IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received October 2, 2020, accepted October 6, 2020, date of publication October 9, 2020, date of current version October 22, 2020.

*Digital Object Identifier 10.1109/ACCESS.2020.3029858*

# New Insights Into the Real-Time Performance of a Multicore Processor

**RAIMARIUS DELGADO, (Member, IEEE), AND BYOUNG WOOK CHOI, (Member, IEEE)**
Department of Electrical and Information Engineering, Seoul National University of Science and Technology, Seoul 01811, South Korea

Corresponding author: Byoung Wook Choi (bwchoi@seoultech.ac.kr)

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Ministry of Science and ICT (MSIT) of the Korean Government under Grant 2019R1F1A1063547.

**ABSTRACT** Multicore processors are gaining popularity in various domains because of their potential for maximizing system throughput of average-case tasks. In real-time systems, where processes and tasks are governed by stringent temporal constraints, the worst-case timings should be considered, and migration to multicore processors leads to additional difficulties. Resource sharing between the cores introduces timing overheads, which affect the worst-case timings and schedulability of the entire system. In this article, we provide new insights into the performance of the real-time extensions of Linux, namely, Xenomai and RT-Preempt, for a homogeneous multicore processor. First, complete details on leveraging both real-time extensions are presented. We identify various multicore deployments and discuss their trade-offs, as established through the experimental evaluation of the scheduling latency. Then, we propose a statistical method based on a variation of chi-square test to determine the best multicore deployment. The unexpected effects of interfering loads, such as CPU, memory, and network operations, on the real-time performance, are considered. Feasibility of the best multicore deployment is verified through the analysis of its periodicity and deterministic response times in a pre-emptive multitasking environment. This research is the first of its kind and will serve as a useful guideline for developing real-time applications on multicore processors.

**INDEX TERMS** Multicore architecture, Xenomai, RT-preempt, real-time, statistical selection.

## I. INTRODUCTION

Integration of multicore processors has become common in general computing and in various embedded applications, which includes robotics, control systems, and automotive [1]–[3]. Deployment of processes over multiple cores has demonstrated advantages in accelerating task execution, system scalability, and low-power consumption. These benefits are possible if the operating software leverages various techniques of parallel processing [4].

Parallel processing in multicore processors is designed to satisfy average-case requirements, which are usually not time-sensitive. In real-time systems, however, tasks and processes are governed by stringent temporal constraints. Migration to multicore processors introduces additional difficulties. Resource sharing and load balancing induces timing overheads that affect the worst-case execution times. This unpredictable behavior may result in violation of the temporal constraints leading to more aggravated problems, such as severe program errors, system failures, and safety-critical issues [5]–[7]. We note that formulation of a real-time scheduling algorithm is not the focus of this article. But instead, interested readers may refer to [8]–[10] for examples of real-time scheduling on multicore processors.

Commercial real-time operating systems (RTOSs), such as VxWorks [11], Neutrino [12], and Nucleus [13], offer real-time scheduling solutions on multicore processors. However, these RTOSs are usually distributed in a black box, which hinders integration to more complex software. Expensive licensing and royalty costs are also serious issues, especially for research and academic institutions. Open-source software overcome these problems. Linux, the most popular open-source operating system, enables developers to freely access and modify the kernel source code in accordance with their application requirements. It has a regularly active ecosystem, which can greatly contribute to code evaluation and problem debugging.

The real-time capabilities of Linux have greatly enhanced owing to the continuous efforts of the community. Starting from version 4.x and above, it is included with a low-latency

The associate editor coordinating the review of this manuscript and approving it for publication was Laxmisha Rai.

pre-emptible kernel configuration, which enables user-space tasks to satisfy *soft* real-time requirements. In other words, tasks are expected to meet their respective temporal deadlines *most* of the time. This trend led to the integration of Linux on a variety of soft real-time projects, for example, multimedia streaming [14], image processing [15], and virtual reality [16].

Conversely, applications that would result in tragic events when a temporal deadline is missed should meet *hard* real-time requirements. Hard real-time in Linux is achieved by revising the kernel and is divided into two major categories: The dual-kernel and the fully pre-emptible kernel approaches.

The dual-kernel approach adapts a co-kernel architecture, where a real-time kernel runs alongside the standard Linux through the ADEOS [17] hardware abstraction layer. In this configuration, the real-time kernel holds the highest priority; meaning that standard Linux tasks are allowed to run only if there is no real-time waiting for execution. Open source projects such as Xenomai [18] and RTAI [19], are some of the popular examples of the dual-kernel approach.

On the contrary, RT-Preempt [20] reworks the internals of the Linux kernel. In particular, elements involving the timers, schedulers, and locking-mechanisms are enhanced to reduce latencies and convert Linux into a fully pre-emptible kernel [21]. One of the attractive features of RT-Preempt in contrast to the dual-kernel approach is better integration with Linux-based libraries, which makes application development easier and flexible [22]. As a trade-off, its real-time performance is inferior according to the experimental studies reported in [23]–[25].

Currently, neither of the real-time extensions administers real-time scheduling on multicore processors. This problem encourages many researchers to modify the Linux scheduler, while maintaining a desirable real-time performance. Several scheduler tools have been proposed in literature. Among them, the concept of integrating plugin-based schedulers has been widely adopted [26]–[28]. Particularly, LITMUS$^{RT}$ [26] is a kernel patch which provides a testbed to implement and evaluate real-time scheduling algorithms. Similarly, loadable kernel modules such as RESECH [27] and ExSched [28] eliminate the need of modifying the kernel code, while offering a similar environment with LITMUS$^{RT}$. These projects have allowed observing the performance of various real-time multicore scheduling algorithms in Linux; however, their high runtime overheads result to unpredictable behavior which render them unfit for practical real-time applications [21], [29].

Another popular approach is to replace the standard Linux scheduler with earliest-deadline first (EDF) [30], [31]. Though this method reduces runtime overheads, sharing resources between tasks on multicore processors introduces blocking delays and priority inversion. To overcome such limitation, Han *et al*. [30] proposed a blocking-aware partitioned scheme which incorporates several resource-guided mapping heuristics. Because dynamic deadlines on the EDF scheduler constitute task priorities, Lelli *et al.* [31] has adopted deadline inheritance. These approaches enable EDF scheduling of real-time tasks on multicore processors; however, practical real-time applications, which usually handle I/O bound tasks and interrupt handlers, require constant and deterministic response times. In the case of EDF, dynamic priorities of the scheduled tasks often results to variable response times [32].

Due to the absence of a practical real-time scheduler that considers task migration on multicore processors, several studies have suggested to either isolate a specific core and associate real-time task onto the isolated core; or emulate a uniprocessor system by disabling all physical and logical cores. In particular, Cereia *et al*. [25] and Betz *et al.* [33] investigated the performance of RT-Preempt and RTAI under several types of simulated loads. They found out that disabling the multicore features results to better periodicity of a single real-time control task. On the other hand, only logical cores affect the real-time performance of Xenomai [34]. Considering that most of embedded platforms in the market nowadays are based on multicore processors, disabling multicore features is a huge waste of resources. Thus, there is a current demand for a general rule to determine the best multicore deployment, which exhibits feasible real-time behavior and deterministic response times, depending on the real-time extension of Linux.

With this motivation, the goal of this article, therefore, is to present new insights into the real-time performance of two real-time Linux extensions, Xenomai and RT-Preempt, on a homogeneous Intel multicore system. First, the details in configuring the Linux kernel to successfully leverage both real-time extensions are provided. Herein, we enumerate the critical kernel options to avoid unwanted timing overheads that may affect the real-time performance. Then, we identify a set of multicore deployments contingent on several processor features: C-State [35], hyperthreading (logical cores), multicore (physical cores), and CPU isolation. The real-time performance of each multicore deployment is evaluated in terms of the scheduling latency, which yields valuable perception on the schedulability and worst-case timings of real-time tasks [36]. Also, calibration methods of timers are also presented to ensure that all measurements are close to the ideal time.

Comparison of the averages and extreme values may not yield meaningful results because of the slight differences between those of the multicore deployments. This article also proposes a statistical method based on a variation of chi-square ($\chi^2$) test to analyze the distributions of scheduling latencies for each multicore deployment and to determine the best multicore deployment among them.

After the best multicore deployment has been identified for each real-time extension, the real-time performance under the effects of interfering loads and its feasibility in a multitasking environment are analyzed to build confidence on the suitability of multicore processors for practical real-time applications. In this regard, the main contributions of this work are to:

1) Provide the complete details on leveraging Xenomai and RT-Preempt on a homogeneous multicore processor.
2) Identify various multicore deployments and evaluate their trade-offs, in terms of the scheduling latency.
3) Propose a variation of $\chi^2$ test to determine the best multicore deployment for the real-time Linux extensions.
4) Investigate the unexpected effects of interfering loads, such as CPU, memory, and network operations, on the performance of the best multicore deployment, then verify its feasibility through rate-monotonic analysis in a pre-emptive multitasking environment

To the best of our knowledge, this study is the first of its kind and will serve as a useful guideline for developing real-time applications on multicore processors.

