Konstantinos Sagonas. 2017. Stateless model checking of the Linux kernel's hierarchical read-copy-update (Tree RCU). In *Proceedings of the 24th ACM SIGSOFT International SPIN Symposium on Model Checking of Software (SPIN'17)*. ACM, New York, 172–181. DOI: https://doi.org/10.1145/3092282.3092287

Kushal Koolwal. 2009. Investigating latency effects of the Linux real-time preemption patches (PREEMPT RT) on AMD's GEODE LX platform. In *Proceedings of the 11th OSADL Real-Time Linux Workshop*.

Hermann Kopetz. 2011. *Real-Time Systems: Design Principles for Distributed Embedded Applications*. Springer US. DOI: https://doi.org/10.1007/978-1-4419-8237-7

Shuhei Kume, Yoshikazu Kanamiya, and Daisuke Sato. 2009. Towards an open-source integrated development and real-time control platform for robots. In *2008 IEEE International Conference on Robotics and Biomimetics*. IEEE, 204–209. DOI: https://doi.org/10.1109/ROBIO.2009.4913004

Walter Fetter Lages. 2016. *Implementation of Real-Time Joint Controllers*. Springer International Publishing, Cham, 671–702. DOI:https://doi.org/10.1007/978-3-319-26054-9_26

Butler W. Lampson and David D. Redell. 1980. Experience with processes and monitors in Mesa. *Communications of the ACM* 23, 2 (Feb. 1980), 105–117. DOI:https://doi.org/10.1145/358818.358824

Phillip A. Laplante. 2004. *Real-Time Systems Design and Analysis* (3rd ed.). Wiley New York. DOI:https://doi.org/10.1002/0471648299

E. L. Lawler and J. M. Moore. 1969. A functional equation and its application to resource allocation and sequencing problems. *Management Science* 16, 1 (1969), 77–84. DOI:https://doi.org/10.1287/mnsc.16.1.77

Edward A. Lee. 2008. Cyber physical systems: Design challenges. In *2008 11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC'08)*. 363–369. DOI:https://doi.org/10.1109/ISORC.2008.25

Hannu Leppinen. 2017. Current use of Linux in spacecraft flight software. *IEEE Aerospace and Electronic Systems Magazine* 32, 10 (Oct. 2017), 4–13. DOI:https://doi.org/10.1109/MAES.2017.160182

Lihao Liang, Paul E. McKenney, Daniel Kroening, and Tom Melham. 2018. Verification of tree-based hierarchical read-copy update in the Linux kernel. In *2018 Design, Automation Test in Europe Conference Exhibition (DATE'18)*. 61–66. DOI:https://doi.org/10.23919/DATE.2018.8341980

Jochen Liedtke. 1995. On micro-kernel construction. In *Proceedings of the 15th ACM Symposium on Operating Systems Principles (SOSP'95)*. ACM, New York, 237–250. DOI:https://doi.org/10.1145/224056.224075

Nabil Litayem and Slim Ben Saoud. 2011. Impact of the Linux real-time enhancements on the system performances for multi-core intel architectures. *International Journal of Computer Applications* 17, 3 (March 2011).

Robert Love and Andrew Morton. 2004. *Linux Kernel Development*. Vol. 1. Sams.

Konstantin Macarenco, Kristina Frye, Benjamin Hamlin, and Karen L. Karavanic. 2016. The effects of system management interrupts on multithreaded, hyper-threaded, and MPI applications. In *2016 45th International Conference on Parallel Processing Workshops (ICPPW'16)*. 338–345. DOI:https://doi.org/10.1109/ICPPW.2016.55

Nicola Manica, Luca Abeni, Luigi Palopoli, Dario Faggioli, and Claudio Scordino. 2010. Schedulable device drivers: Implementation and experimental results. *Proceedings of International Workshop on Operating Systems Platforms for Em-bedded Real-Time Applications (OSPERT'10)*, 53–62. Retrieved from http://www.artist-embedded.org/docs/Events/2010/OSPERT/OSPERT2010-Proceedings.pdf.

