Contents lists available at ScienceDirect

Journal of Systems Architecture

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

EDF scheduling for distributed systems built upon the IEEE 802.1AS clock - A theoretical-practical comparison

Héctor Pérez [*], J. Javier Gutiérrez

Software Engineering and Real-Time Group, Universidad de Cantabria, Santander, Spain

ARTICLE INFO

Keywords:
Distributed systems
EDF
Clock synchronization
Real-time
Schedulability analysis
Scheduling deadline assignment

ABSTRACT

Existing response time analysis and optimization techniques for real-time distributed systems show that EDF schedulers feature better scheduling capabilities when a global clock reference can be used; this form of scheduling is known as global-clock EDF. In this context, precise clock synchronization is an enabling technology for future distributed real-time systems that want to leverage EDF scheduling. The IEEE 802.1AS protocol can be considered a stable technology for this purpose, as it is part of the Time-Sensitive Networking (TSN) family of standards to provide real-time communication over Ethernet. This paper presents a system architecture for applying global-clock EDF scheduling in distributed systems with soft real-time requirements. It also presents experiments to (1) assess the synchronization capabilities of the clock synchronization mechanism in the proposed architecture, (2) evaluate the performance of different scheduling deadline assignment techniques, and (3) contrast the theoretical results obtained by the schedulability analysis against those obtained through the execution of these experiments.

1. Introduction

Earliest Deadline First (EDF) is a preemptible priority-driven scheduling algorithm in which priorities are dynamically assigned to tasks according to their deadlines (i.e., the assigned priority is inversely proportional to the absolute deadline computed at release time). Compared to fixed-priority (FP) scheduling, EDF scheduling can achieve better usage of the CPU in uniprocessor systems [1], and the frequently attributed advantages of FP over EDF are not always present [2]. Furthermore, EDF scheduling is now supported in an increasing number of operating systems (e.g., Linux [3], Zephyr [4], or ERIKA [5]), but it is also used in communication networks (e.g., CAN Bus [6] or switched networks [7]), and real-time programming languages such as Ada [8].

Regarding scheduling real-time distributed systems with EDF, the work in [9] shows how the availability of a global clock (i.e., a clock that is constantly synchronized across all nodes of the distributed system from one and the same timing source) may increase schedulability. According to [9], this improvement is attained when tasks are composed of a sequence of sub-tasks with precedence relations (i.e., activities which are independently scheduled, and may be executed in different CPUs). In this paper, we use the notion of *scheduling deadline* to denote the deadline as scheduling parameter as opposed to the *deadline* as timing requirement [9,10]. In this context, we can distinguish two kinds of scheduling policies: (1) in *local-clock* EDF (LC-EDF), the scheduling deadlines of each sub-task refer to their release times in their own processor, determined from a local clock source; and (2) in *global-clock* EDF (GC-EDF), the scheduling deadlines of each sub-task refer to the release time of the task (first sub-task), determined from a global clock, as the sub-tasks in the concatenation may be located in different processors.

The GC-EDF scheduling can be applied as long as each node belonging to the distributed system draws its time source from a global clock, common to all other nodes of the system. To this end, the IEEE 802.1AS specification [11] proposes a protocol for the transport of timing over bridged local area networks. This specification is part of the Time-Sensitive Networking (TSN) set of standards [12], an ongoing effort to enable deterministic data transfer in Ethernet networks.

Implementations for the clock synchronization mechanism proposed by IEEE 802.1AS are available for different operating systems, including Linux. The Linux kernel is getting more and more adopters in the industry. Results from a recent survey to 120 industry practitioners in the field of real-time embedded systems [13] show that Linux is deployed in 55.88% of respondents' applications, from which 42,2% also use a real-time operating system. For instance, running applications with

* Corresponding author.
E-mail address: perezh@unican.es (H. Pérez).

https://doi.org/10.1016/j.sysarc.2022.102742
Received 21 January 2022; Received in revised form 16 September 2022; Accepted 20 September 2022
Available online 24 September 2022

real-time requirements over Linux has been explored for space or HPC applications [14].

This paper addresses the question of whether EDF scheduling performs better when GC-EDF is used as opposed to LC-EDF, as suggested by theoretical results [10]. In particular, we explore and evaluate the behavior of LC-EDF and GC-EDF in distributed real-time systems using theoretical and experimental results supported by the proposal of a distributed architecture based on Linux and the IEEE 802.1AS standard. This work originates from an earlier and initial experimentation with the proposed architecture [15], and it also completes a previous analysis for LC-EDF that was presented in [16]. Specifically, this paper makes the following contributions:

- Identification of a viable system architecture to apply GC-EDF scheduling. This architecture would enable clock synchronization through the facilities implemented by the Linux kernel, thus making this architecture suitable for soft real-time applications.
- Assess the viability of the approach by (1) developing a distributed real-time platform with support for EDF scheduling and a global clock, and (2) assessing the synchronization capabilities of the platform.
- Evaluate different scheduling deadline assignment techniques that can be applied to distributed real-time systems.
- Verify whether the experimental results follow the same behavior as the theoretical results obtained by schedulability analysis techniques.

This paper is organized as follows. Section 2 reviews the related work. Section 3 presents the proposed system architecture, while Section 4 describes the distributed real-time platform that implements the proposed architecture. Section 5 outlines the real-time model and the schedulability analysis and deadline assignment techniques for EDF scheduling on which this paper is based. Section 6 presents the evaluation of the proposed approach, including the assessment of the synchronization capabilities of using a global clock, and the comparison of the theoretical and experimental results obtained for EDF scheduling in a distributed real-time application. Finally, Section 7 summarizes the main contributions and outlines directions of future work.

2. Related work

Since the initial work by Liu and Layland [1], the EDF scheduling policy has been very widely studied for uniprocessor systems from different perspectives of theory and practice, even in comparison with other industrially accepted scheduling policies such as FP [2,17]. Moreover, the interest that these studies still have is clear as shown in a recent work [18] presenting a toolbox that enables quantitative per-application comparison of EDF and FP uniprocessor scheduling. This toolbox is based on [19] to allow EDF to be used in place of default FP in Ada Ravenscar systems without any change in the application code.

There are also many works concerning EDF and multiprocessor systems [20], where EDF schedulers are well defined and can be classified into global scheduling (tasks may migrate from one processor to another, e.g. [21]) or partitioned scheduling (fixed allocation of tasks to processors, e.g. [22,23]). An evaluation of these two kinds of schedulers in a real-time operating system is presented in [24]. In the context of data-intensive applications, the paper in [25] studies the use of global EDF scheduling with gang scheduled tasks on a homogeneous multiprocessor system.

In the case of real-time distributed systems, the body of knowledge about the behavior of EDF scheduling is much smaller. For instance, the book by Liu [26] shows deadline-assignment algorithms where scheduling deadlines are obtained by distributing the end-to-end deadline of a task to each of its sub-tasks. Later works [10,9] present algorithms for optimizing the assignment of deadlines to tasks and messages in distributed hard real-time systems which outperform the processor utilization of existing ones in the case of LC-EDF schedulers. Some of these algorithms are supported by response time analysis techniques for distributed systems. Most of these analysis techniques are derived from the holistic analysis for GC-EDF proposed by Spuri [27]. The holistic analysis technique for LC-EDF was proposed in [10] as a modification of Spuri's technique. In essence, the holistic analysis assumes that each sub-task is independent of the others, even if they all belong to the same task. After analyzing each sub-task in isolation, the dependencies among them are captured into the jitter term of each sub-task. Response time analysis seeks to determine the longest busy period, found from a critical instant. Notably, the analysis techniques for LC-EDF and GC-EDF also work for systems with arbitrary deadlines, i.e. task's deadlines can be larger than task's periods.

The work presented in [28] discusses an offset-based technique for GC-EDF, which reduces in part the pessimism of the holistic approach. More recently in [29], two equivalent offset-based techniques were proposed for LC-EDF. While the latter techniques yield less pessimistic results, they are extremely complex and a reliable implementation of them is not available yet. So, for the purpose of the work presented here, the holistic technique is used as it can be applied to both LC-EDF and GC-EDF, which also allows comparable results to be obtained.

