Real-Time Systems, 28, 1–22, 2004

Rate Monotonic vs. EDF: Judgment Day

GIORGIO C. BUTTAZZO buttazzo@unipv.it
University of Pavia, Italy

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. Unfortunaltey, 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, specific simulation experiments, or simple counterexamples to show that many common beliefs are either false or only restricted to specific situations.

Keywords:

1. Introduction

Periodic task scheduling is one of the most studied topics within the field of real-time systems, due to the large number of control applications that require cyclical activities. 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 (Locke, 1997). The schedule is then constructed up to the least common multiple of all the periods (called the hyperperiod or the major cycle) and stored in a table. At runtime, tasks are despatched according to the table and synchronized by a timer at the beginning of each minor cycle. On one hand, timeline scheduling is straightforward to implement and does not introduce significant runtime overhead (since scheduling decisions are taken off-line). Moreover, tasks always execute in their preallocated slots, so the experienced jitter is very small.

On the other hand, timeline scheduling is fragile during overload situations, since a task exceeding its predicted execution time could generate (if not aborted) a domino effect on the subsequent tasks, causing their execution to exceed the minor cycle boundary (timeline break). In addition, timeline scheduling is not flexible enough for handling dynamic situations. In fact, a creation of a new task, or a little change in a task rate, might modify the values of the minor and major cycles, thus requiring a complete redesign of the scheduling table.

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 (1973) 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 ordered as the rates, 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.

The major contribution of Liu and Layland was to derive a simple guarantee test to verify the schedulability of a periodic task set under both algorithms. Their results refer to the following task model. Each periodic task τ_i is considered as an infinite sequence of jobs $\tau_{i,k}$ $(k = 1, 2, \ldots)$, where the first job $\tau_{i,1}$ is released at time $r_{i,1} = \Phi_i$ (the task phase) and the generic k-th job $\tau_{i,k}$ is released at time $r_{i,k} = \Phi_i + (k-1)T_i$, where T_i is the task period. Each job is characterized by a worst-case execution time C_i, a relative deadline D_i and an absolute deadline $d_{i,k} = r_{i,k} + D_i$. The ratio $U_i = C_i/T_i$ is called the utilization factor of task τ_i and represents the fraction of processor time used by that task. The value

$$U_p = \sum_{i=1}^{n} U_i$$

is called the total processor utilization factor and represents the fraction of processor time used by the periodic task set. 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 RM algorithm is probably the most used priority assignment in real-time applications, because it is very easy to implement on top of those commercial kernels that do not support explicit timing constraints on the task set. Indeed, an RM scheduler can be implemented just by assigning each task a fixed priority level inversely proportional to its period. On the other hand, implementing a dynamic scheme, like EDF, on top of a priority-based kernel would require the kernel to keep track of all absolute deadlines and perform a dynamic mapping between absolute deadlines and priorities. Such a mapping should be done every time a new task receives a priority falling between two adjacent priority levels, or outside the range of available priorities. Such an additional implementation complexity and runtime overhead due to dynamic priority management often prevents EDF from being implemented on top of commercial real-time kernels, even though it would increase the total processor utilization.

Certainly, the use of EDF would significantly increase if commercial kernels provided direct support for time and deadline management at the task level, because EDF allows a better exploitation of the available resources and would significant improve system's performance. At present, however, the majority of commercial real-time operating systems are still based on a limited set of fixed priority levels, and only a few research kernels support EDF as a native scheduling scheme. Examples of such kernels are Hartik (Buttazzo, 1993), Shark (Gai et al., 2001b}, Erika (Gai et al., 2001a), Spring (Stankovic and Ramamirtham, 1987), and Yartos (Jeffay, 1992). 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 (Rivas and Harbour, 2001).

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 unfairly penalize EDF. 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 Stankovic et al. (1998), in this paper the behavior of these two algorithms is analyzed under several perspectives, including implementation complexity, runtime overhead, schedulability analysis, robustness during transient and permanent 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.

The rest of the paper is organized as follows: Section 2 discusses the implementation complexity of the two algorithms, considering not only the problem of priority mapping, but also the basic kernel operations for a low-level implementation. Section 3 analyses the runtime overhead introduced in task execution by the two methods; both activation overhead and context switches are taken into account. Section 4 briefly summarizes the methods proposed in the literature (with their computational complexity) used to analyze the feasibility of a task set under different simplifying assumptions. Section 5 discusses the robustness of the two algorithms during transient and permanent overload conditions. Section 6 analyses the behavior of RM and EDF with respect to response time jitter and input–output latency. Section 7 discusses other evaluation criteria that should be considered when selecting a scheduling algorithm for a real-time application; they include protocols for resource sharing, aperiodic task handling, and resource reservation approaches. Finally, Section 8 states our conclusions.

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 priorities cannot always be straightforward, especially when, as common in most commercial kernels, the number of priority levels is small (typically, not greater than 256). For example, consider the case in which two deadlines d_a and d_b are mapped into two adjacent priority levels and a new periodic

instance is released with an absolute deadline d_c, such that $d_a < d_c < d_b$. In this situation, there is not a priority level that can be selected to map d_c, even when the number of active tasks is less than the number of priority levels in the kernel. This problem can only be solved by remapping d_a and d_b into two new priority levels which are not consecutive. Notice that, in the worst case, all current deadlines may need to be remapped, increasing the cost of the operation.

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, despatch) 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 complex, as discussed in the following 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.

It is indeed true that, under EDF, deadlines need to be updated by the kernel at each job activation, because in a periodic task the absolute deadline changes from a job to the other. Whenever the k-th job of task τ_i is released at time $r_{i,k}$, its absolute deadline has to be computed as $d_{i,k} = r_{i,k} + D_i$. Such a computation is not needed under RM, because the priority of task τ_i is assigned based on its period T_i or, if using Deadline Monotonic, based on its relative deadline D_i, which does not change from a job to the other.

