Proceedings of the 16th Real-Time Systems Symposium (RTSS 95), Pisa, Italy, pp. 90-99, December 1995.

Value vs. Deadline Scheduling in Overload Conditions

Giorgio Buttazzo, Marco Spuri, Fabrizio Sensini

Scuola Superiore S. Anna
via Carducci, 40 - 56100 Pisa - Italy
giorgio@sssup1.sssup.it, spuri@sssup2.sssup.it

Abstract

In this paper we present a comparative study among scheduling algorithms which use different priority assignments and different guarantee mechanisms to improve the performance of a real-time system during overload conditions. In order to enhance the quality of service, we assume that tasks are characterized not only by a deadline, but also by an importance value. The performance of the scheduling algorithm is then evaluated by computing the cumulative value gained on a task set, i.e. the sum of the values of those tasks that completed by their deadline.

The purpose of this simulation study was twofold. Firstly, we wanted to discover which priority assignment is able to achieve the best performance in overload conditions. Secondly, we were interested in understanding how the pessimistic assumptions made in the guarantee test affect the performance of the scheduling algorithms, and how much a reclaiming mechanism can compensate this degradation.

Simulation results show that, without any admission control, value-density scheduling performs best. Simple admission control based on worst case estimates of the load worsen the performance of all value based algorithms. EDF scheduling performs best if admission control is used along with a reclaiming mechanism that takes advantage of early completions. Finally, scheduling by deadline before overload and by value during overload works best in most practical conditions.

1 Introduction

In a real-time system, a task is usually characterized by a deadline, i.e., the latest time by which the task must complete to produce useful results. In such a system, the objective of the scheduling algorithm is to execute a set of tasks so that all deadlines are met. In this case, the schedule is said to be *feasible*.

In the literature, we find a number of optimal scheduling algorithms that guarantee the feasibility of the schedule under specific assumptions. A scheduling algorithm is said to be optimal if it fails to meet a deadline only if no other scheduling algorithms can produce a feasible schedule. For example, the *Earliest Deadline First* (EDF) algorithm [8] is optimal in the sense that if a task set cannot be feasibly scheduled by EDF, then it cannot be feasibly scheduled by any other priority assignment [5].

When a real-time system is overloaded, however, not all tasks can be completed by their deadlines. Therefore the objective of the scheduling algorithm should be to feasibly schedule at least the most important ones. In order to specify the importance of each task we can add an importance value to the parameters that characterize it. The performance of the scheduling algorithm is then evaluated by computing the cumulative value gained on a task set, i.e. the sum of the values of those tasks that completed by their deadline.

Unfortunately, in overload conditions there are not optimal on-line algorithms that can maximize the cumulative value of a generic task set, hence scheduling decisions must be made using best-effort algorithms, whose objective is to complete the most important tasks by their deadline, avoiding negative phenomena, such as the so called *domino effect*. This happens when the first task that misses its deadline may cause all subsequent tasks to miss their deadlines.

For example, experiments carried out by Locke [9] have shown that EDF is prone to the domino effect and it rapidly degrades its performance during overload intervals. This is due to the fact that EDF gives the highest priority to those processes that are close to

missing their deadlines.

In such a situation, EDF does not provide any type of guarantee on which tasks will meet their timing constraints. This is a very undesirable behavior in practical systems, since in real-world applications intermittent overloads may occur due to exceptional situations, such as modifications in the environment, arrival of a burst of tasks, or cascades of system failures.

A number of heuristic algorithms has been proposed to deal with overloads [2, 3, 4, 6, 7, 10, 11, 13, 15]. They all improve the performance of EDF, however, very few simulation studies have been done to evaluate the importance of the scheduling policy with respect to the guarantee mechanism used to deal with the overload.

Baruah et al. [1] have shown that there exists an upper bound on the performance of any on-line (preemptive) algorithm working in overload conditions. The "goodness" of an on-line algorithm is measured with respect to a clairvoyant scheduler (one that knows the future), by means of the *competitive factor*, which is the ratio r of the cumulative value guaranteed by the on-line algorithm to the cumulative value achieved by the clairvoyant schedule. The value associated to each task is equal to the task's execution time if the task request is successfully scheduled to completion; a value of zero is given to tasks that do not terminate within their deadline. According to this metric, they proved the following theorem:

Theorem 1 *There does not exist an on-line scheduling algorithm with a competitive factor greater than 0.25.*

What the theorem says is that no on–line scheduling algorithm can guarantee a cumulative value greater than $1/4th$ the value obtainable by a clairvoyant scheduler.