## II. REAL-TIME MODEL AND PERFORMANCE METRIC

In order to guarantee the timeliness and deterministic responses of the real-time system, it is necessary to perform schedulability analysis to determine whether a set of real-time tasks will complete its execution according to the specified temporal deadline. Both real-time Linux extensions are based on fixed priority scheduling; thus, we evaluate task response times using rate-monotonic analysis (RMA) [37].

Although RMA is only feasible for uniprocessor scheduling, the choice to use this method is made for trend analysis purposes. It is also worth to note that our focus is to evaluate the effects of multicore deployments on the real-time performance of the system, and not to formulate a novel multicore scheduling algorithm.

In this article, we concentrate on a real-time system that supports periodic tasks, whose execution is triggered after a defined fixed time-interval. A real-time task, $\tau_i$, is defined as a tuple, $\tau_i =< P_i, D_i, C_i, pr_i >$, where $P_i$ represents the period, $D_i$ denotes the deadline, $C_i$ stands for the worst-case execution time, and $pr_i$ represents the priority. These task parameters are essential in scheduling. For instance, scheduling based on priority guarantees that the highest priority task would always execute and should not be interfered by lower-priority tasks.

In conformity with RMA, a set of $n$ number of tasks with harmonic periods, $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$ is schedulable only if the worst-case response time (WCRT), $R_\mathrm{i}$, for every task, $\tau_i$, is less than or equal to its respective deadline, $D_\mathrm{i}$. The WCRT of a real-time task is calculated by [37]:

$$R_i^{x+1} = C_i + B_i + J_i + \underbrace{\sum_{j \in hp(i)} \left\lceil \frac{R_i^x + J_j}{P_j} \right\rceil \cdot C_j}_{I_i}, \quad i = 1, 2, \ldots, n \qquad (1)$$

Herein, $B_i$ represents the blocking time—when a low priority task forcefully owns resources required by higher priority tasks. This occurs when the scheduler does not properly exhibit pre-emptive behavior, or if a low-priority task has not released certain shared locking mechanisms. $I_i$ denotes the interference, which is the sum of all execution times of the tasks within $hp(i)$ that pre-empts task $i$, where $hp(i)$ is the set of tasks with higher priorities than $\tau_i$. Calculation for the WCRT requires iteration of (1) until $R_i^{x+1} = R_i^x$ or $R_i^{x+1} \geq D_i$ is satisfied. Take note that for the first iteration, the response time is equal to the execution time, or $R_i^0 = C_i$.

The jitter $J_i$ is the random deviation from the expected period, or the difference between two iterations ($x$) of the periodic task:

$$J_i^x = P_i^{x+1} - P_i^x \qquad (2)$$

The presence of the jitter is inevitable as a result of the stochastic behavior of RTOS schedulers and its effect on the periodicity and WCRT of real-time tasks is apparent. The jitter is also known as the variance of the scheduling latency [36], which is the interval between the expected release point and the instant when the task actually starts executing. The scheduling latency has numerous sources (e.g., memory paging, interrupt latency, caching) and searching for the exact one is extremely difficult. Nonetheless, the scheduling latency should remain low to ensure that the real-time tasks are schedulable and exhibit deterministic behavior.

In this regard, we will evaluate the real-time performance of two real-time Linux extensions by measuring the scheduling latencies of each system considering various multicore deployments and determine the best one among them.

## III. EXPERIMENTAL SETTINGS AND MULTICORE DEPLOYMENT

In this section, the experimental environment based on two real-time Linux extensions, Xenomai and RT-Preempt, for a homogeneous Intel multicore embedded system (MES) is described in detail. Various multicore deployments are enumerated with discussion of each of their features and effects on the entire system. The experimental tests are carried out to evaluate the scheduling latencies for the multicore deployment on both Xenomai and RT-Preempt in an idle environment.

### A. EXPERIMENTAL REAL-TIME ENVIRONMENT

As the Linux kernel is originally developed for Intel-based processors, the experimental real-time environment is built around an Advantech MIO-5272 MES. The system is equipped with an Intel i7-6600 quad-core processor consisting of two physical cores and two logical cores running at 3.4 GHz and with 16 GB of DDR3L RAM. In order to minimize the effects of integrated graphics controllers, the MES is installed with the minimal Linux distribution, Lubuntu 18.04. We have chosen the most stable version, which is Linux 4.14.134, for both Xenomai and RT-Preempt. Although both approaches support later versions of the kernel, most of them are still in the experimental stage.

#### 1) Xenomai

To implement a dual-kernel environment with Xenomai, the hardware abstraction layer called ADEOS [17] is required

to concurrently leverage both Xenomai and Linux kernels. It is necessary for the ADEOS patch to be compatible with both kernels for a successful implementation. For this specific version, the appropriate ADEOS is ipipe-core-4.14.134-x86-8. We have selected the latest version, v3.1, of Xenomai that can be downloaded either from the Xenomai GIT repository or as a tarball from their download page. To ensure that the latest fixes and updates are included in the implementation, it is preferred to clone the source code from the GIT repository.

After the Linux kernel is patched with ADEOS, several kernel options should be enabled/disabled to guarantee that the system will not experience any unwanted latencies. It is worth to note that some of these configurations may vary depending on the architecture of MES. In the case of the Intel-based MIO-5272, processor dependent kernel options and memory page migration are as follows:

- Enable HIGH_RES_TIMERS
- Enable MCORE2
- Disable CONFIG_SCHED_MC
- Disable CONFIG_TRANSPARENT_HUGEPAGE
- Disable CONFIG_COMPACTION
- Disable CONFIG_MIGRATION

Power management is a vital source of latencies in a real-time system and should be deactivated. The following kernel options should be disabled in the following order:

- CONFIG_CPU_FREQ
- CONFIG_ACPI_PROCESSOR
- CONFIG_INTEL_IDLE
- CONFIG_CPU_IDLE

Debugging features of the kernel are also another source of unwanted latencies. In particular, KGDB uses a source level debugger that breaks into the kernel to inspect variables, call stack information, and memory usage. To prevent latencies, which are due to this feature, the kernel option CONFIG_KGDB should be disabled.

For this particular version, we have encountered a conflict between the Microsoft Hyper-V guest device driver and CPU Idle features that causes errors during kernel compilation. As the former is not necessary in a real-time system, the device driver, CONFIG_HYPERV, should be disabled. In our case, successful compilation and installation of the full kernel was completed after 56 minutes. For easier development of user space real-time tasks, Xenomai user space libraries and tools should be built after installation of the patched kernel.

#### 2) RT-PREEMPT

RT-Preempt patches the kernel to support hard real-time tasks and fully pre-emptible scheduling in Linux, without the need of a co-kernel. One of its attractive features is better integration with Linux-based libraries such as ROS and IgH EtherCAT [38], [39]. However, in comparison with its dual-kernel contemporaries, many studies reported that its real-time performance is still inferior [23]–[25].

The RT-Preempt patches are available in the Linux kernel repository, and the compatible version for our selected kernel is 4.14.134-rt63. The CONFIG_PREEMPT_RT should be enabled to configure the kernel with RT-Preempt. The same processor, memory paging, and power management kernel options should also be enabled/disabled as discussed in the previous section, aside from the Microsoft Hyper-V device driver, which did not impose any conflicts when compiling the RT-Preempt patched Linux kernel.

### B. MULTICORE DEPLOYMENT

Several studies have reported the effects of various multicore configurations on the real-time performance of real-time Linux extensions. Betz *et al*. [33] stated that using multicore worsens the periodicity of real-time tasks in RT-Preempt. To deal with this issue, they permanently disabled multicore features of a dual-core processor. Alternatively, the results of Garre *et al*. [34] has shown that hyperthreading affects the real-time performance of Xenomai. To this end, a general rule to determine the best multicore deployment that exhibits feasible real-time performance and deterministic response times for the real-time extensions of Linux is in demand.

This article, for the first time, details the possible impacts of six multicore deployments on the real-time performance of Xenomai and RT-Preempt. Each deployment is contingent on the following processor features and kernel parameters: C-State, hyperthreading, multicore, and CPU isolation. C-State refers to the power management feature of Intel CPU features, where the processor enters several states when idle [35]. As mentioned in the previous sections, any features regarding power management should be disabled in a real-time system to avoid unwanted latencies. To disable C-State, the kernel should be instructed that the processor must not leave the operating state, with "*intel_idle.max_cstate=0*" as a boot parameter. However, we have observed that the kernel bypasses this parameter, and continuously booting with the C-State still activated. Hence, it is preferable to disable C-State in the BIOS permanently.

