Heuristic partitioning of real-time tasks on multi-processors

Agostino Mascitti, Tommaso Cucinotta, Luca Abeni
Scuola Superiore Sant'Anna, Pisa, Italy
Email: {name.surname}@santannapisa.it

***Abstract*—This paper tackles the problem of admitting real-time tasks onto a symmetric multi-processor platform, where a partitioned EDF-based scheduler is used. We propose to combine a well-known utilization-based test for the first-fit partitioning strategy, with a simple heuristic based on the number of tasks and exact knowledge of the utilization of the first few biggest tasks. This results in an effective and efficient test improving on the state of the art in terms of admitted tasks, as shown by extensive tests performed on task sets generated using the widely adopted `randfixedsum` algorithm.**

I. Introduction

Since the seminal work by Liu and Layland [1] on optimality of EDF in scheduling implicit-deadline, independent and periodic real-time tasks on single-processor systems, researchers have been struggling in looking for equally simple admission tests for multi-processor platforms.

While some optimal scheduling algorithms have been developed for multi-processors too [2]–[5], these algorithms are generally considered complex and/or not too efficient (causing large numbers of tasks migration) and are not used in practice.

Instead, current OS kernels focus on using some variations of the scheduling algorithms used for single-processor systems (mainly fixed priority scheduling and EDF). Algorithms such as EDF can be applied to multi-core/multi-CPU scheduling using the *global scheduling approach* (leading, for example, to G-EDF) or the *partitioned scheduling approach* (leading, for example, to P-EDF).

P-EDF is not work-conserving, making sometimes ineffective use of the available processing power. On the other hand, G-EDF is work-conserving so it would seem to make a better use of the underlying physical resources than P-EDF. However, utilization-based analysis for G-EDF is not useful, as the schedulability bound is 1 independently from the number of processors/cores [6]. Also other kinds of schedulability analysis for G-EDF are not simple and quite pessimistic. On the other hand, P-EDF statically assigns tasks to cores and denies migrations. Thus, analysis is easier since the system designer can apply well-consolidated results for uni-processors, at the cost of less robustness to temporary overheads and the need for offline partitioning.

A. Contributions

In this paper, a simple admission strategy is proposed to admit, on a multi-processor platform under P-EDF scheduling, independent periodic/sporadic real-time tasks with known WCETs and minimum inter-arrival times. The proposed test is based on combining a well-known utilization-based test [7] valid under a first-fit allocation strategy across the available cores, with a heuristic based on peeking at the utilizations of the heaviest few k tasks in the set, bounding the utilization of the remaining tasks with the one of the k^{th} heaviest task.

The proposed test is evaluated against the utilization-based test alone over task sets randomly generated using the well-known `randfixedsum` algorithm [8]. The evaluation focuses on the capability to admit additional tasks into the system, as well as the additional overheads due to the computational complexity needed for the new test. We show that the technique can be proposed as an on-line admission test for dynamic real-time systems where real-time tasks can enter and leave at any time. Our approach does not require to know the whole taskset from the beginning and it allows for incrementally partitioning the taskset.

II. Related Work

The problem of real-time tasks partitioning is implicitly based on the bin-packing problem, where items must be allocated into bins such that the final bins weights are less than 1 and, for example, the number of used bins is minimized. The bin-packing problem is known to be NP-hard in the strong sense, thus making other related interesting problems, like partitioning tasks to achieve the minimum energy consumption, intractable in polynomial time as well. Different bin-packing heuristics (designed to work around the problem complexity) have been investigated in previous works and some interesting conclusions are reported in [9]–[13]. For example, if items are sorted in non-increasing order, then you get closer to the optimum bin-packing. The reader can refer to [14] for a survey.

Efficient heuristics to tackle the bin-packing problem have also been investigated [15]–[20] to cope with tasks arriving and departing dynamically in an on-line fashion. In this case, repacking of previously packed items can improve the algorithm performance.

The problem of partitioning real-time tasks on multiple CPUs/cores has also been studied in the case of uniform multiprocessors [21], [22], where processors may have different speeds.

In order to allow for a better exploitation of the residual capacity in partitioned scheduling approaches for multi-core platforms, semi-partitioned techniques have been proposed [23]. These allow for splitting a real-time task into two pieces that are allocated on two different cores. This way, if the

task cannot complete on the first core, it can continue on the second one under certain conditions. These techniques make schedulability analysis and admission tests particularly more involved than simple utilization-based tests, and require more advanced scheduling features with controlled migrations.

Finally, some previous works consider the partitioning of real-time tasks on heterogeneous multicores. For example, in [24] tasks are allocated on ARM big.LITTLE platforms, also trying to adaptively minimize the energy consumption under a dynamically partitioned EDF scheme. Other works such as [25], [26] use ILP formulations to partition independent constrained-deadline sporadic tasks upon heterogeneous multiprocessor platforms. Finally, [27] extends the study to tasks with shared resources.

