Heuristic Use of Singularities for On-Line Scheduling of Real-Time Mandatory/Reward-Based Optional Systems

R. M. Santos, J. Urriza, J. Santos and J. Orozco

Universidad Nacional del Sur/CONICET
8000 Bahía Blanca, Argentina
(iesantos@criba.edu.ar)

Abstract

The paper addresses the problem of on-line scheduling of mandatory/reward-based systems in which tasks have a hard real-time mandatory part and an optional part with a non-decreasing reward function associated to its execution. Four methods, generically called SH, are proposed. They are based on the detection of singularities, special instants that appear along the execution of the system. By applying some heuristic rules, the singularities and some of the following slots are used to process optional parts. To the best of the authors' knowledge, papers published up to now on the subject of reward maximization require the functions to be continuously differentiable. Moreover, they cannot be executed on-line. On the contrary, the only requirement of the SH methods is that the functions are computable at every instant. After the system is proved to be schedulable by any of the available exact test off-line techniques, SH methods can be executed on-line. Their performance is evaluated using simulations performed on a synthetic set of tasks proposed in one of the outstanding papers on the subject and on sets of tasks randomly generated. The results are analysed and explained. The main conclusion is that, in all cases, the SH methods outperform the Best Incremental Return, often used as a yardstick.

1. Introduction

Research in Real-Time Systems covers several areas, *e.g.* operating systems, programming languages, architecture, fault tolerance, scheduling, etc. [16]. In the classical definition, Real-Time Systems are those in which results must be not only correct from an arithmetic-logical point of view but also produced before a certain instant, called *deadline*. If no deadline can be missed, the system is said to be *hard* as opposed to *soft* in which some deadlines may be missed. Scheduling theory addresses the problem of meeting the specified time constraints, which is the essence of the definition.

Recently, special attention has been focused on the scheduling of *mandatory/reward-based optional* systems in which tasks have a mandatory part that must always be executed meeting its deadline and an optional part with a non decreasing reward function associated with its execution [2, 3, 13]. Reward-based mandatory/ optional systems are so pervasive that they constitute a good candidate to be part of a component-based operating system of the type described in [5] for embedded applications. Maximizing the total reward is, in general, an NP-hard problem [4].

In [12] two methods based on the detection of *singularities*, special instants in the evolution of the system, were proposed for the processing of Rate Monotonic scheduled mixed sets of real and non real-time tasks. In [13] the simplest method, complemented with an heuristic rule, was proposed for the scheduling of embedded systems. In this paper, both methods, combined with two different heuristics, produce four extensions to address the problem of reward-based scheduling.

The rest of the paper is organized as follows: In Section 2, related work is analysed. In Section 3 the necessary and sufficient conditions for the RM schedulabity of hard real-time systems are presented and the notion of singularity introduced. In Section 4, the scheduling conditions are applied to mandatory/optional systems. In Section 5, the single and multiple singularities/heuristic methods are described together with the rationale behind each heuristic. In Section 6, their performance is evaluated *vs.* the Best Incremental Return method and, finally, in Section 7, conclusions are drawn.

2. Related Work

The integration of best effort and fixed priority scheduling was proposed in [1]. Partial computations had been used previously in traditional iterative-refinement algorithms for imprecise results in which the aim is to minimize the weighted sum of errors [8, 10, 15].

In [4], two implementable scheduling policies are proposed for a class of *Increasing Reward with Increasing Service* (*IRIS*) problems. A top-level algorithm, common to both policies, executes at every task arrival to determine the amount of service to be allocated to each task; later, a bottom-level algorithm, different for each policy, determines the order in which tasks are executed. However, tasks are not decomposed in a mandatory (with a minimum service time) and an optional part. Instead tasks receive whatever time can be given before their deadline expires. Although mandatory subtasks may be introduced, they are not guaranteed hard deadlines. In fact, an additional performance metric, the probability of missing a task's deadline before it can receive its mandatory amount of service, is introduced.

In [3] different *mandatory first* schemes were proposed, all sharing the condition that mandatories are always executed first. They differ in the policy according to which optionals parts are scheduled (*earlier-deadline-first, least-laxity-first,* etc.). The slots allocated to optionals are not the result of slack reshuffling but appear naturally as the processing takes place (they are called *background* slots). The schemes were compared in [2] and it was found that allocating each empty slot to the optional contributing most to the total reward at that slot produces the best results among all the mandatory first methods. Because of this, the method is called *Best Incremental Return* (BIR) and it is often used as a yardstick.