Hyperthreading refers to the utilization of logical cores available within the system. Although the physical core only has a single set of execution instructions, it powers the logical cores and delegates the tasks to improve context switching. In the case of MIO-5272, each physical core has its own logical cores totaling to four available cores. The multicore option, on the other hand, refers to the physical cores. When disabled, the system emulates a uniprocessor system.

Isolated CPU refers to the isolation of CPU cores from the standard Linux scheduler and associating real-time tasks to the isolated CPUs. We isolated all CPU cores aside from core 0, where all the interrupts and non-real-time processes are migrated onto. The boot parameter, *isolcpus*, is supplied with the cores to be isolated. For example, "*isolcpus=1,2,3*" refers to isolating cores 1-3 when multicore and hyperthreading are both enabled. Xenomai should be instructed with the masked value of the CPU cores that it could utilize. In the

**TABLE 1. List of multicore deployments.**

| Deployment | C-State | Hyper threading | Multicore | Isolated CPU | CPUs |
|---|---|---|---|---|---|
| CHM | Y | Y | Y | N | 4 |
| HM | N | Y | Y | N | 4 |
| HMI | N | Y | Y | Y | 4 |
| M | N | N | Y | N | 2 |
| MI | N | N | Y | Y | 2 |
| Single | N | N | N | N | 1 |

Y and N refer to the option being enabled or disabled, respectively.

case above, the boot parameter is "*xenomai.supported_cpus* $= 0 \times 7$".

To clearly distinguish the multicore deployments, we employ a naming convention, where each of them is labeled with the first letter of the enabled features and parameters in the order shown in Table 1. For example, deployment CHM refers to C-State (C), Hyperthreading (H), and Multicore (M) options being enabled, while Isolated CPU (I) is disabled. We refer to Single as the deployment, where all the options mentioned above are disabled, emulating a uniprocessor system.

### C. PERFORMANCE EVALUATION IN IDLE ENVIRONMENT

In this article, we evaluate the real-time performance of the multicore deployments in terms of the scheduling latency. We have utilized the *cyclictest* benchmarking tool which measures the scheduling latency using standard POSIX programming interface as shown in Algorithm 1.

**Algorithm 1** Scheduling Latency Measurement

```
1:  clock_gettime((&now));
2:  next = now + expected_period;
3:  while (1)
4:  {
5:      clock_nanosleep((&next));
6:      clock_gettime((&now));
7:      sched_lat = calc_diff(now, next);
8:
9:      update_statistics_buffer(sched_lat);
10:     next + = interval;
11: }
```

This program creates standard configured POSIX threads and promotes them to real-time tasks by assigning necessary task parameters, including priority, scheduling policy, and the CPU affinity. In the measurement cycle, the task starts with a call to *clock_nanosleep*(), which waits for the next release point (period). After the task resumes, the scheduling latency is measured as the difference between the actual and the calculated release times. The next release point is calculated starting a new iteration.

During the experiment, the system being tested is kept isolated to avoid unwanted interrupts in an idle environment and all data analyses are conducted offline.

**TABLE 2. Scheduling latencies of Xenomai tasks in various multicore deployments (in microseconds).**

| Deployment | Ave. | Max. | Min. | $\mu$ |
|---|---|---|---|---|
| HM | 2.549 | 72.824 | 1.907 | 0.229 |
| HMI | 2.554 | 62.197 | 1.864 | 0.218 |
| M | 2.548 | 46.917 | 1.646 | 0.178 |
| MI | 2.547 | 22.041 | 1.521 | 0.156 |
| Single | 2.531 | 21.801 | 1.446 | 0.151 |

#### 1) Xenomai

Because of the dual-kernel architecture of Xenomai, calibration of the timers is required to ensure that the measurements are close to the ideal time. The shortest time (gravity) possible for the MES to deliver an interrupt to the Xenomai interrupt handler, kernel task, or user space task is calculated. The gravity is a static value used for adjusting Xenomai timers depending on the context that the timers are activated which, in our case, is the user space.

In Xenomai 3.1, the gravity is calculated with *autotune*, the calibration tool in the Xenomai testing suite. Note that calibration should be performed every time the system starts. In our case, we acquired an average scheduling latency of 4.01 $\mu s$ when the system is not calibrated, 6.03 $\mu s$ when the gravity is set to 0, and 2.56 $\mu s$ after running *autotune*.

Fig. 1 shows the histograms of the measured scheduling latencies for each multicore deployment on Xenomai. We have utilized a Xenomai-ported *cyclictest*, which is also included in the Xenomai testing suite. The tool was executed with *-m*, which enables memory page locking with *mlockall*() to prevent page faults. To ensure that *clock_nanosleep*() is used in the measurement, we also enabled the *-n* flag. A real-time task is configured with the period, $P = 100\ \mu s$. Note that the deadline, $D$, is equal to the period. To ensure that blocking does not occur, the task is configured with the highest priority of $pr = 99$.

Additionally, the real-time task does not execute other calculations aside from those mentioned in Algorithm to guarantee that the worst-case execution time (WCRT) is kept minimal. The measurements are conducted for 10 minutes, resulting to a reasonable sampling count of 6,000,000. Each measurement results in a histogram of actual scheduling latencies, as shown in Fig. 1. The numerical results of the scheduling latencies are tabulated in Table 2 : average (Ave.), maximum (Max.), minimum (Min.) and the standard deviation ($\mu$).

We have observed multiple cases of overruns, where the task was not able to meet the specified deadlines on the deployment CHM. We have assessed that the contributing factor for this counterintuitive behavior is the C-State. Thus, we have omitted the results for CHM and focused on the remaining multicore deployments.

Each of the experiments resulted in similar average values, with HM having the worst maximum and standard deviation of 72.824 $\mu s$ and 0.229 $\mu s$, respectively. HM and HMI exhibit similar trends with a small difference in $\mu s$. Looking

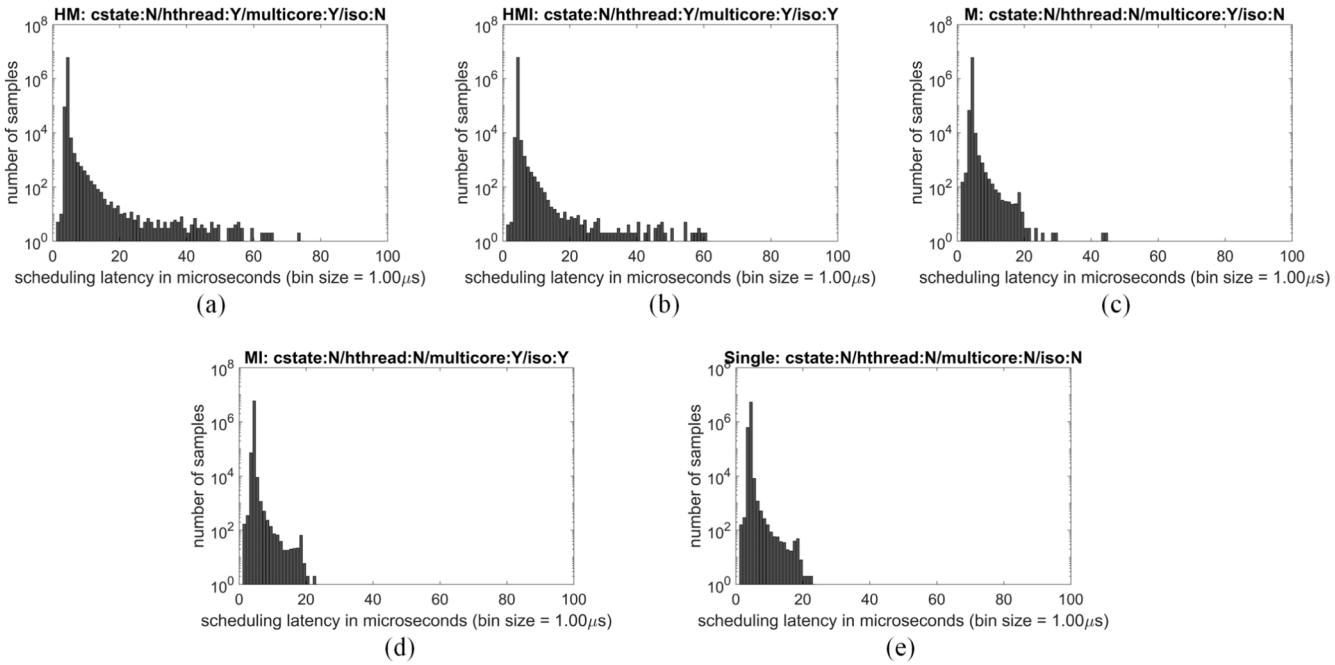


**FIGURE 1.** Histograms from the observed scheduling latencies on Xenomai in an idle environment. (a) HM. (b) HMI. (c) M. (d) MI. (e) Single.

at the shapes of the histograms and the numerical analysis, we conclude that hyperthreading affects the performance of Xenomai as stated in [27]. Although the task is executed in the physical core, sharing resources with the logical cores contribute to unpredictable delays, causing the performance decay.

