

# Improved Heuristics for Partitioned Multiprocessor Scheduling Based on Rate-Monotonic Small-Tasks

Dirk Müller and Matthias Werner  
Operating Systems Group  
Chemnitz University of Technology  
Chemnitz, Germany  
{dirkm|mwerner}@cs.tu-chemnitz.de

**Abstract**—Rate-Monotonic Small-Tasks from 1995 is a well-known algorithm for partitioned rate-monotonic scheduling on a multiprocessor system. While the basic idea of local harmonization of the task set in a sorted list to improve success chances of an allocation strategy is of great value, several improvements/alternatives are possible. A systematic terminology is introduced. The article includes both theoretical reasoning for the suggested improvements and experiments on synthetic task sets. A motivating example explains the four suggested RMST modifications. In contrast to other approaches improving on the approximation ratio which expresses the worst-case behavior, we want to improve on the average case behavior when minimizing the number of processors required for a given task set.

**Keywords**-multicore; partitioned; scheduling; rate-monotonic; RMST; simply periodic;

## I. INTRODUCTION

### A. Background

We are witnessing a revolution in computer hardware, the shift from uniprocessor to multiprocessor and multicore systems. Due to thermal dissipation problems it is no longer suitable to increase clock rates of uniprocessors. Cooling becomes very hard to manage, and power consumption increases beyond a reasonable level which is a particular problem for the more and more important market of mobile computers with their limited accumulator capacities. While the watershed for desktop and laptop computers was in 2005/06, a turning point from uniprocessor to multiprocessor technology for embedded systems is expected for the upcoming decade.

Unfortunately, scheduling on multiprocessors is much harder than on uniprocessors. Due to the simple fact that there are always sequential code parts which cannot be scheduled in parallel, the risk of obtaining undesired idle processor time is much higher than on uniprocessors. This is especially true for different order-of-magnitude execution times, cf. Dhall's effect [1], or *greedy* uniprocessors algorithms like rate-monotonic scheduling (RMS), Earliest Deadline First (EDF) (both in [2]) or Least Laxity First (LLF) [3] imitated on multiprocessors [4].

The scheduling of  $n$  tasks onto  $m$  processors can be broadly classified into *partitioned* and *global* algorithms.

Partitioned scheduling relies on the mathematical principle of *reduction to the known*, uniprocessor scheduling. Contrary to that, global approaches attempt to establish a *holistic* view on the system. The theoretical utilization bound for global algorithms with intra-job migration is  $m$  [5], and, thus, much higher than that for partitioned approaches which equals to only  $(m + 1)/2$  [6].

But the theoretical superiority of global scheduling is contrasted by the following points: First, in practice, costs for migration among processors and preemption of jobs have to be considered additionally. Although migration costs are less critical on multicores due to the tighter integration, they are still existing and not always negligible. A recent study [7] suggests even advantages of partitioned approaches over global ones when viewed in total.

Next, the  $(m+1)/2$  bound of partitioned scheduling refers to a worst-case situation with  $m + 1$  tasks of utilization  $1/2 + \epsilon, \epsilon > 0$  with  $\epsilon \rightarrow 0$  which can be regarded as a pathological setup.

Third, separation aspects should be taken into account. By partitioning, it is possible to cluster tasks under comprehension of their criticalities. Taking an example from the automotive industry, it is not recommended to assign anti-lock braking system (ABS) and air conditioning (AC) tasks to one and the same processor since a faulty AC task compromising ABS would be a disaster in terms of safety.

These three facts blur the at first clear advantage of global scheduling. Thus, in spite of the progress with global scheduling since 2000, partitioned scheduling will continue being important in the future. Note the comparable debate on RMS and EDF on uniprocessors. RMS has lower utilization bounds but is easier to implement, includes a smaller overhead and, thus, is still playing a dominant role in practice, in spite of clarified legends and misconceptions in [8].

There are also *hybrid* scheduling approaches taking the best from both worlds like semi-partitioned and clustering ones [5]. On the other hand, these approaches need some time to mature. For a well-written survey on real-time scheduling on multiprocessors, see, e.g., [5].