In [2] an off-line reward-based method is presented. It is based on the use of linear programming maximizing techniques and this requires the reward functions to be continuously differentiable. Also, in order to reduce the number of unknowns, it imposes the constraint that the optional part of each task receives the same service time at every task instantiation, resulting in a rather rigid system. In addition, the proposed method provides solutions not only subject to that constraint but in which mandatory parts cannot be RM scheduled unless the periods are harmonic. The use of RM is then limited to that particular class of problems.

The methods proposed in this paper have none of those shortcomings: The mandatory subsystem is RM scheduled and the reward-functions must not necessarily be continuously differentiable. As a matter of fact there are no restrictions except their computability at every slot. The methods can be used on-line, they have a light overhead and outperform Best Incremental Return. On top of that, when a reduction in the worst-case execution time of a mandatory part takes place or the arrival of a task is delayed, that *gain* time may be fully used for the execution of optional parts. These, in turn, may even have different reward functions along successive stretches of its execution or change in reaction to the environment [7]. Because of the rigidity of its approach all this is impossible in [2]

3. The Rate Monotonic Discipline, Empty Slots and Singularities

In [9] it has been proved that Rate Monotonic is optimal among the Fixed Priority disciplines. It is supported by the USA Department of Defense and consequently adopted, among others, by IBM, Honeywell, Boeing, General Electric, Magnavox, Mitre, General Dynamics, NASA, Paramax and McDonnell Douglas. It is, then, a *de facto* standard, at least in USA [11]. Therefore it makes sense to use it whenever possible, especially in applied systems that may find their way to the market.

Several methods have been proposed for testing the RM scheduling of real-time systems [6, 9, 14]. All have in common the fact that the ordering relation to define priorities is based on increasing periods, with some additional rules to break ties. In the Empty Slots method [14], time is considered to be slotted and the duration of one slot is taken as unit of time. Slots are notated t and numbered 1, 2, ... The expressions *at the beginning of slot* t and *instant* t mean the same. Tasks are preemptible at the beginning of slots.

A set $\mathbf{S}(n)$ of n independent preemptible periodic tasks is completely specified as $\mathbf{S}(n)=\{(C_1, T_1, D_1), (C_2, T_2, D_2),..., (C_n, T_n, D_n)\}$, where C_i, T_i and D_i, denote the worst case (maximum) execution time, the period (or the minimum period if there is any jitter) and the deadline of task i, respectively. Usually it is assumed that $D_i \le T_i$. In [14] it has been formally proved that $\mathbf{S}(n)$ is RM-schedulable iff

$$\forall i \in (1,2,....,n) \quad T_i \ge D_i \ge \text{least } t \mid t = C_i + \sum_{h=1}^{i-1} C_h \left\lceil \frac{t}{T_h} \right\rceil$$

in other words, if before the next instantiation the task encounters enough empty slots to execute itself and give way to higher priority tasks. Slots go empty only when all the tasks released in the interval [1, empty slot] have been executed.

In [12] two methods based on the detection of *singularities*, special instants in the evolution of the system, have been proposed for optimising the scheduling of mixed systems of tasks where only some of them are hard real-time

A singularity s, is a slot in which all the tasks belonging to $\mathbf{S}(n)$, released in [1, (s-1)], have been executed. Note that s-1 can be either an empty slot or a slot in which a last pending periodic task completes its execution. s is a singularity even if at $t = s$, periodic tasks are released. A singularity s_i is a slot in which all tasks belonging to $\mathbf{S}(i)$, released in [1, (s_i-1)], have been executed. The method based on the detection of s is called *Single Singularity Detection,* SSD. When all s_i are detected, the method is called *Multiple Singularity Detection,* MSD.

4. Mandatory/Optional Systems

In the context of this paper, a generic task, notated τ_i, $i \in \{1, 2, ..., n\}$, can be seen as $\tau_i = \mathrm{M}_i \cup \mathrm{O}_i$, where M_i and O_i denote the mandatory and the optional parts or subtasks of τ_i, with execution times m_i and o_i respectively. Obviously, the previous schedulability condition should be applied only to the mandatory part to be RM scheduled.

The maximum number of slots available for the execution of optionals is independent of the scheduling policy: It is always the number of slots left empty (unused) by the mandatory subsystem. If M denotes the least common multiple of the periods of the n tasks, commonly called the *hyperperiod,* the expression

$$W_n(M) = \sum_{i=1}^{n} m_i \frac{M}{T_i}$$

gives the number of slots necessary to process the mandatory subsystem in the interval [1, M]. Obviously, the number of slots available for the optional parts will be $M - W_n(M)$, and since the tasks are periodic, it suffices to study the evolution of the system in one hyperperiod. If $M - W_n(M) < \sum_{i=1}^{n} o_i$, the system is oversaturated and only some portions of the optionals will be executed. In fact, the optimal upper-bound on the attainable reward over the hyperperiod is dependent on the number of slots, or time, left free for the execution of optional subtasks. This is called *slack-time* and the simplest way to use it is to let empty slots appear only when there are not pending mandatories (background slots). On the contrary, the singularities methods used in this paper as the basis to address the reward problem, belong to the class of active methods that redistribute the slack over the hyperperiod.

