A Reservation Scheduler for Real-Time Operating Systems

David Matschulat, César A. M. Marcon, Fabiano Hessel

PPGCC - FACIN – PUCRS - Av. Ipiranga, 6681, Porto Alegre, RS – Brazil

david@inf.pucrs.br, {Cesar.Marcon, Fabiano.Hessel}@pucrs.br

***Abstract**: The increasing demand for embedded multimedia applications makes evident the need for end-to-end Quality of Service (QoS) provisioning. Particularly, operating systems, despite their location at end systems, switches or routers, must guarantee that resources under their control are adequately managed to fulfill the application requirements. This work proposes the implementation of QoS provisioning in real-time embedded systems scheduler. In order to achieve the end-to-end QoS, we propose the implementation of the control and management of QoS mechanisms in the operating system scheduler. The implementation of such mechanisms includes admission control and resource reservation, as well as process scheduling control and active monitoring of the delivered QoS. As a result, a new scheduling algorithm, named ER-EDF, is proposed and compared to previous scheduler solutions. This approach was validated through a set of benchmarks and we conclude that ER-EDF adds performance and simplified hard real-time support to real-time embedded applications.*

1 Introduction

In the last years, there has been an increase demand for hardware/software platforms with multimedia application support. These platforms are increasingly distributed, real-time, embedded, and must operate under highly unpredictable and changeable conditions.

In order to provide end-to-end Quality of Service (QoS), required by multimedia applications, resource management on the whole operation environment is needed. Indeed, QoS provisioning requires the implementation of several tasks both in the end-systems and in the communication provider, including its switches and routers. In the end-systems, resources controlled by the operating system, like CPU, memory and communication buffers, must be adequately managed to ensure that the interaction of various applications will not cause individual QoS violations.

QoS provisioning has become even harder since new requirements, imposed by new types of multimedia application and new codification techniques have emerged. In fact, the rapid and inexpensive deployment of services with new QoS requirements has become essential to embedded multimedia applications.

Real-Time Operating Systems (RTOS) services and mechanisms (e.g. scheduling) with QoS support emerged to provide predictability to the critical systems. However, the current generation of commercial-off-the-shelf RTOS schedulers lacks adequate support for applications with stringent QoS requirements. Since processing and

communication requirements are distinct for each media type, different QoS guarantees are necessaries to maintain synchronization characteristics, temporal constraints, and reliability, among others, of an application.

The computing infrastructure for these systems must be sufficiently flexible to support workload variation at different times during an application lifecycle, yet maintain highly predictable and dependable behavior. Controlling the real-time behavior of such embedded systems is one important dimension of the delivered QoS.

The recent focus on user control over QoS aspects stems from technology advances in historically challenging research areas, such as allocation policies, synchronization of streams in embedded multimedia applications, and assured communication in the face of high demand. The focus on QoS aspects has led to the development of a number of proposed and implemented improvements to commonly available embedded computing infrastructures. When coupled with embedded software that can recognize and react to environmental changes, these improvements form the basis for constructing appropriate adaptive behavior for next-generation embedded real-time systems.

Two main phases can be identified during the QoS provisioning: negotiation and tuning. The *QoS negotiation* phase involves mechanisms responsible for task (or jobs) admission control. An admission characterizes the establishment of a service contract (or service agreement) between a task and the QoS provisioning environment. During the service offering, both sides can break the previously negotiated contract. The task may not respect anymore its initial load and the environment can be no more able to maintain the service level agreement, since resources are dynamically shared. The *QoS tuning* phase provides mechanisms responsible for monitoring the flow load and the QoS really offered to the application tasks. In case of contract violation, from any side, it should fire actions to reestablish the QoS negotiated level.