Several authors have assessed the timing capabilities of the clock synchronization protocol provided by the IEEE 802.1AS standard. For instance, the results obtained in [30] show a clock offset between 3 and 8 μs in a simple distributed system; and the authors in [31] evaluated the precision of clock synchronization in large-scale realistic networks by means of simulations, which achieved a precision of 2 μs with the assumptions made in their system model.

3. System architecture

The Linux kernel provides developers with notable features such as the high availability of popular development tools and packages, extensive platform support, and ease of customization. This has led users to regard Linux as a relevant choice of operating system in the industry [13]. Whereas the Linux kernel architecture was designed for general-purpose computing, its real-time performance has been continuously improved and the current real-time scheduling performance can be sufficient for soft real-time systems as long as the kernel is appropriately configured [14]. However, additional features are needed for it to become able to support global-clock EDF scheduling:

- ***Support for EDF scheduling service***

  Modern Linux kernels provide an implementation of EDF scheduling combined with the CBS (Constant Bandwidth Server) algorithm [32]. Under this scheduling policy, the current scheduling deadline of a sub-task is initialized to the current time plus a configured period set by CBS, where each sub-task is associated with a budget and a period. Thus, this policy does not fit well with the concept of GC-EDF [9], where scheduling deadlines of each sub-task are refer to the release time of the task.

  To mitigate this issue, our system architecture relies on a real-time operating system to provide scheduling services. MaRTE OS [33] is a real-time kernel mainly written in Ada [8], which follows the Minimal Real-Time POSIX.13 subset. This kernel implements most of the services defined by the Real-Time Systems Annex of Ada [8], including the *EDF_Across_Priorities* dispatching policy. The implementation of the EDF policy in MaRTE OS allows developing applications scheduled with GC-EDF scheduling as long as a global clock is available. Furthermore, MaRTE OS allows the execution of applications not only over bare machine but also over the Linux kernel with some limitations. The latter option is enabled by means of the *linux_lib* architecture [34] which allows MaRTE OS applications to be executed as user processes in Linux.

• ***Support for a clock synchronization mechanism***

Several solutions exist for clock synchronization in a distributed system, which assure different degrees of time-synchronization. Unlike most of them, the Precision Time Protocol (PTP) relies on standard hardware and infrastructure to synchronize clocks in the nodes of a distributed system. The IEEE 802.1AS standard can be seen as a subset or profile of PTP to provide better real-time guarantees. Thus, the proposed architecture provides the notion of global time by means of the implementation of the IEEE 802.1AS protocol [11], which is responsible for synchronizing all the PTP Hardware Clocks (PHC) for each node of the distributed system. Among other functionalities, the protocol deals with (1) the transport of synchronized time, (2) the selection of a timing source, and (3) the indication of timing errors. This standard ensures that the jitter and time synchronization requirements are met for time-sensitive applications.

The synchronization protocol specified by IEEE 802.1AS generates a master-slave hierarchy among the clocks of the network. One clock is used as a reference time source (Grandmaster) and the other clocks are used as slaves. Fig. 1 shows a possible configuration for the synchronization of PHC and system clocks in IEEE 802.1AS systems. In this example, the timing source is located in the switch but other configurations are allowed by the standard.

Fig. 2 shows a multicore system following the proposed software architecture for integrating a global clock in distributed systems with real-time requirements. The Linux kernel provides different services to applications, such as timing or networking. Real-time applications are executed on top of MaRTE OS, which is configured with the *linux_lib* architecture [34]. Under this architecture, a MaRTE OS application can be executed as a standard Linux process in which concurrency and scheduling facilities are provided by MaRTE OS. It is worth noting that these applications are standard processes and therefore may be interrupted by Linux kernel activities.

However, the design of the Linux kernel favors throughput over predictability and therefore it should not be used with its default configuration for real-time systems. Some of the most notable approaches to bound response times of real-time applications in Linux include the PREEMPT_RT kernel patch and the core isolation capabilities [14]. While the former aims at reducing latency by making most of the kernel preemptable, the isolation capabilities aim at reserving a processing core for one or more tasks or sub-tasks (i.e., they prevent the Linux scheduler to schedule other workload on the isolated cores). To enhance the predictability of a MaRTE OS application executed on top of

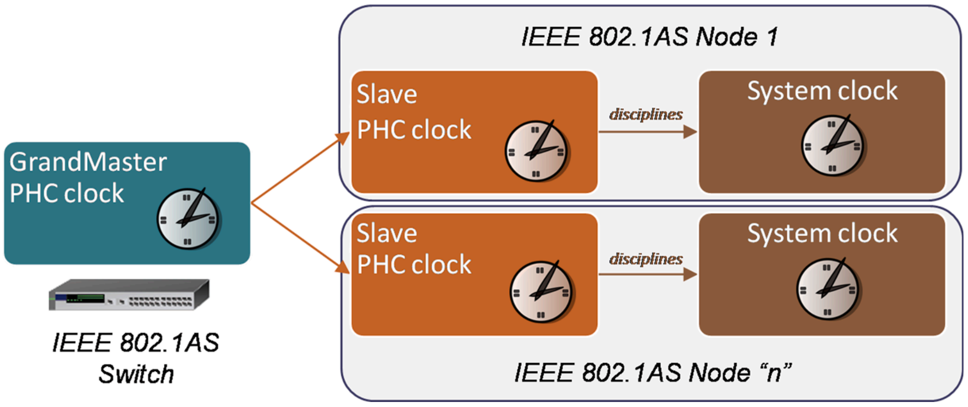


Fig. 1. Example of clock synchronization in IEEE 802.1AS system.

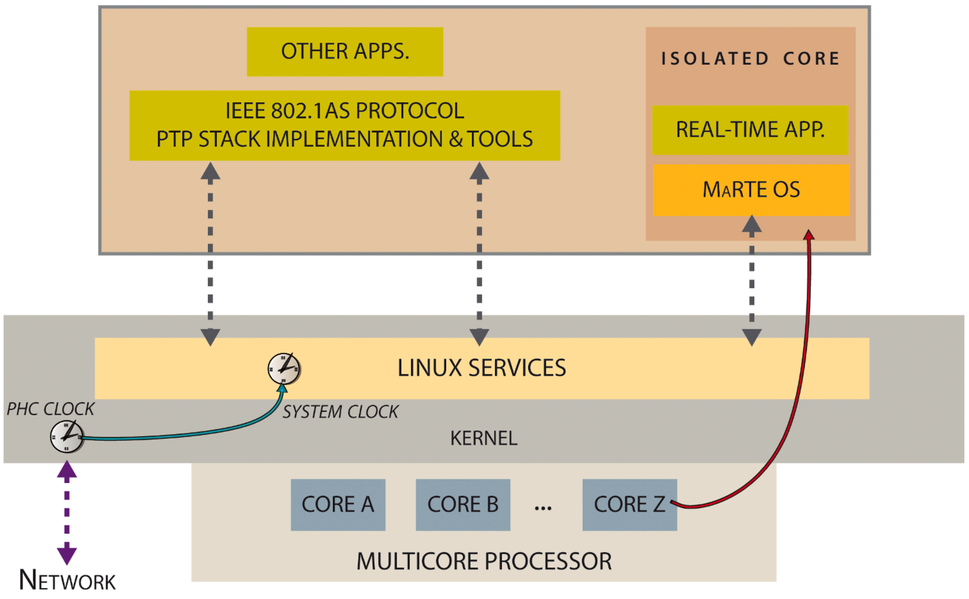


Fig. 2. System architecture.

Linux, our proposal applies the isolation capabilities provided by the Linux kernel. Isolating a core can enhance the predictability of applications by limiting user and kernel preemptions. Whereas the Linux kernel provides several features to facilitate the execution of threads in isolated cores, these features do not provide full isolation and they do not consider some sources of interferences such as shared caches, global work queues or interprocessor interrupts. Furthermore, there are also a few kernel threads bound to each core whose workload cannot be migrated. It is worth noting that the use of MaRTE OS enables the execution of a multithreading application in the isolated core.

Finally, a system clock (e.g., a POSIX clock) could also be synchronized to the PHC clock in order to facilitate access to the global time from user-space applications as shown in Fig. 2.

4. The distributed real-time platform

This section describes the implementation of the architecture introduced in Section 3 in order to validate the integration of the required software and provide experimental results in subsequent sections. The configuration of the Linux kernel requires tuning both compilation and runtime parameters, which are briefly described next. For instance, core isolation at runtime is provided by means of *isolcpus* and *cpuset* facilities [3]. These facilities allow executing the MaRTE OS process as isolated workload on a given core, exempt from competition from other workload.