Let $\mathbf{S}(n)=\{(m_1, o_1, T_1, D_1), (m_2, o_2, T_2, D_2),... , (m_n, o_n, T_n, D_n)\}$ be the set of n independent preemptible periodic mandatory/optional tasks to be scheduled using RM for the mandatory subsystem and a reward-based scheme for the optional subsystem. The optional part of a task must be executed within the period in which the mandatory part was executed.

Associated to each optional part is a reward function. Realistic reward functions are non-decreasing, either linear (constant returns) or concave (diminishing returns). Maximizing the total reward is, in general, an NP-hard problem [4].

It must be pointed out that in what follows the only restriction that will be imposed on the reward functions is their computability at every slot of the optional part. Moreover, they can be different for different intervals and even change in reaction to the environment, making the system a truly *adaptive* one [7]. Consider, for instance, the case of autonomous mobile service robots for transportation tasks in indoor environments [17]. When going towards a T-junction of corridors, the distance resolution may be low for large distances, say from 3 m up, and increase on the final approach.

A mandatory subsystem is said to be k-RM *schedulable* if

$$\forall i \in (1,2,....,n)$$

$$T_i \geq D_i \geq \text{least } t \mid t = m_i + k + \sum_{h=1}^{i-1} m_h \left\lceil \frac{t}{T_h} \right\rceil \qquad (1)$$

From the formal proofs in [12] it can be concluded that if the mandatory subsystem is k-RM schedulable, the k slots of an interval [s, (s+k-1)] can be used to execute tasks of the optional subsystem. In the *Single Singularity Detection* Method, SSD*,* k slots, slated for optional subtasks, are generated whenever possible. The *Multiple Singularity Detection* Method, MSD*,* refines the first one by generating not the lower bound but as many slots, k_i, as each mandatory subtask, $\mathrm{M_i}$ admits**.** In general, k will be a common lower bound on $\{k_i\}$. In turn, each k_i will be

$$k_i \geq \max k \mid T_i \geq D_i \geq \text{ least } t \mid t = C_i + k + \sum_{h=1}^{i-1} C_h \left\lceil \frac{t}{T_h} \right\rceil$$

It must be noted that k and k_i are calculated using the maximum C_i and the minimum T_i. The complexity of the calculation is similar to that of the empty-slots schedulability algorithm which in [14] has been proved to be $O(n*T_n)$, where n denotes the number of tasks and T_n the maximum period in the system.

These methods were first used for scheduling mixed systems of periodic real-time and aperiodic non real-time tasks with aperiodics executed in the singularity and the k-1 slots following it [12]. In that case, the mere advancement of empty slots to process aperiodic tasks in them, is enough to improve the performance of the system. This is simply because non real-time tasks are executed earlier and, consequently, the average response time is reduced. Shorter response times are often associated to a better figure of merit of the system.

However, in the case of reward-based systems, advancing the execution of optional parts is not enough and some heuristic must be added to maximize the total reward accrued over a given period of time.

5. The Singularity/Heuristic Methods

The combination of single and multiple singularities with two sets of heuristic rules (1 and 2) produce four methods, generically called SH, for the on-line scheduling of real-time mandatory/reward-based optional systems. They are specifically designated SSD1, MSD1, SSD2, MSD2. Heuristics must be added because reshuffling slack by advancing empty slots and executing optional parts in them is not enough to improve the performance. There will be cases in which high reward optionals are associated to low priority mandatories and *vice versa*. In those cases, advancing the execution of optionals can produce adverse results if an executed high priority mandatory enables the execution of its low-reward associated optional, preempting the execution of a high-reward optional associated to a low priority mandatory not yet executed.

The rationale supporting the first heuristic is to preclude the execution of a low reward optional when the only reason to do it is that its high priority mandatory has been executed. If a higher reward optional cannot be executed because its associated mandatory has not been executed, then mandatories are executed following the Rate Monotonic discipline. The use of advanced slack for the execution of mandatories eventually enables the execution of high reward optionals as soon as they are ready and a singularity appears.

The rationale behind the second heuristic is similar to the first one. However, if the higher reward optional is not enabled, its associated mandatory is executed in the available advanced slack slots, even at the cost of breaking the Rate Monotonic ordering. If its associated optional cannot be executed immediately after, it will be executed as soon as the next singularity appears.