It is worth pointing out, however, that the above bound is achieved under very restrictive assumptions, such as all tasks in the set have zero laxity, the overload can have an arbitrary (but finite) duration, task's execution time can be arbitrarily small, and task value is equal to computation time. Since in most real world applications tasks characteristics are much less restrictive, the $1/4th$ bound has only a theoretical validity and more work is needed to derive other bounds based on more knowledge of the task set [12].

In this paper we present a comparative study among scheduling algorithms which use different priority assignments to keep the cumulative value high and adopt different guarantee mechanisms to avoid the domino effect and achieve graceful degradation during transient overloads. A robust version of these algorithms is also proposed and simulated to see how these algorithms perform in real situations in which tasks execute less than their worst case computation time.

The purpose of this simulation study was twofold. First, we wanted to discover which priority assignment is able to achieve the best performance in overload conditions, e.g., whether it is convenient to schedule the tasks based on pure values, pure deadlines, or on a suited mixture of both. The second aspect we were interested in was to understand how and how much the guarantee algorithm influences the performance of the system during transient overloads.

To answer these questions, in our study we have considered four classical priority assignments, that we have tested using three different guarantee mechanisms, thus comparing a total number of twelve scheduling algorithms.

2 Terminology and Assumptions

Before we describe the scheduling algorithms we have considered in our performance study, we define the following notation to refer the parameters of task J_i:

a_i denotes the arrival time, i.e., the time at which the task is activated and becomes ready to execute.

C_i denotes the maximum computation time, i.e., the worst case execution time needed for the processor to execute the task without interruption.

c_i denotes the dynamic computation time, i.e., the remaining worst case execution time needed, at the current time, to complete the task without interruption.

d_i denotes the absolute deadline, i.e., the time by which the task should complete its execution to produce a valuable result.

D_i denotes the relative deadline, i.e., the time interval between the arrival time and the absolute deadline.

V_i denotes the task value, i.e., the relative importance of task J_i with respect to the other tasks in the set.

f_i denotes the finishing time, i.e., the time at which the task completes its execution and leaves the system.

We assume that at its arrival, each task is characterized by the following parameters:

$$J_i(C_i, D_i, V_i).$$

Moreover, we assume that tasks are preemptable and their arrival times are not known in advance.

To evaluate the performance of a scheduling algorithm in underload and in overload conditions, we associate to each task J_i a *worth value* v_i defined as follows:

$$v_i = \begin{cases} V_i & \text{if } f_i \leq d_i \\ 0 & \text{otherwise} \end{cases}$$

This means that, if task J_i is completed within its deadline d_i, the algorithm gains a value equal to V_i, otherwise it gains a value equal to zero.

Finally, the performance of a scheduling algorithm A on the task set **J** is evaluated by computing the Cumulative Value (Γ_A), defined as the sum of all worth values v_i gained during the task set execution:

$$\Gamma_A = \sum_{i=1}^{n} v_i$$

3 Algorithms Description

The four priority assignments we have considered in our performance study are the following:

EDF (*Earliest Deadline First*) Task priority is assigned as $p_i = 1/d_i$. Hence, the highest priority task is that one with the earliest absolute deadline.

HVF (*Highest Value First*) Task priority is assigned as $p_i = V_i$. Hence, the highest priority task is that one with the highest importance value.

HDF (*Highest Density First*) Task priority is assigned as $p_i = V_i/c_i$. Hence, the highest priority task is that one with the highest value density V_i/c_i.

MIX (*Mixed rule*) Importance value and deadline are both considered in assigning the task priority, which is computed as a weighted sum of the value and the deadline: $p_i = \alpha V_i - (1 - \alpha)d_i$. Notice that, although this priority assignment depends on the absolute deadline, the ready queue ordering is time independent.

The four scheduling algorithms described above (EDF, HVF, HDF, MIX) will be referred in the following as *plain algorithms*, since they do not provide any form of guarantee. The lack of load awareness makes them prone to domino effect in overload conditions.

To handle overload situations in a more predictable way, the four plain algorithms have been extended in two additional classes: a class of *guaranteed* algorithms, characterized by an acceptance test, and a class of *robust* algorithms, characterized by a more sophisticated rejection strategy and a reclaiming mechanism.

Within the guaranteed class, each algorithm performs an acceptance test at each task activation, so that if an overload is detected the newly arrived task is rejected. The acceptance test allows to avoid the *domino effect* by keeping the actual workload always less than one. However, it does not consider importance values since it always removes the newly arrived task, even though it is the most important one. Another problem with this guarantee mechanism is that the system does not take advantage of early completions: once a task is rejected (based on a pessimistic evaluation of the load) it cannot be recovered when some tasks terminate earlier than their worst case finishing time.