To enhance isolation, the CONFIG_NO_HZ_FULL kernel parameter [3] has been enabled. This parameter allows a core to run in *adaptative* tick mode (i.e., the kernel tick is offloaded to another core when the isolated core is idle or running a single thread). However, this parameter cannot remove a residual 1 Hz tick which still remains as a source of interference. Additionally, the isolated core is switched off during boot time to force the kernel to assign the regular workload to the non-isolated cores.

The PREEMPT_RT patch makes several changes to the kernel to favor predictability over throughput. One of them is converting interrupt handlers into preemptible kernel threads, which somehow compromises the benefit of the *adaptative* tick mode (i.e., several runnable threads can be available at the same time and therefore the kernel tick cannot be offloaded to another core). Furthermore, the design of the *linux_lib* architecture of MaRTE OS relies on POSIX signals for the communication with the Linux kernel, which increases the complexity of the system configuration when POSIX signals are handled by kernel threads.

The proposed architecture relies instead on the low-latency profile for the kernel, which integrates some of the changes proposed by the PREEMPT_RT patch without compromising core isolation. As a side note, the use of MaRTE OS as a threading library on top of Linux allows the execution of a multithreaded application in *adaptative* tick mode since there is only one runnable thread from the Linux kernel perspective.

LinuxPTP[1] is a suite of tools for Linux that enable configuration and access to a global clock in a distributed system. For our purposes, two tools from the toolset are particularly relevant: (1) *ptp4l* to synchronize the PHC clocks through the network, and (2) *phc2sys* to synchronize two local clocks in each node (e.g., a POSIX clock and a PHC clock). Besides following the IEEE 802.1AS specification, both tools are configured to provide monotonically increasing times (i.e., clock corrections are performed by changing clock frequency), as it is more suitable for real-time applications.

At the MaRTE OS level, the *linux_lib* architecture has been modified to provide access to the global clock from the application level. To enable this feature, the timing functions located in the *Hardware Abstraction Layer* have been modified to use the clock with synchronized time provided by the PTP stack. In the original implementation of the kernel, the clock call was based on the Linux system call *gettimeofday* [3]. Since the PHC clock can be mapped to the system clock through *phc2sys*, the new call relies on the *clock_gettime* system call to query the time from the global clock. In a similar way, the Ada runtime system used by MaRTE OS has been also modified to enable the access to the global clock from Ada applications.

5. System model, schedulability analysis and scheduling deadline assignment

This section briefly describes the system model, the schedulability analysis techniques and some of the scheduling deadline assignment algorithms used in this work. MAST (Modeling and Analysis Suite for Real-Time Applications) [35] is a tool suite used for modeling and analysis of real-time and embedded systems. It integrates the tools needed in this work as well as a model [36] aligned with MARTE (Modeling and Analysis of Real-Time Embedded systems) [37], a standard defined by the OMG (Object Management Group). MAST, therefore, has facilitated the process of obtaining the theoretical results presented in this paper.

- ***Real-time model***

In MAST, a real-time system is composed of distributed *end-to-end flows* with periodic or sporadic activations. Each end-to-end flow (e2e flow hereafter) Γ_i is released by a sequence of external events with a period (or a minimum interarrival time) T_i, and contains a set of *steps* that model tasks (CPU-bound scheduling units) and messages (network-bound scheduling units). Each periodic release of an e2e flow causes the execution of the set of steps, each step being released when the preceding one in its e2e flow completes. We assume that steps are statically assigned to processors and networks (migration is not allowed), and that the relative phasing of different e2e flows is arbitrary.
Fig. 3 shows an example of one e2e flow, denoted Γ_i, with three steps. The arrival of the external event that releases the e2e flow is represented by a thick horizontal arrow labelled e_i, which has period T_i. The thin horizontal arrows represent the release of the following steps in the e2e flow; a step cannot be executed before the preceding step has completed. We assume that events represented in Fig. 3 are instantaneous and any activity in the system is modelled as a step. The j-th step of e2e flow Γ_i is identified as τ_{ij}; it is characterized by its worst case execution time C_{ij} and its best-case execution time C^b_{ij}. The timing requirements that we consider are end-to-end deadlines (e2e deadlines hereafter), D_i, relative to that particular release period of the e2e flow, and must be met by the final step in the e2e flow. We

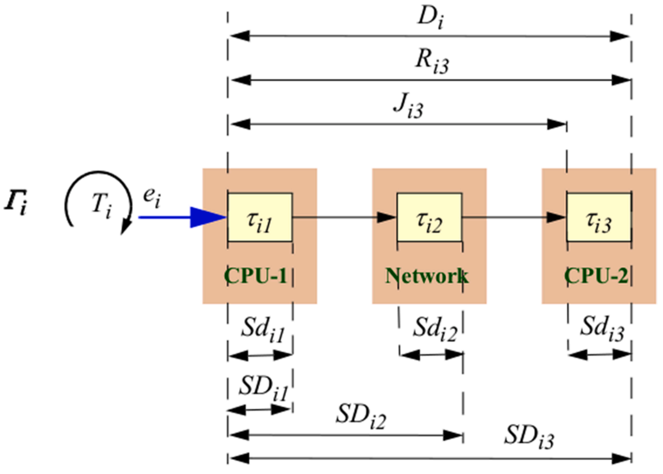


Fig. 3. The model of the e2e flow Γi.

[1] Available at http://linuxptp.sourceforge.net/

allow e2e deadlines to be larger than the corresponding e2e flow periods. As a result of schedulability analysis, each step τ_{ij} also has a worst-case response time (or an upper bound on it) R_{ij}, and a best-case response time (or a lower bound on it) R^b_{ij}. The worst-case response time estimation of the last step is then compared with the e2e deadline in order to determine the schedulability of the system. We allow the external event that triggers an e2e flow to have a maximum release jitter J_{i1} in relation to the corresponding activation of the e2e flow. Other steps τ_{ij} may also have an initial release jitter J_{ij}. Despite this jitter, e2e deadlines and response times always refer to the theoretical start of their respective instance's period, not to the actual release of the e2e flow. We assume that J_{ij} may be larger than the period of its e2e flow, T_i. For each step τ_{ij} scheduled by the EDF policy, two kinds of scheduling deadlines are defined: (1) a local scheduling deadline Sd_{ij}, relative to the release time of its associated step in its own processing resource, thus allowing the use of the LC-EDF policy; and (2) a global scheduling deadline SD_{ij} for the GC-EDF policy, relative to the release period of the corresponding e2e flow.

- ***Schedulability analysis techniques for EDF***

The real-time model described above allows the response time analysis techniques developed in [27] for GC-EDF and in [10] for LC-EDF to be applied in order to obtain an estimation of the worst-case response times. It is known [38] that using the best-case response time of an e2e flow in the holistic analysis makes jitter calculation more precise, which in turn reduces the overall pessimism of the analysis. We compute a lower bound on the best-case response time as the sum of the best-case execution times of the current step and all its predecessors in the e2e flow.

- ***Scheduling deadline assignment techniques***

In this work, we use four scheduling deadline assignment methods and evaluate them according to their interpretation depending on whether LC-EDF or GC-EDF is used, as described in [9]:

- *Ultimate Deadline* (UD) [26], which obtains the scheduling deadlines by assigning the e2e deadline of an e2e flow (D_i) to each one of its steps.
- *Effective Deadline* (ED) [26], which obtains the scheduling deadlines by considering that if a step finishes its execution within its assigned deadline, the following steps in the same e2e flow will have to complete within their worst-case execution time.
- *Proportional Deadline* (PD) [26], which calculates scheduling deadlines by distributing the e2e deadline (D_i) proportionally to the worst-case execution time of each step (C_{ij}.).
- *Proportional Deadline with Global Scheduling Deadline* (PD-GSD) [9], which works as PD does, but converts the scheduling deadlines obtained for LC-EDF into GC-EDF ones. Thus, the scheduling deadline of step τ_{ij} equals the sum of all deadlines (assigned by PD) of the preceding steps in the e2e flow, including itself.

