EVOLVE 2011
Bourglinster Castle, Luxembourg, May 25-27 2011
ISBN 978-2- 87971-106-5 ISSN 2222 - 9434

IMPACT OF JOB DROPPING ON THE *PROBABILISTIC* SCHEDULABILITY OF UNIPROCESSOR *DETERMINISTIC* REAL-TIME SYSTEMS

O. Buffet[a] and **L. Cucu-Grosjean**[a]

[a]LORIA / INRIA / Nancy University
615, rue du Jardin Botanique – 54600 Villers-lès-Nancy – France
firsname.lastname@inria.fr

Abstract

In [1], the authors point out an interesting open problem concerning the schedulability (with Fixed Priorities) of uniprocessor probabilistic real-time systems with variable execution times: What is the impact of dropping some jobs, in particular when they are doomed to fail? The present paper looks at this problem—i.e., assuming that the scheduling criterion depends on the satisfaction of a success rate—while restricting our study to deterministic tasks (no variable execution times or other kind of uncertainty). After formally introducing the problem at hand, we discuss a simple job dropping rule and properties that it satisfies.

Problem Description

Here we consider that execution times, deadlines, offsets and inter-arrival times are all fixed and deterministic. Each task is associated with a required percentage of success. A task τ_i is thus given by a four-tuple (C_i, T_i, D_i, p_i) where

- $C_i > 0$ is its worst-case execution time;
- $T_i \geq C_i$ is its period;
- $D_i \in [C_i, T_i]$ is its deadline; and
- $p_i \in [0, 1]$ is the minimum success rate.

The problem is to schedule n such tasks on a processor so as to respect their respective minimum success rates.
We consider fixed priority (FP) scheduling, i.e., tasks are ordered according to pre-defined priorities which dictate which tasks should run in case of conflict. All tasks can be preempted by higher priority tasks.

Success Rate and Averages – One point that needs clarification is what it means to guarantee a minimum success rate. How is this success rate precisely defined? It must be the average number of successfully executed jobs, but which "average"? Here we consider the *Simple Moving Average* (SMA), an average computed on a sliding window of width $w > 0$:

$$\begin{aligned} A_T &= \frac{1}{w} \sum_{t=T-w}^{T} a_t \\ &= A_{T-1} + \frac{a_t - a_{t-w-1}}{w}, \end{aligned}$$

where A_t is the average at t and a_t is the instant data gathered at t. A small width allows for low memory requirements and high "robustness" (if a success rate is guaranteed for w_1, then it is also guaranteed for any $w_2 > w_1$); conversely, the smaller the width, the harder it is to guarantee a success rate.
Note that the success rate of a task may be guaranteed only after an initial transient phase.

Example – Let us consider a first example with two tasks defined as follow:

	C	T	D	p
τ_1	1	3	1	1
τ_2	1	1	1	$\frac{2}{3}$

By giving the highest (fixed) priority to task τ_1, one gets a schedule with one job of τ_1 and two jobs of τ_2 executed every three time steps.
Let us now consider the success rate of τ_2 with a sliding window of width w. The schedule is periodic and repeats every three time steps so that, for each value of w, the observed success rate is also a periodic sequence of period three (independently of whether the required success rates are rational numbers or not):

w	1	2	3	4	5	6	$\cdots$
$p_{\frac{1}{3}}$	$\frac{0}{1}$	$\frac{1}{2}$	$\frac{2}{3}$	$\frac{2}{4}$	$\frac{3}{5}$	$\frac{4}{6}$	$\cdots$
$p_{\frac{2}{3}}$	$\frac{1}{1}$	$\frac{1}{2}$	$\frac{2}{3}$	$\frac{3}{4}$	$\frac{4}{5}$	$\frac{4}{6}$	$\cdots$
$p_{\frac{3}{3}}$	$\frac{1}{1}$	$\frac{2}{2}$	$\frac{2}{3}$	$\frac{3}{4}$	$\frac{4}{5}$	$\frac{4}{6}$	$\cdots$

Whatever w, the average success rate is always $\frac{1}{3}$. But the instant success rate varies more or less, and the maximum difference between the minimum and maximum instant success rates for a given w tends (progressively but not monotonically) towards zero:

$$\lim_{w \to \infty} (p_{\max}(w) - p_{\min}(w)) = 0.$$

What is Job Dropping – We are in fact interested in "augmenting/complementing" fixed-priority scheduling with *job dropping* rules, i.e., simple rules deciding when to drop a job in a view to find a feasible schedule. The next section presents a simple rule and discusses some of its properties.