In spite of the extra computation needed for updating the absolute deadline, however, EDF introduces less runtime overhead than RM, when context switches are taken into account. In fact, to enforce the fixed priority order, the number of preemptions that typically occur under RM is much higher than under EDF.

The example illustrated in Figure 1 shows that, under RM, to respect the priority order given by periods, the high priority task τ_1 must preempt every instance of τ_2, whereas under EDF preemption occurs only once in the entire hyperperiod.[1] 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.

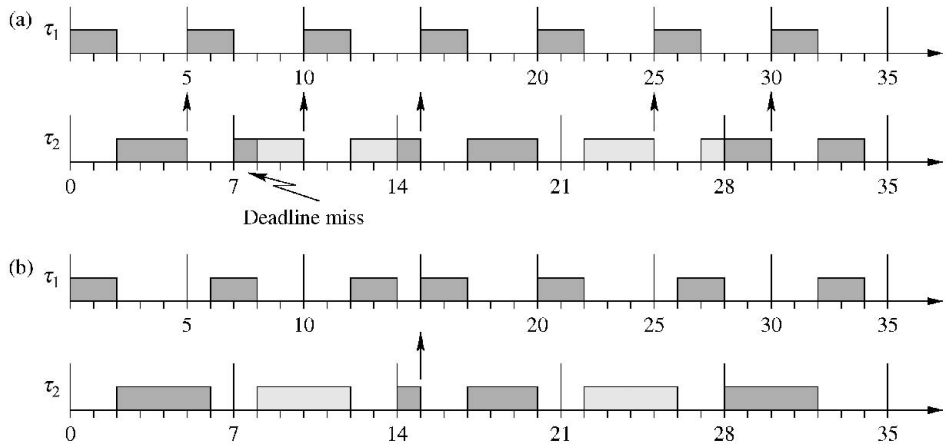

Figure 1. Preemptions introduced by RM (a) and EDF (b) on a set of two periodic tasks. Adjacent jobs of τ_2 are depicted with different colors to better distinguish them.

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 with uniform distribution in the

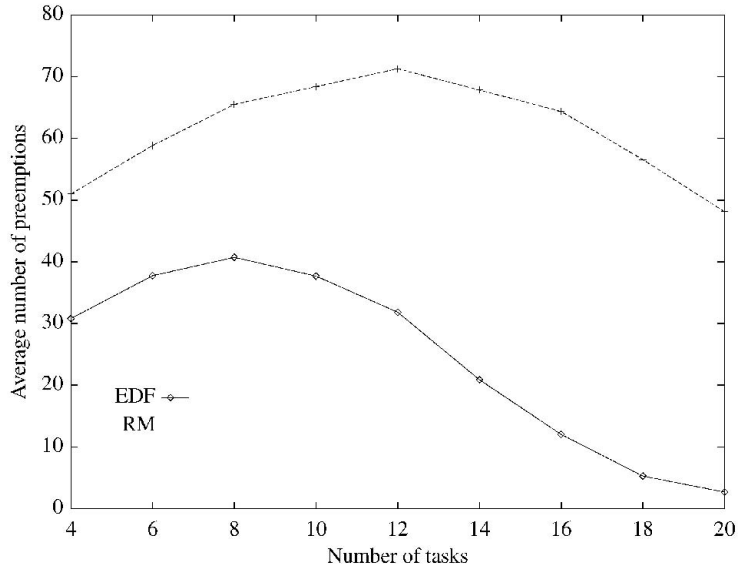

Figure 2. Preemptions introduced by RM and EDF as a function of the number of tasks.

range of 10 to 100 units of time, whereas execution times were computed to create a total processor utilization $U = 0.9$.

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 τ_3 is increased from 4 to 8 units of time. When $C_3 = 4$, the second

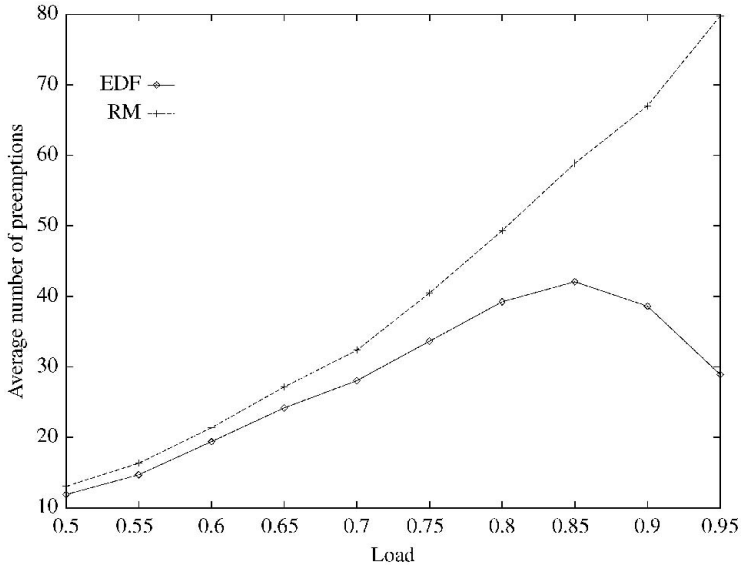

Figure 3. Preemptions introduced by RM and EDF on a set of 10 periodic tasks as a function of the load.