The specification of QoS services can involve the choice of scheduling, admission and classification algorithms, as well as other configuration parameters, such as tasks that will be part of the communication protocol stack or the description of the system initial state for the QoS provisioning (e.g. initial partitioning of resources for each application class). For this sake, diverse high-level adaptability abstractions have been proposed [1] (e.g. reflection, open signaling, active networks, etc). These abstractions usually rely on switches and end systems that can be explicitly programmed during communication infrastructure operation, demanding, therefore, an operating system with sufficient flexibility. Nonetheless, the variety of available embedded RTOS (eRTOSs) hampers the deployment of such high-level adaptability abstractions. For example, the most used scheduling algorithm for embedded real-time applications, which is the Earliest Deadline First (EDF) was not conceived to offer guarantees and has an unpredictable behavior when the system is overloaded [2], thus not providing predictability of execution.

The key issues of this paper is to discuss and propose an adequate support for QoS provisioning and service adaptability that can be built in a general purpose eRTOS. In this sense, we present a new scheduling algorithm for QoS provisioning on eRTOS, named ER-EDF. In this paper, we mainly focus on the QoS provisioning for hard real-time tasks.

In order to validate the proposed approach, three algorithms were implemented in an embedded operating system: EDF, *Reservation* EDF (R-EDF) and *Enhanced* R-EDF

(ER-EDF). The main issue of the ER-EDF is the performance enhancements and support for processing reservation for hard real-time tasks, as we will demonstrate in the experimental results.

The remaining of this paper is organized as follows. Section 2 presents the related work. Next, in Section 3, the basic concepts of job and task model are explained. Section 4 presents the R-EDF algorithm and its limitations. Section 5 presents the new algorithm ER-EDF. Section 6 shows the implementation and experiments created to validate the proposed algorithm. Finally, Section 7 concludes this work.

2 Related work

Resource reservation is a common mechanism to provide separation between real-time applications and best-effort applications, in an open shared environment [3][4][5][6]. This approach allows multimedia application to reserve processor resource and guarantees the resource availability to the admitted applications.

Real-time scheduling algorithms such as Rate Monotonic (RM) and EDF [2][7] are designed to guarantee resource availability to real-time applications. Deng et al. [8] proposed a scheduling scheme for hard real-time applications in open environment. However, these algorithms usually do not work well in a general-purpose open environment where soft real-time applications coexist with best-effort applications and compete for resource.

Abeni and Buttazzo [9] introduced the Constant Bandwidth Server (CBS), which schedules tasks based on budget reservation. It uses dedicated servers to isolate groups of tasks and guarantee protection to other tasks. CBS restricts the execution of tasks to its budget to protect other tasks, thus allowing unnecessary deadline misses.

Zhu et al. [10] proposed the Diff-EDF scheduler, which offers guarantees to tasks by changing a task's deadlines based on its desired miss-rate. Tasks with modified deadlines are then put into an EDF queue. Being focused on continuous media soft real-time applications, Diff-EDF lacks support to multiple classes and hard real-time tasks.

SMART [11] and Rialto [12] allow applications to specify real-time requirements for a computation unit. For example, Rialto uses a primitive *BeginConstraint()* to specify start time, deadline and criticality for a code block. These approaches may incur a large overhead, since multimedia applications usually contain a lot of code blocks with timing constraints and it is necessary to specify time constraints for each individual code block.

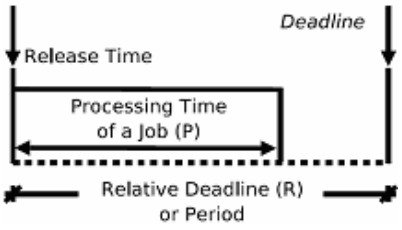
Yuan et al. [13] introduced the R-EDF algorithm, which targets the mix of soft real-time multimedia applications and best-effort applications in open shared environment. It supports multiple classes of multimedia tasks to reserve CPU resource, based on task utilization. The utilization-based reservation is optimistic, and the R-EDF algorithm protects overrun, handling it in a predictable time bound. However, some times the algorithm can deliver unexpected results for soft real-time applications generating undesirable delays. In addition, hard real-time applications are not supported. This work improves R-EDF algorithm by hard real-time supporting and a better overall performance of the application.