In what follows, the algorithms of the four methods are described step by step. SSD1 and SSD2 are implemented by means of one counter, AC (contents denoted *AC*).

SSD1 steps are:

1) $AC = k$ at $t=s$.

2) *If* $AC \neq 0$ and there is not a pending mandatory with an associated optional of reward higher than the reward of the optional to be executed *then* the optional is executed *else* an RM ordered mandatory is executed.

3) *AC* is decremented by one on each slot assigned to an optional part.

SSD2 steps are:

1) $AC = k$ at $t=s$.

2) *If* $AC \neq 0$ and there is not a pending mandatory with an associated optional of reward higher than the reward of the optional to be executed *then* the optional is executed *else* the mandatory associated to the higher reward optional is executed (even if it violates the RM ordering).

3) *AC* is decremented by one on each slot assigned to an optional part or to a mandatory violating the RM ordering.

MSD1 and MSD2 are implemented by means of n counters, AC$_i$, $i \in \{1, 2, ..., n\}$ (contents denoted AC_i).

MSD1 steps are

1) $\forall g \in \{1, 2, ..., i\}$, $AC_g = k_g$ at $t = s_i$.

2) *If* $\forall i \in \{1, 2, ..., n\}$, $AC_i \neq 0$ and there is not a pending mandatory with an associated optional of reward higher than the reward of the optional to be executed *then* the optional is executed *else* an RM ordered mandatory is executed

3) $\forall i \in \{1, 2, ..., n\}$, AC_i is decremented by one on each slot assigned to an optional part.

MSD2 steps are

1) $\forall g \in \{1, 2, ..., i\}$, $AC_g = k_g$ at $t = s_i$.

2) *If* $\forall i \in \{1, 2, ..., n\}$, $AC_i \neq 0$ and there is not a pending mandatory with an associated optional of reward higher than the reward of the optional to be executed *then* the optional is executed *else* the mandatory associated to the higher reward optional is executed (even if it violates the RM ordering).

3) *If* an optional is executed *then* $\forall i \in \{1, 2, ..., n\}$, AC_i is decremented by one *else* only the counters corresponding to tasks inverted by the execution of the violating mandatory are decremented by one.

It must be noticed that since empty slots are singularities, counters will be reloaded at them. The methods are bandwidth preserving in the sense that the slots generated at each singularity must not necessarily be used immediately after it. A given optional may wait until the next release of its associated mandatory or until the next singularity. At that moment, however, the counters are reloaded and k slots are again available. Updating the counters is the only overhead of the SH methods.

Example. A simple example showing the essence of the methods is given. The system is specified in Table 1. Reward functions are exponential and the BIR and SSD1 methods are used.

i	m_i	o_i	T_i	f_i
1	1	2	3	$5(1-e^{-t})$
2	2	2	5	$7(1-e^{-5t})$
3	1	2	15	$2(1-e^{-3t})$

Table 1: The specification of the system

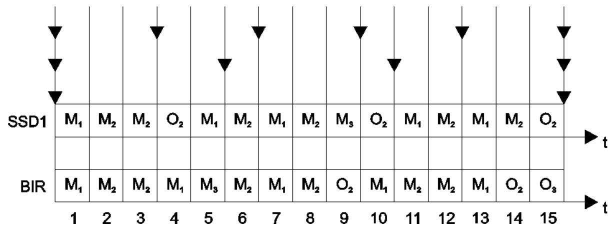
SSD1
M1 M2 M2 O2 M1 M2 M1 M2 M3 O2 M1 M2 M1 M2 O2
t
BIR
M1 M2 M2 M1 M3 M2 M1 M2 O2 M1 M2 M2 M1 O2 O3
t
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

Fig. 1 The evolution of the system. ↓ denotes the release of the task.

The evolution of the system under SSD1 is shown in the first line on Fig. 1. Its *k*, calculated using expression (1), is 1. At *t*=1, therefore, *AC*=1, but no optional can be executed because no mandatory has completed its execution. At *t*=2, O_1 could be executed, but the scheduler prevents it because the execution of M_2 with an associated optional of higher reward is still pending. At *t*=4, after executing M_2, O_2 is executed. Something similar occurs at *t*=10 and *t*=15, after the counter is reloaded. Total reward is 21.

When the same problem is scheduled with the BIR method (second line), O_2 is executed at *t*=9 and *t*=14. At *t*=15, the first optional slot of task 3 is executed because it has a higher reward than the second slot of O_2. Total reward is only 17 and the rewards ratio is 1.23. It must be noted that SSD1 reaches the optimal reward: three slots are available for optional executions and all of them are used to process the optional of highest reward. The mandatory utilization factor is 0.80.

