DOI. 10.1109/SACI.2018.8440932

P_FENP: A Multiprocessor Real-Time Scheduling Algorithm

Eugenia A. Capota, Cristina S. Stangaciu, Mihai V. Micea, Vladimir I. Cretu
Politehnica University, Timisoara, Romania
Computer and Information Technology Department
2, Vasile Parvan Blvd., 300223, Timisoara, Romania
eugenia.capota@dsplabs.cs.up.ro, certejan@dsplabs.cs.upt.ro, mihai.micea@cs.upt.ro, vladimir.cretu@cs.upt.ro

***Abstract*—This paper addresses the problem of real-time scheduling on multiprocessor systems for periodic tasks when scheduling jitter is not allowed. A partitioned real-time scheduling method based on a table-driven uniprocessor algorithm called fixed execution non-preemptive (FENP) is proposed as a solution for this problem. An analysis of the new proposed algorithm is provided in terms of scheduling jitter and schedulability ratio, by comparison against other popular partitioned real-time scheduling algorithms.**

I. INTRODUCTION

The continuous integration of embedded real-time systems in fields such as robotics, avionic, automotive, telecommunications and cyber-physical systems in general has led to an increase in software complexity and processing demands for the real-time systems. Due to the direct interaction between the system and the environment, a real-time system is required not only to function correctly but also to guarantee that the response is given in certain timeframes, therefore the task scheduling algorithm is an essential component in such a scheme. Due to the hardware development, on one hand, and to the increasing application complexity on the other, multiprocessor platforms are becoming more common.

Many application running on the real-time systems have a critical impact on the environment and humans, thus predictability is a key aspect of those systems. Sometimes, in data acquisition, control applications or digital-communications the presence of jitter is intolerable, thus a jitterless scheduling algorithm is required.

A. Task model

In real-time systems a periodic task set τ_i, is defined by a few parameters: the worst-case execution time (WCET) C_i, a relative deadline D_i and a period T_i. Each task set generates a number of jobs that arrive at T_i units apart and each job requires not more than C_i units to execute before its deadline occurs. D_i describes the number of time units a job is allowed to execute after its arrival.

There are three types of tasks [1]: periodic, aperiodic and sporadic. Periodic tasks arrive at regular time intervals and have relative deadlines, while aperiodic tasks are invoked only once, they do not have relative deadlines and their arrival time is unknown. Lastly, sporadic tasks have arbitrary arrival times but a well-defined minimum inter-arrival time between two consecutive invocations. In this paper only periodic tasks will be considered for the evaluation of scheduling policies.

During the execution of tasks, some parameter variations may occur, which can affect the predictability of the system. These disturbances are known as jitter [2]. The start time of a task always suffers from deviations from its release time due to various factors, which must be considered when developing a scheduling policy. Therefore, scheduling jitter refers to the delay between the moment a task activates and its release (start time) [3]. In the context of periodical tasks, task execution jitter can be defined as the deviation in the task periodicity.

B. Scheduling schemes

In this paper we focus on the multiprocessor systems. A multiprocessor system is composed of several processors upon which jobs can execute on. There are three types of classes used in real-time multiprocessor scheduling: global, partitioned and the hybrid approach.

In global scheduling all tasks are stored in a single priority queue, from which the global scheduler selects the highest priority tasks for execution. This scheme allows task migration from one processor to another which is an unpredictable source of overhead which reflects in task execution jitter.

In partitioned scheduling each task is assigned to a single processor and each processor is scheduled independently. This technique presents the following advantages [4]:

- Overhead is reduced because each processor uses a separate run queue.
- There is no penalty in terms of migration.

Unfortunately, the approach has a few negative consequences:

- Processors are often idle, because they cannot be used by tasks assigned to a different processor.
- Some task sets are schedulable if and only if migration takes place.
- Finding an optimal way to allocate tasks to processors is a bin-packing problem, which is NP-hard in the strong sense.

The hybrid scheduling approach combines elements of the previously mentioned scheduling classes, partitioned and global.

Because of its reduced scheduling overhead, and because its predictability, we will consider only the partitioned scheduling class in this article.