3 Job and Task Models

Models of real-time systems may use the concepts of task and job to represent the behavior of applications. A task is a part of an application, since that an application can be seen as a set of tasks. A job computes part of a task, having a release time and deadline. For example, a task could be mapped to a video decoding function and its jobs could be mapped to the processing of each frame, i.e. each frame is a job.

Figure 1(a) illustrates the job model. Together with the release time and deadline, a job has the processing time **P** and the relative deadline **R**, which, for this work, it is also considered the period of the task. The utilization **θ** of a job **J** is **θ**(**J**) = **P**/**R**. The task **T** is composed by a set of jobs (**T** = {$\mathbf{J_1}$, $\mathbf{J_2}$, ..., $\mathbf{J_n}$}, where **n** ≥ 1), as it is illustrated in Figure 1(b). In a task **T** with **n** jobs, the utilization of the task is $\theta(T) = \frac{\sum_{i=1}^{n} \theta(J_i)}{n}$. The current job list (**CJL**) of a task is the set of jobs that have already been released but not yet completed. That is, **CJL**(**T**) = {$\mathbf{J_s}$, …, $\mathbf{J_m}$}, where $\mathbf{J_m}$ is the latest released job.

(a) Job



(b) Task with *n* dependent jobs

Figure 1: Task and Job Models

4 R-EDF Concepts and Limitations

R-EDF is a real-time scheduler, based on the well-known EDF, which proposes to add QoS to task scheduling. It is accomplished by reserving the processing time via parameterization. R-EDF classifies five types of tasks:

- **Periodic Constant Processing Time (PCPT)** jobs have constant processing time and relative deadline, resulting in constant utilization.
- **Events** are a special kind of PCPT with only one job.
- **Periodic Variable Processing Time (PVPT)** jobs have constant relative deadline and variable processing time.
- **Aperiodic/Sporadic Constant Utilization (ASCU)** jobs have arbitrary relative deadlines and processing time, i.e. both parameters may vary at each job. Generally, there is no algorithm to meet deadlines for some sporadic jobs. Hence, the support to ASCU jobs imposes constraints: the jobs have constant utilization and their relative deadline is known at release time.
- **Best-effort** tasks have no timing restrictions, but should not starve.

Utilization **θ** and peak utilization **ψ** are defined, as the average utilization of all jobs

of a task and the maximum utilization among all jobs of a task, respectively. Each task reserves the processing time for all its jobs at the beginning of the task, based on **θ** for soft real-time tasks and **ψ** for hard real-time tasks. For example, if a soft task has **θ** = 20%, a 20% reservation will be effective for the task. When a job exceeds its reservation limit, it enters in the overrun state. The job returns to the ready state when comes the next release time of the task. Figure 2 illustrates the states that a task can be.