If m_3 =2 instead of 1, the mandatory utilization factor is 0.86. There are two empty slots that SSD1 and BIR use at (4, 15) and (14, 15), respectively. The rewards ratio is 1.37.

If m_3 =3, the mandatory utilization factor is 0.93.There is only one empty slot used by SSD at $t = 4$ and by BIR at *t*=15. However, the rewards ratio is only 1.0 because both methods execute the same optional. In this case, advancing the execution does not produce a better result. The rewards ratio decreases after reaching a maximum.

6. Performance Evaluation

In what follows the evaluation process is described to the extent that its correctness can be validated and the results replicated.

6.1 The synthetic set.

Since there are not standard benchmarks, a first comparative evaluation was performed using the synthetic set proposed in one of the outstanding papers published on the subject [2]. The set is presented in Table 2: eleven tasks with periods in the range 20-2160 and whole (mandatory plus optional) worst case execution times in the range 10-300.

There are three general reward functions: exponential, logarithmic and linear, with specific coefficients for each task. For the three reward functions, the evaluation started with a first instantiation in which all mandatory execution times were taken equal to 1 slot, giving a total mandatory utilization factor of 0.19. In the second instantiation, m_i=1 for *i*=1, 2, ..,10, and m_{11}=101. In successive instantiations, the mandatory execution times were varied following an odometer-type mechanism, with different pitches for each task. When the maximum execution time is surpassed, m_i has completed a "turn" and returns to 1. Then, m_{i-1} advances one pitch, etc. This mechanism generates a wide variety of m_i combinations, more than 50,000 for each reward function and each SH method. In the represented utilization's factor range [0.35, 0.95] many of them produce the same total mandatory utilization factor down to the hundredths and this fact allows the comparison between the SH methods and BIR with 99% confidence intervals.

The metric used to evaluate the four SH methods is the ratio between total reward obtained by each of them and the total reward obtained by BIR. The ratio is plotted *vs* mandatory utilization factor for each of them and for each of the three reward functions. For the sake of clarity the curves representing the confidence intervals are not shown but they are very close to the performance ratio curves. Results are depicted in Fig. 2.

Task	T_i	$m_i + o_i$	Exponential	Logarithmic	Linear	Pitch
1	20	10	$15\left(1-e^{-t}\right)$	$7\ln(20t+1)$	5 *t*	3
2	30	18	$20\left(1-e^{-3t}\right)$	$10\ln(50t+1)$	7 *t*	6
3	40	5	$4\left(1-e^{-t}\right)$	$2\ln(10t+1)$	2 *t*	2
4	60	2	$10\left(1-e^{-0.5t}\right)$	$5\ln(5t+1)$	4 *t*	1
5	60	2	$10\left(1-e^{-0.2t}\right)$	$5\ln(25t+1)$	4 *t*	1
6	80	12	$5\left(1-e^{-t}\right)$	$3\ln(30t+1)$	2 *t*	4
7	90	18	$17\left(1-e^{-t}\right)$	$8\ln(8t+1)$	6 *t*	6
8	120	15	$8\left(1-e^{-t}\right)$	$4\ln(6t+1)$	3 *t*	5
9	240	28	$8\left(1-e^{-t}\right)$	$4\ln(9t+1)$	3 *t*	7
10	270	60	$12\left(1-e^{-0.5t}\right)$	$6\ln(12t+1)$	5 *t*	20
11	2160	300	$5\left(1-e^{-t}\right)$	$3\ln(15t+1)$	2 *t*	100