Fig. 4 shows an illustrative example for the application of the scheduling deadline assignment methods in a single e2e flow with a period and an e2e deadline of 12 and 25 units of time, respectively. This e2e flow is composed of 4 steps whose worst-case execution times are 2, 5, 1 and 2 units of time, as shown in Fig. 4. Further details on the description of these algorithms as well as on their scheduling capabilities can be found in [9].

6. Evaluation

After introducing the real-time model, the schedulability analysis algorithm, and the deadline assignment techniques, an evaluation of the proposal is performed with a twofold objective:

- Assessing the synchronization capabilities of using a global clock in the proposed architecture by obtaining performance metrics. In particular, three experiments have been carried out in the context of the proposed architecture to estimate (1) the overhead associated with the reading of the global clock, (2) the latency associated with the handling of an external event and (3) the synchronization capabilities of the global clock.
- Verifying whether the experimental results for EDF scheduling in distributed systems slant in the same direction as the theoretical results obtained by the schedulability analysis techniques. To this end, a distributed real-time application with synthetic workloads is evaluated when different scheduling deadline assignment techniques are applied.

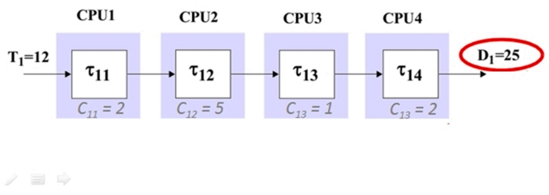
The hardware platform consists of two quad-core 1.9 GHz nodes connected through an isolated 1 Gbps Ethernet network. Both nodes and the switch are compliant with the IEEE 802.1AS standard and run the PTP stack to implement the synchronization of clocks according to the configuration illustrated in Fig. 1.

The implementation of the architecture depicted in Fig. 2 has used the following software components: (1) Linux kernel v4.4.256-rt214 compiled using the parameters described in Section 3; (2) MaRTE OS v2017 using the *linux_lib* architecture, together with the required changes for accessing to the global clock described in Section 4; (3) *ptp4l* v3.0 to implement the PTP stack; and (4) *phc2sys* v3.0 to synchronize system and PHC clocks.

The evaluation has been performed with regular kernel workload and without any additional user workload except the one determined by the experiments themselves.

6.1. Assessment of the synchronization capabilities

The proposed architecture is based on IEEE 802.1AS and MaRTE OS for providing a global clock and EDF scheduling, respectively. Thus, it is worth estimating the synchronization capabilities of the global clock from the perspective of a real-time application executed on top of MaRTE OS. To better compare the measurements obtained, two different scenarios are defined: (1) the *MaRTE OS over Linux* scenario, which represents the proposed approach; and (2) the *Reference* scenario in which the test applications use the same features as the previous



Scheduling deadlines

	τ_{11}	τ_{12}	τ_{13}	τ_{14}
UD	25	25	25	25
ED	17	22	23	25
PD	5	12.5	2.5	5
PD-GSD	5	17.5	20	25

Fig. 4. Example of applying different scheduling deadline assignment techniques.

scenario (e.g., core isolation) but they execute directly on top of the Linux kernel.

6.1.1. *Overhead metrics*

The first experiment aims to estimate the overhead associated with the reading of the global clock. This operation is executed 10,000,000 times, and the average, maximum, and minimum times are estimated, together with the standard deviation.

Table 1 shows the results obtained for both scenarios under evaluation, which report a high maximum time compared to the average and minimum values. This difference may be caused by interferences of the Linux kernel whose design is not specifically tailored to real-time systems. As noted in Sections 3 and 4, the Linux kernel does not provide full isolation and it can suffer from some sources of interferences such as shared caches, global work queues, interprocessor interrupts or the residual 1 Hz tick. According to specialized literature [39], this can lead to worst-case latencies in the range of tens of μs.

It is also worth noting that *MaRTE OS over Linux* scenario obtains higher average and minimum values. In Linux, the *clock_gettime* function is usually supported as vDSO (virtual Dynamic Shared Object) to improve its performance [3]. However, the *clock_gettime* call in the *linux_lib* architecture of MaRTE OS is internally redirected to the Linux kernel through an explicit system call, which incurs additional overhead in relation to the native case. Finally, the results show that the standard deviation is below 0.1 μs in both scenarios.

6.1.2. *Event-handling latency metrics*

The second experiment aims to estimate the latency associated with the handling of an external event. In particular, this test measures the delay in the response to a periodic timer event (i.e., it executes *clock_nanosleep* followed by *clock_gettime* using the global clock). In this case, the requested time through *clock_nanosleep* should be nearly identical to the time measured by *clock_gettime*. This experiment is executed 1,000,000 times with a period of 100 μs, and the average, maximum, and minimum times are estimated, together with the standard deviation.

Table 2 shows the results obtained for this experiment. Again, the measurements obtained for *MaRTE OS over Linux* are higher for average and minimum values. This is due to the design of MaRTE OS, which relies on POSIX signals to communicate with the Linux kernel. However, the results show that this overhead is lower than the maximum interferences which are caused by the Linux kernel. Finally, the standard deviation remains below 1 μs in both scenarios, although it is slightly lower for *MaRTE OS over Linux*. Thus we can conclude that the proposed approach reduces execution-time dispersion (variability) at the cost of acceptable runtime overhead.

6.1.3. *Timing synchronization metrics*

The last experiment aims to evaluate the synchronization of the global clock in a distributed system. To this end, two nodes periodically and synchronously to each other raise a digital signal using the global clock as the timing source. Then, an oscilloscope measures the delay between the two digital signals. This operation is executed 50,000 times, and the average and maximum times together with the standard deviation are estimated. Figs. 5 and 6 show the results taken by oscilloscope for both scenarios using a period of 100 μs. In this experiment, the maximum time is obtained as the maximum absolute value of the Max and Min times captured by the oscilloscope. It is worth noting that negative values indicate that the second signal is raised before the first one.

Table 1
Overhead in accessing the global clock (time in μs).

Scenario	Max	Avg	Min	Std Dev
Reference	27.30	0.07	0.07	0.01
MaRTE OS over Linux	38.80	3.40	3.40	0.07

Table 2
Event-handling latency for a timer event (time in μs).

Scenario	Max	Avg	Min	Std Dev
Reference	30.20	6.50	5.10	0.60
MaRTE OS over Linux	23.70	18.60	16.20	0.32

Both scenarios obtain similar maximum absolute delays below 10 μs. Furthermore, the standard deviation obtained by each scenario is in the same range as the second experiment. These results suggest that the approach may be suitable for real-time applications whose deadlines are in the range of hundreds of microseconds.

6.2. *Theoretical and practical comparison*

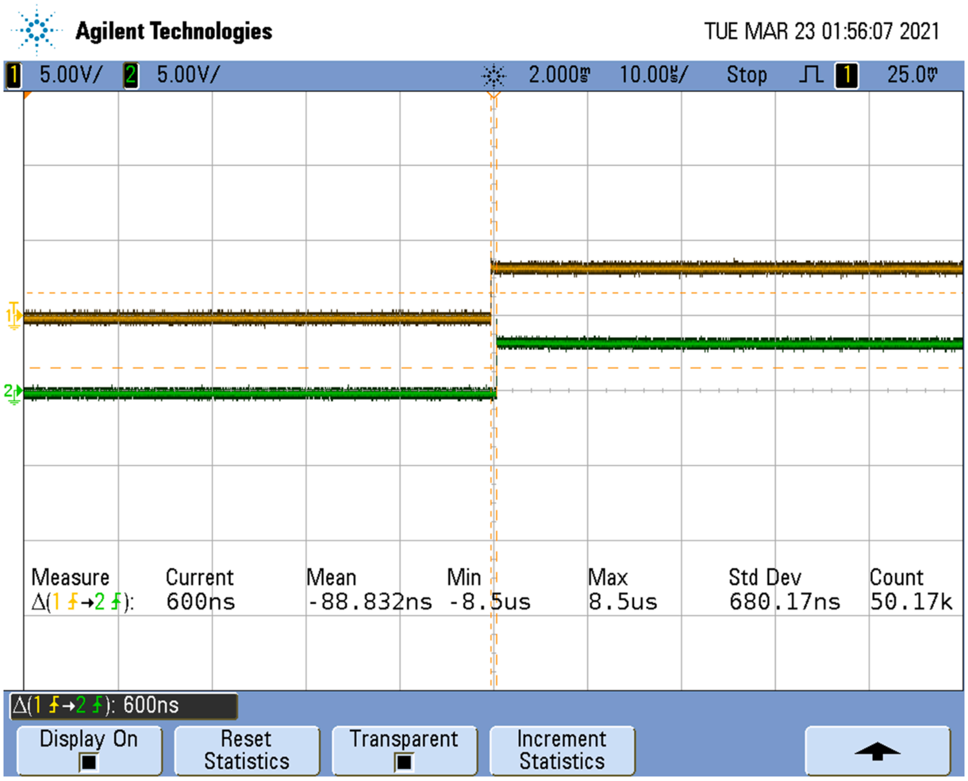
Our second objective is to contrast the worst-case response times obtained by applying the holistic response time analysis techniques for EDF with those obtained from the experiments using the platform described in Section 4. According to the theoretical results from previous research [10], EDF scheduling should be expected to perform better when GC-EDF is used as opposed to LC-EDF.