Minimal Job Dropping

Rule – The job dropping rule mentioned in [1]—hereby called *minimal Job Dropping* (minJD)—is to give up a job if it is doomed to fail. This seems to be the most obvious thing to do, and can lead to (i) not starting a job if there is not enough time left for it, or (ii) stopping a job if some information tells us that it won't be able to finish on time.

When to Quit a Job? – One can verify more or less carefully whether a job is doomed to fail. Here are two possible options:

Basic Test: The most obvious thing to do is look at the time remaining before its deadline without caring about other jobs. Dropping a job because there is not enough time left for it is always a good rule to apply.

Advanced Test: A thing is, if there is competition for CPU time with other jobs, one should try to take this competition into account to detect more job dropping situations. To that end, let us first notice that a job τ_i only has to care about other jobs with a higher priority. This means that one should check for *droppability* starting with high priority jobs.

Plus, to completely check for the *droppability* of τ_i, one has to look at each remaining time step to count how many of them are available. This may require looking at the droppability of jobs that have not been started yet. The consequence is a chain reaction going from low priority jobs to high priority ones and implying longer simulations for higher priority tasks.

Compatibility with Fixed Priorities – As we will see now, minJD is a simple job dropping rule that exhibits nice properties. One would in particular favor job dropping rules for which the next property holds.

Proposition 1 (Soundness of FP+minJD)**.** *Let τ be a system of tasks scheduled using a (given) fixed priorities policy. By applying minJD on τ we improve the observed success rates.*

Proof (sketch). Because a fixed priorities policy is used, dropping a job of priority P gives way to lower-priority jobs, but does not affect higher priority jobs. Thus, dropping a job for a given task can not degrade the success rate of this task or tasks of higher priority, but allows lower priority tasks to run more jobs. □

Proposition 2 (Periodicity of FP+minJD)**.** *When using fixed priorities, minJD produces a periodic schedule that repeats every hyperperiod T^h (after the first hyperperiod).*

Proof (sketch). The proof is similar to the periodicity proof for FP scheduling alone (a proof by induction). Starting with the highest priority task, τ_1, its schedule repeats every T_1 time steps, and thus every T^h time steps as well. Note that, for this first task, either $C_i \leq D_i$ and all jobs can be run, or $C_i > D_i$ and no job can be run.
Then, assuming that the schedule of τ_1 to τ_{i-1} repeats every T^h time steps, this gives a periodic pattern of free time steps in which τ_i can be scheduled, placing jobs iff the job dropping rule is satisfied. As the job dropping rule only looks at the situation over at most $D_i < T^h$ time steps (for both droppability tests), this results in the same decisions in each hyperperiod. □

Conclusion

The idea of dropping jobs early to avoid using CPU time for no reason was first raised in [1]. This preliminary work presents a simple job dropping (minJD) rule, showing that (i) more or less effort can be put in determining which jobs to drop, and (ii) minJD exhibits interesting properties such as the guarantee that it can only improve the success rate of a task.
This work paves the way to develop more complex job dropping rules, pointing out the type of properties that one could expect from them. Other directions are related to extending such work to real-time systems with probabilistic dynamics or other properties like parallelism.

References

[1] O. Buffet and L. Cucu-Grosjean. Impact of job dropping on the schedulability of uniprocessor probabilistic real-time systems with variable execution times. In *Proceedings of the First International Real-Time Scheduling Open Problems Seminar (RTSOPS 2010), joint workshop with the 22nd Euromicro International Conference on Real-Time Systems (ECRTS 2010)*, July 2010.