Global Laxity-Based Scheduling on Multiprocessor Resource Reservations

Open Problem

João Pedro Craveiro and José Rufino
Universidade de Lisboa, Faculdade de Ciências, LaSIGE — Lisbon, Portugal
jcraveiro@lasige.di.fc.ul.pt, ruf@di.fc.ul.pt

I. Introduction

In uniprocessor scheduling, both EDF and LLF are found to be *optimal*, in the sense that both are able to schedule any feasible task set. EDF has received more preference and attention because, both being optimal, it leads to less preemptions than LLF. In global multiprocessor, neither EDF and LLF are optimal [1]. EDF, for instance, is subject to the Dhall effect [2], which causes relatively light task sets to be unschedulable. The prototypical task set for this effect has a high utilization task which, due to its characteristics, is relegated to a lower priority. LLF does not suffer from the Dhall effect, but is still characterized by a higher number of preemptions. To capture the best of these algorithms, Lee [3] has proposed EDZL (Earliest Deadline until Zero Laxity), a variation of EDF that promotes tasks to highest priority when they reach zero laxity; this allows scheduling task sets which are unschedulable with EDF [4].

In this paper, we focus on the problem of scheduling a task set on a *multiprocessor resource reservation*, i.e., in a scenario in which some (or all) processors may be at some times unavailable. We focus on the approach of adding laxity-based priority assignment decisions to EDF, as in the case of EDZL.

II. System model and background

Constrained-deadline sporadic task model: Each task, $\tau_i \overset{\text{def}}{=} (T_i, C_i, D_i)$, releases an infinite sequence of jobs, whose release times are separated by at least T_i time units. The jth job of task τ_i, $J_{i,j} \overset{\text{def}}{=} (a_{i,j}, e_{i,j}, d_{i,j})$ is released at time instant $a_{i,j} \geq a_{i,j-1} + T_i$ and must receive $e_{i,j} \leq C_i$ units of execution capacity within the time interval $[a_{i,j}, d_{i,j}[$ (with $d_{i,j} = a_{i,j} + D_i$). A job which has arrived and has not yet executed $e_{i,j}$ is said to be *active*. At some instant t, the *laxity* of an active job $J_{i,j}$ is the difference between how much time there is until the deadline $(t - d_{i,j})$ and the remaining execution (i.e., the units of execution capacity that such job must receive within the time interval $[t, d_{i,j}[$). We assume the first jobs of all tasks arrive at $t = 0$.

Scheduling algorithms: We consider that jobs are scheduled over the multiprocessor resource reservation using a *work-conserving unrestricted-migration global* policy. This means that, at each instant, the highest priority active jobs will be selected for execution; the number of jobs that will execute is upper-bounded by the number of available processors. An active job may be preempted from one processor and, eventually, resume execution on another processor. We now describe the priority assignment policies used by the algorithms refered to in this paper:

- **EDF** Active jobs are prioritized according to their absolute deadlines. The higher a job's deadline, the higher its priority.
- **LLF** Active jobs are prioritized according to their laxities. The less laxity a job has, the higher its priority.
- **EDZL** Jobs with zero laxity are assigned highest priority. The remaining jobs are prioritized as EDF.

Resource reservation: To describe the open problem, we will consider a simple model for a resource reservation $R \overset{\text{def}}{=} (\Pi, a, m)$, which is defined as a reservation of an identical multiprocessor platform (with m unit-capacity processors) for a consecutive time units every Π time units; this means that, at any instant, there are either 0 or m processors available. When a job is scheduled on one of the processors for one time unit, its remaining execution decreases by one unit. We also assume that the first a-long interval of availability of the resource starts at $t = 0$.