In this is paper, we propose an improved admission test for partitioned EDF systems, starting from the approach appeared in [7]. The latter performs a test based on the overall taskset *utilization*, as well as the utilization of the heaviest task. We combine it with an approach based on the *number of tasks* in the taskset, using not only the highest utilization in the taskset, but also the subsequent highest few ones, resulting in the capability to save a number of partitionable tasksets that the original test would not admit.

III. Proposed Approach

Consider a set $\Gamma = \{\tau_i\}_{i=1,\ldots,n}$ of n tasks with utilizations $\{U_i\}$ to be scheduled under P-EDF on m processors. Without loss of generality, assume the tasks are sorted by decreasing utilizations: $U_i \geq U_{i+1}$ for all $i < n$.

The well-known test in [7] admits the tasks when their overall utilization does not exceed the value:

$$\sum_{i=1}^{n} U_i \leq \frac{m * \beta + 1}{\beta + 1} \tag{1}$$

where $\beta \triangleq \left\lfloor \frac{1}{U_1} \right\rfloor$. As evident, the test relies heavily on the maximum utilization U_1 among the tasks to admit, introducing quite some pessimism for tasksets having a single task significantly heavier than the others.

On the other hand, an alternative simple admission test that can be used is the one that constraints the number n of allowed tasks in the system to be lower than a maximum value n^{max} based on the biggest utilization U_1. Considering that each of the m cores fits at most $\lfloor 1/U_1 \rfloor$ tasks, we have:

$$n \leq n^{max} = m \left\lfloor \frac{1}{U_1} \right\rfloor \tag{2}$$

However, for some task sets the value of U_1 might be particularly higher than the one of the other tasks, thus it would result in admitting a particularly low number of tasks.

The core idea proposed in this paper is the one of a simple admission test that works around this pessimism by considering up to a number of k highest utilizations $U_1, \ldots, U_k$, then relying on U_k as a bound for the utilization of the remaining tasks $\{\tau_{k+1}, \ldots, \tau_n\}$ in the set, which, along with τ_k itself, will be referred to as *small* tasks in what follows. As it will be clear in a moment, this results in a particularly simple and efficient test for small k values, $k = 2$ or $k = 3$.

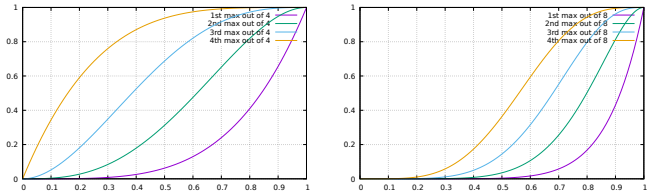


Figure 1: CDF of the k^{th} maximum (various curves) among 4 (left plot) or 8 (right plot) randomly generated utilizations.

To reinforce the motivations behind our proposal, we consider n tasks with uniformly random and i.i.d. utilization distribution within $[0, 1] \subset \mathbb{R}$. Looking at the cumulative distribution function (CDF) of the k^{th} maximum among the utilizations (which is the $(n-k+1)-$order statistic), we have a significant shift of the distribution towards lower values as k increases. This is shown in Figure 1 for $n = 4$ (left plot) and $n = 8$ (right plot), and $k = 1, \ldots, 4$ (various curves). Also, it is evident that in Equation (2) particularly important "bending" points for the number of admitted tasks are those where U_1 falls below $1/2$, $1/3$, $1/4$, etc... In Figure 2 we visualize the probability of the k^{th} maximum (on the X axis) among n randomly drawn utilizations falling below said thresholds (various curves), for $n = 4$ (left plot) and $n = 8$ (right plot). For example, for $n = 8$ tasks, the probability of $U_k \leq 0.5$ goes from a negligible value of ~ 0.0039 for $k = 1$, up to consistent values of ~ 0.14 for $k = 3$ and ~ 0.36 for $k = 4$.

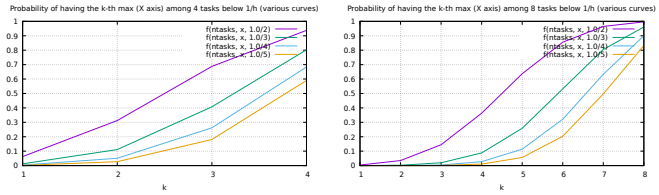


Figure 2: CDF of the k^{th} maximum (various curves) among 4 (left plot) or 8 (right plot) randomly generated utilizations.

For simplicity and without loss of generality, we assume $n \geq m \geq k$ in what follows.