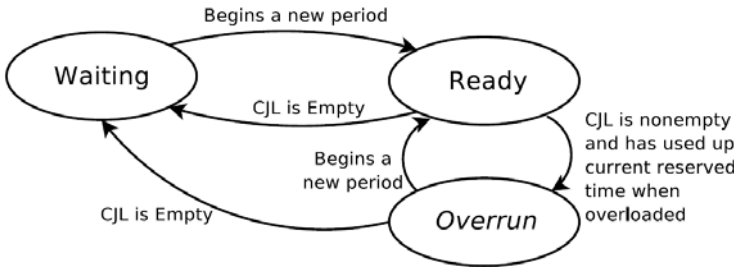


Figure 2: Finite state automata of real-time tasks

The number 1 defines 100% of the processor capacity. Therefore, a system with **M** processor has capacity **M**. R-EDF statistically multiplexes the processor capacity between real-time and best-effort tasks. The time-sharing capacity $\mathbf{C_{TS}}$ is the unreserved capacity, which is shared among all best-effort tasks. $\mathbf{C_{TS}}$ has a lower bound **β**, such that $\mathbf{C_{TS}} \geq \beta$, to protect best-effort tasks from starvation. Real-time capacity $\mathbf{C_{RTp}}$ and peak capacity $\mathbf{PC_{RTp}}$ of a processor **p** ($1 \leq \mathbf{p} \leq \mathbf{M}$) are, respectively, the sum of the utilizations of the tasks and the sum of peak utilization of tasks bound to a processor. That is, $\mathbf{C_{RTp}} = \sum_{\mathbf{i}=1}^{\mathbf{m}} \boldsymbol{\theta}(\mathbf{T_i})$ and $\mathbf{PC_{RTp}} = \sum_{\mathbf{i}=1}^{\mathbf{m}} \boldsymbol{\Psi}(\mathbf{T_i})$, where $\mathbf{T_i}$ ($1 \leq \mathbf{i} \leq \mathbf{m}$) are real-time tasks bound to a processor **p**. The system is classified as being real-time *overloaded* if $\mathbf{PC_{RTp}} > 1$, or $\sum_{\mathbf{p}=1}^{\mathbf{M}} \mathbf{PC_{RTp}} > \mathbf{M} - \boldsymbol{\beta}$ for the whole system. Otherwise, the system is *underloaded*. Analyzing the R-EDF algorithm's behavior, a limitation was found, as Figure 3 shows.

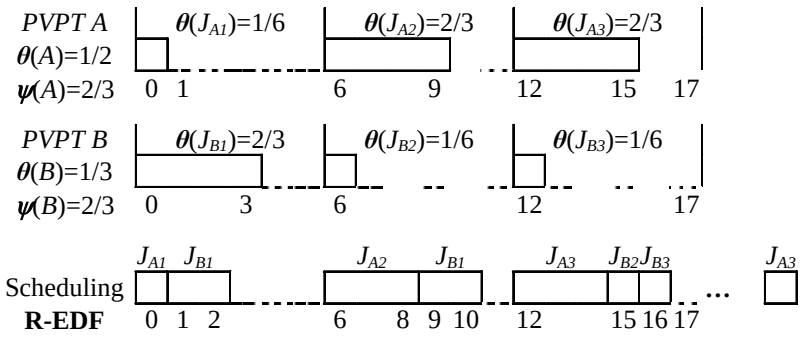


Figure 3: Reservation in R-EDF with two PVPT tasks

Two PVPT tasks are illustrated in Figure 3. Task **A** has a reservation **θ(A) = 1/2** and task **B** has reservation **θ(B) = 1/3**. At the beginning of execution, the job $\mathbf{J_{B1}}$ executes after job $\mathbf{J_{A1}}$. However, the job $\mathbf{J_{B1}}$ uses all its reserved time and enters in the overrun state. There would be time available to execute the job $\mathbf{J_{B1}}$ in the **3** and **4** time ticks. This example shows the restrictive reservation problem present in this algorithm. In the proposed algorithm (ER-EDF) we solve this issue by allowing tasks to execute in the extra available time (see Section 5).

In addition, R-EDF does not support hard real-time tasks, since it assumes that a job can miss its deadline when the reservation is reached. In this algorithm, if a task enters in the overrun state, it will miss its deadline.

5 The ER-EDF Algorithm

Enhanced R-EDF (ER-EDF) is an improvement of R-EDF. It was conceived to improve the QoS delivered to soft real-time tasks and provide reservation for hard-real time ones. The analysis of R-EDF showed that it can be improved; hence modifications were introduced to allow the prevention of restrictive reservations and allow the reservation of hard real-time tasks.

The hard real-time reservation is accomplished with the establishment of reservations based on worst case [1], i.e. the reservation is the task peak utilization ψ. This allows the scheduling of hard real-time, soft real-time and best effort tasks in the same system.

5.1 Admission Control

Changes in the admission control were introduced to effect the proposed alterations. At the creation time, each task informs the scheduler if it is a soft or hard real-time task. The admission control algorithm is presented below.