Paolo Mantegazza, Lorenzo Dozio, and S. Papacharalambous. 2000. RTAI: Real time application interface. *Linux Journal* 2000, 72es, Article 10 (2000), 10 pages. Retrieved from http://dl.acm.org/citation.cfm?id=348554.348564

Chen-Nien Mao, Mu-Han Huang, Satyajit Padhy, Shu-Ting Wang, Wu-Chun Chung, Yeh-Ching Chung, and Cheng-HsiHsu. 2015. Minimizing latency of real-time container cloud for software radio access networks. In *2015 IEEE 7th Inter-national Conference on Cloud Computing Technology and Science (CloudCom'15)*. 611–616. DOI:https://doi.org/10.1109/CloudCom.2015.67

Mastura D. Marieska, Paul G. Hariyanto, M. Firda Fauzan, Achmad Imam Kistijantoro, and Afwarman Manaf. 2011. On performance of kernel based and embedded real-time operating system: Benchmarking and analysis. In *International Conference on Advanced Computer Science and Information System (ICACSIS'11)*. IEEE, 401–406.

Miguel Masmano, Ismael Ripoll, Alfons Crespo, and Jorge Real. 2004. TLSF: A new dynamic memory allocator for real-time systems. In *IEEE Proceedings of 16th Euromicro Conference on Real-Time Systems (ECRTS'04)*. 79–88. DOI:https://doi.org/10.1109/EMRTS.2004.1311009

Paul McKenney and Jonathan Walpole. 2007a. What is RCU, fundamentally? *LWN.net* (Dec. 2007). Retrieved from https://lwn.net/Articles/262464/

Paul McKenney and Jonathan Walpole. 2007b. What is RCU? Part 2: Usage. *LWN.net* (Dec. 2007). Retrieved from https://lwn.net/Articles/263130/

Paul E. McKenney. 2015. Motivating lazy RCU callbacks under out-of-memory conditions. US Patent 8,972,801.

Paul E. McKenney, Jonathan Appavoo, Andi Kleen, Orran Krieger, Rusty Russell, Dipankar Sarma, and Maneesh Soni. 2001. Read-copy update. In *AUUG Conference Proceedings*. AUUG, Kensington, Australia, 175–184.

Paul E. McKenney, Silas Boyd-Wickizer, and Jonathan Walpole. 2013. *RCU Usage in the Linux Kernel: One Decade Later*. Technical Report. Retrieved from http://rdrop.com/users/paulmck/techreports/survey.2012.09.17a.pdf.

Paul E. McKenney and Dipankar Sarma. 2005. Towards hard realtime response from the Linux kernel on SMP hardware. In *linux.conf.au (LCA)*. Australian National University, Camberra, Australia.

Larry McVoy, Silicon Graphics, and Carl Staelin. 1996. lmbench: Portable tools for performance analysis. In *Proceedings of the 1996 USENIX Annual Technical Conference*. San Diego, CA, 279–294.

José Ramón Medina, Martin Lawitzky, Alexander Mörtl, Dongheui Lee, and Sandra Hirche. 2011. An experience-driven robotic assistant acquiring human knowledge to improve haptic cooperation. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2416–2422. DOI:https://doi.org/10.1109/IROS.2011.6095026

Frank Mehnert, Michael Hohmuth, and Hermann Hartig. 2002. Cost and benefit of separate address spaces in real-time operating systems. In *23rd IEEE Real-Time Systems Symposium, 2002 (RTSS'02)*. 124–133. DOI:https://doi.org/10.1109/REAL.2002.1181568

Paul Menage, Paul Jackson, and Christoph Lameter. 2008. Cgroups. Retrieved from https://www.kernel.org/doc/Documentation/cgroup-v1/cgroups.txt.

Ronaldo Mercado, Ian Gillingham, J. Rowland, and K. Wilkinson. 2011. Integrating EtherCAT based IO into EPICS at diamond. In *Proceedings of 13th International Conference on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'11)*. 662–665.

