

# Rate Monotonic vs. EDF: Judgment Day

G.C. Buttazzo

University of Pavia, Italy  
buttazzo@unipv.it

**Abstract.** Since the first results published in 1973 by Liu and Layland on the Rate Monotonic (RM) and Earliest Deadline First (EDF) algorithms, a lot of progress has been made in the schedulability analysis of periodic task sets. Unfortunately, many misconceptions still exist about the properties of these two scheduling methods, which usually tend to favor RM more than EDF. Typical wrong statements often heard in technical conferences and even in research papers claim that RM is easier to analyze than EDF, it introduces less runtime overhead, it is more predictable in transient overload conditions, and causes less jitter in task execution. Since the above statements are either wrong, or not precise, it is time to clarify these issues in a systematic fashion, because the use of EDF allows a better exploitation of the available resources and significantly improves system's performance. This paper compares RM against EDF under several aspects, using existing theoretical results or simple counterexamples to show that many common beliefs are either false or only restricted to specific situations.

## 1 Introduction

Before a comprehensive theory was available, the most critical control applications were developed using an off-line table-driven approach (timeline scheduling), according to which the time line is divided into slots of fixed length (*minor cycle*) and tasks are statically allocated in each slot based on their rates and execution requirements. Although very predictable, such a technique is fragile during overload conditions and it is not flexible enough for supporting dynamic systems [24].

Such problems can be solved by using a priority-based approach, according to which each task is assigned a priority (which can be fixed or dynamic) and the schedule is generated on line based on the current priority value. In 1973, Liu and Layland [23] analyzed the properties of two basic priority assignment rules: the Rate Monotonic (RM) algorithm and the Earliest Deadline First (EDF) algorithm. According to RM, tasks are assigned fixed priorities that are proportional to their rate, so the task with the smallest period receives the highest priority. According to EDF, priorities are assigned dynamically and are inversely proportional to the absolute deadlines of the active jobs.

Assuming that each task is characterized by a worst-case execution time  $C_i$  and a period  $T_i$ , Liu and Layland showed that the schedulability condition of

a task set can be derived by computing the *processor utilization factor*  $U_p = \sum_{i=1}^n C_i/T_i$ . Clearly, if  $U_p > 1$  no feasible schedule exists for the task set with any algorithm. If  $U_p \leq 1$ , the feasibility of the schedule depends on the task set parameters and on the scheduling algorithm used in the system.

The Rate Monotonic algorithm is the most used priority assignment in real-time applications, because it is very easy to implement on top of commercial kernels that do not support explicit timing constraints. On the other hand, implementing a dynamic scheme, like EDF, on top of a priority-based kernel would require to keep track of all absolute deadlines and perform a dynamic mapping between absolute deadlines and priorities. Such an additional implementation complexity and runtime overhead often prevents EDF to be implemented on top of commercial real-time kernels, even though it would increase the total processor utilization.

At present, only a few research kernels support EDF as a native scheduling scheme. Examples of such kernels are Hartik [8], Shark [15], Erika [14], Spring [33], and Yartos [17]. A new trend in some recent operating system is to provide support for the development of a user level scheduler. This is the approach followed in MarteOS [27].

In addition to resource exploitation and implementation complexity, there are other evaluation criteria that should be taken into account when selecting a scheduling algorithm for a real-time application. Moreover, there are a lot of misconceptions about the properties of these two scheduling algorithms, that for a number of reasons penalize EDF more than it should be. The typical motivations that are usually given in favor of RM state that RM is easier to implement, it introduces less runtime overhead, it is easier to analyze, it is more predictable in transient overload conditions, and causes less jitter in task execution.

In the following sections we show that most of the claims stated above are either false or do not hold in the general case. Although some discussion of the relative costs of RM and EDF appear in [35], in this paper the behavior of these two algorithms is analyzed under several perspectives, including implementation complexity, runtime overhead, schedulability analysis, robustness during transient overloads, and response time jitter. Moreover, the two algorithms are also compared with respect to other issues, including resource sharing, aperiodic task handling, and QoS management.

## 2 Implementation Complexity

When talking about the implementation complexity of a scheduling algorithm, we have to distinguish the case in which the algorithm is developed on top of a generic priority based operating system, from the case in which the algorithm is implemented from scratch, as a basic scheduling mechanism in the kernel.

When considering the development of the scheduling algorithm on top of a kernel based on a set of fixed priority levels, it is indeed true that the EDF implementation is not easy, nor efficient. In fact, even though the kernel allows task priorities to be changed at runtime, mapping dynamic deadlines to pri-

orities cannot always be straightforward, especially when, as common in most commercial kernels, the number of priority levels is small. If the algorithm is developed from scratch in the kernel using a list for the ready queue, then the only difference between the two approaches is that, while in RM the ready queue is ordered by decreasing fixed priority levels, under EDF it has to be ordered by increasing absolute deadlines. Thus, once the absolute deadline is available in the task control block, the basic kernel operations (e.g., insertion, extraction, getfirst, dispatch) have the same complexity, both under RM and EDF.

An advantage of RM with respect to EDF is that, if the number of priority levels is not high, the RM algorithm can be implemented more efficiently by splitting the ready queue into several FIFO queues, one for each priority level. In this case, the insertion of a task in the ready queue can be performed in  $O(1)$ . Unfortunately, the same solution cannot be adopted for EDF, because the number of queues would be too large (e.g., equal to  $2^{32}$  if system time is represented by four byte variables).

Another disadvantage of EDF is that absolute deadlines change from a job to the other and need to be computed at each job activation. Such a runtime overhead is not present under RM, since periods are typically fixed. However, the problem of evaluating the runtime overhead introduced by the two algorithms is more articulated, as discussed in the next section.