When the hyperthreading feature is disabled as in the M and MI deployments, the scheduling latency has distinctly improved across all analytics. The results are also remarkably similar with that of the Single deployment, which should have displayed the best results in accordance with [21], [26]. As a result of the small difference between M and MI, further statistical analysis is necessary to determine the best multicore deployment in Xenomai.

### 2) RT-PREEMPT

In the case of RT-Preempt, no specific calibration is required owing to the implementation of standard POSIX libraries. The experiments were conducted with the same real-time parameters and experimental conditions as in Xenomai. The resulting histograms are shown in Fig. 2 and the numerical analyses are shown in Table 3.

Looking at both the visual and numerical results, it is quite fascinating that RT-Preempt appears not to be significantly affected by the various multicore deployment. Contradicting the results of Xenomai, only a slight improvement occurred when hyperthreading is disabled as shown by the results of HM and M.

Rather, Isolated CPU has shown to have improved the scheduling latencies for both deployments with compelling standard deviations of HMI and MI. In this context, the best multicore deployment for RT-Preempt is, conspicuously, the deployment MI. In terms of the scheduling latency, these results contradict the studies in [25], [33], which have stated that the best deployment for a multicore processor is the uniprocessor emulation.

**TABLE 3.** Scheduling latencies of RT-Preempt in various multicore deployments (in microseconds).

| Deployment | Ave. | Max. | Min. | $\mu$ |
|---|---|---|---|---|
| HM | 2.061 | 66.354 | 1.670 | 0.342 |
| HMI | 2.046 | 65.692 | 1.564 | 0.248 |
| M | 2.075 | 61.487 | 1.446 | 0.338 |
| MI | 2.040 | 50.541 | 1.421 | 0.202 |
| Single | 2.071 | 66.101 | 1.445 | 0.331 |

From the experiment results for both real-time extensions, we conclude that the best multicore deployment is MI, where the multicore feature is enabled, and real-time tasks are executed on an isolated core. RT-Preempt shows that it has a better scheduling latency with an average of 2.040 $\mu s$, which is approximately 0.5 $\mu s$ faster than that of Xenomai. In this instance, Xenomai shows superior results in terms of the maximum and standard deviation. Note that this difference is small, and the performance should be tested under the influence of interfering load to guarantee that the real-time system is applicable to practical applications.

As this study is the first of its kind, the results are especially important for selecting between the two real-time Linux extensions. However, our goal is not only to compare their performance but also to present a guideline for developers aiming to integrate real-time applications with multicore processors.

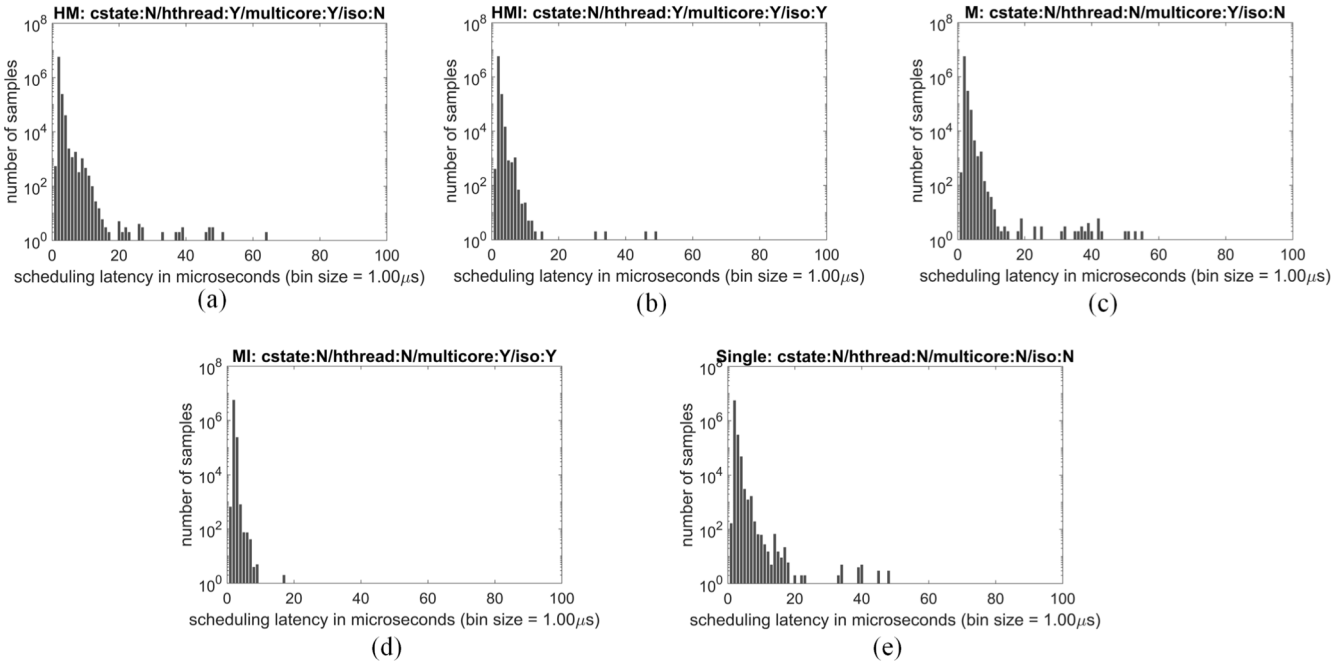


**FIGURE 2.** Histograms from the observed scheduling latencies on RT-Preempt in an idle environment. (a) HM. (b) HMI. (c) M. (d) MI. (e) Single.

## IV. STATISTICAL SELECTION METHOD

Due to the stochastic behavior of the real-time scheduler, comparison of the averages and extreme values of the scheduling latencies may not yield meaningful results to determine the best multicore deployment, as evident from the results of Xenomai in the previous section.

Herein, we define the best multicore deployment as the distribution that has the lowest statistical error and ''fits'' the performance of a uniprocessor system. The proposed statistical analysis focuses on the comparison of the acquired scheduling latencies from the previous section. To the best of our knowledge, this study is the first to highlight the effects of multicore deployments on the real-time performance of the system and to select the best multicore deployment.

In contrast to the results of [25], [33] for RT-Preempt, it is numerically and visually easy to determine that MI is the best multicore deployment in terms of the scheduling latency. For Xenomai, worst results were acquired from the deployments HM and HMI; however, the results from deployments M and MI produced similar extreme values and standard deviations, which require further statistical analysis to determine the best multicore deployment. In this context, we are more focused on finding the best multicore deployment for Xenomai.

Fig. 3 illustrates the quantile-to-quantile (Q-Q) plot of the various multicore deployments against the uniprocessor (Single) deployment. Herein, we could clearly see that deployments HM, HMI, and M do not belong on the same distribution with Single. Each of their distributions did not produce a straight line coincident with the calculated linear regression. Conversely, MI and Single came from the same distribution as evident in Fig. 3(d). Thus, we could assume that MI is the best multicore deployment for Xenomai.

To prove this assumption, we conducted statistical analysis on the distributions from all multicore deployments. We start by stating the null hypothesis, which implies that the measured scheduling latencies of a multicore deployment are from the same distribution with the results from the Single deployment. The alternate hypothesis is that they are not from the same distribution.

Hence, we define the results from the Single deployment as the expected values, with the scheduling latencies from the other deployments as the observed values.

The *cyclictest* produced a histogram with the distribution of the measured scheduling latencies. The bins were divided into an equal width of 1 $\mu s$ (refer to Fig. 1) to contain the measured scheduling latencies. As we are dealing with multiple histograms with different extreme values, we considered the maximum and minimum bin ranges equal to that of the Single histogram for all analyses.