To solve these problems, the algorithms in the robust class perform a rejection based on importance values and include a reclaiming mechanism which allows to apply a recovery strategy to the rejected tasks. In particular, each algorithm in the robust class also performs a guarantee test at each task activation. However, if an overload is detected the least value task that can remove the overload is rejected. Robust algorithms also include a resource reclaiming mechanism which takes advantage of the unused processor time deriving from early terminations. To realize this reclaiming, rejected tasks are not removed, but temporarily parked in a reject queue, from which they can be possibly recovered later on. At each early completion, hence, the system executes the guarantee test again to attempt a recovery of rejected tasks with the highest values. An example of robust algorithm having these features has been described by Buttazzo and Stankovic in [3].

A summary of the twelve algorithms described above, is shown in Table 1, which also provides a useful scheme for understanding the simulation experiments presented in the following section.

Plain	Guaranteed	Robust
EDF	GEDF	REDF
HVF	GHVF	RHVF
HDF	GHDF	RHDF
MIX	GMIX	RMIX

Table 1: Scheme of the tested algorithms: priority assignments vs. guarantee mechanisms.

4 Performance evaluation

In this section we present the performance results obtained by simulating the scheduling algorithms described in Table 1. Our first aim was to compare the behaviour of the four priority assignments at different load conditions. In order to do so, we performed three sets of simulation experiments, each one dedicated to a particular scheduling class (plain, guaranteed, and robust). In a second set of experiments, we tested the effectiveness of the guarantee mechanism and the robust recovery strategy over the plain priority-based scheduling scheme.

In summary, the simulation experiments we have conducted are well synthesized in Table 1. Each row of the table represents a performance study which compares different scheduling classes using the same priority assignment, whereas each column describes a simulation experiment conducted within the same scheduling class on different priority assignments.

Each plot on the graphs shown in this section represents the average of a set of 100 independent simulations, the duration of each was chosen to be 300,000 time units long. All algorithms have been executed on task sets consisting of 100 aperiodic tasks, whose parameters were generated as follows.

- The worst-case execution time C_i was chosen as a random variable with uniform distribution between 50 and 350 time units.
- The average interarrival time T_i of each task was computed to produce a given workload ρ. In particular, it was modeled as a Poisson distribution with average value equal to

$$T_i = \frac{N \cdot C_i}{\rho}$$

being N the total number of tasks in the set. The load ρ was computed based on task parameters (arrival time, deadline, and worst case execution time), as described in [3].

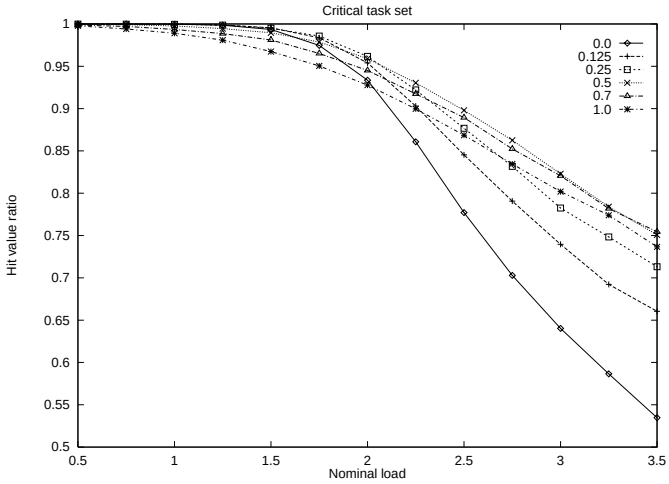


Figure 1: Performance of the MIX priority assignment as a function of α.

- The laxity of a task was computed as a random value with uniform distribution from 150 and 1850 time units (i.e., with an average of 1000 time units).
- The relative deadline of a task was computed as the sum of its worst-case execution time and its laxity.
- Unless otherwise stated, the actual execution time of a task was computed as a random variable with uniform distribution between zero and its worst-case execution time. In this way, the average execution time of a task was equal to a half of the worst-case execution time.

To reduce the number of simulations, we first measured the performance of MIX as a function of α. As shown in figure 1, the priority assignment which provides the best cumulative value for almost any load condition is the one obtained with $\alpha = 0.5$. As a consequence, all simulation experiments involving the MIX priority assignment have been done with this value of α.

4.1 Column Experiments

The priority assignments belonging to each scheduling class have been tested under two different task set situations, identified as *random set* and *linear set* respectively. In the random set, the importance value of each task is independent from any other task parameter. In particular, the task value was generated as a random variable ranging from 150 to 1850 (being 1000

the average value). This range is the same as that one chosen for the deadlines. In the linear set, the importance value of each task is proportional to its relative deadline. Notice that this situation is critical for all algorithms, since the important tasks have a higher probability to miss their deadline. The motivation of using two different task sets is that we wanted to see how sensitive an algorithm is with respect to the task parameters and how rapidly it degrades.