### 3 Runtime Overhead

It is commonly believed that EDF introduces a larger runtime overhead than RM, because in EDF absolute deadlines need to be updated from a job to the other, so slightly increasing the time needed to execute the job activation primitive. However, when context switches are taken into account, EDF introduces less runtime overhead than RM, because it reduces the number of preemptions that typically occur under RM.

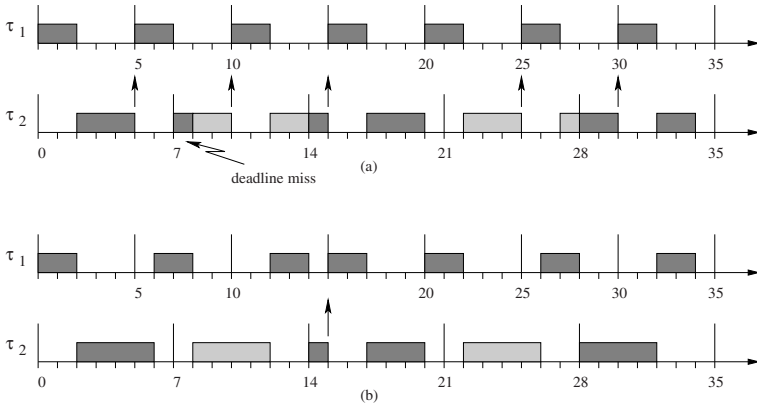
The example illustrated in Figure 1 shows that, under RM, to respect the priority order given by periods, the high priority task  $\tau_1$  must preempt every instance of  $\tau_2$ , whereas under EDF preemption occurs only once in the entire hyperperiod<sup>1</sup>. For larger task sets, the number of preemptions caused by RM increases, thus the overhead due to the context switch time is higher under RM than EDF.

To evaluate the behavior of the two algorithms with respect to preemptions, a number of simulation experiments have been performed using synthetic task sets with random parameters.

Figure 2 shows the average number of preemptions introduced by RM and EDF as a function of the number of tasks. For each point in the graph, the average was computed over 1000 independent simulations, each running for 1000 units of time. In each simulation, periods were generated as random variables

---

<sup>1</sup> The hyperperiod is defined as the smallest interval of time after which the schedule repeats itself and it is equal to the least common multiple of the task periods.



**Fig. 1.** Preemptions introduced by RM (a) and EDF (b) on a set of three periodic tasks. Adjacent jobs of  $\tau_2$  are depicted with different colours to better distinguish them.

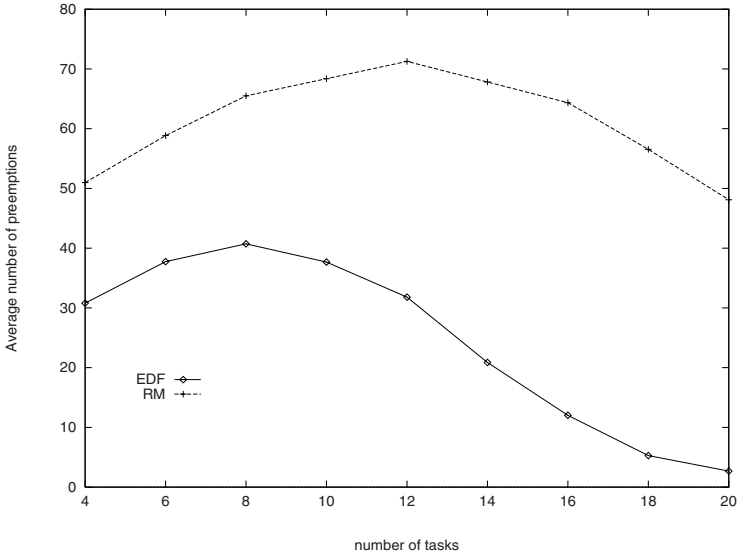
with uniform distribution in the range of 10 to 100 units of time, whereas execution times were computed to create a total processor utilization of 0.9.

As shown in the plot, each curve has two phases: the number of preemptions occurring in both schedules increases for small task sets and decreases for larger task sets. This can be explained as follows. For small task sets, the number of preemptions increases because the chances for a task to be preempted increase with the number of tasks in the system. As the number of tasks gets higher, however, task execution times get smaller in the average, to keep the total processor utilization constant, hence the chances for a task to be preempted reduce. As evident from the graph, such a reduction is much more significant under EDF.

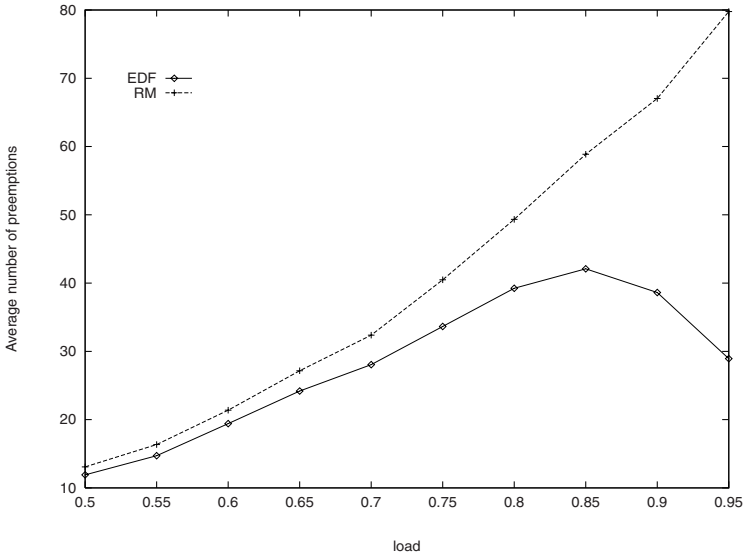
In another experiment, we tested the behavior of RM and EDF as a function of the processor load, for a fixed number of tasks. Figure 3 shows the average number of preemptions as a function of the load for a set of 10 periodic tasks. Periods and computation times were generated with the same criterion used in the previous experiment, but to create an average load ranging from 0.5 to 0.95.

