Partitioned EDF Scheduling in Multicore systems with Quality of Service constraints

Nadine Abdallah, Audrey Queudet, Maryline Chetto, Rafic Hage Chehade

Partitioned EDF Scheduling in Multicore systems with Quality of Service constraints

Nadine Abdallah*, Audrey Queudet†, Maryline Chetto*, Rafic Hage Chehade§
*IRCCyN
University of Nantes, Nantes, France
Email: firstname.lastname@irccyn.ec-nantes.fr
†LINA
University of Nantes, Nantes, France
Email: audrey.queudet@univ-nantes.fr
§IUT Saida
Lebanese University, Saida, Lebanon
Email: rhagechehade@ul.edu.lb

***Abstract*— In this paper we study the partitioned EDF scheduling in a homogeneous multiprocessor environment with Quality of Service (QoS) constraints. The system considered here is a real-time multiprocessor system assumed to be powered by rechargeable batteries. We address the issue of how to best partition a set of firm real-time tasks that can occasionally skip one instance according to a predefined QoS threshold. The main goal is to minimize the energy consumption of the system while offering solutions with respect to transient energy starvation situations the system can experiment. The contribution of the paper is twofold. First, we present a schedulability analysis of firm multiprocessor task sets under QoS constraints. Second we propose new partitionning heuristics integrating skips. The evaluation is conducted from several points of view (minimization of the total processor number, maximization of the spare capacity on each processor).**

I. Introduction

The computer science literature generally divides real-time systems in three main categories: *soft, hard and firm* [10]. *Soft* real-time systems allow some jobs to miss their deadlines in order to improve resource usage or average performance. *Hard* real-time systems embody guaranteed timing and cannot miss deadlines. A single failure in one timing constraint can cause an intolerable cost (in terms of human lives, equipment damage or economic loss). *Firm* real-time systems instead allow some of their constraints to be occasionally lost, the performance being then quantified in terms of *Quality of Service (QoS)*.

In recent years, there has been increasing interest to incorporate real-time scheduling techniques that deal with power/energy conservation. Many energy-oriented real-time scheduling techniques have been proposed to reduce energy consumption. Among them, Dynamic Voltage and Frequency Scaling (DVFS) algorithms [5] reduce energy consumption by changing processor speed and voltage at run-time depending on the needs of the applications running. Another trend is based upon Dynamic Power Management (DPM) policies [3] which trade off the performance for the power consumption by selectively placing components into low-power states. All these techniques have been proposed for uniprocessor real-time systems. However, with todays computational demands and as the miniaturization of integrated circuits reaches its physical limits, a valid solution to supply sufficient resources is to use multicore platforms, but actually, much less work has been done for power awareness in multicore systems.

In our research we consider multicore real-time systems with both QoS and energy constraints. The objective is to exploit the flexibility offered by QoS-based real-time tasks to face both processor overload and energy starvation situations, skipping the execution of some task instances. For that purpose, we propose to tackle the problem of the repartition of QoS-constrained tasks over such platforms. Our contribution is twofold. First, we design a schedulability test for firm multiprocessor task sets under QoS constraints. Second, based on this test, we propose new partitioned scheduling heuristics to assign tasks with QoS constraints to processors, so as to minimize the number of processors, thus minimizing the energy consumption. To the best of our knowledge, this paper is the first to introduced QoS contraints into partitioning heuristics.

We rely on previous partitioning results [12][9][13] to preselect which partitioning heuristic and which task set sorting criterion can best fit both QoS and energy constraints. The performance of each heuristic is analyzed according to the success ratio (i.e. the number of schedulable task sets among the total number of generated task sets). We also study the influence of the task sorting criteria on the schedulability of each heuristic so as to underline the impact of task sorting criterion on the performance of the heuristics chosen.

The remainder of this paper is organized as follows: in Section 2, we describe some background material. Section 3 presents the models and definitions considered in this paper. Section 4, 5 and 6 presents the main contribution which begins with a schedulability analysis under QoS constraints and then relies on the partitioning under QoS contraints to end with performance evaluation. In Section 7, we conclude and give some future lines of investigation.