In all graphs, the independent variable on the X-axis is the *nominal load*, i.e., the workload estimated based on the worst-case execution times. The nominal load ranges from 0.5 to 3.5. The result reported on the Y-axis is the *Hit Value Ratio (HVR)*, i.e., the ratio between the cumulative value obtained by an algorithm A and the total value of the task set:

$$HVR = \frac{\Gamma_A}{\sum_{i=1}^{n} V_i}$$

For each simulation, the standard deviations of the hit value ratios were computed and they were never greater than 1%.

4.1.1 Plain class

Figures 2a and 2b show the results of a simulation conducted on the plain scheduling class. The graphs of figure 2a concern the case of a random task set, with independent importance values. It can be noticed that, for low load conditions, EDF shows its optimality, whereas the other three algorithms have about the same performance, achieving a resulting Hit Value Ratio greater than 95 percent of the total value. As soon as the nominal load approaches the value of two (which corresponds to an actual load of one), EDF performance falls down, while the other algorithms degrade more gracefully. It is worth to notice that, although MIX is defined as a linear combination of EDF and HVF, in overload conditions, its behaviour is not an average of their performance. On the contrary, MIX performs better than both for any load.

The best behaviour for high overloads is achieved by HDF, which however has the disadvantage of a heavier overhead, due to the dynamic priorities, which depend on the remaining execution time.

The graphs in figure 2b show the situation for a linear task set, where importance values are proportional to deadlines. In particular, the least sensitive algorithms with respect to task set variations are EDF and HDF, whereas HVF and MIX are more influenced by the task parameters. We can summarize the results of this experiment in the following observation.

Observation 1 *Without any guarantee mechanism, the most effective priority assignment in overload conditions is the one based on value density, namely HDF. It exhibits a very graceful degradation during overloads and it is not much sensitive to task set parameters.*

4.1.2 Guaranteed class

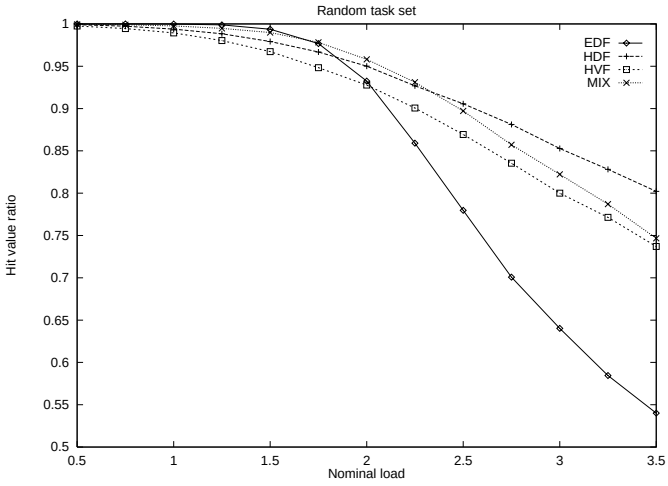
This experiment illustrates the performance of the same four priority assignments, which now include a guarantee mechanism consisting in the execution of an acceptance test at each task activation. Whenever a new task is activated, the guarantee routine estimates the schedule based on the current scheduling algorithm and on the nominal task parameters, and verifies whether some tasks could miss their deadline. If a time-overflow is predicted, the newly arrived task is rejected, otherwise it is accepted and inserted into the ready queue.

We refer to these new versions of the algorithms as GEDF, GHVF, GHDF, and GMIX. Figure 3a shows the case of a random task set with independent importance values, whereas figure 3b is relative to the case of a linear task set with values proportional to deadlines. Notice that for both task sets and for any load conditions, GEDF is the most effective scheduling strategy. On the contrary, the density-based priority assignment, which was the most effective among the plain algorithms, degrades its performance if used with a guarantee mechanism.
The relevant result of this experiment can be synthesized in the following observation.