III. Motivating example and problem

We describe the open problem resorting to a contrived task set $\mathcal{T}$ — with independent tasks $\tau_1 = (20, 6, 20)$, $\tau_2 = (20, 7, 20)$, $\tau_3 = (20, 8, 20)$, and $\tau_4 = (21, 12, 21)$ — and a resource reservation $R = (20, 12, 3)$. When simulating[1] scheduling this example using EDF, we have the first job of task τ_4 miss a deadline at $t = 21$ (Fig. 1a); grayed-out boxes represent platform unavailablity, and the bar traced in orange is the job's execution beyond the deadline (its response time is 5 time units beyond the deadline). This is because the jobs of the remaining three tasks, having earlier deadlines, occupied the three processors for enough time to push the execution of τ_4's job forward. The observed phenomenon is the same we see in classical examples of the Dhall effect [2], which EDZL aims to eliminate. However, when simulating the scheduling of the same example using EDZL we see no difference (Fig. 1b). When the laxity of τ_4's job reaches zero, the jobs of the remaining tasks have already finished, so the promotion to the highest priority has no practical effect.

This work was partially supported by FCT/Égide (PESSOA programme), through the transnational cooperation project SAPIENT; by the EC, through project IST-FP7-STREP-288195 (KARYON); and by FCT, through project PTDC/EEI-SCR/3200/2012 (READAPT), multiannual funding to LaSIGE (UI 408), and Doctoral Grant SFRH/BD/60193/2009.

[1]Source code available at http://lasige.di.fc.ul.pt/~jcraveiro/EDzetaL/.

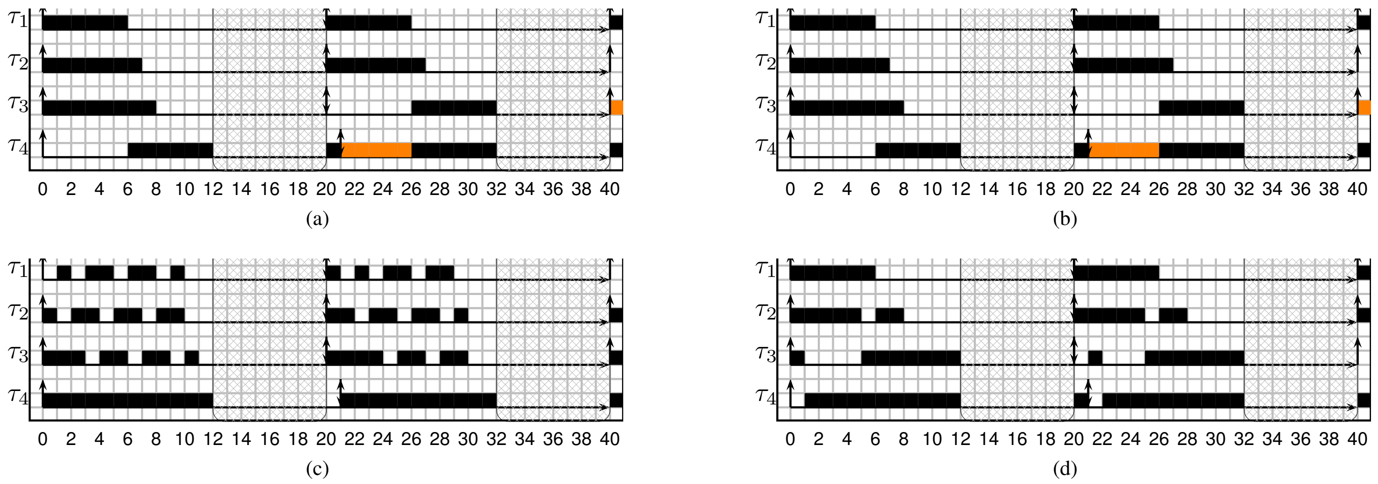

Fig. 1: Scheduling $\mathcal{T}$ on $R = (20, 12, 3)$ using (a) EDF; (b) EDZL; (c) LLF; (d) EDζL.

Problem: *The classical notion of laxity does not consider the unavailability of the platform. As such, comparing laxity to a constant value (e.g., EDZL) is inappropriate when dealing with resource reservations.*

We did not observe deadline misses when simulating with LLF either (Fig. 1c); the classical notion of laxity is not an issue for LLF, since it only compares the laxities of different jobs among them (not against a constant value, like EDZL). A higher number of preemptions is, however, clearly seen.