On the other hand, at the processor level, four classes of scheduling algorithms can be considered [4]:

Fixed task-priority (FTP) - where all the invocations, or jobs of the same task are given the same priority.

Fixed job-priority (FJP) - here different priorities can be allocated to different jobs of the same task. However, the priority cannot change between the release and completion time of a job.

Dynamic-priority (DP) - unlike FJP, the priority of a job can change between its arrival and completion time.

Hybrid-priority (HP) - it combines features of multiple scheduling algorithm classes.

Even though there are great efforts in bounding and reducing the task execution jitter for several single processor scheduling algorithms [5], none of the scheduling classes described above can provide zero task execution jitter like a table-driven scheduling algorithm. While table-driven algorithms are common in single processor real-time systems, they are very rarely used on multiprocessors. One of the reasons is that for a table-driven algorithm, the table is static and determined offline, before the system is started; thus making the use of this type of algorithms very difficult for multiprocessor systems.

In this paper, we propose a solution to this problem, by using a partitioned table-driven approach, based on the monoprocessor scheduling algorithm FENP.

Tests were conducted and a comparison is made between three uniprocessor schedulers: Rate Monotonic Non-preemptive (RMNP) [6], Earliest Deadline First Non-preemptive (EDFNP) [7] and the First Execution Non-preemptive (FENP) [8] technique. Furthermore, their partitioned approaches were also compared. On the global level, we used several partitioning schemes [9]: First-Fit Decreasing (FFD), Next-Fit Decreasing (NFD), Best-Fit Decreasing (BFD) and Worst-Fit Decreasing (WFD). An additional heuristic will be integrated for the FENP algorithm.

The remainder of this paper is organized as follows. Section II covers some related work, introducing real-time scheduling algorithms on uniprocessor and multiprocessor platforms, relevant to the conducted experiments. In Section III, the P_FENP scheduling approach is described, while Section IV presents results from simulation experiments, comparing a few known scheduling algorithms to the FENP technique. Lastly, the paper concludes in Section V.

II. Related Work

As mentioned before, a partitioned multiprocessor scheduling algorithm consists of two components: a global scheduler that assigns tasks to processors, and a local scheduler in each partition, used to handle the execution of tasks.

Because of its reduced overhead and predictability [10] only partitioned scheduling is considered.

As mentioned previously, four known heuristics are used for task partitioning to processors [9]: First-Fit Decreasing (FFD), Next-Fit Decreasing (NFD), Best-Fit Decreasing (BFD) and Worst-Fit Decreasing (WFD). The goal of these algorithms is to process the tasks from the ready queue one by one and assign each task to a processor.

It was proved that FFD and BFD have an absolute worst-case ratio of $3/2$ [11]. However the performance of these heuristics depends on the sequence of requests. Research has shown that both FFD and BFD are better than WFD in terms of decreasing time utilization and storage [12]. In addition, between the two, FFD is faster. NFD is also less efficient than FFD [13].

After tasks are assigned to partitions, they are processed locally by a scheduling algorithm. Some of the most popular scheduling algorithms at the processor level are:

A. Earliest Deadline First (EDF)

The EDF algorithm [7] is one of the most popular of the fixed job-priority class. Its main advantage is that it is optimal at the processor level; it was proved that EDF allows using resources efficiently and improves responsiveness of aperiodic tasks because the processor is fully utilized [14]. Regarding the jitter, it provides a higher task execution jitter than the FTP scheduling algorithms but its jitter has been proved to be bounded [15].

B. Rate Monotonic (RM)

Considering the FTP class, RM [6] is one of the most popular. According to RM, tasks have fixed priorities ordered by rates. In this case the task with the smallest period has the highest priority. RM is one of the mostly used algorithms in real-time applications because it is easy to implement [14]. Task execution jitter increases while the priority decreases, thus only the highest priority task can have zero task execution jitter.

One of the techniques used for reducing task jitter which is applicable to both of the algorithms mentioned above is to prohibit task preemption by transforming the algorithms into non-preemptive ones [16]. While there are also other techniques for reducing task jitter, in this article we will only focus on this one.