Step 1: Initially, the real-time capacity $\mathbf{C_{RTp}}$ and peak real-time capacity $\mathbf{PC_{RTp}}$ of each processor **p** are set to 0 ($1 \leq \mathbf{p} \leq \mathbf{M}$), and the time-sharing capacity $\mathbf{C_{TS}}$ is set to **M**.

Step 2: A real-time task with utilization $\boldsymbol{\theta}$ and peak utilization ψ requests reservation:

If the task is hard real-time **then** (reserve using ψ)

If the time-sharing capacity can be reduced to admit this task $\mathbf{C_{TS}} - \psi > \beta$, and a processor **p** ($1 < \mathbf{p} < \mathbf{M}$) can fulfill the requirement $\mathbf{C_{RTp}} + \psi \leq 1$ **then**

Task is bound to the processor **p**, with: $\mathbf{C_{RTp}} = \mathbf{C_{RTp}} + \psi$; $\mathbf{PC_{RTp}} = \mathbf{PC_{RTp}} + \psi$; $\mathbf{C_{TS}} = \mathbf{C_{TS}} - \psi$

Else

Task is rejected.

End if

Else (reserve using utilization $\boldsymbol{\theta}$):

If the time-sharing capacity can be reduced to admit this task $\mathbf{C_{TS}} - \boldsymbol{\theta} > \beta$ and a processor **p** ($1 \leq \mathbf{p} \leq \mathbf{M}$) can fulfill the requirement $\mathbf{C_{RTp}} + \boldsymbol{\theta} \leq 1$ **then**

Task is bound to the processor **p**, with: $\mathbf{C_{RTp}} = \mathbf{C_{RTp}} + \boldsymbol{\theta}$; $\mathbf{PC_{RTp}} = \mathbf{PC_{RTp}} + \psi$; $\mathbf{C_{TS}}$ = CTS - $\boldsymbol{\theta}$

Else

Task is rejected

End if

End if

Step 3: If a real-time task with utilization $\boldsymbol{\theta}$ and peak utilization ψ, bound to a processor **p**, releases its reservation, **then**:

If the task is hard real-time, **then**

$\mathbf{C_{RTp}} = \mathbf{C_{RTp}} - \psi$; $\mathbf{PC_{RTp}} = \mathbf{PC_{RTp}} - \psi$; $\mathbf{C_{TS}} = \mathbf{C_{TS}} + \psi$.

Else

$\mathbf{C_{RTp}} = \mathbf{C_{RTp}} - \boldsymbol{\theta}$; $\mathbf{PC_{RTp}} = \mathbf{PC_{RTp}} - \psi$; $\mathbf{C_{TS}} = \mathbf{C_{TS}} + \boldsymbol{\theta}$.

End if

5.2 Scheduling

ER-EDF incorporates modifications to allow better use of the processing capacity.

These modifications are conceived to allow a task to exit its overrun state and execute in the available time. The modifications include:

1. Forbid a task to enter in the overrun state when there is not any real-time task ready;

2. At the end of a job, remove the task with the earliest deadline from the overrun state if no other real-time task is ready to execute.

Like its predecessor, ER-EDF only activates the overrun protection mechanism when the system is overloaded. Consequently, ER-EDF has analogous behavior as the EDF algorithm, when the system is under loaded. The ER-EDF algorithm is described next.

Step 1: Selection a task for execution.
If any real-time task is ready, **then**
Select one whose latest released job has the earliest deadline and execute jobs in the CJL in order;
Else if there is a task in the overrun state, **then**
Select the task in the overrun state whose latest released job has the earliest deadline, put it in the ready state and execute jobs in the CJL in order.
Else
Invoke the best-effort task scheduler.
End if

Step 2: The scheduler waits until the next time unit.
If a running task finishes all its jobs, **then**
It enters the waiting state;
Else if the system is overloaded and the CJL of the current task is not empty and the task used all its reserved time, **then**
If there is any real-time task ready **then**
Current task enters the overrun state.
Else if the ran utilization of the current task is greater or equal than (1- β), **then**
It enters the overrun state.
Else
It continues to execute.
Check all tasks for reached release times and set them to the ready state.
End if
End if

Step 3: Go to step 1.