Table 2: The specification of the synthetic set

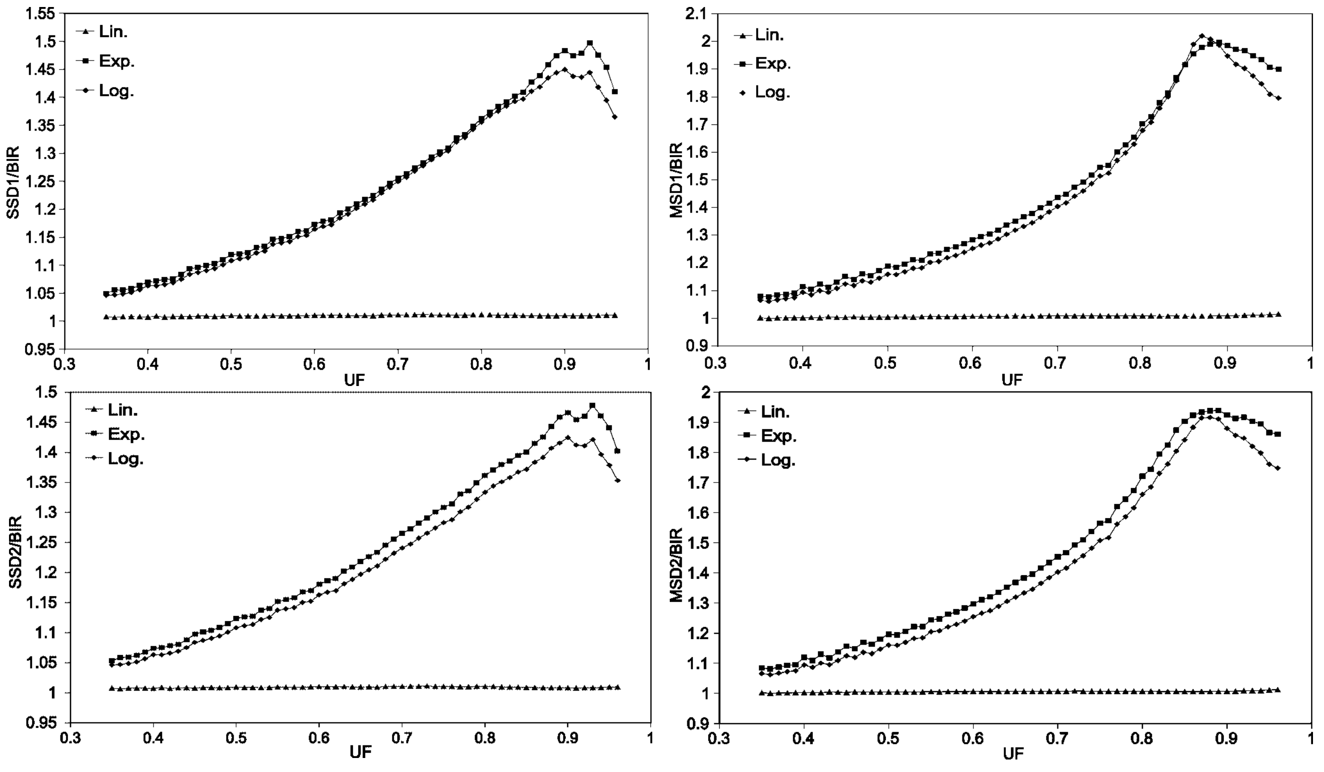

Fig. 2. The Synthetic set. Ratio of SH rewards/BIR rewards

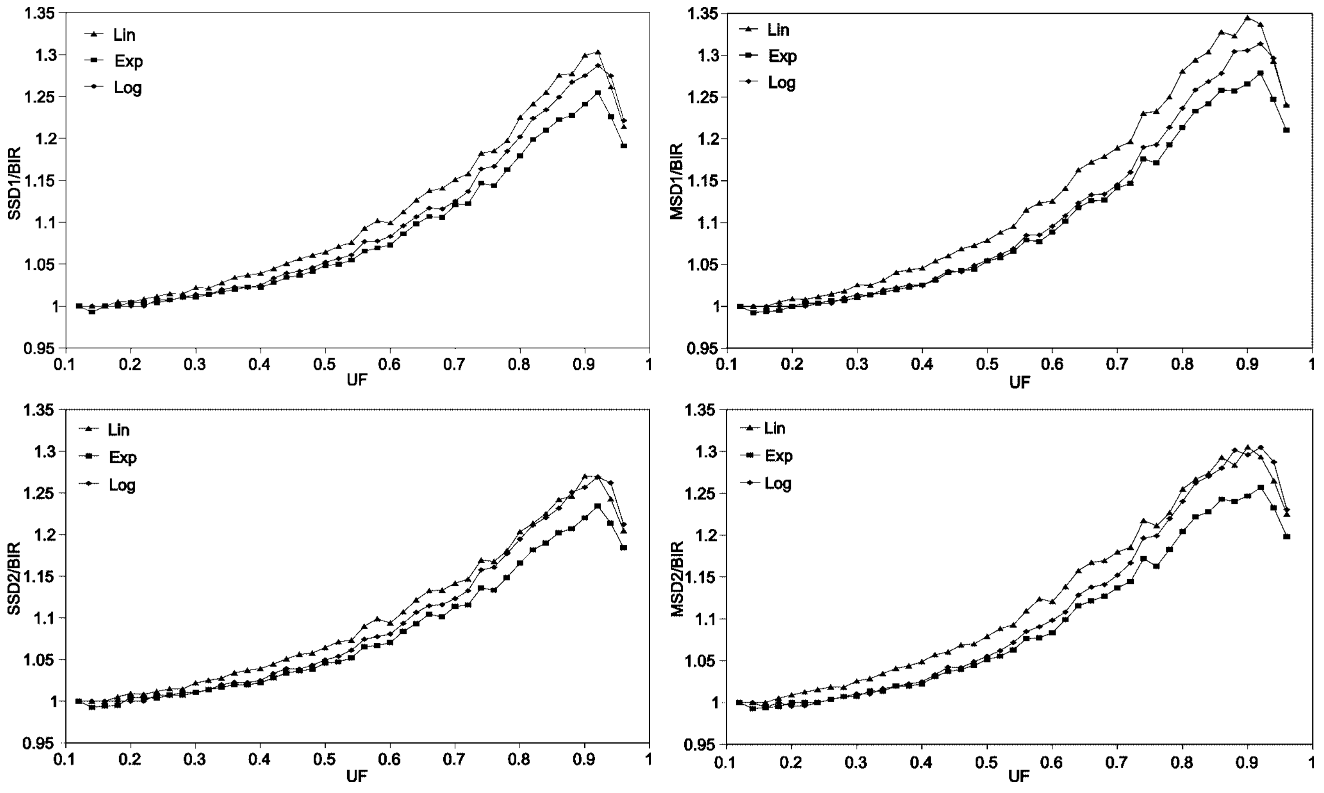

Fig. 3. Randomly generated sets. Ratio of SH rewards/BIR rewards

6.2 Sets of tasks and reward functions randomly generated.

More than 57,000 sets of 10 tasks were randomly generated with all random variables uniformly distributed. The sample space of the 10 periods was (20, 30, 40, ..., 600) with the added condition that the hyperperiod must not exceed 32,000. The total mandatory utilization factor was randomly chosen in the interval [0.12, 0.96]. The total optional utilization factor was calculated as 2 minus the total mandatory; in that way, there would always be optionals ready to be executed in available slots. The worst case mandatory execution time and the optional execution time were randomly assigned to each task in such a way that both utilization factors were met and the condition $m_i+o_i \leq T_i$ always held.

Associated to each task, three reward functions were randomly generated. As in the previous case, they were linear, exponential and logarithmic. The three functions were generated in such a way that, for $t=o_i$, all reach the same maximum value, an integer randomly generated in the interval [4, 40]. The coefficient of the linear function follows immediately as well as the minimum function value, obtained for $t=1$. The first coefficients of the exponential and logarithmic functions were generated at random within intervals derived from the maximum and minimum values of the linear function. The other coefficients were adjusted to produce the desired exponential or logarithmic function values for $t=o_i$. Results are depicted in Fig. 3.

6.3 Analysis of results

The sets of curves depicted in Fig. 2 for the four SH methods show a remarkable similarity. When the utilization factor is low, background slots appear early in the process and there is not much difference between BIR and the SH methods. As the utilization factor increases, the early appearance of reshuffled slack and the possibility of assigning it to the processing of high reward optionals start to bear and the SH methods clearly outperform BIR in the exponential and the logarithmic case. This doesn't happen in the linear