

# Taking advantage of priority inversions generated to improve the processing of non-hard real-time tasks in mixed systems

J. Orozco, R. Santos, J. Santos and R. Cayssials  
Universidad Nacional del Sur/CONICET, 8000 Bahía Blanca, Argentina  
{ieorozco, iesantos}@criba.edu.ar

## Abstract

*In this paper, work in progress about two original on-line algorithms to solve the problem of scheduling a mix of periodic hard real-time and aperiodic soft and non real-time tasks in a single processor is presented. The proposed methods perform well when compared to other methods that address the same problem.*

*Index terms: k-schedulability; priority inversions; hard, soft and non real-time mix; periodic and aperiodic mix*

## 1. Introduction

Scheduling a mix of periodic hard real-time and aperiodic non-hard (soft and non real-time) tasks in multitask-single processor systems is an important problem not only because the mix increases the processor's utilization factor but because some of the aperiodic tasks may be used to improve the periodic computations, for instance by increasing their precision.

The main contributions of this paper are two methods to advance the execution of aperiodic tasks by taking advantage of priority inversions, generated on purpose, without hampering the execution of the periodic ones.

## 2. Related work

Several methods to improve the processing of mixed (periodic and aperiodic) tasks sets have been proposed. In order to describe and analyze them it is useful to assume that time is slotted.

The simplest method to use the available time is to let empty slots appear naturally while executing the periodic set and use them as they appear. The method is called the *background method*, BGR, and, as can be seen, is absolutely passive.

The active methods try to redistribute the available slots over the *hyperperiod* (the least common multiple of the periods) making them appear as soon as possible and

using them for aperiodic traffic. They can be considered as belonging to a class that can be generically called Latest Start Time. Up to now there are, basically, two active methods to do this: *servers* and *slack-stealing*.

The first one creates a new periodic task and uses it as a server of aperiodic tasks. Three main types of servers have been proposed: *polling*, *sporadic* and *deferrable*. The polling server, POS, runs periodically at the minimum period in the set of periodic tasks and the maximum schedulable execution time. This server loses the unused slots in the period.

The sporadic [1, 7] and the deferrable [8] servers, SPS and DES, allow their capacity to be used throughout their periods (*bandwidth preserving* algorithms). They differ only in the way their capacity is replenished.

In slack stealing, SLS [2, 3, 6], all the possible slots that can be taken from the periodic set without causing its deadlines to be missed are "stealed". Results, as presented in [6], are obtained with a large amount of calculations that make it unfeasible as a dynamic on-line algorithm. At the time of its publication it was claimed that the method "*minimizes the response time of every aperiodic task among all scheduling algorithms which meet all periodic tasks deadlines*". Later, however, a counterexample was presented invalidating the previous assertion [9].

The two methods presented here are executed on-line, generate priority inversions on purpose and use them to process aperiodic tasks. When periodic tasks are executed in less than their worst case execution times, both methods automatically adjust themselves to the varying condition and take advantage of that surplus time, called *gain time*, to improve the service of aperiodic tasks. Moreover, the methods are not affected by jitter.

## 3. k-RM schedulable systems

Rate Monotonic scheduling has become a *de facto* standard [4] and several methods have been proposed for the RM scheduling of real-time systems. In what follows, 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.

*Definitions:*

- In the context of this paper, a *priority inversion* is the phenomenon by which a lower priority periodic task is executed instead of a higher one or an aperiodic task is executed instead of a periodic one.
- A system  $\mathbf{S}(m)$  is *k-RM schedulable* if it is RM schedulable in spite of the fact that its tasks can suffer up to  $k$  inversions.
- A *singularity*,  $s$ , is a slot in which all the hard tasks 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.

In [5] it has been formally proved that if a hard system  $\mathbf{S}(m)$  is  $k$ -RM schedulable, the  $k$  slots of an interval  $[s, (s+k-1)]$  can be used to execute tasks of another system. A system  $\mathbf{S}(m)$  is  $k$ -RM schedulable iff

$$\forall i \in (1, 2, \dots, m)$$

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

where  $D_i$ ,  $C_i$  and  $T_i$  denote the deadline, the worst case execution time and the period of  $\tau_i$ , respectively. It must be noted that, in general,  $k$  will be a common lower bound on the number of inversions,  $k_i$ , that each task,  $\tau_i$ , may admit.  $k_i$  will be

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

#### 4. The priority inversion algorithms

Although priority inversions are a hindrance to the schedulability of periodic tasks, they can be used to improve the service of aperiodic ones by reducing the average delay. Two methods are proposed. In the first one, the *Single Singularity Detection Method, SSD*,  $k$  priority inversions are generated whenever possible and used to service aperiodic tasks. The second method, the *Multiple Singularity Detection Method, MSD*, refines the first one by generating not the lower bound but as many priority inversions,  $k_i$ , as each task  $\tau_i$  admits. It must be noted that in both cases they are priority inversions generated on purpose and not produced, for instance, by resource contention.