The main idea is that we have to consider all of the possible allocations of the heaviest $k-1$ tasks partitioning them across a subset of $k-1$ processors. Without loss of generality, we can assume these are the first $k-1$ processors among the m ones available on the platform. Whenever each of said processors j hosts a subset Γ_j^H of the first $k-1$ heavy tasks, with overall utilization $U_{\Gamma_j^H} \triangleq \sum_{i \in \Gamma_j^H} U_i \leq 1$ (note Γ_j^H can be an empty set for one or more processors here), its utilization available to host small tasks is $1 - U_{\Gamma_j^H}$, therefore each processor $j < k$ can host a maximum number of small tasks n_j^S equal to:

$$n_j^S = \left\lfloor \frac{1 - \sum_{i \in \Gamma_j^H} U_i}{U_k} \right\rfloor, \quad \text{for } j < k, \tag{3}$$

where the $m-k+1$ remaining processors will exclusively host small tasks, so the maximum number of tasks n_j^S for $j \geq k$ is:

$$n_j^S = \left\lfloor \frac{1}{U_k} \right\rfloor, \text{ for } j \geq k. \tag{4}$$

Therefore, the number of tasks n^{max} that can be admitted is:

$$\begin{aligned} n^{max} &= \sum_{j=1}^{k-1} \left(\left| \Gamma_j^H \right| + n_j^S \right) + \sum_{j=k}^{m} n_j^S \\ &= k - 1 + \sum_{j=1}^{k-1} \left\lfloor \frac{1 - \sum_{i \in \Gamma_j^H} U_i}{U_k} \right\rfloor + (m - k + 1) \left\lfloor \frac{1}{U_k} \right\rfloor \end{aligned} \tag{5}$$

Now, if we know a-priori where the $k-1$ heaviest tasks will be placed, then the sets $\{\Gamma_j^H\}$ above are known, and we have readily available their overall utilizations $U_{\Gamma_j^H} = \sum_{i \in \Gamma_j^H} U_i$.

On the other hand, if we are able to pin the heaviest tasks down to specific cores, then we can tackle the problem of best assignments $\{\Gamma_j^H\}$ so that n^{max} is maximized, however this is not considered here and left as possible future work.

Instead, we are interested in formulating a test that holds regardless of the exact location of the heaviest tasks $\{\Gamma_j^H\}$. So, we want to compute the minimum possible n^{max} as derived accounting for all the possible distributions of the $k-1$ heaviest tasks over $k-1$ processors respecting the single-processor EDF schedulability test $U_{\Gamma_j} \leq 1 \forall j < k$.

Therefore, the test to be applied requires to verify that $n \leq n^{max}$ with n^{max} as from Equation (5), for any possible partitioning of the $k-1$ tasks with highest utilization across $k-1$ processors, where the saturation bound of $U_{\Gamma_j^H} \leq 1 \forall j < k$ is respected.

Once we have the maximum number of admissible tasks n^{max}, we can compute its corresponding admitted maximum utilization:

$$U^{max} = \sum_{i=1}^{k-1} U_i + (n^{max} - k + 1) U_k \tag{6}$$

Note that it may well happen that tasksets with utilization below said threshold are not admitted by our test in Equation (5), if they have too many tasks.

A. Computational complexity

A straightforward implementation of the general test described above requires:

1) the identification of the $k-1$ tasks with the highest utilization, and of the k^{th} highest utilization value U_k;
2) identify all the possible ways for partitioning the identified $k-1$ heaviest tasks across $k-1$ cores $\Gamma^H = \left\{\Gamma_j^H\right\}_{j=1,\dots,k-1}$ (including those in which one or more subsets Γ_j^H are empty);
3) perform the test in Equation (5) for each said task distribution Γ^H.

We focus on how expensive it is to *update* the information in the above three steps when a new task enters the system, and we need to decide whether to admit it or not. We assume n tasks including the new one to be admitted, and $k \geq 2$ for the sake of simplicity.

The first step can be realized efficiently for large values of n and/or k, by adopting a min-heap for the biggest $k-1$ elements and a max-heap for the remaining $n-k+1$ ones. This leads to a logarithmic complexity of $O(\log(k-1) + \log(n-k+1)) \equiv O(\log(k(n-k))$,

The second step requires an $O\left((k-2)^{k-1}\right)$ complexity, because, no matter where τ_1 is placed, for each additional task among the remaining $k-2$ heaviest ones identified in step 1, we have to evaluate its placement on any of the $k-1$ cores under consideration, and discard those placement actions that would exceed the schedulability bound $U_{\Gamma_j^H} \leq 1$.

The third step requires to compute $k-1$ division and floor operations using the $k-1$ different leftover bandwidths that are easily obtained while enumerating the individual subsets $\Gamma_j^H \subseteq \Gamma^H$. This test needs to be repeated for each of the possible partitions Γ^H as enumerated in step 2.

Therefore, considering also the operations needed for the first step, the final overall complexity turns out to be:

$$O\left(\log(k(n-k)) + (k-1)(k-2)^{k-1}\right) \equiv O\left(\log(k(n-k)) + k^k\right). \tag{7}$$

Note that looking at whether the new task lands within the min-heap or the max-heap, some of the steps above may be simplified and/or optimized away. These details are omitted for the sake of brevity.

B. Practical examples for small values of k

Despite the scaring appearence of Equation (7), for small values of k the test described above can easily and practically be calculated. For example, for $k = 1$ it degenerates into the easy verification of Equation (2). However, it is more interesting to explicitly formulate it for higher values of k.