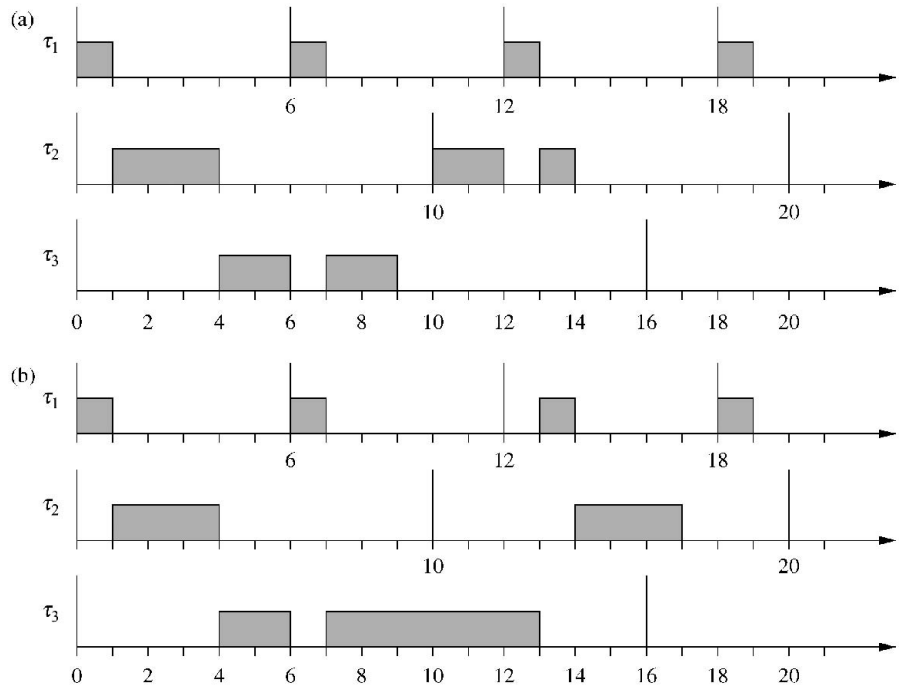

Figure 4. Under EDF, the number of preemptions may decrease when execution times increase: in case (a), where C_3 is small, τ_2 is preempted by τ_1, but this does not occur in case (b), where τ_3 has a higher execution time.

instance of τ_2 is preempted by τ_1, that has a shorter absolute deadline. If $C_3 = 8$, however, the longer execution of τ_3 (which has the earliest deadline among the active tasks) pushes τ_2 after the arrival of τ_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 periods always preempt tasks with longer period, independently of their absolute deadlines. The result of the experiment illustrated in Figure 3 shows that the number of preemptions increases almost linearly with the load under RM, while it decreases for high loads under EDF.

4. Schedulability Analysis

The basic schedulability conditions for RM and EDF proposed by Liu and Layland (1973) were derived for a set Γ of n periodic tasks under the assumptions that all tasks start simultaneously at time $t = 0$ (that is $\Phi_i = 0$ for all $i = 1, \ldots, n$), relative deadlines are equal to periods (i.e., $d_{i,k} = k T_i$) and tasks are independent (i.e., 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

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

Under the same assumptions, a set of n periodic tasks is schedulable by the EDF algorithm if and only if

$$\sum_{i=1}^{n} U_i \leq 1 \tag{2}$$

The schedulability bound of RM is a function of the number of tasks, and it decreases with n. We recall that

$$\lim_{n\to\infty} n\,(2^{1/n} - 1) = \ln 2 \simeq 0.69$$

meaning that any task set can be scheduled by RM if $U \leq 0.69$, but not all task sets can be scheduled if $0.69 < U \leq 1$.

Lehoczky et al. (1989) 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. (2001). 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

$$\prod_{i=1}^{n} (U_i + 1) \leq 2 \tag{3}$$

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.

Here, tasks τ_2 and τ_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 easy to see that increasing the computation time of any task by a small amount τ_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 all pairs of periods in the task set are 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

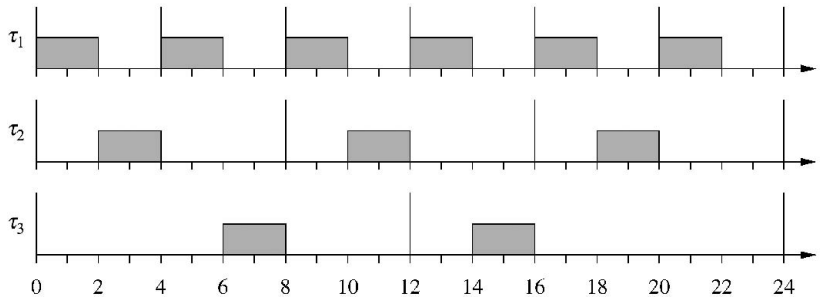

Figure 5. The condition in which all periods are multiple of the shortest period is not sufficient to guarantee a schedulability bound equal to one.

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 Lehoczky et al. (1989), Audsley et al. (1993), Joseph and Pandya (1986). Using the Response Time Analysis (RTA) proposed in Audsley et al. (2004), 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 deadline. The worst-case response time R_i of a task can be computed using the following iterative formula:

$$\begin{cases} R_i^{(0)} = C_i \\ R_i^{(k)} = C_i + \displaystyle\sum_{j:D_j<D_i} \left\lceil \frac{R_i^{(k-1)}}{T_j} \right\rceil C_j \end{cases} \tag{4}$$

where the worst-case response time of task τ_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

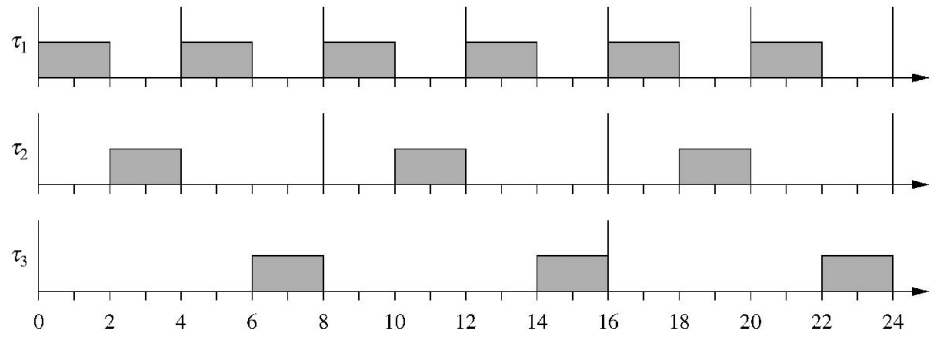

Figure 6. When all pairs of periods are in harmonic relation the schedulability bound of RM is equal to one.

applications with large task sets. To solve this problem, an approximate fesibility test with a tunable complexity has been proposed by Bini and Buttazzo (2002).