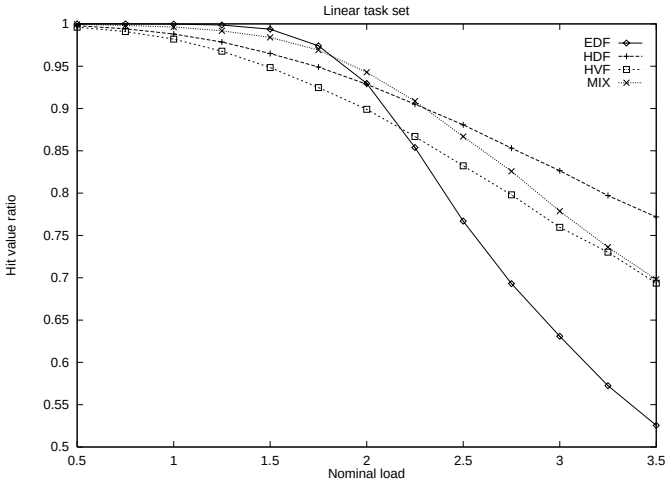
Observation 2 *If the system workload is controlled at each task activation by an acceptance test which under overload conditions rejects the newly arrived task, then the most effective priority assignment is EDF.*

One problem with this form of guarantee is that it is too pessimistic. In fact, since the workload is estimated based on pessimistic parameters (such as the worst case execution times of the tasks), a task could be rejected even though it would have completed in time. Another problem is that a task is rejected regardless of its importance value. For example, a better policy is to reject the least value task that can remove the overload condition.

These aspects are treated in the robust class of scheduling algorithms, whose performance is illustrated next.

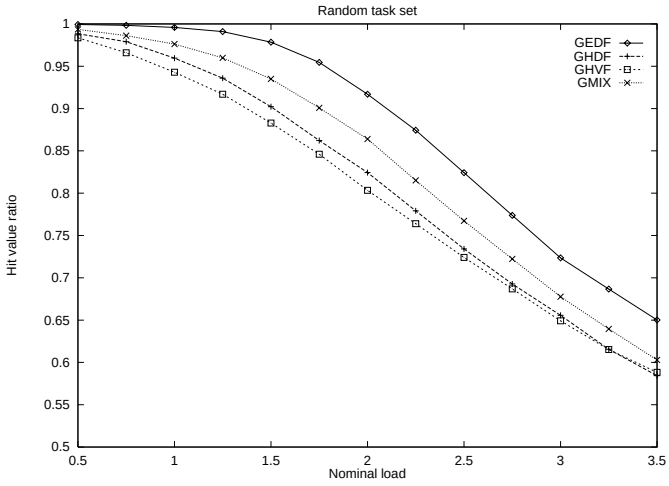


(a)

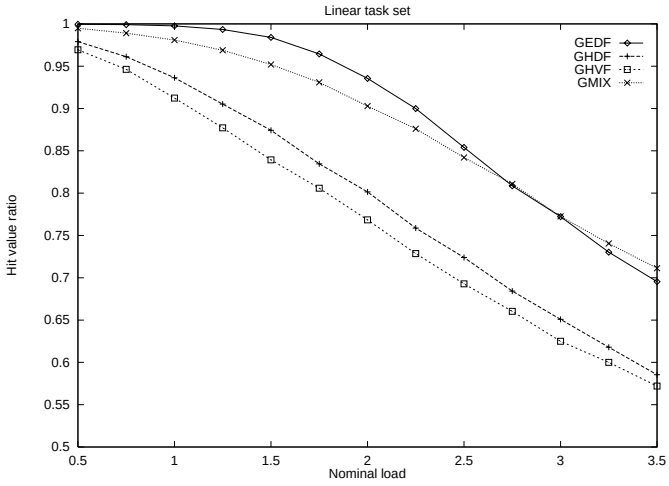


(b)

Figure 2: Performance of the plain algorithms under a random set (a) and a linear set (b)



(a)



(b)

Figure 3: Performance of the guaranteed algorithms under a random set (a) and a linear set (b)

4.1.3 Robust class

The results concerning the performance of the robust algorithms are shown in figures 4a, for the case of a random task set, and in figure 4b, for the case of a linear task set. As a first remark, we can observe that all robust algorithms show a graceful degradation as the load increases and achieve a similar behaviour in the range of overload conditions we have simulated. This result suggests that the performance of the robust algorithms is close to the best one achievable by a on-line algorithm. The improvement obtained by the robust algorithms with respect to the other versions has been evaluated by a specific set of experiments illustrated in the following section.

In both graphs, REDF shows the best performance for almost any load. However, it is worth to notice that for high overloads (nominal load greater than 3) RHDF performs slightly better than REDF. This means that, when the processing demand is much greater than the available processing time and a high percentage of tasks must be rejected, value density ordering intrinsically tries to save the highest cumulative value of the current requests.