In relation to the scheduling deadlines assignment, PD could be the only algorithm that would make sense to apply to LC-EDF, as we might think that the e2e deadline will be guaranteed as long as local deadlines are also guaranteed for each step. However, the other scheduling deadline assignment techniques are known to be able to obtain lower worst-case response times as shown in [9,16]. Hence, we evaluate the four techniques for LC-EDF, and the UD, ED and PD-GSD techniques for GC-EDF.

6.2.1. *Evaluation scenario*

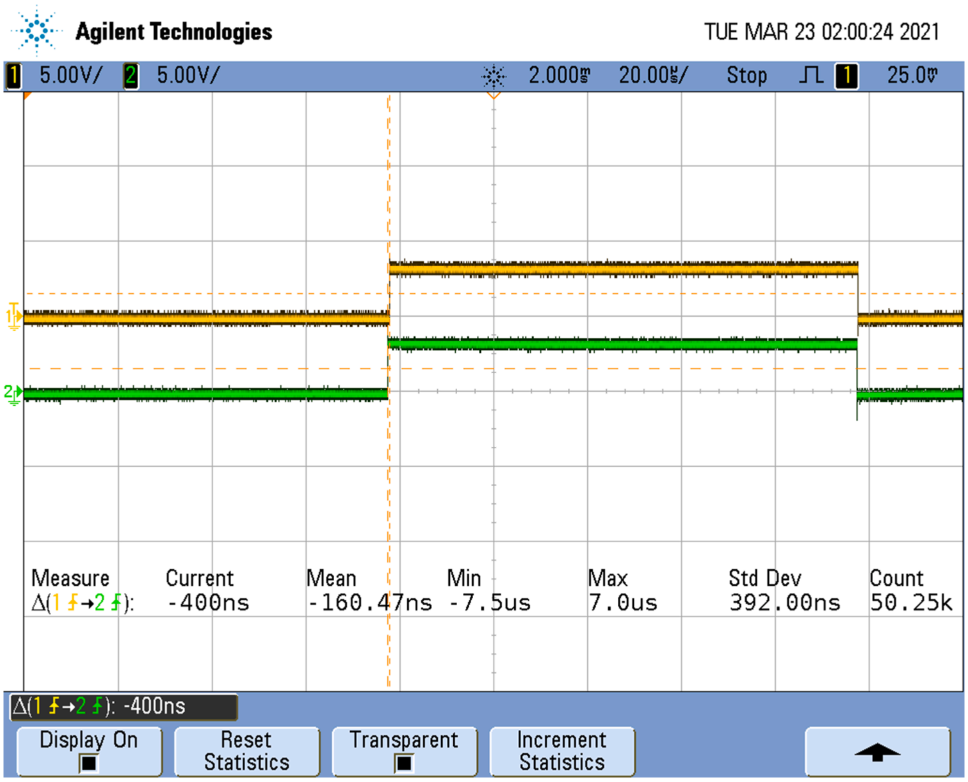
The distributed real-time application used for the analysis is illustrated in Fig. 7. It consists of two processors (CPU1 and CPU2) connected through a communication network. Four e2e flows are defined, all of them with the same structure as shown next: each e2e flow is composed of four steps, which are executed sequentially and alternately in each processor; the first one is activated periodically and the remaining steps are triggered by the arrival of the corresponding message through the network. The execution times, periods and deadlines of each e2e flow are carefully selected to show the differences in the response times obtained by the analysis when different scheduling deadline assignment techniques are used. The objective of this selection is to find configurations that allow obtaining representative theoretical results for the behavior of LC-EDF and GC-EDF scheduling, so that they can be later compared with the experimental ones. For the sake of simplicity, messages sent through the network have a fixed length of 64 bytes. In this example, the overhead associated with the network is considered negligible. This can be done without loss of generality since the transmission times are going to be several orders of magnitude smaller than the shortest of the e2e flow periods, as we show next. Similarly, we also consider negligible the overheads associated with the scheduling of the operating system (e.g., context switches) and the clock synchronization mechanism, which are in the range of microseconds while the application-level execution times are in the range of milliseconds.

In this experiment, we apply two configurations of the system concept outlined above, and whose characteristics are shown in Table 3. Both configurations define the same activation periods for the e2e flows and they also have the same e2e deadlines, which have been set as three times the period of the corresponding e2e flow; this is a common case of arbitrary deadlines in distributed systems. These deadlines enable obtaining a schedulable configuration for high CPU utilization (more than 95%). Furthermore, the execution times shown in Table 3 are constant (that is, the best-case execution times of steps are equal to their worst-case execution times), as it is the normally case for synthetic workloads. Once periods were set, several configurations for the execution times, ranging from mid to high (44% – 97%) CPU



```
Measure      Current   Mean        Min      Max     Std Dev    Count
Δ(1 ⨍→2 ⨍):  600ns     -88.832ns   -8.5us   8.5us   680.17ns   50.17k
Δ(1 ⨍→2 ⨍): 600ns
```

Fig. 5. Synchronization of two digital signals: metrics taken by oscilloscope in the *Reference* scenario.



```
Measure      Current   Mean         Min      Max     Std Dev    Count
Δ(1 ⨍→2 ⨍):  -400ns    -160.47ns    -7.5us   7.0us   392.00ns   50.25k
Δ(1 ⨍→2 ⨍): -400ns
```

Fig. 6. Synchronization of two digital signals: metrics taken by oscilloscope in the *MaRTE over Linux* scenario.

utilizations, were tested by means of schedulability analysis and scheduling deadline assignment techniques. Finally, the two configurations shown in Table 3 were selected, whose CPU utilizations are:

- *Configuration 1*: 74.14% for CPU1 and 74.59% for CPU2. Considering increasing workloads, this is the first configuration in which at least one of the scheduling deadline assignment techniques used (UD for

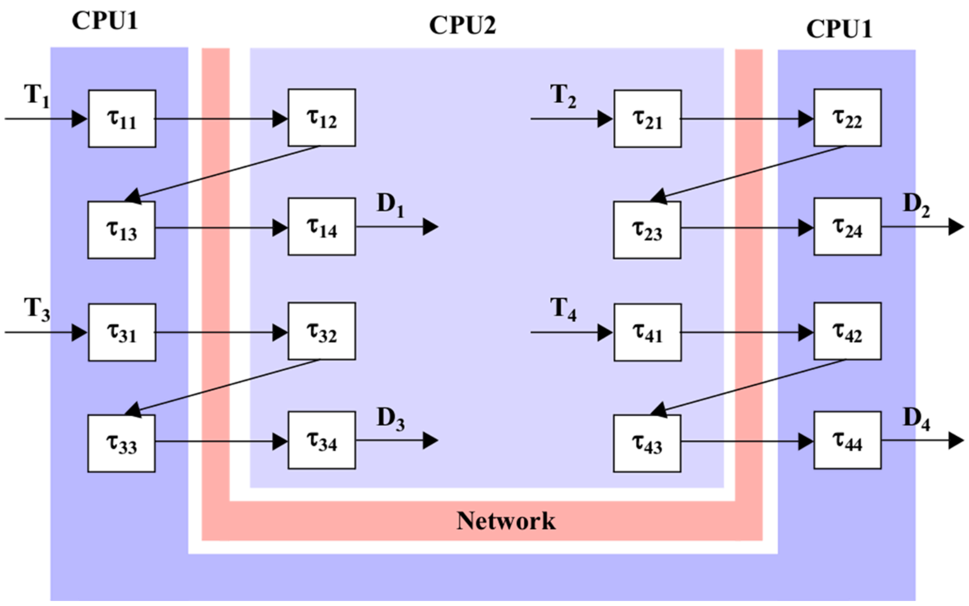


Fig. 7. The distributed real-time application under analysis.