Under EDF, the schedulability analysis of periodic tasks with relative deadlines less than periods can be performed using the processor demand criterion (PDC) proposed by Baruah et al. (1990). According to this method, a set of tasks is schedulable by EDF if and only if

$$\forall L > 0, \quad \sum_{i=1}^{n} \left\lfloor \frac{L + T_i - D_i}{T_i} \right\rfloor C_i \leq L \tag{5}$$

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:

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

where $H = \text{lcm}(T_1, \ldots, T_n)$ 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 schedulabilty 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, although, in the average, the PDC requires more computational steps. Figure 7 shows the average number of steps required to run the (RTA) and the PDC as a function of the number of tasks. The tests were performed on randomly generated

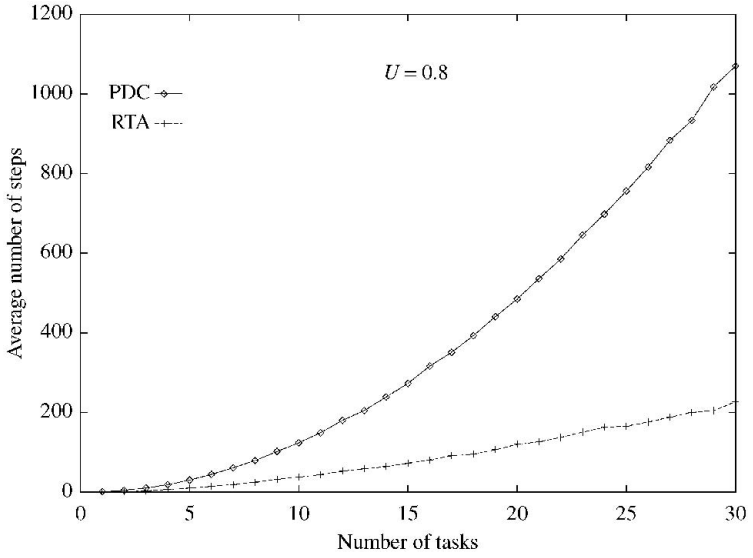

Figure 7. Average number of steps required for the RTA and for the PDC as a function of the number of tasks.

task sets with periods T_i uniformly distributed in $[10, 200]$, relative deadlines uniformly distributed in $[T_i/2, T_i]$, and utilization factor $U = 0.8$. The average was computed on 1000 samples.

5. Robustness During Overloads

In this section we compare the behavior of RM and EDF during overload conditions, that is when the total demand of the task set exceeds the processor capacity. We first consider the two algorithms under permanent overload situations (occurring when $U > 1$), and then under transient overload conditions, caused by sporadic execution overruns in some of the jobs.

5.1. Permanent Overload

An interesting property of EDF during permanent overloads is that it automatically performs a period rescaling, and tasks start behaving as they were executing at a lower rate. This property has been proved by Cervin et al. (2002) and it is formally stated in the following theorem.

THEOREM 1 *(Cervin) Assume a set of n periodic tasks, where each task is described by a fixed period T_i, a fixed execution time C_i, a relative deadline D_i, and a release offset Φ_i. If $U > 1$ and tasks are scheduled by EDF, then, in stationarity, the average period $\overline{T_i}$ of each task τ_i is given by $\overline{T_i} = T_i U$.*

Notice that under RM, a permanent overload may cause a complete blocking of the lower priority tasks. Figure 8 illustrates the schedules produced by EDF and RM for a set of three periodic tasks in a permanent overload condition. The three tasks, in fact, generate a total processor workload $U = 4/8 + 6/12 + 5/20 = 1.25$. According to the previous theorem, this means that EDF schedules the tasks as they were executing with periods $T'_1 = T_1 U = 10$, $T'_2 = T_2 U = 15$, and $T'_3 = T_3 U = 25$ (note that $U' = 4/10 + 6/15 + 5/25 = 1$). Indeed, it can be easily verified that in the first interval of 120 units of time, τ_1 executes 12 times $(120/10 = 12)$, τ_2 executes 8 times $(120/15 = 8)$, and τ_3 executes almost 5 times $(120/25 = 4.8)$.

In conclusion, under permanent overload conditions both the behaviors of RM and EDF are predictable, but, deciding which one is better is highly application dependent.

5.2. Transient Overload

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

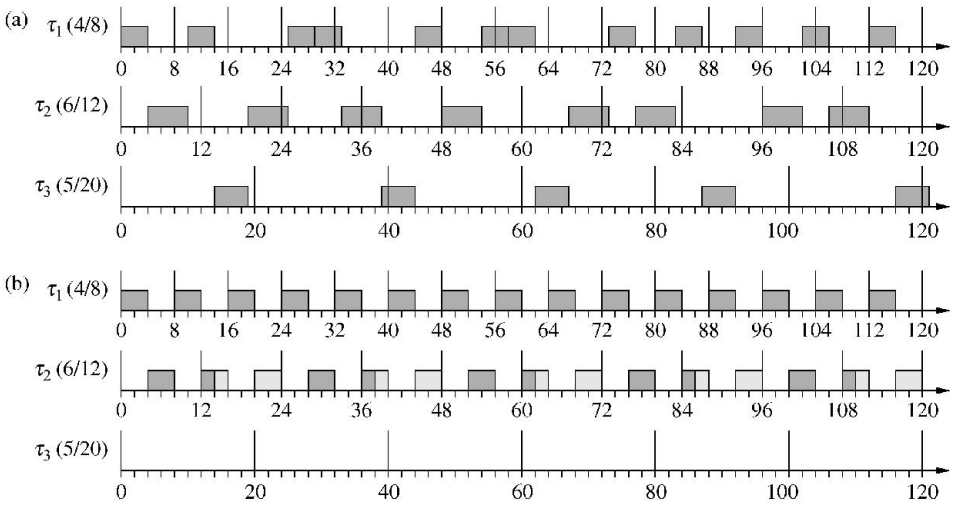

Figure 8. Schedules produced by EDF (a) and RM (b) for a set of three periodic tasks in a permanent overload condition.