It is interesting to observe the different behavior of RM and EDF for high processor loads. Under RM, the number of preemptions constantly increases with the load, because tasks with longer execution times have more chances to be preempted by tasks with higher priorities. Under EDF, however, increasing task execution times does not always imply a higher number of preemptions, because a task with a long period could have an absolute deadline shorter than that of a task with smaller period. In certain situations, an increased execution time can also cause a lower number of preemptions.

This phenomenon is illustrated in Figure 4, which shows what happens when the execution time of  $\tau_3$  is increased from 4 to 8 units of time. When  $C_3 = 4$ , the second instance of  $\tau_2$  is preempted by  $\tau_1$ , that has a shorter absolute



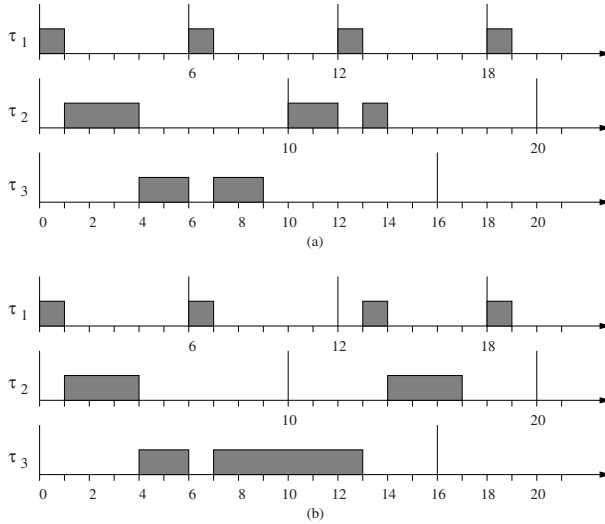
**Fig. 2.** Preemptions introduced by RM and EDF as a function of the number of tasks.



**Fig. 3.** Preemptions introduced by RM and EDF on a set of 10 periodic tasks as a function of the load.

deadline. If  $C_3 = 8$ , however, the longer execution of  $\tau_3$  (which has the earliest deadline among the active tasks) pushes  $\tau_2$  after the arrival of  $\tau_1$ , so avoiding its preemption. Clearly, for a higher number of tasks, this situation occurs more frequently, offering more advantage to EDF. Such a phenomenon does not occur

under RM, because tasks with small period always preempt tasks with longer period, independently of the absolute deadlines.



**Fig. 4.** Under EDF, the number of preemptions may decrease when execution times increase: in case (a), where  $C_3$  is small,  $\tau_2$  is preempted by  $\tau_1$ , but this does not occur in case (b), where  $\tau_3$  has a higher execution time.

## 4 Schedulability Analysis

The basic schedulability conditions for RM and EDF proposed by Liu and Layland in [23] were derived for a set  $I$  of  $n$  periodic tasks under the assumptions that all tasks start simultaneously at time  $t = 0$ , relative deadlines are equal to periods, and tasks are independent (that is, they do not have resource constraints, nor precedence relations). Under such assumptions, a set of  $n$  periodic tasks is schedulable by the RM algorithm if

$$\sum_{i=1}^n U_i \leq n(2^{1/n} - 1). \quad (1)$$

The schedulability bound of RM is a function of the number of tasks, and it decreases with  $n$ . We recall that for large  $n$  the bounds tends to  $\ln 2 \simeq 0.69$ . Under EDF, a task set is schedulable if and only if  $\sum_{i=1}^n U_i \leq 1$ .

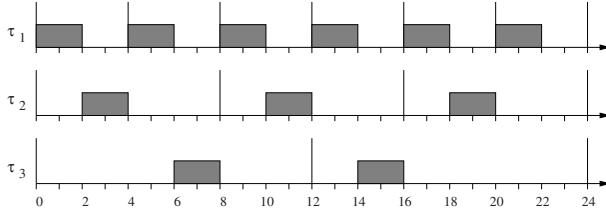
In [21], Lehoczky, Sha, and Ding performed a statistical study and showed that for task sets with randomly generated parameters the RM algorithm is able to feasibly schedule task sets with a processor utilization up to about 88%.

However, this is only a statistical result and cannot be taken as an absolute bound for performing a precise guarantee test.

A more efficient schedulability test, known as the Hyperbolic Bound (HB), was proposed by Bini et al. in [6]. This test has the same complexity as the Liu and Layland one, but improves the acceptance ratio up to a limit of  $\sqrt{2}$  for large  $n$ . According to this method, a set of periodic tasks is schedulable by RM if

$$\prod_{i=1}^n (U_i + 1) \leq 2. \quad (2)$$

For RM, the schedulability bound improves when periods have harmonic relations. A common misconception, however, is to believe that the schedulability bound becomes 1.00 when the periods are multiple of the smallest period. This is not true, as can be seen from the example reported in Figure 5.



**Fig. 5.** A task set in which all periods are multiple of the shortest period is not sufficient to guarantee a schedulability bound equal to one.