There was a tendency to oversimplify the partitioned scheduling problem. Too often it has been reduced to *Bin*

$i$	1	2	3	4	5	6	7	8	9	10
$p_i$	7	16	21	32	48	64	66	75	96	100
$e_i$	2	9	3	10	20	20	16	20	10	11
$u_i$	0.286	0.562	0.143	0.312	0.417	0.312	0.242	0.267	0.104	0.110

Table I  
EXAMPLE TASK SET WITH CALCULATED UTILIZATIONS (ROUNDED)

*Packing* with the approved *First Fit* heuristic and a sufficient uniprocessor schedulability criterion. But, especially for partitioned RMS, the problem is much more complicated. The reason is that the remaining size of each bin heavily depends on its content (*dynamic* variable-size Bin Packing), i.e., the parameters<sup>1</sup> of the tasks already allocated to it, and the parameters of the new task.

Only for the two extreme cases, taking the most pessimistic (but also most general) sufficient test by Liu and Layland [2] for  $n \rightarrow \infty$  with bound  $\ln 2 \approx 0.69$  and for the beneficial case of simply periodic<sup>2</sup> task sets (bound is 1 like for EDF) [9], pure *Bin Packing* is appropriate. The former is too pessimistic, and the latter does not refer to the most general case. Sometimes, simply periodic task sets occur in practice. But often, requirements are implicitly adapted in order to be close to this advantageous case. Thus, raw requirements need to consider a more general case.

Hence, a certain presorting<sup>3</sup> of tasks increasing the local utilization bounds in the sorted list when applying a Bin Packing heuristics is a clever improvement. *Rate-Monotonic Small-Tasks* (RMST) was developed by Burchard *et al.* in 1995 [11]. It uses *period value proximity to powers of two* as sorting criterion. A task's period is close to a power of two iff it is equal to or only slightly above a power of two. Thus, two tasks with a similar such parameter possess a period *ratio* which is almost integer since the deviations from powers of two then almost cancel each other out. Recall that simply periodic task sets have a utilization bound of 1 and that this bound decreases *continuously* with modified period ratios, see, e.g., [12].

This sorting is then the base for a *Next Fit* allocation strategy. RMST makes use of the fact that task sets with periods being close to simply periodic have higher utilization bounds than in the general case.

### B. Contributions

In this paper, we propose improvements and/or alternatives for RMST on the level of the uniprocessor schedulability test, the allocation strategy, the presorting, and the

<sup>1</sup>Note that besides the sizes (execution times), the *periods* have an influence on the remaining capacity.

<sup>2</sup>Another common term for simply periodic is *harmonic*.

<sup>3</sup>Another reasonable presorting is that according to decreasing utilizations. This improves on the number of required bins for First Fit (FF) and was coined First Fit Decreasing Utilizations (FFDU) [10]. But we believe that the harmonization described here is more important. This assumption is backed for sets of exclusively small tasks ( $\forall i : u_i \leq 0.5$ ) in terms of the approximation ratio of  $\frac{1}{1-u_{\max}}$  vs. 2 [11] [10].

base for period similarity estimation. Improving the approximation ratio, i.e., improving the *worst* case compared to the optimum number of processors required for a given task set, has been achieved by Rothvoß with his *RMMatching*,  $\mathcal{O}(n^3)$  and *k-RMM*,  $\mathcal{O}(n^2)$  heuristics (to 1.5), see [13] and [14]. In contrast to that, we focus on the improvement of the *average* case with uniformly distributed utilizations and log-uniformly distributed periods and present a variety of ideas to modify and improve RMST.

### C. Related Work

An article considering the average case improvement [15] uses an equivalent to the Hyperbolic Bound [16] as uniprocessor schedulability test. Their  $\mathcal{O}(n \log n)$  algorithm achieves an approximation ratio of 2.0 [15]. An  $\mathcal{O}(n \log n)$  algorithm called First Fit Matching Periods (FFMP) is given in [17]. There, both analytically and experimentally, the superiority of FFMP to RMGT [11] in terms of lower average *waste* is shown.

### D. Structure

The rest of the article is structured as follows. Section II gives a motivating example including an explicit Subsection II-A on task model and notation and Subsection II-C on the basic ideas of the paper. The original RMST algorithm is revisited in Section III. Section IV gives the suggested improvements and alternatives in more detail in Subsections IV-A to IV-D and closes with a discussion in Subsection IV-E. An evaluation based on experiments on synthetic task sets can be found in Section V. Finally, conclusions will be drawn in Section VI.

## II. MOTIVATING EXAMPLE

### A. Task Model and Notation

Given a synchronous (all phases are zero:  $\forall i : \varphi_i = 0$ ) periodic task set  $\{(p_i, e_i) | i = 1..n\}$  with implicit deadlines ( $\forall i : d_i = p_i$ ) and worst-case execution times (WCETs)  $e_i$ , see Table I, the necessary number of processors ( $m$ ) according to different approximation algorithms for partitioned preemptive RM scheduling shall be calculated and compared to each other and to the optimal value of the *minimization problem*. This problem minimizes the number of necessary processors applying partitioned RMS under a given task set [9]. Due to its close relationship to Bin Packing it is NP-Hard [18].