In order to describe both methods the following notation shall be used:  $AC_s$  and  $AC_i$ , counters associated to the whole set of periodic tasks and to task  $\tau_i$  respectively.  $AC_s$  and  $AC_i$  denote their contents.  $s_i$  denotes a slot in which all periodic tasks belonging to  $\mathbf{S}(i)$  and released in  $[1, (s_i-1)]$  have been executed.

The SSD method is implemented by means of one counter,  $AC_s$ . The algorithm is:

1.  $AC_s = k$  at  $t=s$
2.  $AC_s$  is decremented by one on each slot assigned to an aperiodic task.
3. Aperiodic tasks are executed if  $AC_s \neq 0$

The MSD method is implemented by means of  $m$  counters,  $AC_i$ . The algorithm is:

1.  $\forall g \in \{1, 2, \dots, i\}$ ,  $AC_g = k_g$  at  $t = s_i$
2.  $AC_g$  is decremented by one on each slot assigned to an aperiodic task.
3. Aperiodic tasks are executed if  $\forall i \ AC_i \neq 0$

It must be noticed that what would be background slots are also singularities and the counters are set to the maximum value at them.

#### 5. Simulation results and comparative evaluation

The simulations used to evaluate the method and compare it to other methods addressing the same problem were done with a set of ten periodic tasks with periods in the range 45 to 120. Execution times are adjusted to produce a periodic set utilization factor,  $U_p$ , of 0.4 to 0.6 in steps of 0.1. The aperiodic load is adjusted, also in steps of 0.1, to produce a total (periodic plus aperiodic) tasks utilization factor of up to 0.8. The aperiodic arrival and service times follow a Poisson and an exponential distribution, respectively. The aperiodic mean service time is one tenth of the minimum period, that is 4.5. In addition to background and the active methods, the performance of M/M/1 was also determined. M/M/1 assumes no periodic load and gives therefore a lower bound on the average delay.

In Figs. 1 to 4, the results of the simulations are depicted for three different utilization factors of the periodic set. Average response time is plotted vs the aperiodic utilization factor. In each case, the largest response time (background and maximum  $U_a$ ) is taken as 100% and all the rest referred to it.

As before, the notation used is: BGR, Background; POS, Polling Server; DES, Deferrable Server; SSD, Single Singularity Detection; MSD, Multiple Singularity Detection and SLS, Slack Stealing.

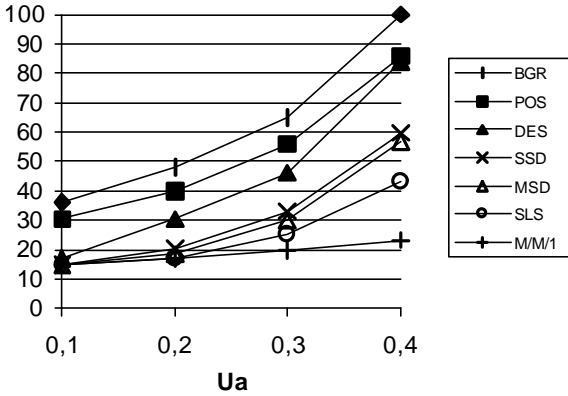


FIG 1: Average response time vs aperiodic utilization factor for  $U_p = 0.4$ .

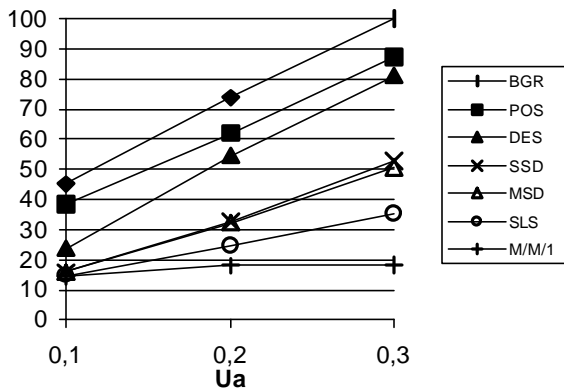


FIG 2: Average response time vs aperiodic utilization factor for  $U_p = 0.5$ .

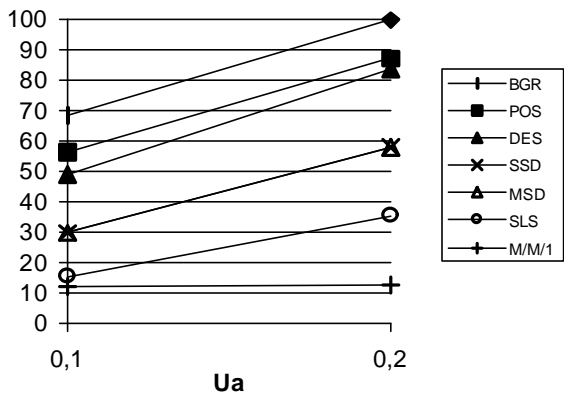


FIG 3: Average response time vs aperiodic utilization factor for  $U_p = 0.6$ .