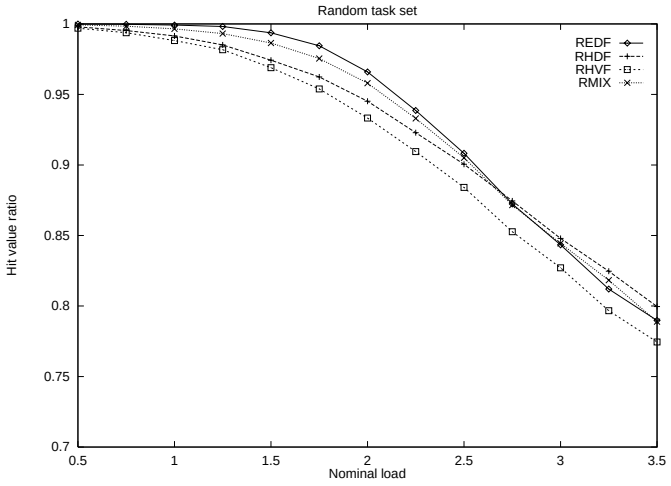
The following observation summarizes the main result of this experiment.

Observation 3 *Among the robust strategies, no algorithm is able to perform better than the others for any load. However, REDF is the most effective algorithm in most practical situations, whereas for very high overloads RHDF seems to be the strategy which gains the highest Cumulative Value.*

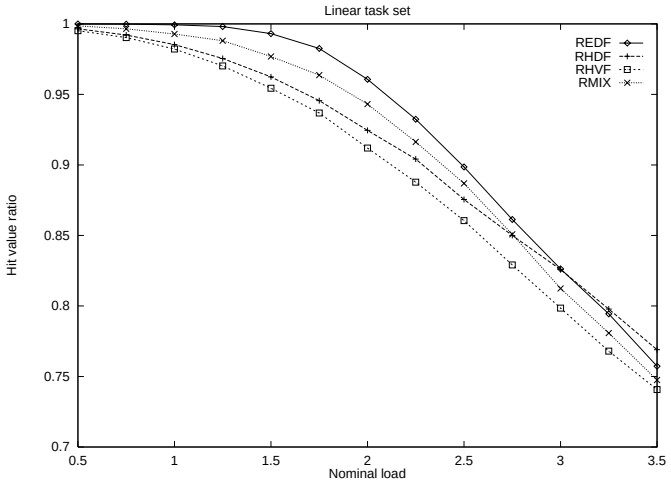
4.2 Rows Experiments

A second set of experiments was conducted along the rows of Table 1 to test the effectiveness of the guaranteed and the robust algorithms with respect to the plain algorithms. In particular, we wanted to see how the pessimistic assumptions made in the guarantee test affect the performance of the algorithms, and how much the reclaiming mechanism introduced in the robust class can compensate this degradation. In order to test these effects, we monitored the Hit Value Ratio obtained by the algorithms as the tasks finish earlier and earlier with respect to their worst case finishing time. The specific parameter we varied in our simulations was the average *Unused Computation Time Ratio* β, defined as follows:

$$\beta = 1 - \frac{Actual\ Computation\ Time}{Worst\ Case\ Computation\ Time}$$



(a)



(b)

Figure 4: Performance of the robust algorithms under a random set (a) and a linear set (b)