To the best of our knowledge, the problem of analysing laxity-based scheduling algorithms on multiprocessor resource reservations has not been approached in the literature.

IV. Proof of concept: EDζL

As a proof of concept, we have implemented an algorithm which we call Earliest Deadline until ζ Laxity (EDζL)[2]. The value of ζ may be greater than zero, and should be appropriate for the available resource reservation. For our simple example and model, we can simply assign $\zeta = \Pi - a$. We implement the priority assignment policy as follows: jobs with laxity $\leq \zeta$ are prioritized as LLF among them, all of them having higher priority than jobs with laxity $> \zeta$; the latter are prioritized among them as EDF. Any remaining ties are broken arbitrarily but consistently. This implementation of EDζL has the advantage of generalizing other algorithms: with appropriate values for ζ, EDζL behaves like EDF ($\zeta \ll 0$), LLF ($\zeta > \max_{\tau_i \in \tau}(D_i - C_i)$), or EDZL ($\zeta = 0$).

We have simulated scheduling our example using EDζL, with $\zeta = 20 - 12 = 8$, for 840 time units — $2 \times \operatorname{lcm}_{\tau_i \in \tau} T_i$. The trace for the initial 40 time units of the simulation can be seen in Fig. 1d. For the whole length of the simulation, no deadlines were missed, and there are less preemptions than with LLF (cf. Fig. 1c).

V. Conclusion and future work

We have presented the inappropriateness of EDZL for the case when scheduling is performed over a multiprocessor resource reservation — i.e., a multiprocessor platform which can be partially or totally unavailable at some times. We have illustrated the problem with a contrived example and (deliberately simple and optimistic) resource model, and shown EDζL as a proof of concept for the general idea of a possible solution. The main open questions are: **(i)** *What is the dynamics of ζ which achieves better performance?* **(ii)** *How can we determine schedulability with such a scheduling policy on various multiprocessor resource reservations* [6]–[9]*?* Our intuition is that the solution for this problem may be based on comparing an adapted notion of laxity to 0 (or, equivalenty, comparing classical laxity to a dynamic ζ); in turn, this adapted notion of laxity should consider the difference between how much execution capacity one single job can have available until its deadline (according to the resource model) and its remaining execution.

References

[1] M. L. Dertouzos and A. K. Mok, "Multiprocessor online scheduling of hard-real-time tasks," *IEEE Trans. Softw. Eng.*, vol. 15, no. 12, Dec. 1989.

[2] S. K. Dhall, "Scheduling periodic-time-critical jobs on single processor and multiprocessor computing systems," Ph.D. dissertation, University of Illinois at Urbana-Champaign, 1977.

[3] S. K. Lee, "On-line multiprocessor scheduling algorithms for real-time tasks," in *TENCON '94)*, Aug. 1994.

[4] T. P. Baker, M. Cirinei, and M. Bertogna, "EDZL scheduling analysis," *Real-Time Syst.*, vol. 40, no. 3, 2008.

[5] R. I. Davis and S. Kato, "FPSL, FPCL and FPZL schedulability analysis," *Real-Time Systems*, vol. 48, no. 6, 2012.

[6] I. Shin, A. Easwaran, and I. Lee, "Hierarchical scheduling framework for virtual clustering of multiprocessors," in *ECRTS '08*, Jul. 2008.

[7] E. Bini, M. Bertogna, and S. Baruah, "Virtual multiprocessor platforms: Specification and use," in *RTSS '09*, Dec. 2009.

[8] G. Lipari and E. Bini, "A framework for hierarchical scheduling on multiprocessors: From application requirements to run-time allocation," in *RTSS '10*, Dec. 2010.

[9] A. Burmyakov, E. Bini, and E. Tovar, "The Generalized Multiprocessor Periodic Resource interface model for hierarchical multiprocessor scheduling," in *RTNS '12*, Nov. 2012.

[2]Akin to Fixed Priority until Static Laxity (FPSL) [5].