Morten Mossige, Pradyumna Sampath, and Rachana G. Rao. 2007. Evaluation of Linux rt-preempt for embedded industrial devices for automation and power technologies-A case study. In *9th RTL Workshop*. Retrieved from http://www.linuxdevices.com/files/article081/Sampath.pdf.

Akhilesh Murikipudi, V. Prakash, and T. Vigneswaran. 2015. Performance analysis of real time operating system with general purpose operating system for mobile robotic system. *Indian Journal of Science and Technology* 8, 19 (2015). DOI: https://doi.org/10.17485/ijst/2015/v8i19/77017

Gábor Márton, Imre Szekeres, and Zoltán Porkoláb. 2017. High-level C++ implementation of the read-copy-update pattern. In *2017 IEEE 14th International Scientific Conference on Informatics*. 243–248. DOI: https://doi.org/10.1109/INFORMATICS.2017.8327254

Fadia Nemer, Hugues Cassé, Pascal Sainrat, Jean-Paul Bahsoun, and Marianne De Michiel. 2006. Papabench: A free real-time benchmark. In *OASIcs-OpenAccess Series in Informatics*, Frank Mueller (Ed.), Vol. 4. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, Dagstuhl, Germany. DOI: https://doi.org/10.4230/OASIcs.WCET.2006.678

Ngoc-Duy Nguyen, Raymond Knopp, Navid Nikaein, and Christian Bonnet. 2013. Implementation and validation of multimedia broadcast multicast service for LTE/LTE-advanced in OpenAirInterface platform. In *38th Annual IEEE Conference on Local Computer Networks - Workshops*. IEEE, 70–76. DOI: https://doi.org/10.1109/LCNW.2013.6758500

Patrick Nielsen. 2012. cpuburn. Retrieved July 8, 2018, from https://patrickmn.com/projects/cpuburn/.

Jan Nowotsch and Michael Paulitsch. 2012. Leveraging multi-core computing architectures in avionics. In *2012 9th European Dependable Computing Conference*. 132–143. DOI: https://doi.org/10.1109/EDCC.2012.27

T. Ogasawara. 1995. An algorithm with constant execution time for dynamic storage allocation. In *Proceedings of the 2nd International Workshop on Real-Time Computing Systems and Applications*. 21–25. DOI: https://doi.org/10.1109/RTCSA.1995.528746

Chandandeep Singh Pabla. 2009. Completely fair scheduler. *Linux Journal* 2009, 184, Article 4 (Aug. 2009), 68–71. Retrieved from http://dl.acm.org/citation.cfm?id=1594371.1594375

Pratyush Patel, Manohar Vanga, and Björn B Brandenburg. 2017. TimerShield: Protecting high-priority tasks from low-priority timer interference. In *2017 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'17)*. 3–12. DOI: https://doi.org/10.1109/RTAS.2017.40

Rodolfo Pellizzoni and Marco Caccamo. 2007. Toward the predictable integration of real-time COTS based systems. In *28th IEEE International Real-Time Systems Symposium (RTSS'07)*. 73–82. DOI: https://doi.org/10.1109/RTSS.2007.15

R. H. Pierce. 2002. *Preliminary Assessment of Linux for Safety Related Systems*. HSE Books, North Lincolnshire, GBR. Aleš Plšek, Frédéric Loiret, Philippe Merle, and Lionel Seinturier. 2008. A component framework for Java-based real-time embedded systems. In *Proceedings of the ACM/IFIP/USENIX 9th International Middleware Conference Leuven, 2008 (Mid-dleware'08)*. Springer, Berlin, Germany, 124–143. DOI: https://doi.org/10.1007/978-3-540-89856-6_7

Aravinda Prasad and K. Gopinath. 2018. A frugal approach to reduce RCU grace period overhead. In *Proceedings of the Thirteenth EuroSys Conference (EuroSys'18)*. ACM, New York, NY, USA, Article 41, 15 pages. DOI: https://doi.org/10.1145/3190508.3190522

Isabelle Puaut. 2002. Real-time performance of dynamic memory allocation algorithms. In *Proceedings 14th Euromicro Conference on Real-Time Systems (Euromicro RTS'02)*. 41–49. DOI: https://doi.org/10.1109/EMRTS.2002.1019184

Pichai Raghavan, Amol Lad, and Sriram Neelakandan. 2005. *Embedded Linux System Design and Development*. CRC Press.