II. Background material

A. EDF Scheduling

Earliest Deadline First (EDF) scheduling algorithm [6] is an algorithm in which jobs with earliest deadlines have higher priority. EDF is an optimal scheduling algorithm on preemptive uniprocessors, in the following sense: if a collection of independent jobs can be scheduled (by any algorithm) such that all the jobs complete by their deadlines, EDF will schedule this collection of jobs such that they all complete by their deadlines.

B. Skip-Over model

We consider a uniprocessor system consisting of firm periodic and preemptable tasks. Tasks are assumed to be independent. Each task is divided into instances where each instance occurs during a single period of the task. The possibility of skipping task instances was introduced by Koren and Shasha [8]. In their model, a task is characterized by its worst case execution time, its period, its deadline and a skip parameter s ($2 \leq s \leq \infty$). This parameter gives the tolerance of a task to miss deadlines. The higher s is, better is the QoS. It represents the minimum QoS required by a task. Every instance of the task can be red or blue. Red instances must complete before their deadline but blue instances can be skipped or aborted at any time. A task set is deeply-red when all tasks instances are initially activated at the same time and are red. The distance between two consecutive skips must be at least s periods. After missing a deadline, the next $s-1$ instances must complete before their deadlines. On the contrary, if a blue instance completes within its deadline, the next instance is still blue.

Here are two algorithms based on the Skip-Over model:

- *RTO (Red Tasks Only)*: this algorithm never tries to execute blue instances. Red instances only are scheduled according to EDF. In the deeply-red model, this algorithm is optimal which means that all feasible task sets will be schedulable using RTO.
- *BWP (Blue When Possible)*: this algorithm is more flexible in the sense that it schedules red instances according to EDF and tries to schedule blue instances when there are no ready red instances.

C. Partitioning

Partitioning a task set is equivalent to the Bin-Packing problem: how to place n objects of different sizes in m identical boxes. This problem is known to be NP-hard. The only known solution for this kind of problem is to enumerate all possible configurations and verify their correctness one by one. Some heuristics [12][9][13] have been proposed in the literature in order to solve it. All of them imply a sequential assignment of tasks to processors: a task is assigned on a processor if it verifies the schedulability test after assignment. According to First Fit (FF) a task is assigned to the first possible processor, starting from π_1 (the first processor). Best Fit (BF) assigns a task τ_i to the processor which minimizes the remaining processor capacity. According to Worst Fit (WF) a task is assigned to the processor which maximizes the remaining processor capacity. Next Fit (NF) assigns a task τ_i to the first possible processor in the range $\pi_j, ..., \pi_m$ (π_j being the current processor). The procedure starts from π_1 .

III. Models and Definitions

A. System and task model

In this paper, we consider π a platform with m identical processors: $\pi = \{\pi_1, ..., \pi_m\}$. We refer to the Skip-Over periodic task model. A periodic task τ_i is defined by a 4-tuple (C_i, P_i, D_i, s_i) where C_i is the worst-case execution time (WCET), P_i the period, D_i the relative deadline and s_i the skip parameter. A task can be instantiated an infinite number of times. The task set $\tau = \{\tau_1, ..., \tau_n\}$ is composed of n periodic tasks with constrained deadline ($D_i \leq P_i$). Tasks are assumed to be independent and preemptable. The system is considered deeply-red.

Each task is characterized by:

- an utilization factor: $u_i = \frac{C_i}{P_i}$
- an equivalent utilization factor integrating skips: $u_i^* = \frac{C_i}{P_i} \times \frac{s_i - 1}{s_i}$
- a density: $\delta_i = \frac{C_i}{min\{Di, Ti\}}$.
- an equivalent density integrating skips: $\delta_i^* = \frac{C_i}{min\{D_i, P_i\}} \times \frac{s_i - 1}{s_i}$

B. Known results

1) Processor utilization factor: Given a set $\tau = \{\tau_i(P_i, D_i, C_i)\}$ of n periodic tasks scheduled on a processor, the processor utilization factor is defined as [11]:

$$U_\tau = \sum_{i=1}^{n} \frac{C_i}{P_i} \tag{1}$$

2) Equivalent processor utilization factor: Given a set $\tau = \{\tau_i(P_i, D_i, C_i, s_i)\}$ of n periodic tasks with implicit deadline that allow skips, the equivalent processor utilization factor integrating skips is defined as [8]:

$$U_\tau^* = max_{L \geq 0}\{\frac{\sum_i D(i, [0, L])}{L}\} \tag{2}$$

where $D(i, [0, L]) = \lfloor \frac{L}{P_i} \rfloor - \lfloor \frac{L}{P_i s_i} \rfloor$.