Figure 9. 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 τ_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., τ_4), but τ_2.

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

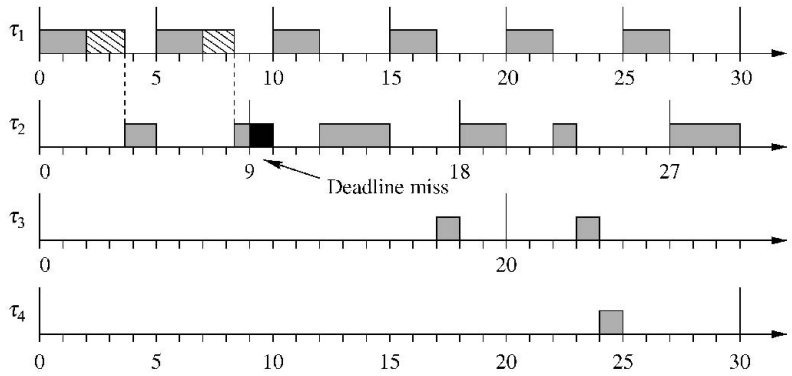

Figure 9. Under overloads, only the highest priority task is protected under RM, but nothing can be ensured for the other tasks.

RM, an overrun in task τ_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 sporadic 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 and Latency

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 different instances of a task defines the jitter for that event. The jitter of an event of a task τ_i is said to be relative if the variation refers to two consecutive instances of τ_i, and absolute if it is computed as the maximum variation with respect to all the instances.

For example, the relative response time jitter (RRJ) of a task is the maximum time variation between the response times of any two consecutive jobs. If $R_{i,k}$ denotes the response time of the k-th job of task τ_i, then the relative response time jitter (RRJ_1) of task τ_i is defined as

$$\text{RRJ}_i = \max_k |R_{i,k+1} - R_{i,k}| \tag{6}$$

whereas the absolute response time jitter (ARJ_i) of task τ_i is defined as

$$\text{ARJ}_i = \max_k R_{i,k} - \min_k R_{i,k} \tag{7}$$

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 (Marti et al., 2002), 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. Clearly, this is true for the highest priority task, but this property does not hold in general, 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 $T_1 = 6$, $T_2 = 8$, $T_3 = 12$. Figure 10 illustrates that, under RM, the three tasks experience a response time jitter (both relative and absolute) equal to 0, 2, and 8, respectively. Under EDF, the same tasks have a response time jitter (both relative and absolute) equal to 1, 2, and 3, respectively. Hence, EDF significantly reduces the jitter experienced by task τ_3 by slightly increasing the one of task τ_1.

Clearly, 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.

A specific simulation experiment has been performed to verify the jitter behavior under the two scheduling algorithms. Task sets of 10 periodic tasks were randomly generated

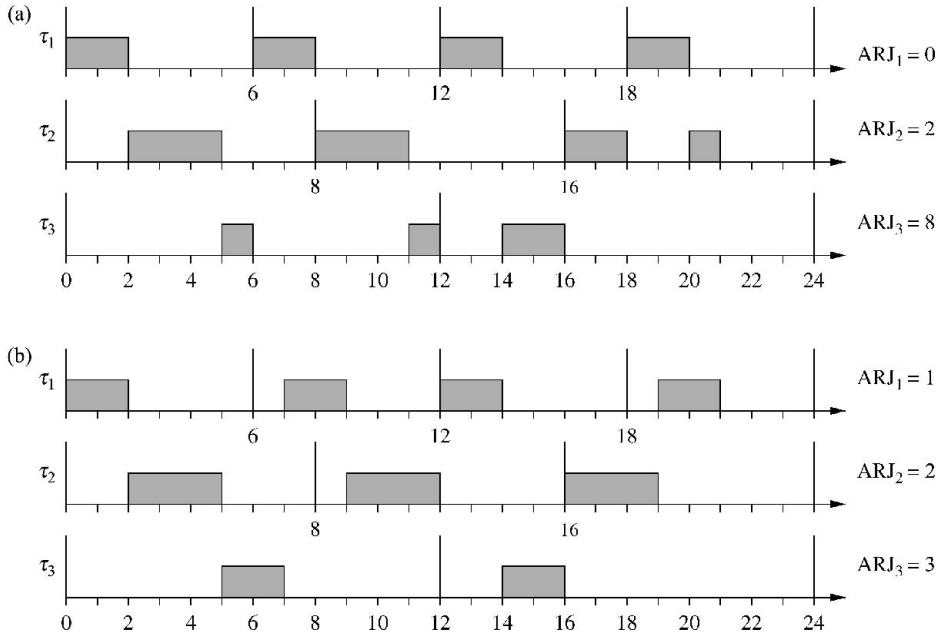

Figure 10. Response time jitter introduced under RM (a) and under EDF (b).

with periods uniformly distributed in $[10, 200]$ and fixed total utilization U. The results refer to the absolute response-time jitter (ARJ), which has been normalized with respect to task periods. Hence a value of 1 on task τ_i corresponds to a jitter equal to its period T_i. The results are reported in Figures 11–13, which show the jitter introduced by the algorithms for each individual task (tasks are ordered by increasing periods). It is worth observing that, when the periodic load is less than 0.7, both algorithms introduce about the same jitter, which linearly increases for tasks with longer periods. For higher loads, the algorithms exhibit the same behavior illustrated in Figure 10: while RM reduces the jitter of high priority tasks at the expenses of tasks with lower priority, EDF treats tasks more evenly, obtaining a significant reduction in the jitter of the tasks with long periods for a small increase in the jitter of tasks with shorter periods.

Another parameter that it is important to minimize in control applications is the input–output latency. Assuming that a control task τ_i acquires inputs at the beginning of each instance and delivers control outputs at the end, the maximum input–output latency is defined as

$$L_i = \max_k (f_{i,k} - s_{i,k}) \tag{8}$$

where $s_{i,k}$ and $f_{i,k}$ are the start time and finishing time of job $\tau_{i,k}$, respectively.