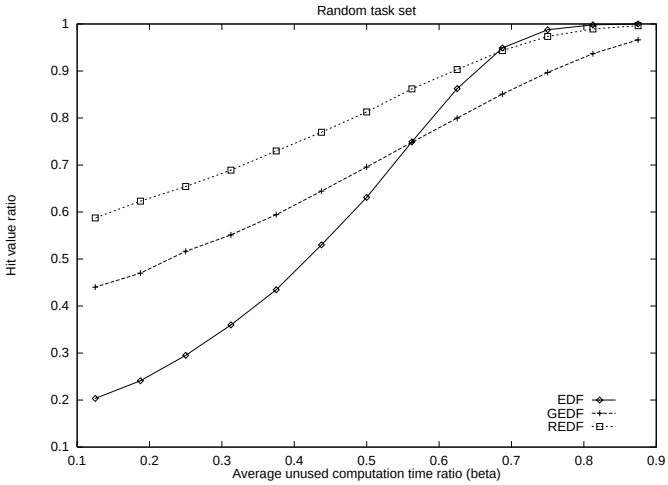


Figure 5: Performance of the EDF priority assignment with a random set

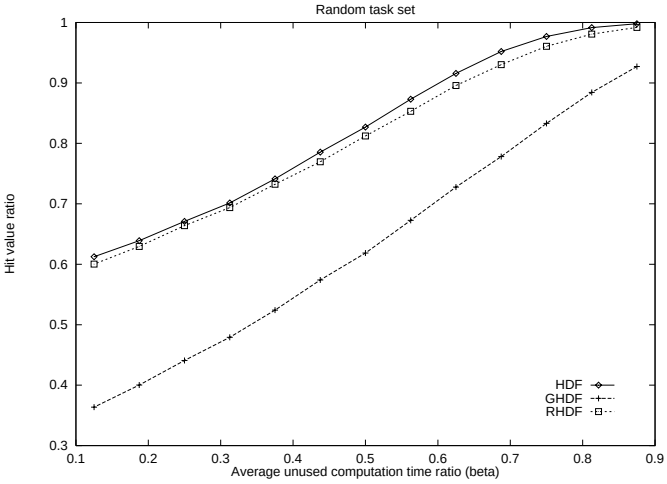


Figure 6: Performance of the HDF priority assignment with a random set

In all graphs, the tasks set was generated with a constant nominal load of 3, while the Average Unused Computation Time Ratio was varied from 0.125 to 0.875. Notice that, as a consequence, the actual mean load changes from a value of 2,635 to a value of 0.375, thus ranging over very different actual load conditions.

As in the previous experiments, each algorithm was tested with the random task set and the linear task set. However, only the results with the random task set are shown, since the experiments with the linear set did not exhibit significant difference.

Figure 5 shows the results obtained with the deadline priority assignment. Under high load conditions, that is, when tasks execute for almost their maximum computation time, GEDF and REDF are able to obtain a significant improvement compared to the plain EDF scheduling. Increasing the unused computation time, however, the actual load falls down and the plain EDF performs better and better, reaching the optimality in underload conditions. Notice that, as the system becomes underloaded ($\beta \simeq 0.7$), GEDF becomes less effective than EDF. This is due to the fact that GEDF performs a worst-case analysis, thus rejecting tasks which still have some chance to execute within their deadline. This phenomenon does not appear on REDF, because the reclaiming mechanism implemented in the robust algorithm is able to recover the rejected tasks that can complete in time.

In figure 6 the performance of the density-based priority assignments was compared. The most relevant observation we can make from these plots is that the guarantee mechanism worsens the performance of the plain HDF algorithm for any load. This

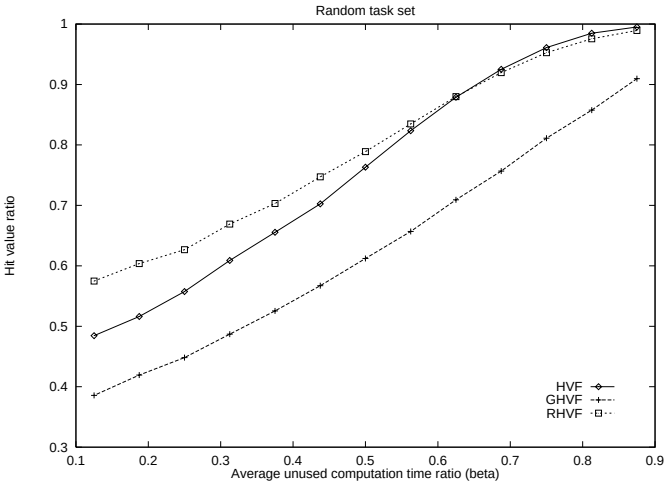


Figure 7: Performance of the HVF priority assignment with a random set

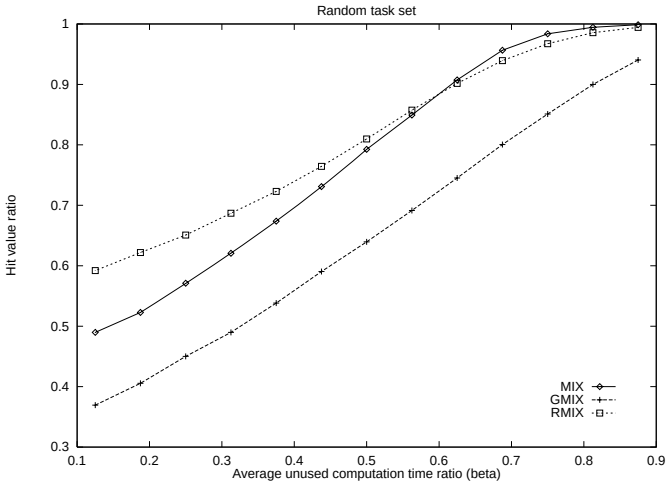


Figure 8: Performance of the MIX priority assignment with a random set

happens because, when the guarantee test fails, the newly arrived task is rejected regardless of its value. Plain scheduling, instead, automatically rejects tasks with very low values, as a consequence of the priority scheme adopted. Notice that the reclaiming mechanism used in the robust algorithm (RHDF) is able to compensate the performance degradation caused by the rejection policy, reaching the same performance of the plain scheduling scheme.