3) Load factor: Given a set $\tau = \{\tau_i(P_i, D_i, C_i)\}$ of n periodic tasks with arbitrary deadlines, the load factor is defined as [7]:

$$Load(\tau) = max\{U_\tau, sup_{L \in [D_{min}, P)} \frac{DBF(\tau, L)}{L}\} \tag{3}$$

where $D_{min} = min\{D_1, ..., D_n\}$ and $P = lcm\{P_1, ..., P_n\}$. For a given t, the Demand Bound Function (DBF) represents the upper bound of the workload generated by all tasks with activation times and absolute deadlines in the same interval $[0, t]$:

$$DBF(\tau, [0, L]) = \sum_{i=1}^{n} \left(1 + \lfloor \frac{L - D_i}{P_i} \rfloor\right) \times C_i \tag{4}$$

DBF will be computed for L corresponding to absolute task deadlines between 0 and D_{min}.

4) Exact schedulability test: A necessary and sufficient (i.e. exact) schedulability test for EDF for arbitrary deadlines on uniprocessor systems is given by [11]:

$$Load(\tau) \leq 1 \tag{5}$$

IV. Schedulability analysis under QoS constraints

5) New equivalent load factor: Given a set $\tau = \{\tau_i(P_i, D_i, C_i, s_i)\}$ of n periodic tasks with arbitrary deadlines that allow skips, we define the equivalent load factor as:

$$Load_{QoS}(\tau) = max\{U^*_\tau, sup_{L\in[D_{min},P)} \frac{DBF_{QoS}(\tau, L)}{L}\} \tag{6}$$

where $D_{min} = min\{D_1, ..., D_n\}$ and $P = lcm\{s_1P_1, ..., s_nP_n\}$. According to the RTO algorithm, the DBF changes. Its new equation is :

$$DBF_{QoS}(\tau, [0, L]) = \sum_{i=1}^{n} \left(1 + \lfloor \frac{L - D_i}{P_i} \rfloor - \lfloor \frac{1}{s_i}\left(1 + \lfloor \frac{L - D_i}{P_i} \rfloor\right) \rfloor\right) \times C_i \tag{7}$$

As the DBF_{QoS} changes its value only at instants corresponding to absolute deadlines, DBF_{QoS} will be computed for L corresponding to absolute tasks deadlines between 0 and D_{min}. The equivalent processor utilization factor U^*_τ integrating skips is defined as it previously appears in equation (2).

6) New exact schedulability test: A necessary and sufficient (i.e., exact) schedulability test for EDF for arbitrary deadlines QoS-constrained tasks on uniprocessor systems is given by :

$$Load_{QoS}(\tau) \leq 1 \tag{8}$$

V. Partitioning under QoS constraints

A. Identification of suitable heuristics

As mentioned previously, partitioning, which reduces to a bin-packing problem, is known to be NP-Hard. According to previous results in [12], we will use two heuristics :

- First Fit (FF) because it minimizes the number of processors used (in our case, it will minimize the energy consumption);
- Worst Fit (WF) because it maximizes the remaining processor capacity, thus offering more flexibility for recharging the system in case of energy starvation.

These heuristics have been adapted in our case. We systematically apply a schedulability test (see equation (8)) before assigning a task to a processor. If the test is not verified we continue testing on others processors until we find the good one able to accept the task. If a task is not assigned to any processor, it means that the task set is not schedulable.