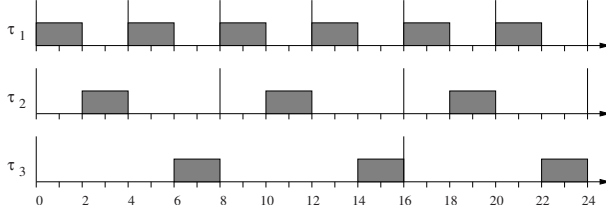
Here, tasks  $\tau_2$  and  $\tau_3$  have periods  $T_2 = 8$  and  $T_3 = 12$ , which are multiple of  $T_1 = 4$ . Since all tasks have a computation time equal to two, the total processor utilization is

$$U = \frac{1}{2} + \frac{1}{4} + \frac{1}{6} = \frac{11}{12} \simeq 0.917$$

and, as we can see from the figure, the schedule produced by RM is feasible. However, it is easy to see that increasing the computation time of any task by a small amount  $\tau_3$  will miss its deadline. This means that, for this particular task set, the utilization factor cannot be higher than  $11/12$ .

The correct result is that the schedulability bound becomes 1.00 only when *any* pair of periods is in harmonic relation. As an example, Figure 6 shows that, if  $T_3 = 16$ , then not only  $T_2$  and  $T_3$  are multiple of  $T_1$ , but also  $T_3$  is multiple of  $T_2$ . In this case, the RM generates a feasible schedule even when  $C_3 = 4$ , that is when the total processor utilization is equal to 1.00.

In the general case, exact schedulability tests for RM yielding to necessary and sufficient conditions have been independently derived in [18,21,2]. Using the Response Time Analysis (RTA) proposed in [2], a periodic task set (with deadlines less than or equal to periods) is schedulable with the RM algorithm if and only if the worst-case response time of each task is less than or equal to its



**Fig. 6.** When *any* pair of periods is in harmonic relation the schedulability bound of RM is equal to one.

deadline. The worst-case response time  $R_i$  of a task can be computed using the following iterative formula:

$$\begin{cases} R_i^{(0)} = C_i \\ R_i^{(k)} = C_i + \sum_{j:D_j < D_i} \left\lceil \frac{R_i^{(k-1)}}{T_j} \right\rceil C_j. \end{cases} \quad (3)$$

where the worst-case response time of task  $\tau_i$  is given by the smallest value of  $R_i^{(k)}$  such that  $R_i^{(k)} = R_i^{(k-1)}$ . It is worth noting, however, that the complexity of the exact test is pseudo-polynomial, thus it is not suited to be used for online admission control in applications with large task sets. To solve this problem, an approximate feasibility test with a tunable complexity has been proposed by Bini and Buttazzo in [7].

Under EDF, the schedulability analysis of periodic tasks with relative deadlines less than periods can be performed using the Processor Demand Criterion proposed by Baruah, Howell, and Rosier [5]. According to this method, a set of tasks is schedulable by EDF if and only if

$$\forall L > 0, \quad \sum_{i=1}^n \left\lfloor \frac{L + T_i - D_i}{T_i} \right\rfloor C_i \leq L \quad (4)$$

As the response time analysis, this test has also a pseudo-polynomial complexity. It can be shown that the number of points in which the test has to be performed can be significantly restricted to those  $L$  equal to deadlines less than a certain value  $L^*$ , that is:

$$\forall L \in \mathcal{D}, \quad \mathcal{D} = \{d_k : d_k < \min(L^*, H)\}$$

where  $H = \text{lcm}(T_1, \dots, T_n)$  is the hyperperiod and

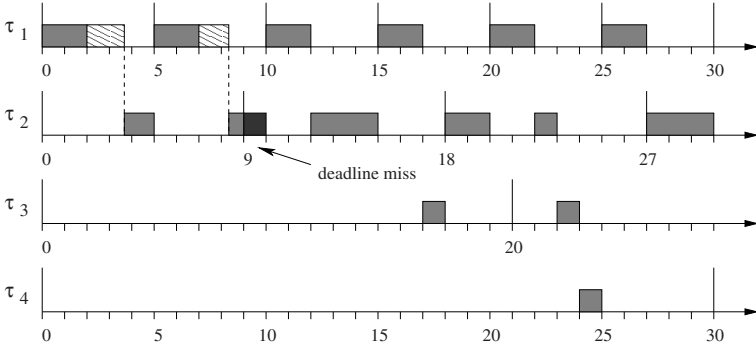
$$L^* = \frac{\sum_{i=1}^n U_i(T_i - D_i)}{1 - U}.$$

In conclusion, if relative deadlines are equal to periods, exact schedulability analysis can be performed in  $O(n)$  under EDF, whereas is pseudo-polynomial under RM. When relative deadlines are less than periods, the analysis is pseudo-polynomial for both scheduling algorithms.



## 5 Robustness During Overloads

Another common misconception about RM is to believe that, in the presence of transient overload conditions, deadlines are missed predictably, that is, the first tasks that fail are those with the longest period. Unfortunately, this property does not hold under RM (neither under EDF), and can easily be confuted by the counterexample shown in Figure 7. In this figure, there are four periodic tasks with computation times  $C_1 = 2$ ,  $C_2 = 3$ ,  $C_3 = 1$ ,  $C_4 = 1$ , and periods  $T_1 = 5$ ,  $T_2 = 9$ ,  $T_3 = 20$ ,  $T_4 = 30$ . In normal load conditions, the task set is schedulable by RM. However, if there is a transient overload in the first two instances of task  $\tau_1$  (in the example, the jobs have an overrun of 1.5 time units), the task that misses its deadline is not the one with the longest period (i.e.,  $\tau_4$ ), but  $\tau_2$ .



**Fig. 7.** Under overloads, only the highest priority task is protected under RM, but nothing can be ensured for the other tasks.

So the conclusion is that, under RM, if the system becomes overloaded, any task, except the highest priority task, can miss its deadline, independently of its period. The situation is not better under EDF. The only difference between RM and EDF is that, under RM, an overrun in task  $\tau_i$  cannot cause tasks with higher priority to miss their deadlines, whereas under EDF any other task could miss its deadline.