Table 3
Parameters for the distributed real-time application (times in ms).

			Configuration 1				Configuration 2			
e2e flow	T_i	D_i	C_{i1}	C_{i2}	C_{i3}	C_{i4}	C_{i1}	C_{i2}	C_{i3}	C_{i4}
Γ_1	50	150	12	7	6	10	15	10	8	11
Γ_2	120	360	5	12	9	5	9	14	13	8
Γ_3	300	900	22	6	19	36	29	9	26	46
Γ_4	650	1950	14	41	83	26	21	59	98	38

instance) causes the system to incur a deadline miss (according to schedulability analysis).

- *Configuration 2*: 96.92% for CPU1 and 96.97% for CPU2. Considering increasing workloads, this is the first configuration in which none of the scheduling deadline assignment techniques used makes the systems schedulable.

6.2.2. Response-time analysis

Once the proposed application has been modelled with MAST, the scheduling deadline assignment techniques can be applied. Table 4 shows the scheduling deadlines obtained by the four techniques described in Section 5. It can be seen that the trivial UD technique assigns the same values for both configurations, and PD is the only technique in which the sum of the scheduling deadlines of the steps belonging to an e2e flow are within the bounds of the e2e deadline.

The results obtained by applying the holistic schedulability analysis techniques [26,10] to the proposed example are shown in Table 5. Only the worst-case response times of the last step in each e2e flow are shown, since they are the ones that can be compared to the e2e deadlines.

6.2.3. Response-time of experimental measurements

An implementation of the proposed application has been coded in C and executed on top of the platform described in Section 4. The two configurations have been executed with the different assignments of scheduling deadlines. Each test has been executed for one hour (approximately, 119 500 e2e flow executions per test) and the worst, best, and average response times of each e2e flow have been calculated from the response times measured. The availability of a global clock facilitates taking end-to-end measurements. In this case, the release time of each e2e flow is transmitted along with the network messages so it can be used by the last step in the corresponding e2e flow to calculate its response time.

Figs. 8 and 9 show the results obtained, expressed as a ratio to the e2e deadline of the corresponding e2e flow (i.e., a value below 100 indicates

Table 4
Scheduling deadlines (Sd_{ij} or SD_{ij}) assigned to steps by each technique (times in ms).

	Configuration 1					Configuration 2				
Step	UD	ED	PD	PD-GSD	D_i	UD	ED	PD	PD-GSD	D_i
τ_{11}	150.000	127.000	51.429	51.429	150	150.000	121.000	51.136	51.136	150
τ_{12}	150.000	134.000	30.000	81.429		150.000	131.000	34.091	85.227	
τ_{13}	150.000	140.000	25.714	107.143		150.000	139.000	27.273	112.500	
τ_{14}	150.000	150.000	42.857	150.000		150.000	150.000	37.500	150.000	
τ_{21}	360.000	334.000	58.065	58.065	360	360.000	325.000	73.636	73.636	360
τ_{22}	360.000	346.000	139.355	197.419		360.000	339.000	114.545	188.182	
τ_{23}	360.000	355.000	104.516	301.935		360.000	352.000	106.364	294.545	
τ_{24}	360.000	360.000	58.065	360.000		360.000	360.000	65.455	360.000	
τ_{31}	900.000	839.000	238.554	238.554	900	900.000	819.000	225.000	225.000	900
τ_{32}	900.000	845.000	65.060	303.614		900.000	828.000	75.000	300.000	
τ_{33}	900.000	864.000	206.024	509.639		900.000	854.000	216.667	516.667	
τ_{34}	900.000	900.000	390.361	900.000		900.000	900.000	383.333	900.000	
τ_{41}	1950.000	1800.000	166.463	166.463	1950	1950.000	1755.000	189.583	189.583	1950
τ_{42}	1950.000	1841.000	487.500	653.963		1950.000	1814.000	532.639	722.220	
τ_{43}	1950.000	1924.000	986.890	1640.850		1950.000	1912.000	884.722	1606.940	
τ_{44}	1950.000	1950.000	309.146	1950.000		1950.000	1950.000	343.056	1950.000	

Table 5
Theoretical worst-case response times of the e2e flows for each scheduling deadline assignment (times in ms).

	Configuration 1 – LC-EDF					Configuration 2 – LC-EDF				
e2e flow	UD	ED	PD	PD-GSD	D_i	UD	ED	PD	PD-GSD	D_i
Γ_1	70.000	54.000	48.000	64.000	150	N/C	N/C	N/C	87.000	150
Γ_2	164.000	118.000	134.208	176.321	360	N/C	N/C	N/C	326.505	360
Γ_3	609.000	356.000	433.286	400.000	900	N/C	N/C	N/C	814.000	900
Γ_4	**2392.000**	1466.000	1025.100	893.000	1950	N/C	N/C	N/C	**2786.940**	1950
	Configuration 1 – GC-EDF					Configuration 2 – GC-EDF				
	UD	ED	PD	PD-GSD	D_i	UD	ED	PD	PD-GSD	D_i
Γ_1	51.000	64.000	–	64.000	150	**1114.000**	**884.000**	–	88.318	150
Γ_2	121.000	141.000	–	178.000	360	**1352.000**	**1084.000**	–	316.818	360
Γ_3	530.000	481.000	–	391.000	900	**1923.000**	**1663.000**	–	761.485	900
Γ_4	1508.000	1457.000	–	893.000	1950	**3037.000**	**2793.000**	–	**2048.820**	1950

N/C: the response time analysis does not converge.
Worst-case response times written in bold correspond to non-schedulable e2e flows.

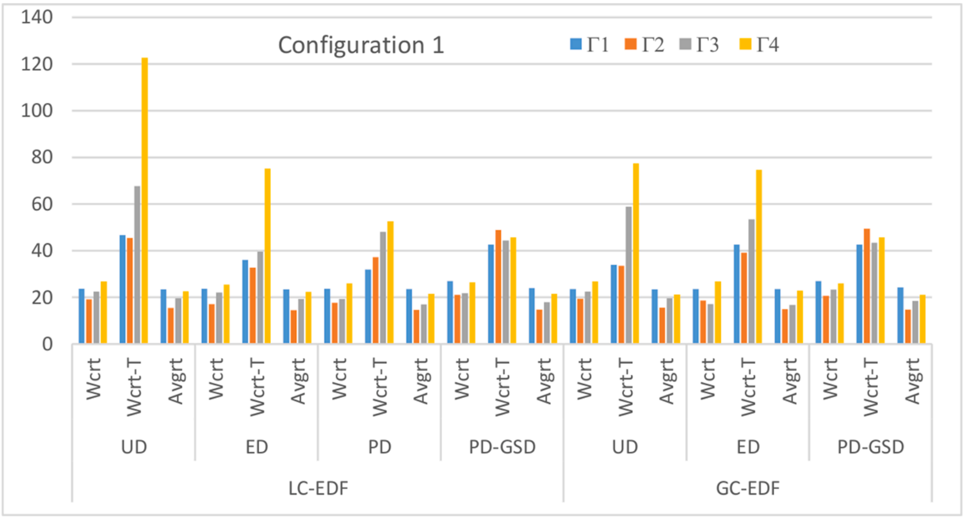


Fig. 8. E2e flow response times relative to the e2e deadlines forConfiguration 1 (in%).
Wcrt/Avgrt: worst-case and average response times for the real execution. Wcrt-T: worst-case response times for the theoretical analysis