In all cases the inversion methods performance is between those of slack stealing and the servers. There are, however, some striking points:

- 1) The performance of the deferrable server is not as good as presented in [8].
- 2) There is not much difference between both servers.
- 3) There is not much difference between both inversion methods.

The first one is explained by the fact that in [8] two conditions are imposed: a) The ratio between the aperiodic utilization factor and the server utilization factor is less than 0.7. b) The mean service time is much less than the server capacity. Most of the time, none of those conditions are met by the load used in these simulations. Therefore, it can be expected that an important fraction of the aperiodic load will be serviced in background slots and the servers' performance will be substantially degraded, which explains the first point. When degraded, the performance of the different types of servers converge and approach background, which explains the second point. In Fig 4, the load, changed to meet the conditions a) and b) stated above, made both servers behave as described in [8]. Even then, the average response time of the inversion methods remains lower than the servers'.

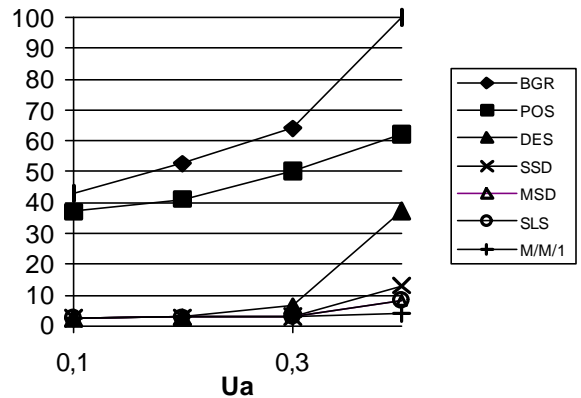


FIG 4: Average response time vs aperiodic utilization factor for  $U_p = 0.5$  and load as specified in [8].

The slight difference between both inversion methods can also be attributed to the particular type of load. When simulations are performed using sets of tasks with approximately the same utilization factor but with a large ratio (say 200) between the maximum and the minimum periods, MSD and SSD approach M/M/1 but SSD diverge for higher utilization factors.

## 6. Conclusions and future work

SSD shows a very acceptable performance: The average response time of the aperiodic load is higher than SLS but lower than DES. Having in mind that it is an on-line method (while SLS is off-line) and its implementation requires only one counter, the cost/performance trade-off seems to be rather good.

However, the performance of all methods is highly sensitive to the type of load. This is shown by the marked differences in the DES performance as reported in [8] and as presented in Figs. 1 to 3. Only when the load is changed to meet the conditions stated in [8], the performance of SLS, presented in Fig. 4, approaches M/M/1 as reported there. It must be noted that in that case, the performance of SSD and MSD are also very close to M/M/1.

It seems sensible then, that future work be directed towards more extensive simulations with the very different kinds of load that can be expected in practical applications ♦

## References

- [1] G. Bernat and A. Burns, "New results on fixed priority aperiodic servers", *IEEE Symposium on Real-Time Systems*, 1999, *IEEE Computer Society Digital Library*.
- [2] R. J. Davis, K.W. Tindell and A. Burns, "Scheduling slack time mixed priority preemptive systems", *Proc. Real-Time Systems Symp.* pp. 222-231, 1993.
- [3] J. P. Lehoczky and S. Ramos-Thuel, "An optimal algorithm for scheduling soft-aperiodic tasks fixed-priority preemptive systems", *Proc. Real-Time Systems Symp.* pp. 110-123, 1992.
- [4] R. Obenza, "Rate monotonic analysis for real-time systems", *IEEE Computer*, 26(3):73-74, 1993.
- [5] J. Orozco, J. Santos, R. Cayssials and E. Ferro, "Mixing real and non real-time processing: Priority inversions are not so bad after all", *Proc WiP and Industrial Experience Sessions*, 12th Euromicro Conference on Real-Time Systems, pp. 7-10, Stockholm, 2000.
- [6] S. Ramos Thuel, "Enhancing fault tolerance of real-time systems through time redundancy", *Ph.D. Thesis*. Dep. of Electr. and Comp. Eng., Carnegie Mellon Univ., 1993.
- [7] B. Sprunt, L. Sha and J. Lehoczky, "Aperiodic task scheduling for hard real-time systems", *Real-Time Systems Journal*, 1(1), pp. 27-60, June 1989.
- [8] J. K. Strosnider, J. P. Lehoczky and L. Sha, "The deferrable server algorithm for enhanced aperiodic responsiveness in hard real-time environments". *IEEE TC*, 44(1), pp. 73-91, 1995.
- [9] T.S. Tia, J.W.S Liu and M.S. Shankar, "Algorithms and optimality of Scheduling soft aperiodic requests in fixed-priority preemptive systems", *Real Time Systems Journal*, 10(1), pp. 23-44, January 1996.