Federico Reghenzani, Giuseppe Massari, and William Fornaciari. 2017. Mixed time-criticality process interferences characterization on a multicore Linux system. In *2017 Euromicro Conference on Digital System Design (DSD'17)*. 427–434. DOI: https://doi.org/10.1109/DSD.2017.18

Federico Reghenzani, Giuseppe Massari, and William Fornaciari. 2018. chronovise: Measurement-based probabilistic timing analysis framework. *Journal of Open Source Software* 3 (2018), 711. DOI: https://doi.org/10.21105/joss.00711

Steven Rostedt. 2009. Finding origins of latencies using ftrace. In *Proceedings of the 11th OSADL Real-Time Linux Workshop.*
TU Dresden Faculty of Computer Science, Dresden, Germany, 117–130.

Steven Rostedt. 2011. Using kernelshark to analyze the real-time scheduler. *LWN.net* (Feb. 2011). Retrieved from https://lwn.net/Articles/425583/.

Steven Rostedt and Darren V. Hart. 2007. Internals of the RT patch. In *Proceedings of the Linux Symposium*, Vol. 2. Ottawa, Canada, 161–172.

Frank Rowand. 2013. *Using and Understanding the Real-Time Cyclictest Benchmark*. The Linux Foundation.

Takashi Sakamoto and Toshiyuki Kondo. 2012. Can passive arm movement affect adaptation to visuomotor rotation? In *2012 IEEE International Conference on Development and Learning and Epigenetic Robotics (ICDL'12)*. IEEE, 1–6. DOI: https://doi.org/10.1109/DevLrn.2012.6400824

Pradyumna Sampath and B. Rachana Rao. 2011. Efficient embedded software development using QEMU. In *Proceedings of the 11th OSADL Real-Time Linux Workshop*. Faculty of Electrical Engineering, Czech Technical University, Prague, Czechia, 155–161.

Dipankar Sarma and Paul E. McKenney. 2004. Making RCU safe for deep sub-millisecond response realtime applications. In *Proceedings of the 2004 USENIX Annual Technical Conference*. University of Wisconsin, Boston, 182–191.

Henning Schild, Adam Lackorzynski, and Alexander Warg. 2009. Faithful virtualization on a real-time operating system. In *Proceedings of the 11th OSADL Real-Time Linux Workshop*, 237–244.

Matthias Schoepfer, Florian Schmidt, Michael Pardowitz, and Helge Ritter. 2010. Open source real-time control software for the Kuka light weight robot. In *2010 8th World Congress on Intelligent Control and Automation*. IEEE, 444–449. DOI: https://doi.org/10.1109/WCICA.2010.5553773

Tobias Schoofs, Sergio Santos, Cassia Tatibana, and José Anjos. 2009. An integrated modular avionics development environment. In *2009 IEEE/AIAA 28th Digital Avionics Systems Conference*. IEEE, 1.A.2–1–1.A.2–9. DOI: https://doi.org/10.1109/DASC.2009.5347582

Claudio Scordino and Giuseppe Lipari. 2006. Linux and real-time: Current approaches and future opportunities. In *ANIPLA International Congress*.