Cervin proved that EDF can always achieve a shorter input–output latency than RM, for any task. This is stated by the following theorem (Cervin, 2003).

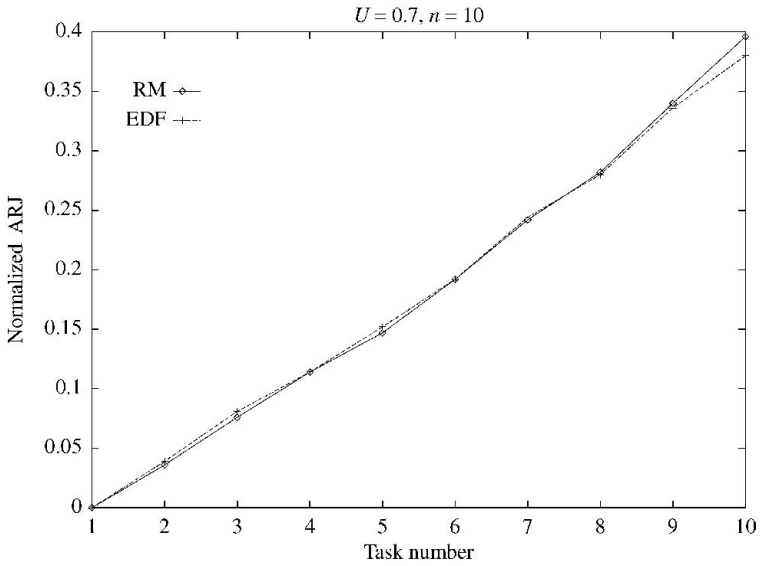

Figure 11. Average normalized jitter for $U = 0.7$.

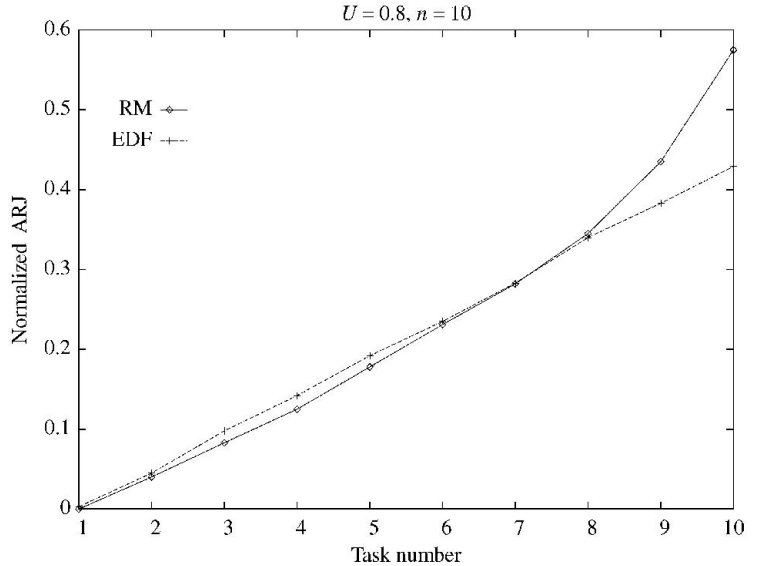

Figure 12. Average normalized jitter for $U = 0.8$.

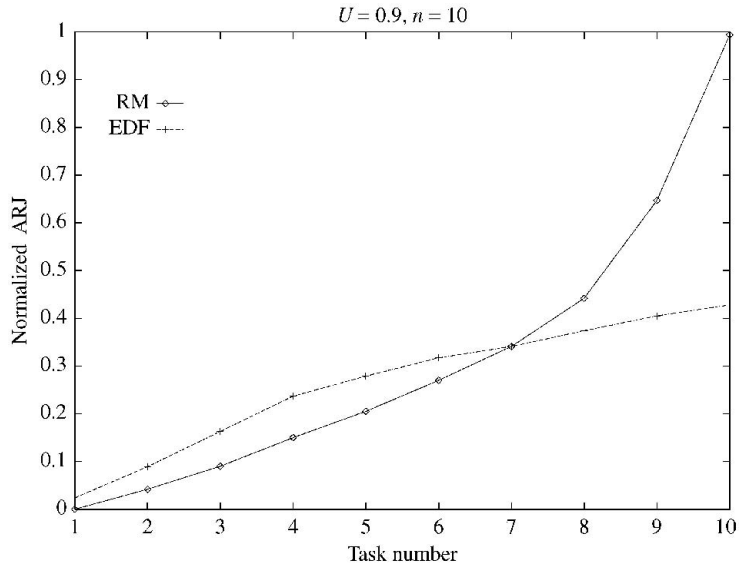

Figure 13. Average normalized jitter for $U = 0.9$.

THEOREM 2 *(Cervin) Given a set of n periodic control tasks performing input at the beginning of each job and output at the end, the maximum input–output latency of each task under EDF is shorter than or equal to the corresponding maximum latency under RM.*

Intuitively, the theorem is true because, under EDF, a task τ_i can never be preempted by tasks with a longer relative deadline, but can only be delayed, if the absolute deadline of the task with longer period is less than the absolute deadline of τ_i. However, such a start time delay does not affect the input–output latency. Moreover, under EDF, the number of preemptions experienced by each job is less than or equal to that experienced under RM. For example, in the schedule illustrated in Figure 10, the maximum input–output latencies of the tasks under RM are 2, 5, and 7, whereas under EDF are 2, 3, and 2, respectively.

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 developed by Sha et al. (1990). For the reason that both PIP and PCP are well known in the real-time literature and were originally deviced 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 inheritance (DPI) (Stankovic et al., 1998), the dynamic priority ceiling (DPC) (Chen and Lin, 1990), the stack resource policy (SRP) (Baker, 1991), and the dynamic deadline modification (DDM) protocol (Jeffay, 1992).