For $k = 2$, we have:

$$n^{max} = 1 + \left\lfloor \frac{1 - U_1}{U_2} \right\rfloor + (m-1) \left\lfloor \frac{1}{U_2} \right\rfloor. \tag{8}$$

For $k = 3$, we need to distinguish two cases. If $U_1 + U_2 \geq 1$, then the two heaviest tasks can only be placed on distinct cores, and the test simplifies in:

$$n^{max} = 2 + \left\lfloor \frac{1 - U_1}{U_3} \right\rfloor + \left\lfloor \frac{1 - U_2}{U_3} \right\rfloor + \left\lfloor \frac{1}{U_3} \right\rfloor (m-2). \tag{9}$$

On the other hand, if $U_1 + U_2 < 1$, then the two heaviest tasks can be found either on the same CPU or on distinct CPUs, so we need to take the worst-case among the two and add a minimum operation:

$$\begin{aligned} n^{max} = \min &\left\{ \left\lfloor \frac{1 - U_1 - U_2}{U_3} \right\rfloor + \left\lfloor \frac{1}{U_3} \right\rfloor, \left\lfloor \frac{1 - U_1}{U_3} \right\rfloor + \left\lfloor \frac{1 - U_2}{U_3} \right\rfloor \right\} \\ &+ 2 + \left\lfloor \frac{1}{U_3} \right\rfloor (m-2). \end{aligned} \tag{10}$$

C. Non-combinatorial bound

As mentioned, the above bound n^{max} on the number of admissible tasks requires a combinatorial evaluation of all the possible placements of the heaviest $k-1$ tasks. This becomes easy to perform only for very small values of k.

However, for higher values of k (which is useful with a higher number n of tasks to be admitted) we can provide a conservative bound for n^{max} that does not suffer of the combinatorial complexity for its computation, and it is still useful to admit additional tasks into the system.

Exploiting the fact that:

$$\forall x, y \in \mathbb{R},\ \lfloor x \rfloor + \lfloor y \rfloor \geq \lfloor x + y \rfloor - 1, \tag{11}$$

n^{max} as from Equation (5) above can be bounded as:

$$n^{max} \geq 1 + \left\lfloor \frac{k - 1 - \sum_{i=1}^{k-1} U_i}{U_k} \right\rfloor + (m - k + 1) \left\lfloor \frac{1}{U_k} \right\rfloor \tag{12}$$

Indeed, the n^{max} formulation in Equation (5) contains the summation of the result of $k-1$ ceil operations (one for each of the $k-1$ first cores possibly hosting heavy tasks), which can be bounded applying the approximation in Equation (11) exactly $k-2$ times, resulting in the first term of $(k-1)$ in Equation (5) being reduced to just 1. Note that this bound is correct for any possible partitioning $\{\Gamma_j^H\}$ of the $k-1$ heaviest tasks across the first $k-1$ cores, and regardless of which subsets of said tasks actually fit onto a single core. Indeed, the final bound for n^{max} turns out to be identical in all these cases, and equal to the one in Equation (12).

Therefore, the approximated test in Equation (12) has generally linear complexity in the number of tasks. However, optimized implementations are possible for repeating the test incrementally whenever a new task joins the taskset. For example, using a two-heaps implementation, as detailed in step 1 in section III-A, leads to a final logarithmic complexity of:

$$O(\log k(n-k)). \tag{13}$$

IV. Evaluation

This section evaluates the combinatorial admission tests given by Equation (5) for the cases $k = \{2, 3, 4\}$, along with their non-combinatorial approximated variants in Equation (12), comparing with the test in Equation (1) (referred to as *utilization-based test* in what follows). As it will become clear, the joint use of the test in Equation (1) and some of the proposed one(s) results in a viable, efficient and useful admission test for P-EDF that admits more tasksets than Equation (5) alone would do.

Tasksets have been generated with the `taskgen.py` program[1] by Emberson et al. [8] and the number of CPUs is fixed to $m = 4$ or $m = 8$. The admission tests have been repeated on a variety of tasksets with different cardinality and overall utilization. For each overall utilization among the values in $\{1.5, 1.6, ..., 2.9, 3.0\}$ for $m = 4$ cores, and those in $\{4.0, 4.1, ..., 5.0\}$ for $m = 8$ cores, 100 random tasksets have been generated and the admitted tasksets have been counted for each configuration. Each taskset contains a number of tasks varying in the range $\{6, 7, \ldots, 17, 18\}$ for $m = 4$ cores, and belonging to the set $\{9, 12, 15, 18, 21, 24\}$ for $m = 8$ cores. Results are shown in Figure 3 for the two cases of $m = 4$ and Figure 4 for $m = 8$ cores.

[1]The tool is available at: http://retis.sssup.it/waters2010/tools.php

A. Motivational example

Consider the taskset of 6 tasks in Table I generated with the taskgen program by Emberson et al. and having total utilization 2.6. It is among the ones used to make the plot in Figure 3a for 4 cores ($m = 4$). The tasks are already sorted for convenience.