Table driven schedulers, on the other hand can provide up to zero task execution jitter.

The idea of integrating table-driven scheduling algorithms on multi processors has been exploited before, but far less than using other algorithms at the processor level. While the algorithm proposed in [17] uses tables at the global level and not at the processor level, the algorithm proposed in [16] consists of a hierarchical scheduling algorithm which integrates a table-driven component at one of its levels. Here tasks are statistically assigned to each processor where a table-driven scheduler is used. This forms a container which encapsulates the high-criticality tasks, while reducing slack and preserving temporal isolation.

In this work, we will propose a multiprocessor version of the FENP monoprocessor scheduling algorithm. To the best of our knowledge no multiprocessor algorithms which include FENP at the processor level have been considered so far.

III. P_FENP: Partitioned FENP

A. FENP

FENP is a non-preemptive algorithm used for scheduling a set perfectly periodic tasks, with fixed-execution times during their periods. The technique was developed to provide a maximum predictability for processing applications and acquiring critical-digital signals that are synchronized under a hard real-time compact kernel, HARETICK [18].

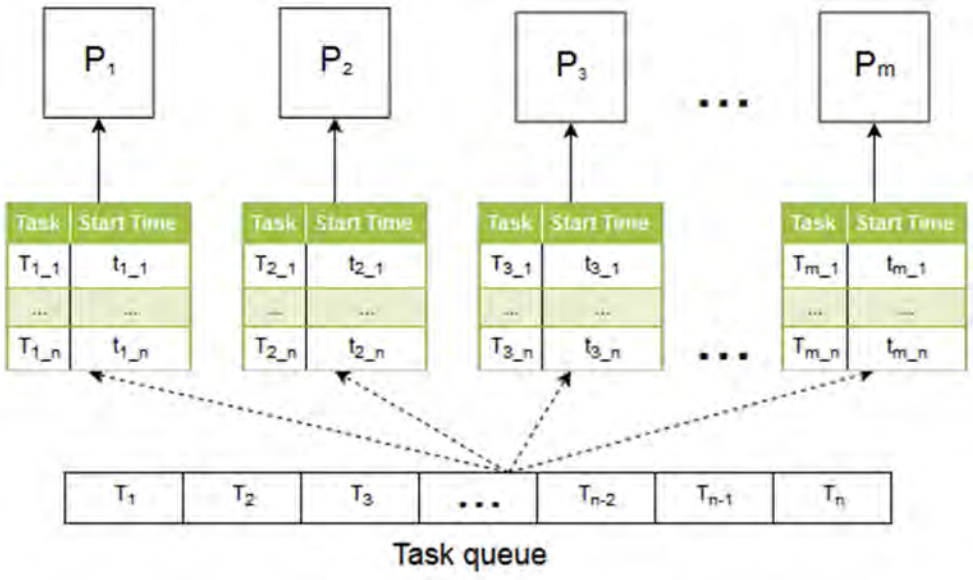

Figure 1. Table-driven partitioning method.

FENP can be briefly described as follows: the execution (start time - s_i) of any consecutive jobs of a task is separated by a time interval equal to the task period (T_i) [8], [19]:

$$s_{i,k+1} = s_{i,k} + T_i, 1 \leq i \leq m \qquad (1)$$

The performance of the FENP have already been analyzed and compared against the non-preemptive version of the EDF, called EDFNP and are available in [20].

Moreover, schedulability tests for this algorithm are available in [8].

The main advantage of the FENP algorithm, beside the fact that it offers a jitter-less task execution, is that it provides a method for determining scheduling tables, thus the tables are not determined manually. Until now, the scheduling table was determined offline on a host and not on the embedded platform, because the complexity of the algorithm. But, due to the hardware performance increase, this is not a restriction anymore.

B. *Partitioned FENP*

An efficient partitioning algorithm for the FENP scheduling policy is proposed in this paper. The algorithm is a table-driven approach, where each processor has a scheduling table associated to it. Tasks are then selected one by one from the ready queue and added in each table associated with a processor, where a test is performed. The test verifies if a task subset is deemed schedulable on that particular processor, in which case, the task will remain in the corresponding scheduling table and the next task is removed from the global queue and tested. If a test returns failure, the task is removed from the list and added in the next processor scheduling table, where the same test is performed. After all the tasks are added to corresponding lists, their start times are calculated. A schematic representation of the process is available in Figure 1.