The problem caused by execution overruns can be solved by enforcing temporal isolation among tasks through a resource reservation mechanism in the kernel. This issue has been investigated both under RM and EDF and it is discussed in Section 7.

## 6 Jitter

In a feasible periodic task system, the computation performed by each job must start after its release time and must complete within its deadline. Due to the presence of other concurrent tasks that compete for the processor, however, a

task may evolve in different ways from instance to instance; that is, the instructions that compose a job can be executed at different times, relative to the release time, within different jobs. The maximum time variation (relative to the release time) in the occurrence of a particular event in two consecutive instances of a task defines the jitter for that event. So, for example, the start time jitter of a task is the maximum time variation between the relative start times of any two consecutive jobs. If  $s_{i,k}$  denotes the start time of the  $k^{th}$  job of task  $\tau_i$ , then the start time jitter ( $STJ_i$ ) of task  $\tau_i$  is defined as

$$STJ_i = \max_k |s_{i,k+1} - s_{i,k}|. \quad (5)$$

Similarly, the response time jitter is defined as the maximum difference between the response times of any consecutive jobs. If  $R_{i,k}$  denotes the response time of the  $k^{th}$  job of task  $\tau_i$ , then the response time jitter ( $RTJ_i$ ) of task  $\tau_i$  is defined as

$$RTJ_i = \max_k |R_{i,k+1} - R_{i,k}|. \quad (6)$$

In real-time applications, the jitter can be tolerated when it does not degrade the performance of the system. In many control applications, however, a high jitter can cause instability or a jerky behavior of the controlled system [25], hence it must be kept as low as possible.

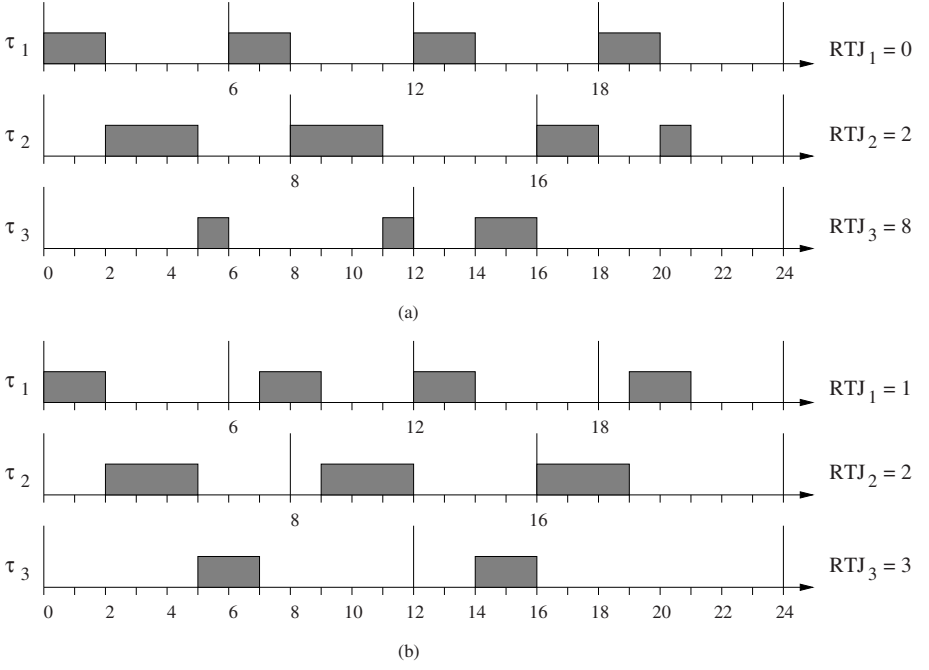
Another misconception about RM is to believe that the fixed priority assignment used in RM reduces the jitter during task execution, more than under EDF. This is false, as it is shown by the following example. Consider a set of three periodic tasks with computation times  $C_1 = 2$ ,  $C_2 = 3$ ,  $C_3 = 2$ , and periods  $T_1 = 6$ ,  $T_2 = 8$ ,  $T_3 = 12$ . For this example we consider the response time jitter, as defined by equation (6). Figure 8 illustrates that, under RM, the three tasks experience a response time jitter equal to 0, 2, and 8, respectively. Under EDF, the same tasks have a response time jitter equal to 1, 2, and 3, respectively. Hence, under RM, the jitter experienced by task  $\tau_3$  is much higher than that given under EDF.

Notice that this example does not prove that EDF always introduces less jitter than RM, but just confutes the common belief that RM outperforms EDF in reducing jitter.

## 7 Other Issues

### 7.1 Resource Sharing

If tasks share mutually exclusive resources, particular care must be used for accessing shared data. In fact, if critical sections are accessed through classical semaphores, then tasks may experience long delays due to a phenomenon known as *priority inversion*, where a task may be blocked by a lower priority task for an unbounded amount of time. The problem can be solved by adopting specific concurrency control protocols for accessing critical sections, such as the Priority Inheritance Protocol (PIP) or the Priority Ceiling Protocol (PCP), both



**Fig. 8.** Response time jitter introduced under RM (a) and under EDF (b).

developed by Sha, Rajkumar and Lehoczky in [28]. For the reason that both PIP and PCP are well known in the real-time literature and were originally devised for RM, some people believe that only RM can be predictably used and analyzed in the presence of shared resources. However, this is not true, because a number of protocols also exist for accessing shared resources under EDF, such as the Dynamic Priority Ceiling (DPC) [12], the Stack Resource Policy (SRP) [3], and other similar methods specifically developed for deadline-based scheduling algorithms [16,35].