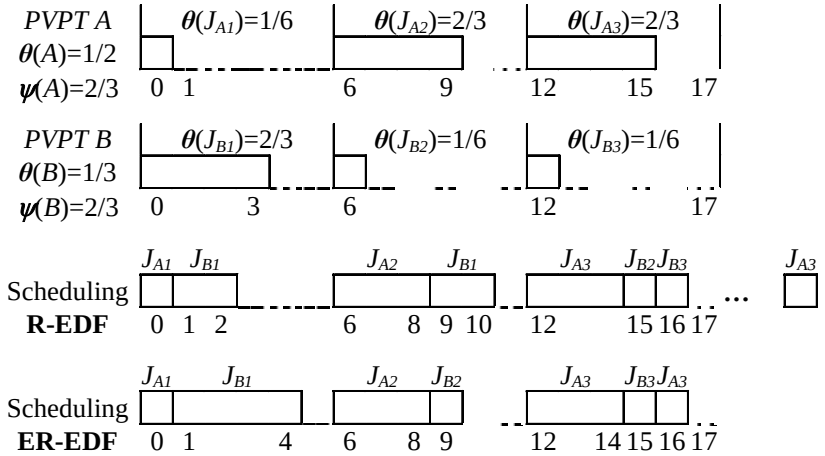


Figure 4: Restrictive reservation in R-EDF and ER-EDF with two PVPT tasks

In Figure 4, two PVPT tasks are show executing under R-EDF and ER-EDF algorithms according to the example presented in Section 4. While in R-EDF execution the job $\mathbf{J_{B1}}$ enters the overrun state, ER-EDF verifies that no other real-time task is available to run and allows that job to execute in the 3 and 4 time ticks, passing its reserved time. In the time tick 15, the job $\mathbf{J_{A3}}$ enters the overrun state, allowing $\mathbf{J_{B3}}$ to execute. Soon after $\mathbf{J_{B3}}$ execution, $\mathbf{J_{A3}}$ exits the overrun state, finishing its execution in the remaining available time.

6 Implementation/Experiments

The Spartan-3 Starter Board [14], together with Plasma soft-core processor, was used to validate the proposed algorithms. The border contains 1 MiB of SRAM memory, displays, leds, serial port and JTAG for bit stream loading. Plasma implements a reduced MIPS instruction set. It also is open-source, allowing a flexible hardware support.

The operating system used is EPOS[1]. EPOS is an application-oriented operating system, i.e. it adapts automatically to the application requirements. The two algorithms were implemented in this OS and two experiments are shown next.

In the first experiment, a situation where four real-time tasks are executed is shown. Each task has 500 jobs to execute simultaneously. The first three are PCPT tasks, i.e. have constant utilization. PCPT tasks have peak utilization equal to task utilization ($\boldsymbol{\theta} = \boldsymbol{\psi}$). Thus, PCPT tasks have similar behavior to tasks marked as hard real-time.

Table 1 presents the parameters used to generate the experiment data. Task 2 is the only one marked as hard real-time. However, the reservation is similarly made to the first three tasks. Task 4 is a PVPT task where each job receives a generated utilization based on a linear mathematical distribution, where the minimum is 10% and the maximum is 42%. The system is classified overloaded with $\sum_{i=1}^{4} \Psi(T_i) = 115\%$. The total reservation of the system is 100%.