IV. EVALUATION

The analysis of the P_FENP is made by implementing the algorithm in SimSo [21]. Four global partitioning methods are already available in the simulator and a fifth has been implemented during this research.

An experimental comparison was conducted between several partitioned multiprocessor scheduling policies that use FENP, or one of the two described algorithms already integrated in SimSo, as local schedulers.

SimSo disposes of a built in, two-phase task set generator. The first step is to create a list of task utilization rates by running one of the three available approaches proposed by: Kato et. al [22], Davis et. al [23] and Emberson et. al [24]. The generated utilization sets must then be combined with a list of periods. Therefore, SimSo also implements three period generators: uniform distributions in various fixed ranges [21], log-uniform choice of periods [23] and random draw among a fixed set of values [21]. The UU-Fast Discard algorithm, introduced by Davis et. al [23], was used to determine the task utilizations, while the periods were generated randomly within a uniform period distribution between 10 and 100 ms [21].

A. *Uniprocessor scheduling*

Three non-preemptive uniprocessor scheduling policies were compared: EDFNP (Earliest Deadline First Non-preemptive), RMNP (Rate Monotonic Non-preemptive) and FENP (First Execution Non-preemptive), by varying the total processor utilization for a set of $N = 5$ tasks and $N = 10$ tasks. The total utilization factor is depicted on the x-axis, while the success ratio on the y-axis. Each data-point was determined by randomly generating 10 task sets, and testing their schedulabilty for the three specified algorithms (Figures 2-4).

It can be noted that overall, the success ratio decreases with high processor utilizations. Furthermore, the number of tasks has an inversely proportional impact on the schedulability of the task set, leading to a decrease of the latter (Figure 2-3).

Next, for a total utilization of 0.5, the number of tasks in a task set is varied: 2, 4, 6, 8 and 10.

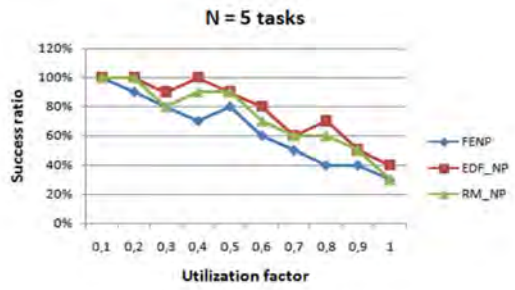

Figure 2. Success ratios (SR) versus total utilization factors for $N = 5$ tasks in a uniprocessor system.

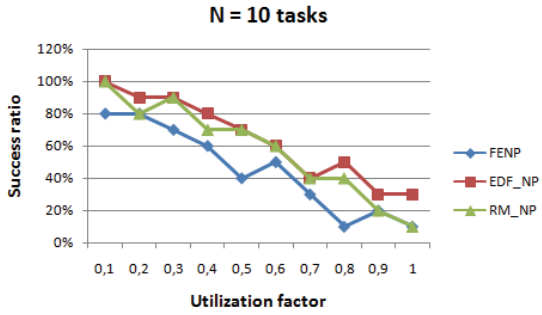

Figure 3. Success ratios (SR) versus total utilization factors for $N = 10$ tasks in a uniprocessor system.

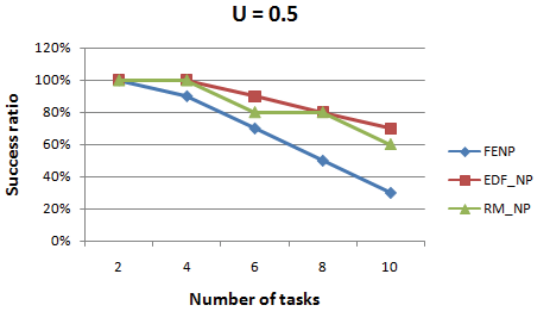

Figure 4. Success ratios (SR) versus number of tasks for a total utilization factor of $U = 0.5$ in a uniprocessor system.