Euiseong Seo, Jinkyu Jeong, Seonyeong Park, Jinsoo Kim, and Joonwoon Lee. 2009. Catching two rabbits: Adaptive real-time support for embedded Linux. *Software: Practice and Experience* 39, 5 (2009), 531–550. DOI: https://doi.org/10.1002/spe.911 Lui Sha, Ragunathan Rajkumar, and John P. Lehoczky. 1990. Priority inheritance protocols: An approach to real-time synchronization. In *Proceedings of 13th Real-Time Linux Workshop on IEEE Transactions on Computers* 39, 9 (Sep. 1990), 1175–1185. DOI: https://doi.org/10.1109/12.57058

Jianping Shen, Michael Hamal, and Sven Ganzenmüller. 2011. Dynamic memory allocation on real-time Linux. 187–193.

Katherine K. Sheridan-Barbian. 2015. *A Survey of Real-Time Operating Systems and Virtualization Solutions for Space Systems*. Ph.D. Dissertation. Monterey, CA: Naval Postgraduate School. Advisor(s) Nguyen, Thuy; Gondree, Mark. DOI: https://doi.org/10945/45256

Suresh Siddha, Venkatesh Pallipadi, and A. V. D. Ven. 2007. Getting maximum mileage out of tickless. In *Proceedings of the Linux Symposium*, Vol. 2. Ottawa, Canada, 201–207.

Lukas Sigrist, Georgia Giannopoulou, Pengcheng Huang, Andres Gomez, and Lothar Thiele. 2015. Mixed-criticality runtime mechanisms and evaluation on multicores. In *21st IEEE Real-Time and Embedded Technology and Applications Symposium*. 194–206. DOI: https://doi.org/10.1109/RTAS.2015.7108442

Dimitri Sivanich. 2007. Low latency real-time computing on multiprocessor systems running standard Linux. In *Proceedings of 11th High Performance Embedded Computing (HPEC'07) Workshop*. MIT Lincoln Laboratory, Lexington, KY.

Ben Smith, Rick Grehan, Tom Yager, and D. C. Niemi. 2011. Byte-unixbench: A Unix benchmark suite. Retrieved from https://github.com/kdlucas/byte-unixbench.

Jesper Smith, Douglas Stephen, Alex Lesman, and Jerry Pratt. 2014. Real-time control of humanoid robots using OpenJDK. In *Proceedings of the 12th International Workshop on Java Technologies for Real-time and Embedded Systems (JTRES'14)*. ACM, New York, Article 29, 8 pages. DOI: https://doi.org/10.1145/2661020.2661027

Wonjun Song, Hyung-Joon Jung, Jung Ho Ahn, Jae W. Lee, and John Kim. 2016. Evaluation of performance unfairness in NUMA system architecture. *IEEE Computer Architecture Letters* 16, 1 (2016), 26–29. DOI: https://doi.org/10.1109/LCA.2016.2602876

Paulo Baltarejo Sousa, Nuno Pereira, and Eduardo Tovar. 2012. Enhancing the real-time capabilities of the Linux kernel. *SIGBED Review* 9, 4 (Nov. 2012), 45–48. DOI: https://doi.org/10.1145/2452537.2452546

Patrick H. Stakem. 2001. FlightLinux: A new option for spacecraft embedded computers. In *2001 Earth Science Technology Conference*. University of Maryland Conference Center, Collage Park, MD.

John A. Stankovic and Raj Rajkumar. 2004. Real-time operating systems. *Real-Time Systems* 28, 2 (2004), 237–253.

John A. Stankovic and Krithi Ramamritham. 1990. What is predictability for real-time systems? *Real-Time Systems* 2, 4 (Nov. 1990), 247–254. DOI: https://doi.org/10.1007/BF01995673

Alexander D. Stoyenko. 2012. *Constructing Predictable Real Time Systems*. Springer International Series in Engineering and Computer Science, Vol. 146. Springer US. DOI: https://doi.org/10.1007/978-1-4615-4032-8

Yaw-Ren Tan, David Peake, and Daniel Tavares. 2017. Fast orbit feedback with Linux PREEMPT_RT. In *5th International Beam Instrumentation Conference (IBIC'16)*. JACOW, Geneva, Switzerland, 631–634.