case because all reward functions start at the origin, they do not cross each other, the returns are constant and, therefore, while it has optionals to be executed, the steepest one is sooner or later chosen, whatever the method. For instance, if the reward functions of the example in Section 5 were linear, with the function associated to task 2 the steepest one, the BIR method would have chosen a second O_2 instead of a first O_3 in slot 15, and both methods would have yielded the same total reward over the hyperperiod.

The synthetic set is particularly prone to this kind of behaviour. Note, for instance, that the steepest linear function is associated to task 2, with a high total execution time ($m_2+o_2=18$) and a low period ($T_2=30$). For low mandatory utilization factors, background empty slots appear early and the number of task 2 optionals that may be executed in the hyperperiod is high. For high mandatory utilization factors the number of empty slots available for optionals decreases but, because of the high reward associated to them, task 2 optionals are executed in background slots, even if they appear late.

The ratio SH rewards/BIR rewards increases steadily, reaches a maximum in the vicinity of a 0.9 utilization factor, and then decreases. This is due to the fact that the number of slots available for optionals is so small that the potentiality of the SH methods cannot be fully exploited. Although still better than BIR, the rewards ratio decreases, as shown in the example of Section 5.

As could be expected, MSD1 and MSD2 perform better than their single counterparts. Although the number of empty slots is the same in every case, multiple singularities produce an earlier appearance of slots that can be used to execute high reward optionals that may otherwise be lost.

The curves corresponding to the second set, depicted in Fig. 3, show a pattern similar to the first set in all cases except in the linear one. In this case, the number of optionals of the high reward linear functions is not always large and therefore optionals of lower reward are executed. The advance of the empty slots allows a full use of the SH capabilities. The result is that in the linear case, the rewards ratio not only increases with the mandatory utilization factor but produces the higher rewards ratio.