Table I: Taskset for the motivational example

U_1	U_2	U_3	U_4	U_5	U_6	$\sum_i U_i$
0.9237	0.5331	0.3762	0.2627	0.2528	0.2514	2.6

Equation (1), which is a utilization-based test, would not admit the taskset because it only considers the first maximum

$$\begin{aligned} \beta &= \left\lfloor \frac{1}{U_1} \right\rfloor = 1 \\ 2.6 &> \frac{4 * 1 + 1}{1 + 1} = 2.5 \end{aligned} \tag{14}$$

while Equation (5) and Equation (12) would admit it for $k = 3$, for example. Notice that, differently from Equation (1), these equations are based on the number of tasks in the taskset.

Equation (5) for k = 2 would behave as Equation (8). It would not admit the taskset since it does not meet the schedulability check, being n^{max} smaller than the number of tasks in the taskset:

$$6 > 4 = 1 + \left\lfloor \frac{1 - 0.9237}{0.5331} \right\rfloor + (4 - 1) \left\lfloor \frac{1}{0.5331} \right\rfloor \tag{15}$$

Equation (5) for k = 3 would behave as Equation (9). It would admit the taskset, since the schedulability test holds:

$$6 \leq 7 = 2 + \left\lfloor \frac{1 - 0.9237}{0.3762} \right\rfloor + \left\lfloor \frac{1 - 0.5331}{0.3762} \right\rfloor + (4 - 2) \left\lfloor \frac{1}{0.3762} \right\rfloor \tag{16}$$

Equation (12) for k = 2 would not admit the taskset too:

$$6 > 4 = 1 + \left\lfloor \frac{2 - 1 - 0.9237}{0.5331} \right\rfloor + (4 - 2 + 1) \left\lfloor \frac{1}{0.5331} \right\rfloor \tag{17}$$

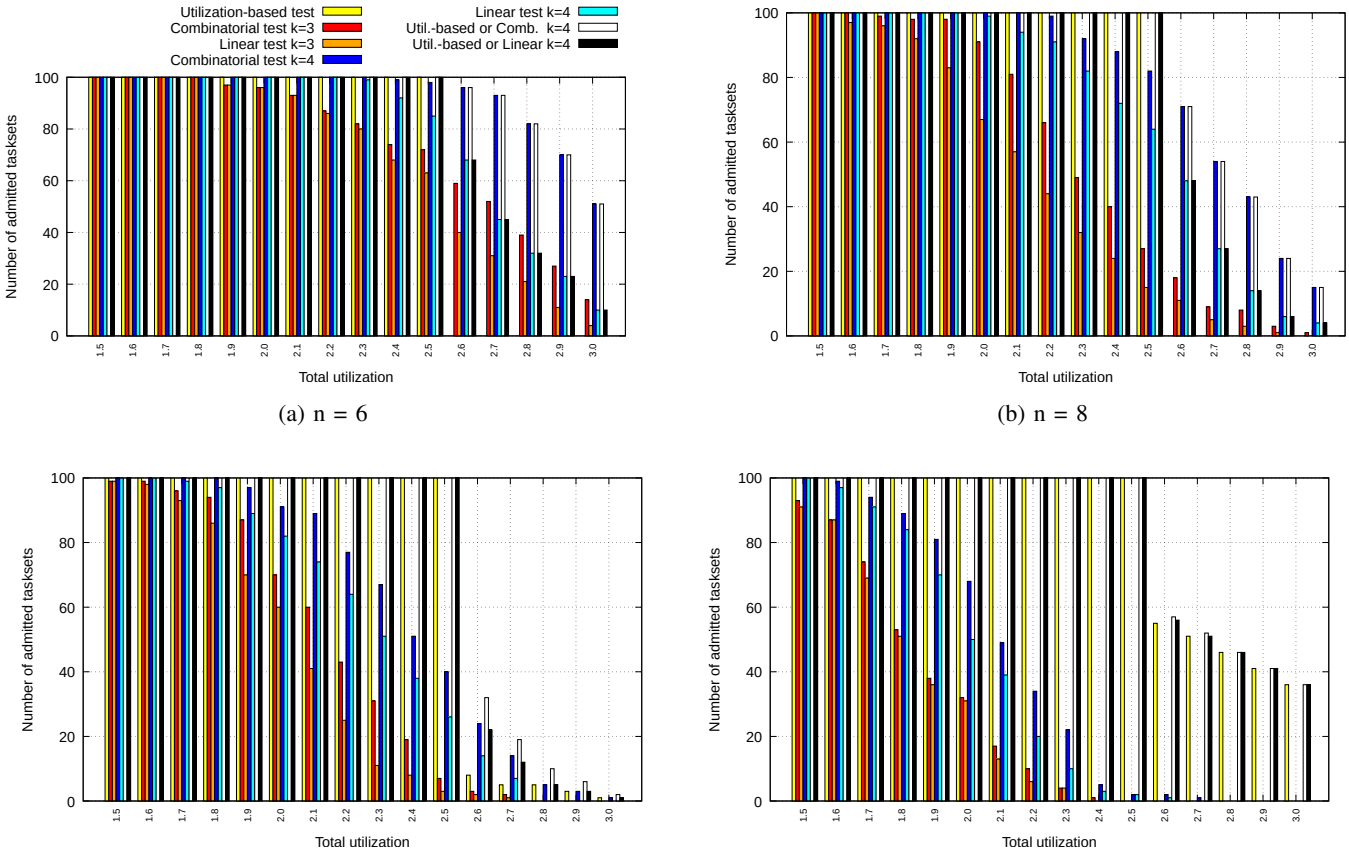
Equation (12) for k = 3 would admit the taskset:

$$6 \leq 6 = 1 + \left\lfloor \frac{3 - 1 - (0.9237 + 0.5331)}{0.3762} \right\rfloor + (4 - 3 + 1) \left\lfloor \frac{1}{0.3762} \right\rfloor \tag{18}$$

Finally, Equation (12) would admit the taskset for $k = 4$ as well:

$$6 \leq 8 = 1 + \left\lfloor \frac{4 - 1 - (0.9237 + 0.5331 + 0.3762)}{0.2627} \right\rfloor + (4 - 4 + 1) \left\lfloor \frac{1}{0.2627} \right\rfloor \tag{19}$$

In fact, Equation (5) has combinatorial complexity and it tries to partition the first $k-1$ heaviest tasks in all the



(a) n = 6

(b) n = 8

(c) n = 11

(d) n = 18

Figure 3: Comparison of equations behavior for $m = 4$ and varying the number of tasks in each taskset

possible ways over the first $k-1$ cores as shown in Figure 5. Therefore, it constitutes a viable test only for small values of k. On the other hand, the test in Equation (12) has a reduced computational complexity, making it more suitable for real-world admission tests if higher values of k are needed. Figure 5 shows the possible partitioning configurations for $k = 3$ and $k = 4$ that must be checked to determine whether the taskset is admissible or not.

B. Experimental comparison of the admission tests

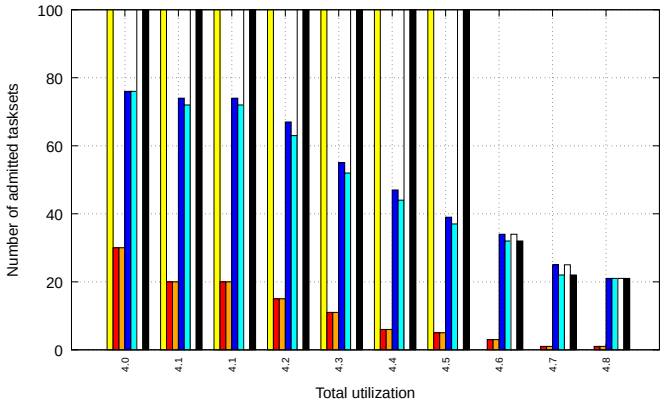
This section evaluates the mentioned admission tests for P-EDF with the settings described in section IV. In Figure 3 and Figure 4, *Utilization-based test* refers to Equation (1), *Combinatorial test* refers to Equation (5) and *Linear test* refers to Equation (12). *Util.-based or Comb. k = 4* is the *logical or* between Equation (5) and Equation (1) for each taskset, while *Util.-based or Linear k = 4* is the *logical or* between Equation (12) and Equation (1) for each taskset.

While the utilization-based test generally achieves higher rates of accepted tasks for lower system utilizations, the new ones proposed in this paper tend to save an interesting number of tasksets when the total utilization increases, particularly for smaller numbers of tasks in tasksets.

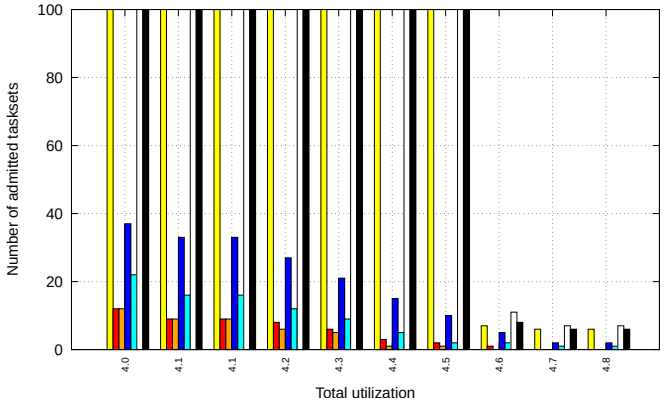
As shown in Figure 3a and Figure 3b, the tests based on the number of tasks in the taskset tend to be stricter and admit less tasksets. However, there are cases in which they admit tasksets that the utilization-based test would not admit (e.g., Figure 3a). This results in an interesting increase in the total number of admitted tasksets, when the tests are used in a combined way. These cases are highlighted by the *or* bins in the histograms, counting how many tasksets satisfy either the original utilization-based test in [7], or the new ones from Section III. In all the experiments using the exact combinatorial tests, increasing k (i.e., reasoning on more utilizations) also enhances the capability of the paper equations, and in particular $k = 4$ achieves the best results.