## 7.2 Aperiodic Task Handling

When real-time systems include soft aperiodic activities to be scheduled together with hard periodic tasks, RM and EDF show a significant difference in achieving good aperiodic responsiveness. In this case, the objective is to reduce the aperiodic response times as much as possible, still guaranteeing that all periodic tasks complete within their deadlines.

Several aperiodic service methods have been proposed under both algorithms. The common approach adopted in such cases is to schedule aperiodic requests through a periodic server, which allocates a certain amount of budget  $C_s$ , for aperiodic execution, in every period  $T_s$ . Under RM, the most used service mechanisms are the Deferrable Server [20,34], and the Sporadic Server [29], which

have good responsiveness and easy implementation complexity. Under EDF, the most efficient service mechanism in terms of performance/cost ratio is the Total Bandwidth Server (TBS), proposed by Spuri and Buttazzo in [30,32].

The superior performance of EDF-based methods in aperiodic task handling comes from the higher processor utilization bound. In fact, the lower schedulability bound of RM limits the maximum utilization ( $U_s = C_s/T_s$ ) that can be assigned to the server for guaranteeing the feasibility of the periodic task set. As a consequence, the spare processor utilization that cannot be assigned to the server is wasted as a background execution. This problem does not occur under EDF, where, if  $U_p$  is the processor utilization of the periodic tasks, the full remaining fraction  $1 - U_p$  can always be allocated to the server for aperiodic execution.

A different approach used under RM to improve aperiodic responsiveness is the one adopted in the Slack Stealing algorithm, originally proposed by Lehoczky and Thuel in [22]. The main idea behind this method is that there is no benefit in completing periodic tasks much before their deadlines. Hence, when an aperiodic request arrives, the Slack Stealer steals all the available slack from periodic tasks (pushing them as much as possible towards their deadlines) and uses the slack to execute aperiodic requests as soon as possible. Although the Slack Stealer performs much better than the Deferrable Server and the Sporadic Server, it is not optimal, in the sense that it cannot minimize the response times of the aperiodic requests.

Tia, Liu, and Shankar [36] proved that, if periodic tasks are scheduled using a fixed-priority assignment, no algorithm can minimize the response time of every aperiodic request and still guarantee the schedulability of the periodic tasks. In particular, the following theorems were proved in [36]:

**Theorem 1 (Tia-Liu-Shankar).** *For any set of periodic tasks ordered on a given fixed-priority scheme and aperiodic requests ordered according to a given aperiodic queueing discipline, there does not exist any valid algorithm that minimizes the response time of every soft aperiodic request.*

**Theorem 2 (Tia-Liu-Shankar).** *For any set of periodic tasks ordered on a given fixed-priority scheme and aperiodic requests ordered according to a given aperiodic queueing discipline, there does not exist any on-line valid algorithm that minimizes the average response time of the soft aperiodic requests.*

Notice that Theorem 1 applies both to clairvoyant and on-line algorithms, whereas Theorem 2 only applies to on-line algorithms. These results are not true for EDF, where optimal algorithms have been found for minimizing aperiodic response times. In particular, the Improved Total Bandwidth server (ITB) [10] assigns an initial deadline to an aperiodic request according to a TBS with a bandwidth  $U_s = 1 - U_p$ . Then, the algorithm tries to shorten this deadline as much as possible to enhance aperiodic responsiveness, still maintaining the periodic tasks schedulable. When the deadline cannot be furtherly shortened, it can be used to schedule the task with EDF, thus minimizing its response

time. The algorithm that stops the deadline shortening process after  $N$  steps is denoted as  $TB(N)$ . Thus,  $TB(0)$  denotes the standard TBS and  $TB^*$  denotes the optimal algorithm which continues to shorten the deadline until its minimum value.

Figure 9 shows the performance of the Slack Stealer against  $TB(0)$ ,  $TB(1)$ ,  $TB(3)$ , and  $TB^*$  as a function of the aperiodic load, when the periodic utilization is  $U_p = 0.85$ . The periodic task set has been chosen to be schedulable both under RM and EDF. As can be seen from the plots, the Slack Stealer performs better than the standard TBS. However, just one iteration on the TBS deadline assignment,  $TB(1)$ , dominates the Slack Stealer.

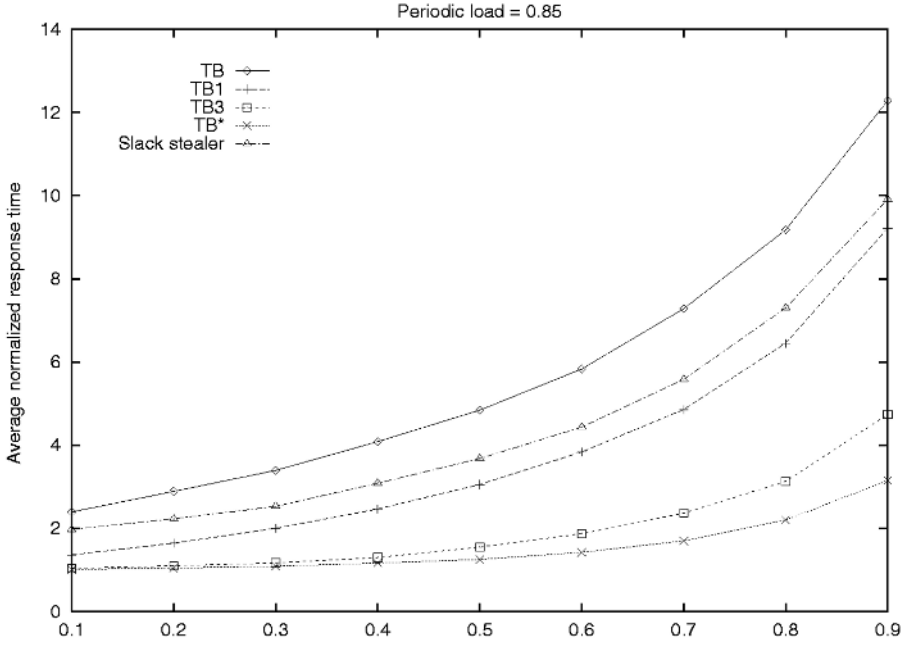


Fig. 9. Performance of the  $TB(N)$  against the Slack Stealer.