To avoid zero-value frequencies, histograms from the multicore deployment were reshaped. Contents of bins that are greater than that of the maximum are transferred to the maximum bin, whereas all samples that are located on those lesser than the minimum, are placed on the minimum bin. Then, we performed $\chi^2$ goodness of fit test:

$$\chi_T^2 = \sum_{j=1}^{n} \frac{(O_j - E_j)^2}{E_j} \tag{3}$$

where $O$, the observed values, represents the respective multicore deployment, $n$ is the total number of samples and

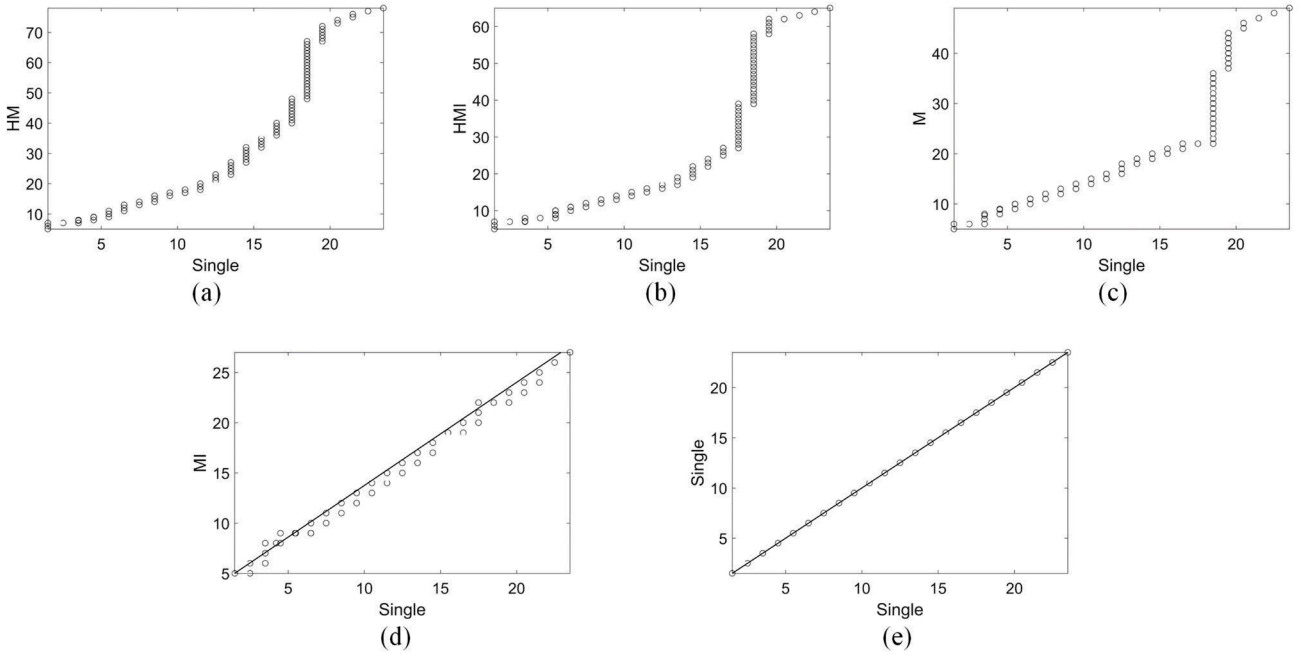


**FIGURE 3.** Quantile-to-quantile (Q-Q) plot of multicore deployments versus the Single deployment of Xenomai. (a) HM. (b) HMI. (c) M. (d) MI. (e) Single.

$E$ denotes the expected value, which are the results from the Single deployment. It is clear from this equation that a smaller $\chi_T^2$ means that the difference between the expected and observed values are smaller, signifying that both came from the same distribution.

We may choose a significance level, $\alpha$, to calculate the critical value $\chi_\alpha^2$. The null hypothesis is rejected when the calculated test statistic is greater than the critical value, or it is accepted otherwise. In the selection method, we determine the best multicore deployment to be the distribution which does not reject the null hypothesis and with the least $\chi_T^2$. As it is crucial that the scheduling latency remains low in a real-time system, we consider a one sided lower-tail $\chi^2$ test with a significance level of $\alpha = 0.05$, setting a threshold of 95% accuracy of the results. This ensures that the results are not determined by randomness. As we are directly comparing the histograms of the results, the critical value is calculated with the degrees of freedom (DOF) equal to the number of bins of the histogram from the Single deployment.

Note that the statistical analysis is conducted offline using Matlab to avoid any unwanted effects during measurement of the scheduling latencies. The steps in conducting the $\chi^2$ test and selection method are summarized below.

- Configure the maximum and minimum bin ranges as equal to values from the Single histogram;
- If needed, recreate the histograms from the various multicore deployments. Contents of bins that are greater than that of the maximum are transferred to the maximum bin. Whereas all samples that are located on those lesser than the minimum are placed on the minimum bin;
- Obtain the test statistic ($\rho$ value) with (3) and check whether it exceeds $\chi_T^2$, based on $\alpha$;
- Determine the multicore deployments that do not reject the null hypothesis and select the one with the least test statistic.

**TABLE 4.** Results of the $\chi^2$ tests with 95% level of significance For The Various Multicore Deployments of Xenomai.

| Deployment | DOF | $\chi^2_{(\alpha=0.05)}$ | $\rho$ value | Reject? |
|---|---|---|---|---|
| HM | 22 | 33.924 | 55.98 | Y |
| HMI | 22 | 33.924 | 65.12 | Y |
| M | 22 | 33.924 | 37.15 | Y |
| MI | 22 | 33.924 | 23.26 | N |
| Single | 22 | 33.924 | 0 | N |

Following these steps, we analyze the scheduling latencies from the multicore deployments of Xenomai and the results are shown in Table 4.

In Fig. 1, the Single deployment distribution produced a histogram with a total number of 22 bins ($n = 22$), which results to a critical value of $\chi_T^2 = 33.924$.

In the table, the calculated $\rho$ value for the deployments HM, HMI, and M is greater than that of the critical limit. Thus, these deployments reject the null hypothesis which meaning that they did not come from the same distribution as the Single distribution. As for M and MI, although there were trivial differences measured between the extreme values and standard deviation, we can see in these results that only MI satisfies the null hypothesis when compared with Single. Thus, we have statistically proven the Q-Q plot in Fig. 3 and

**TABLE 5. Scheduling latency of The selected Multicore deployments for each RT linux extension under interfering best-effort load (in microseconds).**

| RT Linux Extension | Ave. | Max. | Min. | $\mu$ |
|---|---|---|---|---|
| Xenomai | 2.779 | 71 | 1.503 | 1.216 |
| RT-Preempt | 2.504 | 69 | 1 | 0.594 |

determined that the best multicore deployment for Xenomai is indeed the MI multicore deployment.

## V. EFFECTS OF INTERFERING LOAD AND FEASIBILITY ANALYSIS

To build confidence that both real-time extensions of Linux on a multicore processor are suitable for practical real-time applications, we have measured the scheduling latencies of each extension under the effects of interfering loads. In this instance, we consider three best-effort loads: CPU, memory, and network operations.

The experiments were conducted on the best multicore deployment determined in the previous section (MI for both Xenomai and RT-Preempt), following the same procedures in Section III. The real-time task has a period of 100 $\mu s$, acquiring a total number of 6,000,000 samples for a runtime time of 10 minutes. The three interfering loads being considered correspond to the following: CPU and memory load utilizing *stress-ng* tool [40], and network stress with *iperf* [41].

The *stress-ng* was configured to run multiple floating-point arithmetic operations in a very tight infinite loop to simulate loads fully utilizing the CPU. We have selected to sequentially exercise all methods available within the *stress-ng* tool. Virtual memory stress test is also performed with 5 virtual memory stressors. Due to the memory requirement for storing observed values, the memory stress can only occupy 70% of the system memory during the experiment.

In order to simulate network operations, we implemented a multi-node environment emulating a practical scenario, where a server should attend to asynchronous data requests from multiple clients. In this instance, the real-time system serves as the server connected a remote-load generator (another PC). In the load generator, we simulated 100 clients with each requesting 64 KB of data, resulting in a throughput of 1 Mbps/client. The histogram of the scheduling latencies under interfering loads for both Xenomai and RT-Preempt are illustrated in Fig. 4.