The combinatorial test produces higher schedulability ratios than the linear one for all k, at the cost of higher computational complexity. Increasing k achieves better results too, since more utilizations are taken into account for the checks.

Analyzing the graphs in Figure 3 when the number of tasks in each taskset increases (i.e., n increases), all admission checks show generally better performance with lower system utilizations. Moreover, the effectiveness of the paper equations gets poorer when increasing n and for $n = 18$ the utilization-based test dominates. The effectiveness of the paper equations with smaller k tends to decrease quite fast. The same trend



(a) n = 12



(b) n = 21

Figure 4: Comparison of equations behavior for $m = 8$ and varying the number of tasks in each taskset

is in Figure 4 for the case of 8 cores. For lower system utilizations, Equation (1) is the only one admitting all tasksets. However, for lower tasks number ($n = 12$), the number of admitted tasksets with Equation (1) falls down and the paper equations take over. The same does not hold for higher number of tasks ($n = 21$), where the utilization-based test always dominates. Generally speaking, the proposed tests perform better and dominate for lower number of tasks in the tasksets. In these cases Equation (1) collapses. The contrary applies for higher number of tasks in tasksets. Therefore, the main outcome is that using both Equation (1) and the proposed tests makes the admission check more robust, as highlighted with the *or* cases in the graphs.

Finally, note that the conclusions drawn from the comparison performed in this section refer to tasksets randomly generated with the *taskgen.py* program [8]. However, things might be different for alternative distributions of the task utilizations. For example, when the utilizations are better balanced among the tasks and when the first few highest utilizations are not too different from the ones of the very next task, the proposed technique would not result advantageous. A more realistic comparison should consider benchmarks or tasksets used in certain industrial domains. Our choice of

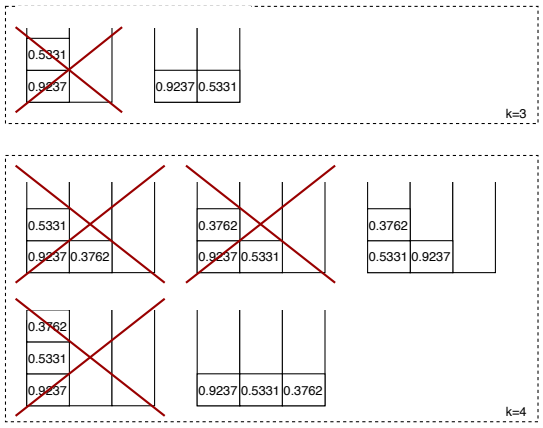


Figure 5: Some of the possible partitioning configurations that Equation (5) must check, for $k = 3$ (top) and $k = 4$ (bottom). Configurations that are impossible due to violation of EDF schedulability are marked with a red cross.

using the taskset generator by Emberson et al. has been dictated by its popularity and wide acceptance in the real-time community.

V. Conclusions and Future Work

In this paper, the problem of admitting real-time tasks onto a SMP platform under partitioned EDF scheduling has been tackled. We proposed an easy-to-compute test that combines the well-known utilization based test for first-fit allocation heuristics as studied in [7] with a test based on looking at the k biggest utilizations.

An extensive validation performed on randomly generated tasksets shows that our proposed test is able to admit onto the system a number of tasksets that would be discarded by the original test in [7] due to its pessimistic behavior.

Concerning possible future works on the topic, we plan to extend the technique for uniform and/or heterogeneous multi-processor systems, where processors may have different computational capabilities, and to make the technique also usable on SMP platforms with DVFS energy-saving features.

References

[1] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *J. ACM*, vol. 20, no. 1, p. 46–61, Jan. 1973. [Online]. Available: https://doi.org/10.1145/321738.321743

[2] S. K. Baruah, N. K. Cohen, C. G. Plaxton, and D. A. Varvel, "Proportionate progress: A notion of fairness in resource allocation," *Algorithmica*, vol. 15, no. 6, pp. 600–625, Jun 1996. [Online]. Available: https://doi.org/10.1007/BF01940883

[3] G. Levin, S. Funk, C. Sadowski, I. Pye, and S. Brandt, "Dp-fair: A simple model for understanding optimal multiprocessor scheduling," in *Proceesings of the 22nd Euromicro Conference on Real-Time Systems (ECRTS 2010)*. Brussels, Belgium: IEEE, July 2010, pp. 3–13.

[4] S. Funk, "Lre-tl: an optimal multiprocessor algorithm for sporadic task sets with unconstrained deadlines," *Real-Time Systems*, vol. 46, no. 3, pp. 332–359, Dec 2010. [Online]. Available: https://doi.org/10.1007/s11241-010-9109-2

[5] P. Regnier, G. Lima, E. Massa, G. Levin, and S. Brandt, "Run: Optimal multiprocessor real-time scheduling via reduction to uniprocessor," in *Proceesings of the 32nd IEEE Real-Time Systems Symposium (RTSS 2011)*. Vienna, Austria: IEEE, Nov 2011, pp. 104–115.