### 7.3 Resource Reservation

The problem caused by execution overruns can be solved by enforcing temporal isolation among tasks through a resource reservation mechanism in the kernel. This issue has been investigated both under RM and EDF and several approaches have been proposed in the literature. Under RM, the problem has

been investigated by Mercer, Savage and Tokuda [26], who proposed a capacity reserve mechanism that assigns each task a given budget in every period and downgrades the task to a background level when the reserved capacity is exhausted.

Under EDF, a similar mechanism has been proposed by Abeni and Buttazzo, through the Constant Bandwidth Server [1]. This method also uses a budget to reserve a desired processor bandwidth for each task, but it is more efficient in that the task is not executed in background when the budget is exhausted. Instead a deadline postponement mechanism guarantees that the used bandwidth never exceeds the reserved value.

Resource reservation mechanisms are essential for preventing task interference during execution overruns, and this is particularly important in application tasks that have highly variable computation times (e.g., multimedia activities). In addition, isolating the effects of overruns within individual tasks allows relaxing worst case assumptions, increasing efficiency and performing much better quality of service control.

## 8 Conclusions

In this paper we compared the behavior of the two most famous policies used today for developing real-time applications: the Rate Monotonic (RM) and the Earliest Deadline First (EDF) algorithm. Although widely used, in fact, there are still many misconceptions about the properties of these two scheduling methods, mainly concerning their implementation complexity, the runtime overhead they introduce, their behavior during transient overloads, the resulting jitter, and their efficiency in handling aperiodic activities. For each of these issues we tried to confute some typical misconception and tried to clarify the properties of the algorithms by illustrating simple examples or reporting formal results from the existing real-time literature. In some cases, specific simulation experiments have also been performed to verify the overhead introduced by context switches and the effectiveness in improving aperiodic responsiveness.

In conclusion, the real advantage of RM with respect to EDF is its simpler implementation in commercial kernels that do not provide explicit support for timing constraints, such as periods and deadlines. Other properties typically claimed for RM, such as predictability during overload conditions, or better jitter control, only apply for the highest priority task, and do not hold in general. On the other hand, EDF allows a full processor utilization, which implies a more efficient exploitation of computational resources and a much better responsiveness of aperiodic activities. These properties become very important for embedded systems working with limited computational resources, and for multimedia systems, where quality of service is controlled through resource reservation mechanisms that are much more efficient under EDF. In fact, most resource reservation algorithms are implemented using service mechanisms similar to aperiodic servers, which have better performance under EDF.

Finally, both RM and EDF are not very well suited to work in overload conditions and to achieve jitter control. To cope with overloads, specific extensions have been proposed in the literature, both for aperiodic [9] and periodic [19,11] load. Also a method for jitter control under EDF has been addressed in [4] and can be adopted whenever needed.

## References

1. L. Abeni and G. Buttazzo, "Integrating Multimedia Applications in Hard Real-Time Systems", *Proc. of the IEEE Real-Time Systems Symposium*, Madrid, Spain, December 1998.
2. N.C. Audsley, A. Burns, M. Richardson, K. Tindell and A. Wellings, "Applying New Scheduling Theory to Static Priority Preemptive Scheduling", *Software Engineering Journal* 8(5), pp. 284–292, September 1993.
3. T.P. Baker, Stack-Based Scheduling of Real-Time Processes, *The Journal of Real-Time Systems* 3(1), pp. 76–100, (1991).
4. S. Baruah, G. Buttazzo, S. Gorinsky, and G. Lipari, "Scheduling Periodic Task Systems to Minimize Output Jitter," *Proceedings of the 6th IEEE International Conference on Real-Time Computing Systems and Applications*, Hong Kong, December 1999.
5. S. K. Baruah, R. R. Howell, and L. E. Rosier, "Algorithms and Complexity Concerning the Preemptive Scheduling of Periodic Real-Time Tasks on One Processor," *Real-Time Systems*, 2, 1990.
6. E. Bini, G. C. Buttazzo and G. M. Buttazzo, "A Hyperbolic Bound for the Rate Monotonic Algorithm", *Proceedings of the 13<sup>th</sup> Euromicro Conference on Real-Time Systems*, Delft, The Netherlands, pp. 59–66, June 2001.
7. E. Bini and G. C. Buttazzo, "The Space of Rate Monotonic Schedulability", *Proceedings of the 23<sup>rd</sup> IEEE Real-Time Systems Symposium*, Austin, Texas, December 2002.
8. G. C. Buttazzo, "HARTIK: A Real-Time Kernel for Robotics Applications", *Proceedings of the 14th IEEE Real-Time Systems Symposium*, Raleigh-Durham, December 1993.
9. G. Buttazzo and J. Stankovic, "Adding Robustness in Dynamic Preemptive Scheduling", in *Responsive Computer Systems: Steps Toward Fault-Tolerant Real-Time Systems*, Edited by D. S. Fussell and M. Malek, Kluwer Academic Publishers, Boston, 1995.
10. G. Buttazzo and F. Sensini, "Optimal Deadline Assignment for Scheduling Soft Aperiodic Tasks in Hard Real-Time Environments," *IEEE Transactions on Computers*, Vol. 48, No. 10, October 1999.
11. G. Buttazzo, G. Lipari, M. Caccamo, and L. Abeni, "Elastic Scheduling for Flexible Workload Management", *IEEE Transactions on Computers*, Vol. 51, No. 3, pp. 289–302, March 2002.
12. M.I. Chen, and J.K. Lin, Dynamic Priority Ceilings: A Concurrency Control Protocol for Real-Time Systems, *Journal of Real-Time Systems* 2, (1990).
13. M.L. Dertouzos, "Control Robotics: the Procedural Control of Physical Processes," *Information Processing*, 74, North-Holland, Publishing Company, 1974.
14. P. Gai, G. Lipari, M. Di Natale, "A Flexible and Configurable Real-Time Kernel for Time Predictability and Minimal Ram Requirements," *Technical Report*, Scuola Superiore S. Anna, Pisa, RETIS TR2001–02, March 2001.