Table 5 lists the corresponding summary statistics. From these data, we can interpret that RT-Preempt, with lower average, extreme values, and standard deviation, should show better deterministic timings in comparison to Xenomai. However, these results only correspond to the scheduling latencies of a single real-time task. To furtherly explore on the feasibility of the best multicore deployments in both real-time Linux extensions, evaluation of the periodicity and deterministic response times of the system in a multitasking environment is required. To analyze the timeliness and responsiveness with

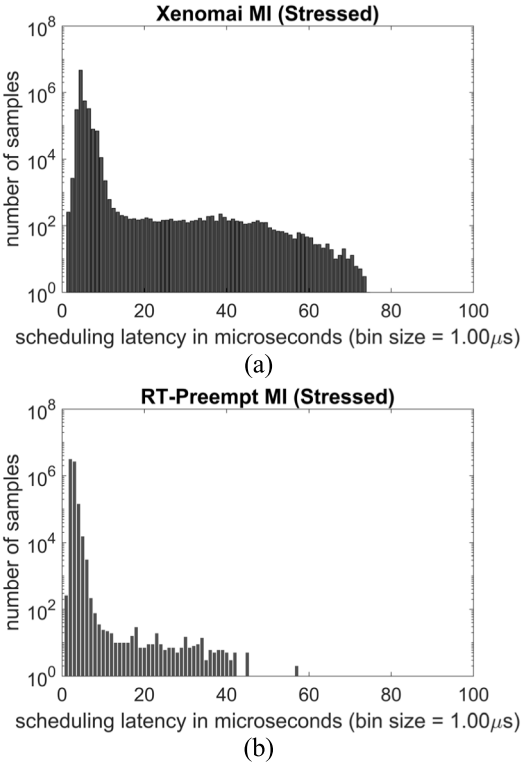


**FIGURE 4. Histograms of the observed scheduling latencies in the presence of CPU, memory, and network bound best-effort workload. (a) Xenomai. (b) RT-Preempt.**

**TABLE 6. Task parameters for the feasibility analysis (in milliseconds).**

| $T$ | $P$ | $D$ | $C$ | $pr$ |
|---|---|---|---|---|
| $\tau_1$ | 10 | 10 | 3 | 99 |
| $\tau_2$ | 20 | 20 | 5 | 80 |
| $\tau_3$ | 40 | 40 | 10 | 50 |

RMA, we consider a harmonic taskset, where each of the period ($P$) is an integral multiple (or sub-multiple) of all other periods within the taskset. The deadline ($D$) of each task is equal to its respective period and the worst-execution time ($C$) is the maximum computation time. Also, the priorities ($pr$) are assigned so that the task with the shortest period has the highest priority (rate-monotonic). In particular, $\tau_1$ has the shortest period of 10 *ms* and the highest period of 99 (specified as the maximum priority for both Xenomai and RT-Preempt). On the other hand, $\tau_3$ which has the longest cycle of 40 *ms*, is assigned the lowest priority of 50. The taskset, together with the task parameters are specified in Table 6.

With the assumption that both Xenomai and RT-Preempt have been successfully implemented, each task should have a blocking time of zero, as long as they do not share a resource locking mechanism. Fig. 5 illustrates the timeline of the expected execution of the taskset in one *hyperperiod* (least common multiple of all periods).

When all of the tasks have the same activation point, from (1), the WCRT of each task are calculated as $R_1 = 3\,ms$, $R_2 = 8\,ms$, and $R_3 = 29$ ms, respectively for $\tau_1$, $\tau_2$, and $\tau_3$.

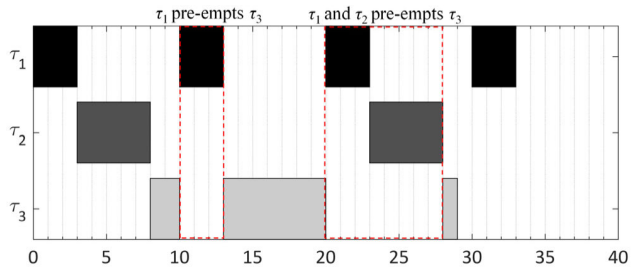


**FIGURE 5.** Execution timeline of a multitasking environment with three real-time tasks scheduled with pre-emption (in milliseconds): $\tau_1$, with execution time of 3 ms and period of 10 ms, $\tau_2$ with execution time of 5 ms and period of 20 ms, and $\tau_3$ with execution of 10 ms and period of 40 ms.

Because the highest priority task ($\tau_1$) does not experience any interference during its execution, the calculated response time is equal to its execution time. $\tau_2$, on the other hand, runs whenever $\tau_1$ completed its execution. We have observed that the task finishes its execution without pre-emption. However, this is not the same for $\tau_3$. As expected, the lowest priority task is pre-empted (red dashed line) twice by $\tau_1$ and once by $\tau_2$, resulting to the longest WCRT.

To assess the real-time performance of the best multicore deployments of both Xenomai and RT-Preempt, we expect to measure actual WCRT that are approximately equal to the calculated values. In such, three periodic tasks are created on each real-time extension with the parameters specified in Table 6. Priorities and the periods are configured before the tasks enter their respective infinite task loop and the tasks are configured to run only on the first available core.

Within the task loop, each task performs measurements of three performance metrics: actual period, jitter, and actual response time. A busy waiting loop is realized to simulate computational load, which burns CPU resources for 1 *ms* and iterates continuously until the task has completed its required execution time. The pseudo code for the task loop, which is identical for the three real-time tasks is depicted in Algorithm 2.

The experiments are conducted on each real-time Linux extension with a considerable runtime of 10 minutes in idle environment. Similar with the previous experiments, all the measured values were stored in a buffer to prevent any overheads that can affect data integrity. Timing analysis are performed offline using Matlab and the results are summarized in Table 7.

From the experimental results, it is clearly observed that the measured WCRT (maximum response time) for the tasks on both Xenomai and RT-Preempt show good accordance with the expected values.

Regarding the response times, Xenomai has shown lower average, maximum and standard deviations; however, the differences are less than 10 $\mu s$, which are exceedingly small and negligible. In terms of periodicity, better results across all tasks and metrics were observed with RT-Preempt. This is expected considering the direct relationship of scheduling latency with periodicity [32].

**Algorithm 2** Real-time Task Loop

```
1:  set_task_periodic(P);
2:  set_task_priority(pr);
3:  task_exec = C;
4:  prev_period = read_tsc(); // read timestamp counters
5:  while (1)
6:  {
7:    wait_for_next_period();
8:    runtime = 0;
9:    curr_period = read_tsc();
10:
11:   while (runtime < task_exec) // busy waiting loop
12:   {
13:     end = read_tsc() + 1000000; // 1 millisecond
14:     while (read_tsc() < end)
15:     __sync_syncrhonize(); // memory barrier
16:
17:     runtime++;
18:   }
19:   response_end = read_tsc();
20:
21:   period = curr_period - prev_period;
22:   response = response_end - curr_period;
23:   jitter = abs(P - period);
24:
25:   prev_period = curr_period;
26:  }
```

**TABLE 7.** Periodicity and response times of multiple tasks on the best multicore deployment in an idle environment (in milliseconds).

| $\tau_1$ | Xenomai | | | RT-Preempt | | |
|---|---|---|---|---|---|---|
| | $P$ | $R$ | $J$ | $P$ | $R$ | $J$ |
| Ave. | 10.000 | 3.015 | 0.014 | 10.000 | 3.022 | 0.001 |
| Max. | 10.069 | 3.125 | 0.068 | 10.061 | 3.105 | 0.061 |
| Min. | 9.954 | 3.004 | 0.000 | 9.942 | 3.003 | 0.000 |
| $\mu$ | 0.017 | 0.014 | 0.013 | 0.002 | 0.019 | 0.001 |
| $\tau_2$ | | | | | | |
| Ave. | 20.000 | 8.006 | 0.056 | 20.000 | 8.007 | 0.042 |
| Max. | 20.110 | 8.057 | 2.533 | 21.349 | 8.063 | 1.248 |
| Min. | 17.085 | 8.006 | 0.000 | 19.900 | 8.005 | 0.000 |
| $\mu$ | 0.057 | 0.001 | 0.010 | 0.043 | 0.003 | 0.007 |
| $\tau_3$ | | | | | | |
| Ave. | 40.000 | 29.039 | 0.002 | 40.000 | 29.042 | 0.003 |
| Max. | 40.077 | 29.084 | 0.077 | 41.422 | 29.092 | 1.383 |
| Min. | 39.925 | 26.063 | 0.000 | 39.930 | 28.015 | 0.000 |
| $\mu$ | 0.015 | 0.014 | 0.014 | 0.010 | 0.052 | 0.010 |

To evaluate the behavior under interfering loads, the same experiment procedures are conducted with the following stress conditions: *stress-ng* utilizing 100% of the CPU and 70% of system memory [41], and *iperf* providing an average throughput of 1 Mbps/client for 100 simulated socket clients [42]. Table 8 shows the results of the experiments in a stressed environment.