The experiments relative to the MIX and HVF based schedulers are shown in figures 7 and 8. The performance shown by these algorithms is quite similar to that obtained by HDF, as far as the guarantee scheme is concerned. Similarly to the case of the HDF-based scheduler (figure 6), the acceptance test executed at each arrival time worsens the performance achieved by the plain class algorithms. The robust scheduling schemes, instead, are able to enhance the performance of their plain versions for small values of β (i.e., for high load values).

The main results of these experiments can be summarized in the following observation.

Observation 4 *The acceptance test executed at each arrival time by the guarantee mechanism worsens the performance of all priority assignments that consider importance values in their ordering discipline. On the contrary, the robust scheduling schemes perform very well both in overload and in underload conditions, proving that the reclaiming strategy is effective for increasing the Cumulative Value in all practical situations.*

5 Conclusions

In this paper we have presented a comparative study among four priority assignments which use values and deadlines to achieve graceful degradation and improve the performance in overload conditions. The four algorithms have been simulated in three versions, which differ from the guarantee mechanism used to detect the overload and select a rejection. Simulation experiments proved that the robust version of these algorithms is the most flexible one, due to the resource reclaiming strategy, which is able to take advantage of early completions, when tasks execute less than their worst case computation time.

One important result derived from this simulation study, is that scheduling by deadlines and rejecting by value (as done by the REDF algorithm) proved to be the most effective strategy for a wide range of overload conditions. However, it is worth pointing out that the REDF strategy is not the best one for all load situations. When the system is underload, in fact, EDF is optimal, whereas for very high overloads RHDF performs slightly better than REDF. This fact suggests that a further improvement on the Cumulative Value could be obtained if the system were able to change its scheduling strategy depending on the current workload, using, for instance, EDF in underload condition, REDF for normal overloads, and RHDF for high overloads.

References

[1] S. Baruah, G. Koren, D. Mao, B. Mishra, A. Raghunathan, L. Rosier, D. Shasha, and F. Wang, "On the Competitiveness of On-Line Real-Time Task Scheduling," *Proceedings of IEEE Real-Time Systems Symposium*, December 1991.

[2] S. Biyabani, J. Stankovic, and K. Ramamritham, "The Integration of Deadline and Criticalness in Hard Real-Time Scheduling," *Proceedings of the IEEE Real-Time Systems Symposium*, December 1988.

[3] G. Buttazzo and J. Stankovic, "RED: A Robust Earliest Deadline Scheduling Algorithm", *Proc. of 3rd International Workshop on Responsive Computing Systems*, Austin, 1993.

[4] S. Cheng, J. Stankovic, and K. Ramamritham, "Dynamic Scheduling of Groups of Tasks with Precedence Constraints in Distributed Hard

Real-Time Systems," *Real-Time Systems Symposium*, December 1986.

[5] M.L. Dertouzos, "Control Robotics: the Procedural Control of Physical Processes," *Information Processing 74*, North-Holland Publishing Company, 1974.

[6] J. R. Haritsa, M. Livny, and M. J. Carey, "Earliest Deadline Scheduling for Real-Time Database Systems," *Proceedings of Real-Time Systems Symposium*, December 1991.

[7] G. Koren and D. Shasha, "D-over: An Optimal On-Line Scheduling Algorithm for Overloaded Real-Time Systems," *Proceedings of the IEEE Real-Time Systems Symposium*, December 1992.

[8] C.L. Liu and J.W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment", *Journal of the ACM* 20(1), 40–61, 1973.

[9] C. D. Locke, "Best-effort Decision Making for Real-Time Scheduling," PhD thesis, Computer Science Department, Carnegie-Mellon University, 1986.

[10] R. McNaughton, "Scheduling With Deadlines and Loss Functions," *Management Science*, 6, pp. 1-12, 1959.

[11] J. Stankovic and K. Ramamritham, "The Spring Kernel: A New Paradigm for Real-Time Systems," *IEEE Software*, Vol. 8, No. 3, pp. 62-72, May 1991.

[12] J. Stankovic, M. Spuri, M. Di Natale and G. Buttazzo, "Implications of Classical Scheduling Results for Real-Time Systems", IEEE Computer, to appear.

[13] P. Thambidurai and K. S. Trivedi, "Transient Overloads in Fault-Tolerant Real-Time Systems," *Proceedings of Real-Time Systems Symposium*, December 1989.

[14] T.-S. Tia, J. W.-S. Liu and M. Shankar, "Algorithms and Optimality of Scheduling Aperiodic Requests in Fixed-Priority Preemptive Systems", *The Journal of Real-Time Systems*, 1994.

[15] G. Zlokapa, J. A. Stankovic, and K. Ramamritham, "Well-Timed Scheduling: A Framework for Dynamic Real-Time Scheduling," submitted to *IEEE Transactions on Parallel and Distributed Systems*, 1991.