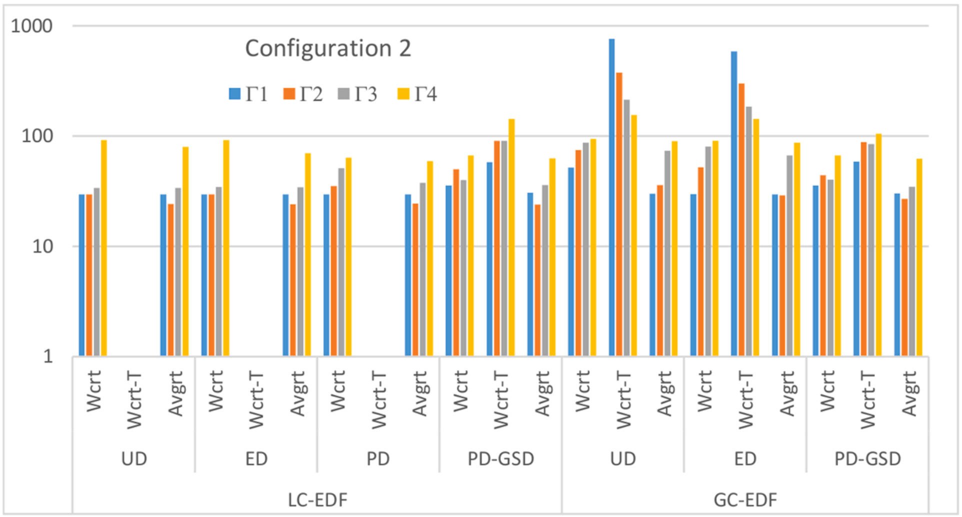


Fig. 9. Configuration 2 (in%).
Wcrt/Avgrt: worst-case and average response times for the real execution. Wcrt-T: worst-case response times for the theoretical analysis" and do the needful.

that the observed response time is below the deadline). The values represented in these figures refer to the measured worst-case response times (Wcrt), the theoretical results that come from the analysis (Wcrt-T, except the cases that do not converge in Configuration 2 which are not represented) and the average response times (Avgrt). Furthermore, a logarithmic scale is used for Configuration 2 in order to better appreciate the differences, since some theoretical results far exceed the e2e deadline.

6.2.4. Discussion

The theoretical results from Table 5 show that the predicted worst-case response times are strongly dependent on the assignment of scheduling deadlines; thus, the results from the response time analysis for the configuration with the highest CPU load (Configuration 2) show that the algorithm that computes the response time for LC-EDF does not converge (i.e., the worst-case response time of at least one e2e flow goes to infinite) when UD, ED and PD assignments are used. UD and ED assignments do converge for GC-EDF, although with very high times. According to the other configurations analysed (not shown in this paper), this lack of convergence begins to appear from a CPU utilization of 86% when UD, ED and PD assignments are used.

For the configuration with the lowest CPU load (Configuration 1), the results obtained for LC-EDF and GC-EDF are very similar except for the UD assignment. Under this configuration, the lowest response times are usually obtained by the PD-GSD assignment. Similarly to Configuration 1, in general, the lowest response times for Configuration 2 are also obtained with the PD-GSD assignment. In this case, the use of GC-EDF scheduling provides better results, as it significantly reduces the response times of three e2e flows (i.e., $\mathbf{\Gamma_2}$, Γ_3 and Γ_4) at the cost of slightly increasing it in $\mathbf{\Gamma_1}$.

These theoretical results are similar to those obtained in our previous works [9,16] in which GC-EDF also provides better results than LC-EDF, although the difference between the two schedulers is smaller for the example presented here.

The experimental results from Figs. 8 and 9 shows that the assignment techniques that did not converge in the theoretical analysis (i.e., those using LC-EDF in Configuration 2) obtain very similar worst-case response times among them in the real execution. Furthermore, some e2e flows present even lower worst-case response times than those obtained using the assignment techniques that converged. In any case, for the configuration with the highest CPU load, the best results come from the PD and the PD-GSD assignments techniques. In general, the UD and ED assignments for GC-EDF have obtained the highest response times in the real execution, following the trend of the theoretical results in which the predicted worst-case response times were remarkably high.

Likewise, Configuration 1 presents very similar experimental results for all the scheduling deadline assignment techniques, with small differences among them. Under this configuration, the PD-GSD assignment shows the smallest differences between theoretical and experimental results for both LC-EDF and GC-EDF scheduling. As can be seen in Fig. 8, the average response times for this configuration follow the same behavior as the worst-case response times (i.e., the assignment techniques with lower worst-case response times also obtain lower average response times). For CPU utilizations much lower than those in Configuration 1, diverse theoretical and experimental results have been observed. We consider that this is not an issue for low workloads when the system stays comfortably schedulable; e.g., for the lowest CPU utilization tested (44%), the ratio response time/deadline is not higher than 32% and 15% for the theoretical and the experimental results, respectively.

7. Conclusions and future work

This paper has explored and evaluated an EDF scheduling policy based on global scheduling deadlines for distributed real-time system. In order to apply GC-EDF scheduling, the paper has firstly proposed an architecture for distributed systems with support for a globally synchronized clock based on the IEEE 802.1AS standard. Under this architecture, global and local timing services are provided by the Linux kernel, and threading and real-time scheduling services are provided by MaRTE OS. This is appropriate since MaRTE OS supports EDF scheduling as described in the Real-Time Annex of Ada, which fits well with the concept of GC-EDF. The proposed architecture only aims at soft real-time systems since MaRTE OS applications are executed as standard Linux processes. To achieve better predictability, the proposed approach is also built upon the isolation facilities provided by the Linux kernel in order to execute MaRTE OS applications in an isolated core. Full isolation, however, cannot be guaranteed as the 1 HZ tick still remains present in the isolated core.

Furthermore, the paper also presents a distributed real-time platform where the proposed system architecture is implemented. The experimental results obtained in the platform tests show that the proposed approach could suit soft real-time applications with deadlines in the range of hundreds of microseconds.

This implementation is key to verifying whether the experimental results for EDF scheduling in distributed systems follow the same behavior as the theoretical results obtained by state-of-the-art schedulability analysis techniques. Since these experimental results are obtained from a limited simulation, the worst-case response times of the e2e flows may not have been actually observed. However, the measurements obtained do show a trend in the behavior followed by the different deadline assignment techniques: (1) the UD and ED assignments have generally obtained the highest response times for the theoretical analysis, and also obtained high response times for the real execution with high CPU load; (2) in general, the PD-GSD assignment can be considered the best technique for both LC-EDF and GC-EDF; and (3) the results of the schedulability analysis technique are highly dependent on the assignment of scheduling deadlines.

From these results, we can highlight that the theoretical dominance of GC-EDF over LC-EDF has not been observed in the experimental results. Even though the analysis techniques for both GC-EDF and LC-EDF are pessimistic (i.e., they obtain large upper bounds of the actual worst-case response times), the results suggest that the analysis is more pessimistic for LC-EDF as there is a larger disparity between the theoretical results and the experimental measurements. Finally, we can conclude that, except for simple systems, the relation between the theoretical-practical behaviors of EDF scheduling in distributed systems is rather difficult to understand and it would require further study.

In the short term, we plan to continue our investigation by exploring new emerging technologies such as ACRN, an hypervisor for Linux built with real-time and safety-criticality requirements in mind, which could enhance the isolation capabilities of the proposed architecture.

Declaration of Competing Interest

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

Data availability

Data will be made available on request.

Acknowledgments

This work was partially supported by MCIN/ AEI /10.13039/ 501100011033/ FEDER “Una manera de hacer Europa” under grant TIN2017-86520-C3-3-R (PRECON-I4).

References

[1] C.L. Liu, J.W. Layland, Scheduling algorithms for multiprogramming in a hard real-time environment, J. ACM 20 (1) (1973) 46–61.
[2] G. Buttazzo, Rate monotonic vs. EDF: judgment day, Real-Time Syst. 29 (1) (2005) 5–26.
[3] The Linux kernel documentation. Available online: https://www.kernel.org /doc/Documentation/[R*etrieved January* 2022].