7. Conclusions

Four methods have been presented for scheduling mandatory/optional systems in which the mandatory subset is hard real-time. Generically called SH methods, they use singularities, special instants that appear along the execution of the system, combined with some heuristic to allocate available slots to optional subtasks. They are implemented by means of counters. Updating the counters is the only overhead of the methods.

Reward functions are non decreasing, either concave or linear. To the best of the authors' knowledge, papers published up to now on the subject of reward maximization require the functions to be continuously differentiable. Moreover, they cannot be executed on-line. On the contrary, the only requirement of the SH methods is that the functions are computable at every instant. Rate-monotonic, a *de facto* standard at least in USA, is used for scheduling the mandatory subsystem. The optional subsystem is scheduled trying to maximize the total reward accrued over the hyperperiod. After the system is proved to be schedulable by any of the available exact off-line tests, SH methods can be executed on-line.

The performance of the methods was evaluated using simulations performed on a synthetic set of tasks proposed in one of the outstanding paper on the subject and on sets of tasks randomly generated. The results were analysed and explained. The main conclusion is that, in all cases, the SH methods outperform Best Incremental Return, often used as a yardstick.

Acknowledgment

The authors are pleased to thank the three anonymous reviewers for their valuable comments and observations.

References

[1] Audsley, N. C., R. I. Davis and A. Burns, “Mechanism for enhancing the flexibility and utility of hard real-time systems”, *Proc. 15th IEEE Rea-Time System Symposium*, pp. 12-21, 1994.

[2] Aydin, H., R. Melhem, D. Mossé and P. Mejía-Alvarez, “Optimal reward-based scheduling for periodic real-time tasks”, IEEE Transactions on Computers, 50, 2, pp. 111-130, 2001.

[3] Chung, J. Y., J. W.-S. Liu and K. J. Lin, "Scheduling periodic jobs that allow imprecise results", IEEE Trans. on Computers, 19, 9, pp. 1156-1173, 1990.

[4] Dey, J. K. and J. Kurose, “On line scheduling policies for a class of IRIS (Increasing reward with increasing service) real-time tasks”, 46, 7, pp. 802-813, 1986.

[5] Friedrich, L., J. Stankovic, M. Humphrey, M. Marley and J. Haskins, "A survey of configurable component-based operating systems for embedded applications", IEEE Computer, 21, 3, pp. 54-67, 2001.

[6] Joseph M. and P. Pandya, “Finding response times in a real-time system”, The Computer Journal, 29, 5, pp. 390-395, 1986.

[7] Kuo T.-W and A. K. Mok, “Incremental reconfiguration and load adjustment in adaptive real-time systems”, IEEE Transactions on Computers, 48, 12, pp. 1313-1324, 1997.

[8] Lin K-J., S. Natarajan and J. W. S. Liu, “Imprecise results: utilizing partial computations in real-time systems”, Proc. 8th IEEE Real-Time Systems Symposium, pp. 210-217, 1987.

[9] Liu C. L. and J. W. Layland, “Scheduling algorithms for multiprogramming in hard real-time environment”, J. ACM, 20, 1, pp. 46-61, 1973.

[10] Liu J. W.-S, K.-J. Lin, W.-K. Shih, A. C.-S. Yu, C. Chung, Y. Yao and W. Zhao, “Algorithms for scheduling imprecise computations”, Computer, 24, 5, pp. 58-68, 1991.

[11] Obenza R., “Rate monotonic analysis for real-time systems”, IEEE Computer, 26, 3, pp. 73-74, 1993.

[12] Orozco J., R. Santos, J. Santos and R. Cayssials, “Taking advantage of priority inversions to improve the processing of non-hard real-time tasks in mixed systems”, Proc. WIP 21st IEEE Real-Time Systems Symposium, pp. 13-16, 2000.

[13] Orozco, J., R. M. Santos, J. Santos and E. Ferro, "Hybrid rate-monotonic/reward based scheduling of real-time embedded systems", Proc. IEEE/IEE Workshop on Real-Time Embedded Systems, London, December, 2001.

[14] Santos, J. and J. Orozco, "Rate monotonic scheduling in hard real-time systems", Information ProcessingLetters, 48, pp. 39-45, 1993.

[15] Shih, W.-K and J. W. S Liu, “Algorithms for scheduling imprecise computations with timing constraints to minimize maximum error”, IEEE Transactions on Computers, 44, 3, pp. 466-471, 1995.

[16] Stankovic, J. and K. Ramamritham, "Advances in Real-Time Systems", IEEE Computer Society Press, pp. 1-16, ISBN 0-8186-3792-7, 1992.

[17] Surmann, H. and A. Morales, "A five layer sensor architecture for autonomous robots in indoor environments", Proc International Ssymposium on Robotics and Automation, ISRA 2000, Mexico, pp. 533-538, 2000.