Table 1: Parameters for experiment data generation – first experiment

Task	1	2	3	4
Jobs	500			
Period	50ms			
Best-effort (β)	0%			
Distribution	Cst	Cst	Cst	Linear
Distribution Param.	26%	21%	26%	10-42%
Utilization (θ)	26%	21%	26%	27%
Peak Utilization (ψ)	26%	21%	26%	42%
Total Utilization (θ)	100%			
Total Peak Util. (ψ)	115%			
Total Reservation	100%			

The deadline miss results for the four tasks of the execution are presented in Figure 5. In this experiment, the hard real-time parameter was disabled in R-EDF and ER-EDF to verify the similar behavior for PCPT tasks. Even though only the second task is marked

[1] Available in http://epos.lisha.ufsc.br/

as hard real-time, all first three behave similarly, losing 0% of its deadlines. In the execution of the EDF algorithm, all tasks miss deadlines. R-EDF and ER-EDF present no deadline miss for PCPT tasks, which is therefore compensated in the PVPT task. However, ER-EDF presents lower deadline miss rate compared to R-EDF.

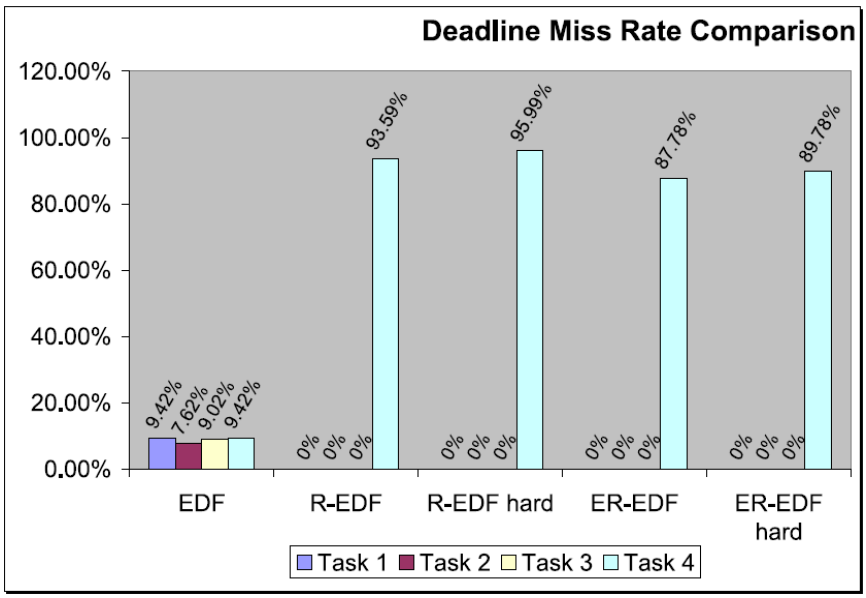


Figure 5: Deadline miss comparison for EDF, R-EDF and ER-EDF with/without hard real-time marks

The next experiment shows two tasks with different periods executing in an overloaded environment. The first task is a hard real-time PCPT with constant utilization of 50%. The second is a PVPT with numbers generated by a linear distribution. Table 2 shows the parameters provided to generate data to be used in the execution.

Table 2: Parameters for experiment data generation – second experiment

Task	1	2
Jobs	500	250
Period	50ms	100ms
Best-effort (β)	0%	
Distribution	Constant	Linear
Distribution Param.	50%	20-75%
Utilization(θ)	50%	49%
Peak Utilization(ψ)	50%	75%
Total Utilization (θ)	99%	
Total Peak Util.(ψ)	125%	
Total Reservation	99%	

Figure 6 compares EDF, R-EDF and ER-EDF showing their deadline miss rate for the execution described in Table 2. EDF presents an undesirable behavior showing considerable deadline miss rate for both tasks. R-EDF shows the first task correctly being treated as hard real-time with 0% deadline miss. ER-EDF shows analogous execution to R-EDF. However, ER-EDF presents an improvement of 30% on the second task over R-EDF.