[4] Zephyr real-time operating system home page: https://zephyrproject.org. [Retrieved January 2022].
[5] ERIKA enterprise, evidence home page, http://www.evidence.eu.com/. [Retrieved January 2022].
[6] P. Pedreiras, L. Almeida, EDF message scheduling on controller area network, Comput. Control Eng. J. 13 (4) (2002) 163–170.
[7] T. Qian, F. Mueller, Y. Xin, Hybrid EDF packet scheduling for real-time distributed systems, in: Proc. of the 27th Euromicro Conference on Real-Time Systems, Lund (Sweden), 2015, pp. 37–46.
[8] ISO/IEC, 2012. Ada 2012 reference manual. Language and standard libraries - International Standard ISO/IEC 8652:2012(E), doi: 10.1007/978-3-642-45419-6, (2012).
[9] J.M. Rivas, J.J. Gutiérrez, J.C. Palencia, M.González Harbour, Deadline assignment in EDF schedulers for real-time distributed systems, IEEE Trans. Parallel Distrib. Syst. 26 (10) (2015) 2671–2684.
[10] J.M. Rivas, J.J. Gutiérrez, J.C. Palencia, M.González Harbour, Optimized deadline assignment and schedulability analysis for distributed real-time systems with local EDF scheduling, in: Proc. of the 8th International Conference on Embedded Systems and Applications, ESA'2010, Las Vegas (Nevada), USA, 2010, pp. 150–156.
[11] IEEE 802.1AS - "Timing and synchronization for time-sensitive applications in bridged local area networks". (2020).
[12] IEEE 802.1 Time-sensitive networking (TSN) task group. Available online: https://1.ieee802.org/tsn/[*Retrieved January* 2022].
[13] B. Akesson, M. Nasri, G. Nelissen, S. Altmeyer, R.I. Davis, An empirical survey-based study into industry practice in real-time systems, in: Proceedings of the 41st Real-Time Systems Symposium (RTSS 2020), 2020, pp. 3–11.
[14] M. Madden. "Challenges using Linux as a real-time operating system". AIAA SciTech Forum (Software Challenges in Aerospace). doi:10.2514/6.2019-0502(2019).
[15] H. Pérez Tijero, D. García, J.J. Gutiérrez, First steps towards an IEEE 802.1AS clock for EDF scheduling in distributed real-time systems, in: 25th Ada-Europe International Conference on Reliable Software Technologies (AEiC), *Work in Progress* Session, in Ada User Journal 42, 2021, pp. 121–124.
[16] J.J. Gutiérrez, H. Pérez, Theory and practice of EDF scheduling in distributed real-time systems, in: Proc. of the 23rd International Conference on Reliable Software Technologies, Ada-Europe 2018, Lisbon (Portugal), in Lecture Notes in Computer Science 10873, LNCS, 2018, pp. 123–137.
[17] R.I. Davis, A. Burns, S. Baruah, T. Rothvoß, L. George, O. Gettings, Exact comparison of fixed priority and EDF scheduling based on speedup factors for both pre-emptive and non-pre-emptive paradigms, Real-Time Syst. 51 (5) (2015) 561–601.
[18] D. Perale, T. Vardanega, Removing bias from the judgment day: a Ravenscar-based toolbox for quantitative comparison of EDF-to-RM uniprocessor scheduling, J. Syst. Arch. 119 (2021), 102236.
[19] P. Carletto, T. Vardanega, Ravenscar-EDF: comparative benchmarking of an EDF variant of a Ravenscar runtime. Reliable Software Technologies – Ada-Europe 2017, LNCS, Springer International Publishing, 2017, pp. 18–33.
[20] R.I. Davis, A. Burns, A survey of hard real-time scheduling for multiprocessor systems, ACM Comput. Surv. 43 (4) (2011) 35.
[21] M. Bertogna, M. Cirinei, G. Lipari, Schedulability analysis of global scheduling algorithms on multiprocessor platforms, IEEE Trans. Parallel Distrib. Syst. 20 (4) (2009) 553–566.
[22] S. Baruah, N. Fisher, Non-migratory feasibility and migratory schedulability analysis of multiprocessor real-time systems, Real-Time Syst. 39 (1–3) (2008) 97–122.
[23] S. Baruah, Partitioned EDF scheduling: a closer look, Real-Time Syst. 49 (6) (2013) 715–729.
[24] G. Gracioli, A.A. Fröhlich, R. Pellizzoni, S. Fischmeister, Implementation and evaluation of global and partitioned scheduling in a real-time OS, Real-Time Syst. 49 (6) (2013) 669–714.
[25] Z. Dong, K. Yang, N. Fisher, C. Liu, Tardiness bounds for sporadic gang tasks under preemptive global EDF scheduling, in: IEEE Transactions on Parallel and Distributed Systems 32, 2021, pp. 2867–2879, https://doi.org/10.1109/TPDS.2021.3081019.
[26] J. Liu, Real-Time Systems, Prentice Hall, 2000.
[27] M. Spuri, Tech. Rep. RR-2873. "Holistic Analysis for Deadline Scheduled Real-Time Distributed Systems", INRIA, France, 1996.
[28] J.C. Palencia, M. González Harbour, Offset-based response time analysis of distributed systems scheduled under EDF, in: Proc. 15th Eur. Conf. Real-Time Syst., Porto, (Portugal), 2003, pp. 3–12.
[29] U. Diaz De Cerio, M. González Harbour, J.C. Palencia, J.P. Uribe, Adding precedence relations to the response-time analysis of EDF distributed real-time systems, in: 22nd International Conference on Real-Time Networks and Systems, RTNS, 2014.
[30] K. Yong Ju, et al., Performance of IEEE 802.1 AS for automotive system using hardware timestamp, in: The 18th IEEE International Symposium on Consumer Electronics (ISCE), 2014.
[31] M. Gutiérrez, W. Steiner, R. Dobrin, S. Punnekkat, Synchronization quality of IEEE 802.1AS in large-scale industrial automation networks, in: IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS), 2017, pp. 273–282, https://doi.org/10.1109/RTAS.2017.10.
[32] L. Abeni, G. Buttazzo, Integrating multimedia applications in hard real-time systems, in: Proceedings of the 19th IEEE Real-Time Systems Symposium, Madrid, 1998, pp. 4–13.
[33] M. Aldea, M. González, MaRTE OS: an Ada kernel for real-time embedded applications, in: Proc. of the International Conference on Reliable Software Technologies, Ada-Europe 2001 2043, Leuven, Belgium, in Lecture Notes in Computer Science, LNCS, 2001.
[34] M. Masmano, J. Real, I. Ripoll, A. Crespo, Extending the capabilities of real-time applications by combining MaRTE-OS and Linux, in: Proc. of the International Conference on Reliable Software Technologies, Ada-Europe 2004. Lecture Notes in Computer Science 3063, 2004, https://doi.org/10.1007/978-3-540-24841-5_11.
[35] MAST home page: http://mast.unican.es/[*Retrieved January* 2022].
[36] M. González Harbour, J.J. Gutiérrez, J.M. Drake, P. López, J.C. Palencia, Modeling distributed real-time systems with MAST 2, J. Syst. Arch. 59 (6) (2013) 331–340.
[37] Object Management Group. UML profile for MARTE: modeling and analysis of real-time embedded systems. OMG document, v1.1 formal/2011-06-02 (2011).
[38] J.C. Palencia, J.J. Gutierrez, M.Gonzalez Harbour, Best-case analysis for improving the worst-case schedulability test for distributed hard real-time systems, in: Proc. of the 10th EUROMICRO Workshop on Real-Time Systems, 1998, pp. 35–44, https://doi.org/10.1109/EMWRTS.1998.684945.
[39] F. Reghenzani, G. Massari, W. Fornaciari, The real-time Linux kernel: a survey on PREEMPT_RT, ACM Comput. Surv. 52 (1) (2019) 36, https://doi.org/10.1145/3297714. Article 18.

Héctor Pérez Tijero has been participating in intense teaching and research activity in the Software Engineering and Real-Time Group at the University of Cantabria (Spain) since 2008. He received his M.*Sc*. and Ph.D. in 2008 and 2012, respectively. His-Ph.D. was concerned with the integration of a real-time model into distribution middleware to facilitate the development process of distributed real-time systems. He works in software engineering for real-time systems and has been involved in several research and industrial projects using emerging distribution middleware technologies to build distributed and deterministic applications.

J. Javier Gutiérrez received his B.S. and Ph.D. Degrees from the University of Cantabria (Spain) in 1989 and 1995 respectively. He is an Associate Professor in the Software Engineering and Real-Time Group at the University of Cantabria since 1996, where he works in software engineering for real-time. His research activity deals with the scheduling, analysis and optimization of embedded real-time distributed systems (including communication networks). He has been involved in several research projects building real- time controllers for robots, evaluating Ada for real-time applications, developing middleware for real-time distributed systems, and proposing models along with the analysis and optimization techniques for distributed real-time applications.