The most important asspect is that the FENP algorithm is able to schedule and execute tasks in a jitter-less manner. In order to avoid the so called release jitter, the execution of a job may start at a later time, however the time between the execution of two successive jobs that belong to the same task must be bounded to $[T, 2 \times T)$, where T is the task period. Figure 5 illustrates a scheduling example for a set of $N = 2$ tasks using EDFNP, RMNP and FENP. While the task execution period is constant under FENP, under EDFNP and RMNP it is not (Figure 5).

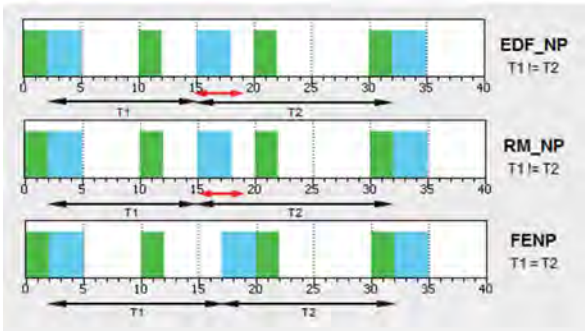

Figure 5. Scheduling of a task set with $N = 2$ tasks using EDF_NP, RM_NP and FENP.

B. *Multiprocessor scheduling*

Another set of similar experiments were conducted on three multiprocessor approaches: partitioned EDF, partitioned RM and partitioned FENP (Figures 6-8). The heuristic used for partitioning tasks to processors, in this case, is First-Fit Decreasing (FFD). A system composed of 2 processors is considered.

The success ratio for the partitioned approaches of EDFNP, RMNP and FENP decrease logarithmically with the total utilization factor. A high utilization results in low schedulability. Increasing the number of tasks in a task set will also cause a decrease in schedulability.

Lastly, a comparison is conducted by modifying the number of tasks in a task set, while preserving the total utilization factor (Figure 8). The same conclusion arises as in the case of the monoprocessor scheduling (the success ratio decreases with the increase in the number of tasks).

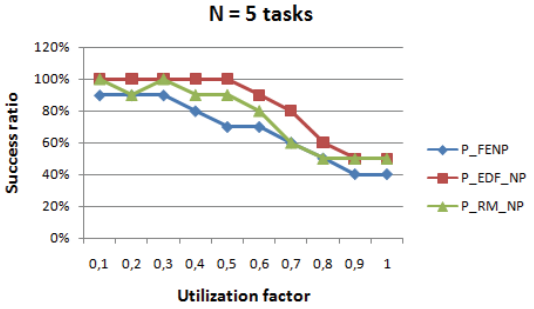

Figure 6. Success ratios (SR) versus total utilization factors for $N = 5$ tasks in a multiprocessor system.

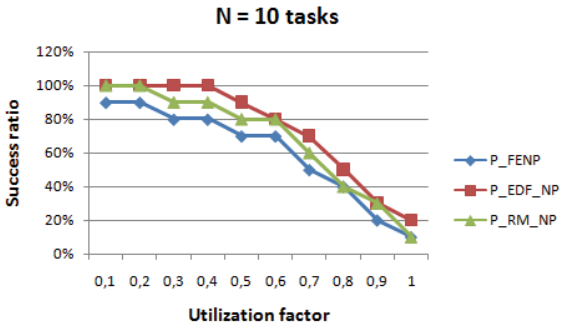

Figure 7. Success ratios (SR) versus total utilization factors for $N = 10$ tasks in a multiprocessor system.

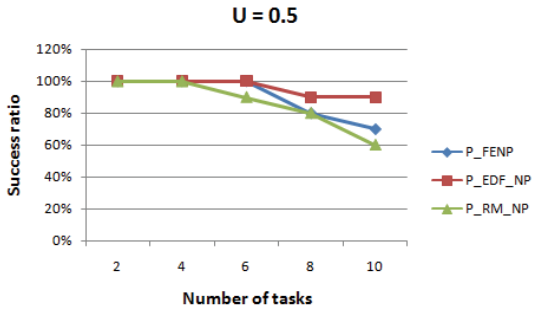

Figure 8. Success ratios (SR) versus number of tasks for a total utilization factor of $\boldsymbol{U = 0.5}$ in a multiprocessor system.