[6] S. K. Dhall and C. L. Liu, "On a real-time scheduling problem," *Operations research*, vol. 26, no. 1, pp. 127–140, 1978.

[7] J. M. López, M. García, J. L. Diaz, and D. F. Garcia, "Worst-case utilization bound for EDF scheduling on real-time multiprocessor systems," in *Proceedings of the 12th Euromicro Conference on Real-Time Systems (ECRTS 2000)*. Stockholm, Sweden: IEEE, June 2000, pp. 25–33.

[8] 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)*, Brussels, Belgium, 2010, pp. 6–11.

[9] D. Simchi-Levi, "New worst-case results for the bin-packing problem," *Naval Research Logistics (NRL)*, vol. 41, no. 4, pp. 579–585, 1994.

[10] D. S. Johnson, "Approximation algorithms for combinatorial problems," *Journal of Computer and System Sciences*, vol. 9, no. 3, pp. 256 – 278, 1974.

[11] D. Johnson, A. Demers, J. Ullman, M. Garey, and R. Graham, "Worst-case performance bounds for simple one-dimensional packing algorithms," *SIAM Journal on Computing*, vol. 3, no. 4, pp. 299–325, 1974.

[12] J. Boyar, G. Dósa, and L. Epstein, "On the absolute approximation ratio for first fit and related results," *Discrete Applied Mathematics*, vol. 160, no. 13, pp. 1914 – 1923, 2012.

[13] R. L. Graham, "Bounds on multiprocessing anomalies and related packing algorithms," in *Proceedings of the May 16-18, 1972, Spring Joint Computer Conference*, ser. AFIPS '72 (Spring). New York, NY, USA: ACM, 1972, pp. 205–217.

[14] M. R. Garey and D. S. Johnson, *Approximation Algorithms for Bin Packing Problems: A Survey*. Vienna: Springer Vienna, 1981, pp. 147–172.

[15] G. Gambosi, A. Postiglione, and M. Talamo, "Algorithms for the relaxed online bin-packing model," *SIAM journal on computing*, vol. 30, no. 5, pp. 1532–1551, 2000.

[16] Z. Ivković and E. L. Lloyd, "A fundamental restriction on fully dynamic maintenance of bin packing," *Information Processing Letters*, vol. 59, no. 4, pp. 229–232, 1996.

[17] J. Balogh, J. Békési, G. Galambos, and G. Reinelt, "Lower bound for the online bin packing problem with restricted repacking," *SIAM J. Comput.*, vol. 38, pp. 398–410, 01 2008.

[18] S. Berndt, K. Jansen, and K.-M. Klein, "Fully dynamic bin packing revisited," *Mathematical Programming*, pp. 1–47, 2018.

[19] J. Balogh, J. Békési, G. Galambos, and G. Reinelt, "On-line bin packing with restricted repacking," *Journal of Combinatorial Optimization*, vol. 27, no. 1, pp. 115–131, 2014.

[20] J. Balogh, J. Békési, G. Dósa, L. Epstein, and A. Levin, "A new and improved algorithm for online bin packing," *arXiv preprint arXiv:1707.01728*, 2017.

[21] S. Funk and S. Baruah, "Task assignment on uniform heterogeneous multiprocessors," in *17th Euromicro Conference on Real-Time Systems (ECRTS'05)*. IEEE, 2005, pp. 219–226.

[22] B. Andersson and E. Tovar, "Competitive analysis of partitioned scheduling on uniform multiprocessors," in *2007 IEEE International Parallel and Distributed Processing Symposium*. IEEE, 2007, pp. 1–8.

[23] D. Casini, A. Biondi, and G. Buttazzo, "Semi-partitioned scheduling of dynamic real-time workload: A practical approach based on analysis-driven load balancing," in *29th Euromicro Conference on Real-Time Systems (ECRTS 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

[24] A. Mascitti, T. Cucinotta, and M. Marinoni, "An adaptive, utilization-based approach to schedule real-time tasks for arm big.little architectures," in *EWiLi*, 2019.

[25] S. K. Baruah, V. Bonifaci, R. Bruni, and A. Marchetti-Spaccamela, "Ilp-based approaches to partitioning recurrent workloads upon heterogeneous multiprocessors," in *2016 28th Euromicro Conference on Real-Time Systems (ECRTS)*. IEEE, 2016, pp. 215–225.

[26] ——, "Ilp models for the allocation of recurrent workloads upon heterogeneous multiprocessors," *Journal of Scheduling*, vol. 22, no. 2, pp. 195–209, 2019.

[27] A. Wieder and B. B. Brandenburg, "Efficient partitioning of sporadic real-time tasks with shared resources and spin locks," in *2013 8th IEEE International Symposium on Industrial Embedded Systems (SIES)*. Porto, Portugal: IEEE, June 2013, pp. 49–58.