Even in the presence of best-effort interfering loads, we have observed no significant performance degradation from the measurements. Following the same trend as in the

**TABLE 8. Periodicity and response times of multiple tasks on the best multicore deployment in a stressed environment (in milliseconds).**

| $\tau_1$ | Xenomai | | | RT-Preempt | | |
|---|---|---|---|---|---|---|
| | $P$ | $R$ | $J$ | $P$ | $R$ | $J$ |
| Ave. | 10.000 | 3.016 | 0.014 | 10.000 | 3.033 | 0.002 |
| Max. | 10.071 | 3.127 | 0.071 | 10.240 | 3.159 | 0.234 |
| Min. | 9.956 | 3.004 | 0.000 | 9.814 | 3.004 | 0.000 |
| $\mu$ | 0.019 | 0.014 | 0.013 | 0.004 | 0.030 | 0.003 |
| $\tau_2$ | | | | | | |
| Ave. | 20.000 | 8.006 | 0.059 | 20.000 | 8.011 | 0.061 |
| Max. | 20.111 | 8.100 | 2.914 | 21.248 | 8.090 | 1.349 |
| Min. | 17.467 | 8.006 | 0.001 | 19.842 | 8.006 | 0.000 |
| $\mu$ | 0.060 | 0.002 | 0.012 | 0.053 | 0.006 | 0.017 |
| $\tau_3$ | | | | | | |
| Ave. | 40.000 | 29.060 | 0.004 | 40.000 | 29.069 | 0.007 |
| Max. | 40.123 | 29.111 | 0.123 | 41.383 | 29.230 | 1.422 |
| Min. | 39.890 | 26.061 | 0.000 | 39.816 | 28.023 | 0.000 |
| $\mu$ | 0.016 | 0.015 | 0.014 | 0.014 | 0.054 | 0.012 |

idle environment, Xenomai has slightly better response times, and RT-Preempt with finer periodicity. This behavior offers a new perspective on the performance of real-time extensions of Linux in contrast to the results presented in [23]–[25].

It is worth to note that the real-time performance may vary depending on the software versions and the underlying hardware system. This is one of the consequences of open source-based systems. In this article, however, we are more focused on the effects of the various multicore deployments on the real-time performance of the system, rather than the comparison between the real-time extensions.

Thus, in our developed system, we therefore conclude that with the best multicore deployment, Xenomai and RT-Preempt should be able to satisfy real-time tasks with a minimum period of 100 $\mu s$ as proven by the results of scheduling latency experiments, even under the effects of interfering loads. Also, real-time constraints, in terms of periodicity and deterministic response times, are satisfied in multitasking environment. These results would build confidence that the real-time extensions of Linux, configured to their respective best multicore deployment, are feasible in practical real-time applications.

## VI. CONCLUSION AND DISCUSSION

This paper presented new insights into real-time performance of real-time extensions of Linux, namely Xenomai and RT-Preempt, on a homogeneous multicore processor.

We provide the complete details on leveraging each real-time extension on an Intel-based multicore embedded system considering the necessary Linux kernel configuration. We described a set of multicore deployments contingent on several processor features such as C-State, hyperthreading, multicore, and CPU isolation. The effects of each multicore deployment on the real-time performance was evaluated for each real-time Linux extension. Performance evaluation was focused on the scheduling latency, owing to its apparent relationship with both the periodicity and WCRT of real-time tasks. The results of the experiment showed that disabling the hyperthreading features (logical cores) of a multicore system greatly improves real-time performance.

We have observed that it is relatively easy to determine the best multicore deployment for RT-Preempt based on the statistical summary and the visual representation of the scheduling latencies as shown in Fig. 2 and Table 3. These results contradict the prior studies presented in [25], [33], and thus highlighting the importance of this study in integrating real-time applications based on RT-Preempt on multicore processors.

However, in Xenomai, further analysis is required to select the best deployment because of the small difference between scheduling latencies of the multicore deployments, especially M and MI. Assuming that emulating a uniprocessor system is the best possible deployment as stated in [25], [33], we have presented a selection method based on a variation of $\chi^2$ goodness-of-fit test. The actual best multicore deployment is determined by comparing the scheduling latencies of each multicore deployment with that of the single core. We have found out that the Xenomai system has also shown the best results when the hyperthreading features is disabled, proving the results of a prior study in [34].

After the best multicore deployment was determined for both real-time extensions, we have evaluated them under interfering loads, such as CPU, memory, and network operations. This step was performed to build confidence in the suitability of the real-time systems on multicore processors for practical real-time applications. Feasibility analysis on a multitasking environment was also performed by analyzing the periodicity and deterministic responses of three periodic tasks that exhibits pre-emptive behavior. In contrast to prior studies [23]–[25], our results showed that RT-Preempt has better periodicity and scheduling latencies over Xenomai, even in the presence of best-effort interfering loads.

This counterintuitive behavior may be attributed to several reasons, such as the different hardware platform and software versions. As we are more focused on the effects of multicore deployments on the real-time performance, analysis of these differences is beyond the scope of this article.

We concluded that the best multicore deployments of both Xenomai and RT-Preempt should be able to satisfy real-time tasks with a minimum period of 100 $\mu s$, even under the effects of interfering loads. We have also proven that both systems were able to satisfy real-time constraints in a multitasking environment, in terms of periodicity and deterministic response times. These results are particularly important since they are the first of their kind and will serve as a guideline for real-time developers who require integration of real-time applications on multicore processors.

There are several directions for the future work. More evaluation on larger platforms and other architectures will be conducted. Based on our findings, CPU power management schemes and hyperthreading features are the main source of performance degradation on an Intel multicore system, we assume that the same behavior can be expected from other homogeneous multicore processors as well. More significantly, we hope to evaluate the real-time performance of a multiprocessor system on chip (MPSoC). Although we

have conceptually reported its expected performance [42], the real-time scheduler should be improved to consider the heterogeneity of such system. Also, refined schedulability analysis is required to evaluate the response times of real-time tasks, considering resource sharing between multicores and multiprocessors. Finally, automation of developing a real-time working environment and evaluation of real-time Linux-based systems are being considered.