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 (Lehoczky et al., 1987; Strosnider et al., 1995), and the sporadic server (Sprunt et al., 1989), 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 (1994, 1995, 1996).

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 (1992). 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.

Indeed, Tia et al. (1996) 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. The same is not true for EDF, as discussed below. In particular, the following theorems were proved in Tia et al. (1996):

THEOREM 3 *(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 4 *(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 3 applies both to clairvoyant and on-line algorithms, whereas Theorem 4 only applies to on-line algorithms. These results are not true for EDF, where otpimal algorithms have been found for minimizing aperiodic response times. In particular, the improved total bandwith server (ITB) (Buttazzo and Sensini, 1999) 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 14 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

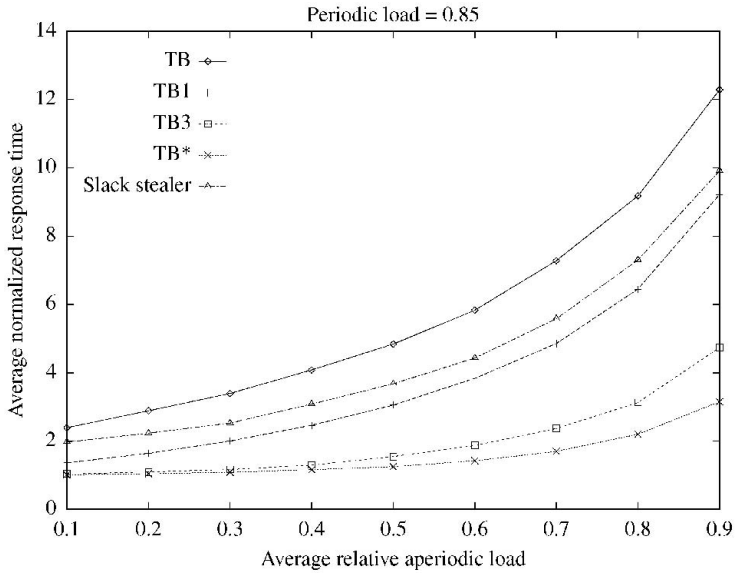

Figure 14. Performance of the TB(N) against the slack stealer.

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.

When on-line admission control of aperiodic tasks is performed through a utilization bound, the difference between fixed and dynamic priority scheduling is even more significant. In fact, Abdelzaher et al. (2004) proved that the aperiodic utilization bound is $(2 - \sqrt{2}) \simeq 0.586$ for deadline monotonic scheduling, and 1 for EDF.

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 et al. (1993), 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 (Abeni and Buttazzo, 1998; 2004). 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 RM and the 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, the response time jitter, 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 (Buttazzo and Stankovic, 1995) and periodic (Koren and Shasha, 1995) load. Also a method for jitter control under EDF has been addressed in Baruah et al. (1999) and can be adopted whenever needed.

Notes

1. 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.

References

Abdelzaher, T., Sharma, V., and Lu, C. 2004. A utilization bound for aperiodic tasks and priority driven scheduling. *IEEE Transactions on Computers* 53(3): 334–350.

Abeni, L., and Buttazzo, G. 1998. Integrating multimedia applications in hard real-time systems. In *Proceedings of the IEEE Real-Time Systems Symposium.* Madrid, Spain.

Abeni, L., and Buttazzo, G. 2004. Resource reservation in dynamic real-time systems. *Real-Time Systems* 27(2): 123–167.

Audsley, N. C., Burns, A., Richardson, M., Tindell, K., and Wellings, A. 1993. Applying new scheduling theory to static priority preemptive scheduling. *Software Engineering Journal* 8(5): 284–292.

Baker, T. P. 1991. Stack-based scheduling of real-time processes. *Journal of Real-Time Systems* 3(1): 76–100.

Baruah, S., Buttazzo, G., Gorinsky, S., and Lipari, G. 1999. Scheduling periodic task systems to minimize output jitter. In *Proceedings of the 6th IEEE International Conference on Real-Time Computing Systems and Applications.* Hong Kong.

Baruah, S. K., Howell, R. R., and Rosier, L. E. 1990. Algorithms and complexity concerning the preemptive scheduling of periodic real-time tasks on one processor. *Real-Time Systems* 2.

Bini, E., Buttazzo, G. C., and Buttazzo, G. M. 2001. A hyperbolic bound for the rate monotonic algorithm. In *Proceedings of the 13th Euromicro Conference on Real-Time Systems*. Delft, The Netherlands, pp. 59–66.

Bini, E., and Buttazzo, G. C. 2002. The space of rate monotonic schedulability. In *Proceedings of the 23rd IEEE Real-Time Systems Symposium.* Austin, Texas.

Buttazzo, G. C. 1993. HARTIK: a real-time kernel for robotics applications. In *Proceedings of the 14th IEEE Real-Time Systems Symposium.* Raleigh-Durham.

Buttazzo, G. C. 2003. Rate monotonic vs. EDF: Judgment day. In *Proceedings of the 3rd International Conference on Embedded Software*. Philadelphia, PA, pp. 67–83.

Buttazzo, G., and Stankovic, J. 1995. Adding robustness in dynamic preemptive scheduling. In Fussell, D. S., and Malek, M. (eds), *Responsive Computer Systems: Steps Toward Fault-Tolerant Real-Time Systems*. Boston: Kluwer Academic Publishers.

Buttazzo, G., and Sensini, F. 1999. Optimal deadline assignment for scheduling soft aperiodic tasks in hard real-time environments. *IEEE Transactions on Computers* 48(10).

Buttazzo, G., Lipari, G., Caccamo, M., and Abeni, L. 2002. Elastic scheduling for flexible workload management. *IEEE Transactions on Computers* 51(3): 289–302.

Cervin, A. 2003. Integrated control and real-time scheduling. Doctoral Dissertation, ISRN LUTFD2/TFRT-1065-SE, Department of Automatic Control, Lund.

Cervin, A., Eker, J., Bernhardsson, B., and Arzen, K.-E. 2002. Feedback-feedforward scheduling of control tasks. *Real-Time Systems* 23(1): 25–53.