15. P. Gai, L. Abeni, M. Giorgi, G. Buttazzo, "A New Kernel Approach for Modular Real-Time Systems Development", Proceedings of the 13th Euromicro Conference on Real-Time Systems, Delft (NL), June 2001.
16. K. Jeffay, "Scheduling Sporadic Tasks with Shared Resources in Hard-Real-Time Systems," Proceedings of IEEE Real-Time System Symposium, pp. 89–99, December 1992.
17. K. Jeffay, D.L. Stone, and D. Poirier, "YARTOS: Kernel support for efficient, predictable real-time systems," Real-Time Systems Newsletter, Vol. 7, No. 4, pp. 8–13, Fall 1991. Republished in Real-Time Programming, W. Halang and K. Ramamritham, eds. Pergamon Press, Oxford, UK, 1992.
18. M. Joseph and P. Pandya, "Finding Response Times in a Real-Time System," *The Computer Journal*, 29(5), pp. 390–395, 1986.
19. G. Koren and D. Shasha, "Skip-Over: Algorithms and Complexity for Overloaded Systems that Allow Skips," *Proc. of the IEEE Real-Time Systems Symposium*, 1995.
20. J. P. Lehoczky, L. Sha, and J. K. Strosnider, "Enhanced Aperiodic Responsiveness in Hard Real-Time Environments," *Proceedings of the IEEE Real-Time Systems Symposium*, pp. 261–270, 1987.
21. J. P. Lehoczky, L. Sha, and Y. Ding, "The Rate-Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behaviour", *Proceedings of the IEEE Real-Time Systems Symposium*, pp. 166–171, 1989.
22. J. P. Lehoczky and S. Ramos-Thuel, "An Optimal Algorithm for Scheduling Soft-Aperiodic Tasks in Fixed-Priority Preemptive Systems," Proceedings of the IEEE Real-Time Systems Symposium, 1992.
23. C.L. Liu and J.W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard real-Time Environment", *Journal of the ACM* 20(1), 1973, pp. 40–61.
24. J. Locke, "Designing Real-Time Systems", Invited talk, *IEEE International Conference of Real-Time Computing Systems and Applications (RTCSA '97)*, Taiwan, December 1997.
25. P. Marti, G. Fohler, K. Ramamritham, and J.M. Fuertes, "Control performance of flexible timing constraints for Quality-of-Control Scheduling," Proc. of the 23rd IEEE Real-Time System Symposium, Austin, TX, USA, December 2002.
26. C.W. Mercer, S. Savage, and H. Tokuda, "Processor Capacity Reserves for Multimedia Operating Systems," Technical Report, Carnegie Mellon University, Pittsburgh (PA), CMU-CS-93-157, May 1993.
27. M. Aldea Rivas and M. González Harbour, "POSIX-Compatible Application-Defined Scheduling in MaRTE OS," Euromicro Conference on Real-Time Systems (WiP), Delft, The Netherlands, June 2001.
28. L. Sha, R. Rajkumar, and J. P. Lehoczky, "Priority Inheritance Protocols: An Approach to Real-Time Synchronization," *IEEE Transactions on Computers*, 39(9), pp. 1175–1185, 1990.
29. B. Sprunt, L. Sha, and J. Lehoczky, "Aperiodic Task Scheduling for Hard Real-Time System," *Journal of Real-Time Systems*, 1, pp. 27–60, June 1989.
30. M. Spuri, and G.C. Buttazzo, "Efficient Aperiodic Service under Earliest Deadline Scheduling", *Proceedings of IEEE Real-Time System Symposium*, San Juan, Portorico, December 1994.
31. M. Spuri, G.C. Buttazzo, and F. Sensini, "Robust Aperiodic Scheduling under Dynamic Priority Systems", *Proc. of the IEEE Real-Time Systems Symposium*, Pisa, Italy, December 1995.
32. M. Spuri and G.C. Buttazzo, "Scheduling Aperiodic Tasks in Dynamic Priority Systems," *Real-Time Systems*, 10(2), 1996.



33. J. Stankovic and K. Ramamritham, "The Design of the Spring Kernel," *Proceedings of the IEEE Real-Time Systems Symposium*, December 1987.
34. J.K. Strosnider, J.P. Lehoczky, and L. Sha, "The Deferrable Server Algorithm for Enhanced Aperiodic Responsiveness in Hard Real-Time Environments", *IEEE Transactions on Computers*, 44(1), January 1995.
35. J. Stankovic, K. Ramamritham, M. Spuri, and G. Buttazzo, *Deadline Scheduling for Real-Time Systems*, Kluwer Academic Publishers, Boston-Dordrecht-London, 1998.
36. T.-S. Tia, J. W.-S. Liu, and M. Shankar, "Algorithms and Optimality of Scheduling Aperiodic Requests in Fixed-Priority Preemptive Systems," *Journal of Real-Time Systems*, 10(1), pp. 23–43, 1996.