## REFERENCES

[1] A. Biondi and M. D. Natale, "Achieving predictable multicore execution of automotive applications using the LET paradigm," in *Proc. IEEE Real-Time Embedded Technol. Appl. Symp. (RTAS)*, Porto, Portugal, Apr. 2018, pp. 240–250.
[2] H. Wei, Z. Shao, Z. Huang, R. Chen, Y. Guan, J. Tan, and Z. Shao, "RT-ROS: A real-time ROS architecture on multi-core processors," *Future Gener. Comput. Syst.*, vol. 56, pp. 171–178, Mar. 2016.
[3] C. E. Tuncali, G. Fainekos, and Y.-H. Lee, "Automatic parallelization of multirate block diagrams of control systems on multicore platforms," *ACM Trans. Embedded Comput. Syst.*, vol. 16, no. 1, pp. 1–26, Nov. 2016.
[4] G. Kornaros, *Multi-Core Embedded Systems*. Boca Raton, FL, USA: CRC Press, 2010.
[5] J. Chen, C. Du, F. Xie, and B. Lin, "Scheduling non-preemptive tasks with strict periods in multi-core real-time systems," *J. Syst. Archit.*, vol. 90, pp. 72–84, Oct. 2018.
[6] H. Chniter, O. Mosbahi, M. Khalgui, M. Zhou, and Z. Li, "Improved multi-core real-time task scheduling of reconfigurable systems with energy constraints," *IEEE Access*, vol. 8, pp. 95698–95713, 2020.
[7] J. Chen, C. Du, P. Han, and Y. Zhang, "Sensitivity analysis of strictly periodic tasks in multi-core real-time systems," *IEEE Access*, vol. 7, pp. 135005–135022, 2019.
[8] S. Moulik, R. Devaraj, and A. Sarkar, "HETERO-SCHED: A low-overhead heterogeneous multi-core scheduler for real-time periodic tasks," in *Proc. IEEE 20th Int. Conf. High Perform. Comput. Commun.; IEEE 16th Int. Conf. Smart City; IEEE 4th Int. Conf. Data Sci. Syst. (HPCC/SmartCity/DSS)*, Jun. 2018, pp. 659–666.
[9] G. Levin, S. Funk, C. Sadowski, I. Pye, and S. Brandt, "DP-FAIR: A simple model for understanding optimal multiprocessor scheduling," in *Proc. 22nd Euromicro Conf. Real-Time Syst.*, Jul. 2010, pp. 3–13.
[10] R. I. Davis and A. Burns, "A survey of hard real-time scheduling for multiprocessor systems," *ACM Comput. Surv.*, vol. 43, no. 4, pp. 1–44, Oct. 2011.
[11] J. Liu, X. Gao, B. Jiang, S. Yang, and Z. Zhang, "Deterministic replay for multi-core VxWorks applications," in *Proc. Int. Conf. Dependable Syst. Appl. (DSA)*, Oct. 2017, pp. 118–125.
[12] *Nucleus RTOS*, Mentor Graph., Wilsonville, OR USA, Oct. 2009.
[13] J. Wetzels and A. Abbasi, "Dissecting QNX," in *Proc. Blackhat Briefings*, Singapore, Mar. 2018, pp. 1–22.
[14] W. Wang, Y. Wang, J. Dai, and Z. Cao, "Dynamic soft real-time scheduling with preemption threshold for streaming media," *Int. J. Digit. Multimedia Broadcast.*, vol. 2019, Jan. 2019, Art. no. 5284968.
[15] H. Salman, M. N. Uddin, S. Acheampong, and H. Xu, "Design and implementation of iot based class attendance monitoring system using computer vision and embedded Linux platform," in *Proc. Workshops Int. Conf. Adv. Inf. Netw. Appl.*, 2019, pp. 25–34.
[16] J. J. Roldán, E. Peña-Tapia, D. Garzón-Ramos, J. de León, M. Garzón, J. del Cerro, and A. Barrientos, "Multi-robot systems, virtual reality and ROS: Developing a new generation of operator interfaces," in *Robot Operating System (ROS): The Complete Reference*, vol. 3, A. Koubaa, ed. Cham, Switzerland: Springer, 2019, pp. 29–64
[17] P. Gerum, "Life with adeos," Xenomai, Munich, Germany, Tech. Rep., 2005. Accessed: Aug. 23, 2020. [Online]. Available: https://xenomai.org/documentation/xenomai-2.4/pdf/ Life-with-Adeos-rev-B.pdf https://xenomai.org/documentation/branches/v2.3.x/pdf/xenomai.pdf
[18] P. Gerum, "Xenomai-implementing a RTOS emulation framework on GNU/Linux," Xenomai, Munich, Germany, White Paper, 2004, pp. 1–12. Accessed: Aug. 30, 2020. [Online]. Available: https://xenomai.org/documentation/branches/v2.3.x/pdf/xenomai.pdf
[19] D. Beal *et al.*, "RTAI: Real-time application interface," *Linux J.*, vol. 29, no. 72, p. 10, 2000.
[20] D. B. de Oliveira and R. S. de Oliveira, "Timing analysis of the PREEMPT RT Linux Kernel," *Softw., Pract. Exp.*, vol. 46, no. 6, pp. 789–819, Jun. 2016.
[21] F. Reghenzani, G. Massari, and W. Fornaciari, "The real-time Linux Kernel: A survey on preempt_rt," *ACM Comput. Surv. (CSUR)*, vol. 52, no. 1, pp. 1–36, 2019.
[22] J. Park, R. Delgado, and B. W. Choi, "Real-time characteristics of ROS 2.0 in multiagent robot systems: An empirical study," *IEEE Access*, vol. 8, pp. 154637–154651, 2020.
[23] J. H. Brown and B. Martin, "How fast is fast enough? Choosing between Xenomai and Linux for real-time applications," in *Proc. 12th Real-Time Linux Workshop*, 2010, pp. 1–17.
[24] A. Barbalace, A. Luchetta, G. Manduchi, M. Moro, A. Soppelsa, and C. Taliercio, "Performance comparison of VxWorks, Linux, RTAI, and xenomai in a hard real-time application," *IEEE Trans. Nucl. Sci.*, vol. 55, no. 1, pp. 435–439, 2008.
[25] M. Cereia, I. C. Bertolotti, and S. Scanzio, "Performance of a real-time EtherCAT master under Linux," *IEEE Trans. Ind. Informat.*, vol. 7, no. 4, pp. 679–687, Nov. 2011.
[26] J. M. Calandrino, H. Leontyev, A. Block, U. C. Devi, and J. H. Anderson, "LITMUSRT: A testbed for empirically comparing real-time multiprocessor schedulers," in *Proc. 27th IEEE Int. Real-Time Syst. Symp.*, Dec. 2006, pp. 111–126.
[27] Y. Saito, F. Sato, T. Azumi, S. Kato, and N. Nishio, "ROSCH: Real-time scheduling framework for ROS," in *Proc. IEEE 24th Int. Conf. Embedded Real-Time Comput. Syst. Appl. (RTCSA)*, Aug. 2018, pp. 52–58.
[28] T. Gupta, E. J. Luit, M. M. H. P. V. D. Heuvel, and Re. J. Bril, "Extending ExSched with mixed criticality support—An experience report," in *Proc. IEEE Int. Conf. Softw. Archit. Workshops (ICSAW)*, Apr. 2017, pp. 23–28.
[29] F. Cerqueira and B. Brandenburg, "A comparison of scheduling latency in Linux, PREEMPT-RT, and LITMUS RT," in *Proc. 9th Annu. Workshop Operating Syst. Platforms Embedded Real-Time Appl.*, 2013, pp. 19–29.
[30] J.-J. Han, S. Gong, Z. Wang, W. Cai, D. Zhu, and L. T. Yang, "Blocking-aware partitioned real-time scheduling for uniform heterogeneous multicore platforms," *ACM Trans. Embedded Comput. Syst.*, vol. 19, no. 1, pp. 1–25, Feb. 2020.
[31] J. Lelli, C. Scordino, L. Abeni, and D. Faggioli, "Deadline scheduling in the Linux Kernel," *Softw., Pract. Exp.*, vol. 46, no. 6, pp. 821–839, Jun. 2016.
[32] G. C. Buttazzo, "Rate monotonic vs. EDF: Judgment day," *Real-Time Syst.*, vol. 29, no. 1, pp. 5–26, Jan. 2005.
[33] W. Betz, M. Cereia, and I. C. Bertolotti, "Experimental evaluation of the Linux RT patch for real-time applications," in *Proc. IEEE Conf. Emerg. Technol. Factory Autom.*, Palma de Mallorca, Spain, Sep. 2009, pp. 598–601.
[34] C. Garre, "Performance comparison of real-time and general-purpose operating systems in parallel physical simulation with high computational cost," SAE Tech. Paper 2014-01-0200, 2014.
[35] E. Rotem, A. Naveh, A. Ananthakrishnan, E. Weissmann, and D. Rajwan, "Power-management architecture of the intel microarchitecture code-named sandy bridge," *IEEE Micro*, vol. 32, no. 2, pp. 20–27, Mar. 2012.
[36] R. Souza, M. Freitas, M. Jimenez, J. Magalhães, A. C. Kubrusly, and N. Rodriguez, "Real-time performance assessment using fast interrupt request on a standard Linux Kernel," *Eng. Rep.*, vol. 2, no. 1, Jan. 2020, Art. no. e12114.
[37] M. Joseph and P. Pandya, "Finding response times in a real-time system," *Comput. J.*, vol. 29, no. 5, pp. 390–395, May 1986.
[38] R. Delgado and B. Choi, "Network-oriented real-time embedded system considering synchronous joint space motion for an omnidirectional mobile robot," *Electronics*, vol. 8, no. 3, p. 317, Mar. 2019.
[39] R. Delgado, B.-J. You, and B. W. Choi, "Real-time control architecture based on xenomai using ROS packages for a service robot," *J. Syst. Softw.*, vol. 151, pp. 8–19, May 2019.
[40] C. I. J. U. H. K. U. C. G. C. S. G. King. (2017). *Stress-Ng*. Accessed: Aug. 23, 2020. [Online]. Available: https://kernel.ubuntu.com/~cking/stress-ng/
[41] V. J. I. F. N. Gueant. (2017). *iPerf—The TCP, UDP and SCTP Network Bandwidth Measurement Tool*. Accessed: Aug. 23, 2020. [Online]. Available: https://iperf.fr/
[42] R. Delgado, J. Park, and B. W. Choi, "MPSoC: The low-cost approach to real-time hardware simulations for power and energy systems," *IFAC-PapersOnLine*, vol. 52, no. 4, pp. 57–62, 2019.

**RAIMARIUS DELGADO** (Member, IEEE) received the B.S. and M.S. degrees in electrical and information engineering from the Seoul National University of Science and Technology, Seoul, South Korea, in 2014 and 2016, respectively, where he is currently pursuing the Ph.D. degree under the supervision of Prof. Byoung Wook Choi. His research interests include real-time systems, industrial control and automation, embedded systems, systems and software architecture, and service robotics.

**BYOUNG WOOK CHOI** (Member, IEEE) received the M.S. and Ph.D. degrees in electrical engineering from the Korea Advanced Institute of Science and Technology (KAIST), Seoul, South Korea, in 1988 and 1992, respectively. He was a Principal Research Engineer with LG Industrial Systems, from 1992 to 2000, and a Professor with Sun Moon University, from 2000 to 2005. He was the CEO of Embedded Web Company Ltd., from 2001 to 2003. He was a Senior Fellow with Nanyang Technological University, Singapore, from 2007 to 2008. He is currently a Professor with the Department of Electrical and Information Engineering, Seoul National University of Science and Technology. He has published textbooks on embedded Linux. His current research interests include real-time systems design, embedded systems, and intelligent robot software.