Chen, M. I., and Lin, J. K. 1990. Dynamic priority ceilings: a concurrency control protocol for real-time systems. *Journal of Real-Time Systems* 2.

Dertouzos, M. L. 1974. Control robotics: the procedural control of physical processes. *Information Processing* 74.

Gai, P., Lipari, G., Di Natale, M. 2001. A flexible and configurable real-time kernel for time predictability and minimal ram requirements. Technical Report, Scuola Superiore S. Anna, Pisa, RETIS TR2001-02.

Gai, P., Abeni, L., Giorgi, M., and Buttazzo, G. 2001. A new kernel approach for modular real-time systems development. In *Proceedings of the 13th Euromicro Conference on Real-Time Systems*. Delft (NL).

Jeffay, K. 1992. Scheduling sporadic tasks with shared resources in hard-real-time systems. In *Proceedings of IEEE Real-Time System Symposium*. pp. 89–99.

Jeffay, K., Stone, D. L., and Poirier, D. 1991. YARTOS: Kernel support for efficient, predictable real-time systems. *Real-Time Systems Newsletter* 7(4): 8–13 (Republished in Halang, W., and Ramamritham, K. (eds), *Real-Time Programming*. Pergamon Press, Oxford, UK, 1992).

Joseph, M., and Pandya, P. 1986. Finding response times in a real-time system. *The Computer Journal* 29(5): 390–395.

Koren, G., and Shasha, D. 1995. Skip-over: algorithms and complexity for overloaded systems that allow skips. In *Proceedings of the IEEE Real-Time Systems Symposium*.

Kuo, T.-W., and Mok, A. K. 1991. Load adjustment in adaptive real-time systems. *IEEE Real-Time Systems Symposium*.

Lehoczky, J. P., and Ramos-Thuel, S. 1992. An optimal algorithm for scheduling soft-aperiodic tasks in fixed-priority preemptive systems. In *Proceedings of the IEEE Real-Time Systems Symposium*.

Lehoczky, J. P., Sha, L., and Strosnider, J. K. 1987. Enhanced aperiodic responsiveness in hard real-time environments. In *Proceedings of the IEEE Real-Time Systems Symposium*. pp. 261–270.

Lehoczky, J. P., Sha, L., and Ding, Y. 1989. The rate-monotonic scheduling algorithm: exact character-ization and average case behavior. In *Proceedings of the IEEE Real-Time Systems Symposium*. pp. 166–171.

Liu, C. L., and Layland, J. W., 1973. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the ACM* 20(1): 40–61.

Locke, J. 1997. Designing real-time systems. In *IEEE International Conference of Real-Time Computing Systems and Applications (RTCSA '97)*. Taiwan (invited talk).

Marti, P., Fohler, G., Ramamritham, K., and Fuertes, J. M. 2002. Control performance of flexible timing constraints for quality-of-control scheduling. In *Proceedings of the 23rd IEEE Real-Time System Symposium*. Austin, TX, USA.

Mercer, C. W., Savage, S., and Tokuda, H. 1993. Processor capacity reserves for multimedia operating systems. Technical Report, Carnegie Mellon University, Pittsburg, PA, CMU-CS-93-157.

Rivas, M. A., and Harbour, M. G. 2001. POSIX—compatible application-defined scheduling in MaRTE OS. In *Euromicro Conference on Real-Time Systems (WiP)*. Delft, The Netherlands.

Sha, L., Rajkumar, R., and Lehoczky, J. P. 1990. Priority inheritance protocols: an approach to real-time synchronization. *IEEE Transactions on Computers* 39(9): 1175–1185.

Sprunt, B., Sha, L., and Lehoczky, J. 1989. Aperiodic task scheduling for hard real-time system. *Journal of Real-Time Systems* 1: 27–60.

Spuri, M., and Buttazzo, G. C. 1994. Efficient aperiodic service under earliest deadline scheduling. In *Proceedings of IEEE Real-Time System Symposium*. San Juan, Portorico.

Spuri, M., and Buttazzo, G. C. 1996. Scheduling aperiodic tasks in dynamic priority systems. *Real-Time Systems* 10(2).

Spuri, M., Buttazzo, G. C., and Sensini, F. 1995. Robust aperiodic scheduling under dynamic priority systems. In *Proceedings of the IEEE Real-Time Systems Symposium*. Pisa, Italy.

Stankovic, J., Ramamritham, K., Spuri, M., and Buttazzo, G. 1998. *Deadline scheduling for real-time systems*. Dordrecht: Kluwer Academic Publishers.

Stankovic, J., and Ramamritham, K. 1987. The design of the spring kernel. In *Proceedings of the IEEE Real-Time Systems Symposium*.

Strosnider, J. K., Lehoczky, J. P., and Sha, L. 1995. The deferrable server algorithm for enhanced aperiodic responsiveness in hard real-time environments. *IEEE Transactions on Computers* 44(1).

Tia, T.-S., Liu, J. W.-S., and Shankar, M. 1996. Algorithms and optimality of scheduling aperiodic requests in fixed-priority preemptive systems. *Journal of Real-Time Systems* 10(1): 23–43.

Giorgio Buttazzo is an associate professor of Computer Engineering at the University of Pavia, Italy. He graduated in Electronic Engineering at the University of Pisa in 1985, received a Master in Computer Science degree at the University of Pennsylvania in 1987, and a Ph.D. in Computer Engineering at the Scuola Superiore S. Anna of Pisa in 1991. During 1987, he worked on active perception and real-time control at the G.R.A.S.P. Laboratory of the University of Pennsylvania, in Philadelphia. From 1991 to 1998, he held a position of Assistant Professor at the Scuola Superiore S. Anna of Pisa, doing research on robot control systems and real-time scheduling. His main research interests include real-time operating systems, dynamic scheduling algorithms, quality of service control, multimedia systems, advanced robotics applications and neural networks. He is a member of the IEEE and the IEEE Computer Society.