C. P_FENP scheduling

A final evaluation is done where the proposed partitioning algorithm for FENP is compared in terms of schedulability to four different heuristics: FFD, NFD, BFD and WFT. 20 task sets with $N = 4$ tasks are used, each schedulable by the partitioned multiprocessor algorithm proposed for the FENP policy. A test is then conducted to quantify the percentage of task sets schedulable if one of the four known partitioning methods is used instead: FFD, NFD, BFD and WFT. The system is considered dual-core.

From our evaluation, 50% of the task sets schedulable with the proposed heuristic are also schedulable by the FFD partitioning algorithm, while a higher percentage, 75% are compatible with NFD, BFD and WFD (Figure 9).

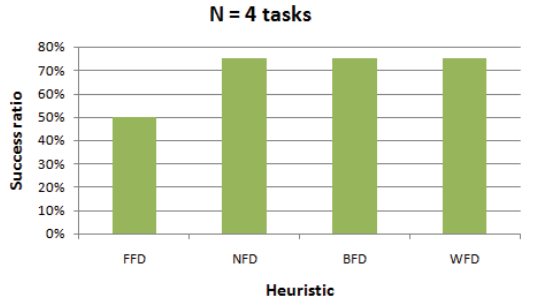

Figure 9. Success ratios (SR) for partitioning heuristics used with the FENP scheduling policy on task sets known to be schedulable by our proposed heuristic.

V. Conclusions and Future Work

In this paper, a partitioned approach is implemented for the FENP scheduling algorithm. The uniprocessor policy is first adapted to a simulation environment, namely SimSo, before developing a heuristic for partitioning tasks to processors effectively in a multi-core setting. Two known scheduling algorithms are studied and a comparison is made both in monoprocessor and multiprocessor systems between their non-preemptive versions and the FENP algorithm.

As it can be seen from our experimental results, a FENP approach is feasible for scheduling real-time task without execution jitter, both in monoprocessor and multi-processor systems. The price to pay is a relatively small decrease in the schedulability ratio, while the predictability coming from the jitterless execution is the main benefit.

Currently SimSo only allows simulation of scheduling algorithms for regular real-time task systems. As future work, the environment can be customized to support mixed-criticality systems, which have components with different levels of criticality and FENP based mixed criticality algorithms can also be developed.

References

[1] D. Isovic and G. Fohler, "Efficient scheduling of sporadic, aperiodic, and periodic tasks with complex constraints," in *Real-Time Systems Symposium, 2000. Proceedings. The 21st IEEE*, 2000, pp. 207-216.

[2] A. Maaita and M. J. Pont, "Using 'planned pre-emption'to reduce levels of task jitter in a time-triggered hybrid scheduler," in *Proceedings of the Second UK Embedded Forum (Birmingham, UK*, 2005, pp. 18-35.

[3] J. Martyna, "Scheduling algorithm for delay and jitter reduction of periodic tasks in real-time systems," *PRZEGLĄD ELEKTROTECHNICZNY (Electrical Review), ISSN*, pp. 0033-2097, 2011.

[4] A. Crespo, A. Alonso, M. Marcos, A. Juan, and P. Balbastre, "Mixed criticality in control systems," *IFAC Proceedings Volumes,* vol. 47, pp. 12261-12271, 2014.

[5] G. Buttazzo and A. Cervin, "Comparative assessment and evaluation of jitter control methods," in *Proceedings of the 15th conference on Real-Time and Network Systems*, 2007, pp. 163-172.

[6] M. Naghibzadeh, P. Neamatollahi, R. Ramezani, A. Rezaeian, and T. Dehghani, "Efficient semi-partitioning and rate-monotonic scheduling hard real-time tasks on multi-core systems," in *Industrial Embedded Systems (SIES), 2013 8th IEEE International Symposium on*, 2013, pp. 85-88.