B. New task sorting criteria

All the partitioning algorithms proposed in the literature often include task sorting criteria with the purpose of increasing their success ratio (i.e. ratio of schedulable task sets). A task sorting criteria consists in ordering the tasks before assigning them to a processor. It has an influence on the tasks assignment on processors. For instance, sorting tasks in order of decreasing density will force to first assign heaviest tasks on the processors.
Based on the observation of previous results in the literature [12], we retained three criteria: the density, the utilization factor and the period. We adapted them to QoS constraints, analyzing the general influence of the 4 following criteria in increasing/decreasing order: equivalent density δ^*_i, equivalent utilization factor u^*_i, period multiplied by skip parameter P_is_i and the skip parameter itself s_i

VI. Performance Evaluation

In this section, we evaluate the performance of all possible combinations between a series of 2 partitioning heuristics (adapted FF and WF) and 8 task sorting criteria (δ^*_i, u^*_i, P_is_i, s_i considered in increasing/decreasing order).

A. Task generation methodology

In our task generation methodology we consider tasks with constrained deadlines ($D_i \leq T_i$) which is the hardest assumption to take.

- P_i is uniformly chosen from $[20, 40]$;
- C_i is set to provide a task utilization factor equal to the one given in input;
- D_i is uniformly chosen between C_i and P_i;
- s_i is uniformly chosen from $[s_{min}, s_{max}]$ where s_{min} and s_{max} are input parameters;
- m is the number of processors used and is given as a parameter.

We consider m-identical processor platforms. For m processors, we generate a system that contains $2 \times m$ tasks. $\forall i \in [1, n], \delta_i \leq 1$, which means that task density doesn't exceed the utilization capacity of a processor. For our simulations, we generated 1000 different task sets.

B. Performance criteria

We evaluate each combination of the previously mentioned parameters according to a performance parameter named *Success_Ratio* which is defined by:

$$Success\_ratio = \frac{N_{success}}{N_{generated}} \tag{9}$$

where $N_{success}$ denotes the number of task sets successfully scheduled and $N_{generated}$ the total number of task sets generated. This criteria helps us to determine which combination schedules the largest number of task sets.

C. Experimental results

In this section, we present an initial empirical investigation for QoS-constrained tasks defined according to the Skip-Over model, examining the effectiveness of our heuristics on 4 processors. This section deals with the impact of a task sorting criterion on the success ratio of schedulability tests.
In the corresponding graphs, D means "decreasing", I means "increasing" and E means "equivalent".

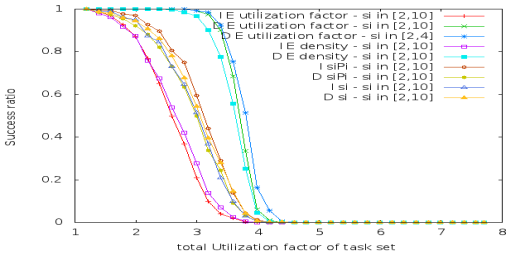

Fig. 1. EDF-based sorting criteria with WF heuristic

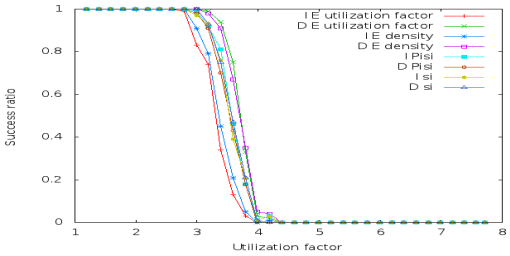

Fig. 2. EDF-based sorting criteria with FF heuristic with s_i in $[2, 10]$

The simulation results depicted in Figure 1 and 2 show that, taking all heuristics together, the sorting criteria which maximize the success ratio are: *Decreasing Equivalent Utilization* and *Decreasing Equivalent Density*. Figure 1 shows that for task sets with a total utilization factor less than 25% of the platform capacity, all sorting criteria give the same performance. However, for task sets with total density greater than 25% of the platform capacity, *Decreasing Equivalent Utilization* factor and *Decreasing Equivalent Density* exhibit the best behavior. *Decreasing Equivalent Utilization* factor is slightly more efficient than *Decreasing Equivalent Density*. Figure 2 shows that for task sets with the total utilization factor less than 75% of the platform capacity, all sorting criteria give the same performance. However, for task sets with total density greater than 75% of the platform capacity, *Decreasing Equivalent Utilization* factor and *Decreasing Density* exhibit the best behavior. In fact, for a total utilization factor equals to 3.2, at most 99% of task sets are schedulable with *First Fit Decreasing Equivalent Utilization* and 84% with *Worst Fit Decreasing Equivalent Utilization* and *Density*. In Figure 1, we also notice for *Decreasing Equivalent Utilization* that when skips are comprised in $[2, 4]$ we have a better success ratio than when skips are comprised in $[2, 10]$. In fact, the skip parameter has also an influence on schedulability: the more we authorize skips higher is the success ratio.