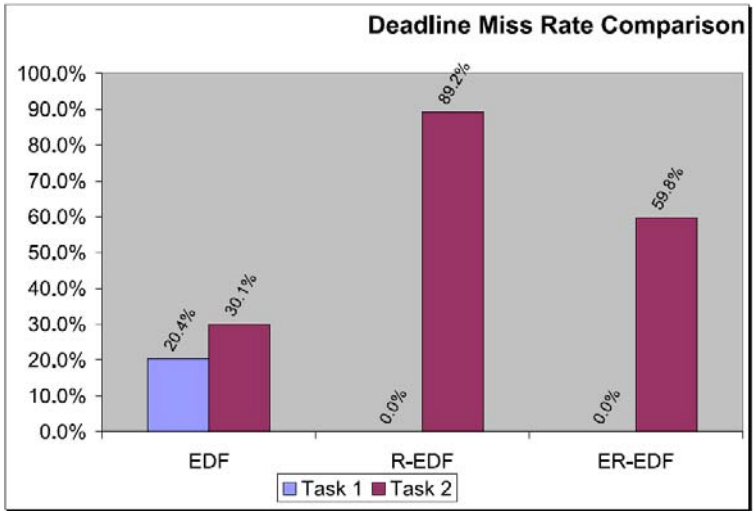


Figure 6: Deadline miss comparison for EDF, R-EDF and ER-EDF

7 Conclusions

This work introduced a new real-time scheduler algorithm to provide quality of service to applications. The new algorithm – *Enhanced* R-EDF – is based on R-EDF, a multiclass real-time scheduler. R-EDF presents some limitations that are overcome by the new algorithm. In addition, the support for hard real-time tasks was added, which is fundamental to applications that require great responsiveness, and allows the existence of hard real-time, soft-real time and best effort tasks in the same system.

ER-EDF showed significant improvement over its predecessor R-EDF. The addition of hard real-time support allows developers to parameterize the application to fulfill application's real-time requirements. However, the enhancement of real-time execution costs to the best-effort tasks more starvation time.

References

[1] A. Campbell et al. A Survey of Programmable Networks. ACM SIGCOMM Computer Communications Review, v. 29, n. 2, pp. 7-23, 1999.

[2] C. Liu and J. Layland. Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. **Journal of ACM**, v. 20, n. 1, pp. 46-61, 1973.

[3] H. Chu and K. Nahrstedt. A Soft Real Time Scheduling Server in UNIX Operating System. **IDMS**, pp. 153-162, 1997.

[4] H. Chu and K. Nahrstedt. CPU Service Classes for Multimedia Applications. **ICMCS**, v. 1, pp. 296-301, 1999.

[5] M. Jones, D. Rosu, and M. Rosu. CPU Reservations and Time Constraints: Efficient, Predictable Scheduling of Independent Activities. **SOSP**, pp. 198-211, 1997.

[6] C. Mercer et al. Processor Capacity Reserves: Operating System Support for Multimedia Applications. **ICMCS**, pp. 90-99, 1994.

[7] J. Liu. Real-Time Systems. **Prentice Hall**, NJ, 2000.

[8] Z. Deng and J. Liu. Scheduling Real-Time Applications in an Open Environment. **IEEE Real-Time Systems Symposium**, p. 308-319, 1997.

[9] L. Abeni and G. Buttazzo. Resource Reservation in Dynamic Real-Time Systems. **Real-Time Systems**, v. 27, n. 2, pp. 123-167, 2004.

[10] H. Zhu et al. Diff-EDF: A Simple Mechanism for Differentiated EDF Service. **IEEE Real Time Technology and Applications Symposium**, pp. 268–277, 2005.

[11] J. Nieh and M. Lam. The Design, Implementation and Evaluation of SMART: A Scheduler for Multimedia Applications. **SOSP**, pp. 184–197, 1997.

[12] M. Jones et al. An Overview of the Rialto Real-Time Architecture. **ACM SIGOPS European Workshop**, pp. 249-256, 1996.

[13] W. Yuan, K. Nahrstedt, and K. Kim. R-EDF: A reservation-based EDF scheduling algorithm for multiple multimedia task classes. **IEEE Real Time Technology and Applications Symposium**, pp. 149-154, 2001.

[14] Xilinx, Inc. Spartan-3 Starter Kit Board - User Guide, 2005. Available at www.xilinx.com/bvdocs/userguides/ug130.pdf.