[7] M. Kargahi and A. Movaghar, "A method for performance analysis of earliest-deadline-first scheduling policy," *The Journal of Supercomputing,* vol. 37, pp. 197-222, 2006.

[8] M. V. Micea, V.-I. Cretu, and V. Groza, "Maximum predictability in signal interactions with HARETICK kernel," *IEEE Transactions on Instrumentation and Measurement,* vol. 55, pp. 1317-1330, 2006.

[9] H. Aydin and Q. Yang, "Energy-aware partitioning for multiprocessor real-time systems," in *Parallel and Distributed Processing Symposium, 2003. Proceedings. International*, 2003, p. 9 pp.

[10] G. Gracioli, A. A. Fröhlich, R. Pellizzoni, and S. Fischmeister, "Implementation and evaluation of global and partitioned scheduling in a real-time OS," *Real-Time Systems,* vol. 49, pp. 669-714, 2013.

[11] D. Simchi-Levi, "New worst-case results for the bin-packing problem," *Naval Research Logistics,* vol. 41, p. 579, 1994.

[12] T. C. Ferreto, M. A. Netto, R. N. Calheiros, and C. A. De Rose, "Server consolidation with migration control for virtualized data centers," *Future Generation Computer Systems,* vol. 27, pp. 1027-1034, 2011.

[13] A. Lodi, S. Martello, and D. Vigo, "Recent advances on two-dimensional bin packing problems," *Discrete Applied Mathematics,* vol. 123, pp. 379-396, 2002.

[14] G. C. Buttazzo, "Rate monotonic vs. EDF: judgment day," *Real-Time Systems,* vol. 29, pp. 5-26, 2005.

[15] S. Baruah, G. Buttazzo, S. Gorinsky, and G. Lipari, "Scheduling periodic task systems to minimize output jitter," in *Real-Time*

Computing Systems and Applications, 1999. RTCSA'99. Sixth International Conference on, 1999, pp. 62-69.

[16] M. S. Mollison, J. P. Erickson, J. H. Anderson, S. K. Baruah, and J. A. Scoredos, "Mixed-criticality real-time scheduling for multicore systems," in *Computer and Information Technology (CIT), 2010 IEEE 10th International Conference on*, 2010, pp. 1864-1871.

[17] B. Chattopadhyay and S. Baruah, "A lookup-table driven approach to partitioned scheduling," in *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2011 17th IEEE*, 2011, pp. 257-265.

[18] M. V. Micea, "HARETICK: A real-time compact kernel for critical applications on embedded platforms," in *Proc. 7th Intl. Conf. Development and Applic. Syst., DAS*, 2004, pp. 16-23.

[19] M. V. Micea, C.-S. Stangaciu, V. Stangaciu, and D.-I. Curiac, "Novel Hybrid Scheduling Technique for Sensor Nodes with Mixed Criticality Tasks," *Sensors,* vol. 17, p. 1504, 2017.

[20] M. V. Micea, C. S. Stangaciu and V. I. Cretu, "Analysis of Non-Preemptive Scheduling Techniques for HRT Systems," *Buletinul Stiintific al Universitatii Politehnica Timisoara - Transactions on Electronics and Communications*, Volume 60 (74), Issue 1, 2015, p 9-14.

[21] M. Chéramy, P.-E. Hladik, and A.-M. Déplanche, "SimSo: A simulation tool to evaluate real-time multiprocessor scheduling algorithms," in *5th International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)*, 2014, p. 6 p.

[22] S. Kato and N. Yamasaki, "Portioned EDF-based scheduling on multiprocessors," in *Proceedings of the 8th ACM international conference on Embedded software*, 2008, pp. 139-148.

[23] R. I. Davis and A. Burns, "Priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems," in *Real-Time Systems Symposium, 2009, RTSS 2009. 30th IEEE*, 2009, pp. 398-409.

[24] P. Emberson, R. Stafford, and R. I. Davis, "Techniques for the synthesis of multiprocessor tasksets," in *proceedings 1st International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS 2010)*, 2010, pp. 6-11.