VII. Conclusion

While low-power uniprocessor real-time systems have fueled much recent work on energy-aware scheduling, the same issue upon multicore platforms has been somewhat neglected. Motivated by this, we proposed a flexible solution based on QoS-constrained real-time tasks, allowing to skip some task instances in case of either processor overload or energy starvation.

The major contributions of our study are as follows: **(i)** we provide a schedulability test for multiprocessor task sets under QoS contraints, **(ii)** we propose and evaluate two new partitionning heuristics (First Fist variant that minimizes the energy consumption, and Worst Fist variant that leaves maximum spare capacity on processors in order to allow the battery to charge). From the simulations, we showed that the best task sorting criterion are *equivalent decreasing density* and *equivalent decreasing utilization*. In the future, we want to extend this study to semi-partitioning algorithms, providing a complete analysis with new evaluation criteria such as the number of migrations and preemptions.

Acknowledgment

The work presented in this paper is sponsored by a CE-DRE project, namely GreenEmbedded, which is a bilateral collaboration between University of Nantes and the Lebanese University.

References

[1] S. Baruah and N. Fisher, *The partitioned multiprocessor scheduling of deadline-constrained sporadic task systems*, IEEE Transactions on Computers, Vol.55, No.7, pp. 918-923, 2006.
[2] S. Baruah, R. Howell, and L. Rosier, *Algorithms and complexity concerning the preemptive scheduling of periodic real-time tasks on one processor*, Journal of Real-Time Systems, Vol. 2, pp. 301-324, 1990.
[3] L. Benini, A. Bogliolo, and G. De Micheli, *A Survey of Design Techniques for System-Level Dynamic Power Management*, IEEE Trans. VLSI Systems, Vol. 8, No.3, pp. 299-316, 2000
[4] M. Caccamo, G. C. Buttazzo, *Optimal Scheduling for Fault-Tolerant and Firm Real-Time Systems*, Proceedings of the IEEE Real-Time Computing Systems and Applications, 1998.
[5] J.-J. Chen and T.-W. Kuo. Energy-efficient scheduling for real-time systems on dynamic voltage scaling (DVS) platforms. *In 13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 28-38. IEEE Computer Society, August 2007.
[6] M.-L. Dertouzos. Control Robotics: The Procedural Control of Physical Processes, *Proceedings of International Federation for Information Processing Congress*, pp. 807-813, 1974.
[7] L. George and J. Hermant, *A norm approach for Partitioned EDF Scheduling of Sporadic Task Systems*. Proceedings of the 21st Euromicro Conference on Real-Time Systems, Dublin, Ireland, July 2009.
[8] G. Koren , D. Shasha *Skip-over: Algorithms and Complexity for Overloaded Real-Time Systems*, Proceedings of the IEEE Real Time Systems, 1995.
[9] C.-L. Liu *Scheduling algorithms for multiprocessors in a hard real-time environment*, JPL Space Programs Summary, Vol. 37-60, pp. 28-31, 1969.
[10] J.-W.-S. Liu, *Real Time Systems*, Prentice Hall, 2000.
[11] C.-L. Liu and W. Layland, *Scheduling algorithms for multi-programming in a hard real time environment*, Journal of ACM, Vol.20, No.1, pp. 46-61, 1973.
[12] I. Lupu, P. Courbin, L. George and J. Goossens, *Multi-Criteria Evaluation of Partitioning Schemes for Real-Time Systems*, IEEE Conference on Emerging Technologies and Factory Automation (ETFA), 2010.
[13] O. U. Peirera Zapata and P. Mejia-Alvarez, *Analysis of Real-Time Multiprocessors Scheduling Algorithms*, Proceedings of the RTSS, 2003.