Robin Verschueren, Stijn De Bruyne, Mario Zanon, Janick V. Frasch, and Moritz Diehl. 2014. Towards time-optimal race car driving using nonlinear MPC in real-time. In *53rd IEEE Conference on Decision and Control*. IEEE, 2505–2510. DOI: https://doi.org/10.1109/CDC.2014.7039771

Nicholas C. H. Vun, H. F. Hor, and J. W. Chao. 2008. Real-time enhancements for embedded Linux. In *2008 14th IEEE International Conference on Parallel and Distributed Systems*. 737–740. DOI: https://doi.org/10.1109/ICPADS.2008.108

Amos Waterland. 2013. Stress POSIX workload generator. Retrieved January 8, 2018, from http://people.seas.harvard.edu/apw/stress.

Noah Watkins, Jared Straub, and Douglas Niehaus. 2009. A flexible scheduling framework supporting multiple programming models with arbitrary semantics in Linux. In *Proceedings of the 11th OSADL Real-Time Linux Workshop*. Dresden, Germany.

Hannes Weisbach, Björn Döbel, and Adam Lackorzynski. 2011. Generic user-level PCI drivers. In *Proceedings of the 13th OSADL Real-Time Linux Workshop*. Faculty of Electrical Engineering, Czech Technical University, Prague, Czechia.

Elmar Wings, Marcel Müller, and Marc Rochler. 2015. Integration of real-time Ethernet in LinuxCNC. *International Journal of Advanced Manufacturing Technology* 78, 9–12 (Jun. 2015), 1837–1846. DOI: https://doi.org/10.1007/s00170-015-6786-y

Karim Yaghmour. 2001. *Adaptive Domain Environment for Operating Systems*. Technical Report. Opersys, Sherbrooke, Canada.

Victor J. Yodaiken. 1999a. Adding real-time support to general purpose operating systems. Holder Wind River Systems, US Patent 5,995,745.

Victor J. Yodaiken. 1999b. *The RTLinux Manifesto*. Technical Report. Department of Computer Science, New Mexico Institute of Technology, Socorro, NM.

Peijie Yu, Mingyuan Xia, Qian Lin, Min Zhu, Shang Gao, Zhengwei Qi, Kai Chen, and Haibing Guan. 2010. Real-time enhancement for Xen hypervisor. In *IEEE/IFIP International Conference on Embedded and Ubiquitous Computing*. IEEE, 23–30. DOI: https://doi.org/10.1109/EUC.2010.14

Sangwon Yun, Woongryol Lee, Taegu Lee, Mikyung Park, Sangil Lee, André C. Neto, Anders Wallander, and Young-Kuk Kim. 2013. Evaluating performance of MARTe as a real-time framework for feed-back control system at Tokamak device. *Fusion Engineering and Design* 88, 6 (2013), 1323–1326. DOI: https://doi.org/10.1016/j.fusengdes.2013.03.028

Richard Zappulla, Josep Virgili Llop, Hyeongjun Park, Costantinos Zagaris, and Marcello Romano. 2016. Floating spacecraft simulator test bed for the experimental testing of autonomous guidance, navigation, & control of spacecraft proximity maneuvers and operations. In *AIAA/AAS Astrodynamics Specialist Conference*. 5268. DOI: https://doi.org/10.2514/6.2016-5268

Fangfang Zhu, Yucong Chen, Jianqiang Wang, Gaofeng Zhang, and Qingguo Zhou. 2016. Experimental validation and exploration of a new kind of synchronization in Linux. In *International Symposium on System and Software Reliability (ISSSR'16)*. IEEE, 91–96. DOI: https://doi.org/10.1109/ISSSR.2016.023

Baojing Zuo, Kai Chen, Alei Liang, Haibing Guan, Jun Zhang, Ruhui Ma, and Hongbo Yang. 2010. Performance tuning towards a KVM-based low latency virtualization system. In *IEEE 2nd International Conference on Information Engineering and Computer Science (ICIECS'10)*. IEEE, 1–4. DOI: https://doi.org/10.1109/ICIECS.2010.5678357