The total utilization  $u$  is defined as sum of all individual task utilizations  $u_i = \frac{e_i}{p_i}$ , thus  $u = \sum_{i=1}^n u_i$ . The sum of utilizations of all tasks on a processor is coined:  ${}_j u = \sum_{\text{task } i \text{ on CPU } j} u_i$ . The prefix subscript  $j$  denotes the processor index in order to distinguish it from the postfix subscript for the task index.

*Sensitivity* is the ratio between the number of task sets deemed to be schedulable and the number of task sets that are really schedulable. It is the most interesting parameter of a sufficient test.

*Acceleration* is a kind of task set transformations where all parameters but period values are kept. Periods can be shortened individually. Technically, to keep them the same is allowed, too. Thus, a task set can even be accelerated to itself. For task sets with implicit deadlines regarded here, the deadlines have to be adapted to the modified periods according to  $\forall i : d_i = p_i$ .

### B. Example

The total utilization of the task set given in Table I is  $u \approx 2.756$ . A good partitioning which achieves the minimum value of processors of  $m = 3$  is given in Table II. This partitioning can be obtained using a brute force search on all set partitionings, cf. [9].

Note that a simple ceiling of the utilization does not give the minimum number of processors, i.e.,  $m = \lceil u \rceil$  does not hold in general. Further, RM uniprocessor schedulability has to be checked for each processor. The simple condition  ${}_j u \leq 1$  is only a necessary one but not sufficient.

Time Demand Analysis (TDA) [19] is the classical exact test for RM schedulability. The test is sufficient and necessary. Thus, its sensitivity is always 100%. On the other hand, the test's drawback is its pseudo-polynomial complexity rendering it unsuitable for quick testing.

We propose to use *accelerated simply periodic task sets* [9] as a framework for sufficient tests of high sensitivity and medium complexity. In the example, tasks on processor 1

$i$	4	8	9	10	2	5	1	3	6	7
$p_i$	32	75	96	100	16	48	7	21	64	66
$e_i$	10	20	10	11	9	20	2	3	20	16
${}_j$	1				2		3			
${}_j u$	$\approx 0.793$				$\approx 0.979$		$\approx 0.983$			

Table II  
OPTIMALLY PARTITIONED EXAMPLE TASK SET ONTO  $m = 3$   
PROCESSORS WITH PROCESSOR UTILIZATIONS (ROUNDED)

can be accelerated to  $\{(32, 10), (64, 20), (64, 10), (64, 11)\}$  with  ${}_1 u' = 61/64 \leq 1$ . Thus, all tasks on processor 1 meet their deadlines. On processor 2, there is already a simply periodic task set with  ${}_2 u = 47/49 \leq 1$ . Hence, it is schedulable. Third, an accelerated simply periodic task set for processor 3 is  $\{(7, 2), (21, 3), (63, 20), (63, 16)\}$  with  ${}_3 u' = 63/63 \leq 1$ . So, there will be no deadline violations on processor 3. Summing up, a RM partitioning onto only

$m = 3$  processors for the task set in Table I has been given in Table II.

The problem is that all classical scheduling algorithms yield a result of 4 instead of only 3 processors. Reasons are unsuitable uniprocessor scheduling tests including complete ignorance of period parameters (e.g. those based on Liu/Layland [2] or on the Hyperbolic Bound [16]), a bad Bin Packing allocation strategy, a wrongly presumed linear instead of the correct circular similarity measure for the simplicity of periods, and a too restricted base of only 2 for calculating period simplicity. These terms will be explained subsequently. Hence, it is worthwhile to improve RMST in terms of the mentioned shortcomings. But before, we will revisit the original version of RMST in Section III.

### C. Ideas

The classical RMST algorithm has a modular structure with a uniprocessor test and an allocation strategy. Both modules can be improved in terms of minimizing the average number of processors necessary to schedule a task set. While the approximation ratio taking the *maximum* ratio of actual and optimal processor count considers the worst case, an average case with uniformly distributed utilizations and log-uniformly distributed periods taking the *mean* of occurring ratios (*average approximation ratio*) might be of more practical relevance since worst-case situations are often pathological cases.

When examining these two modifications, another two alterations seem to be valuable. The first one is to soften the rigid rule of always starting with task number one when it comes to allocation by considering an offset in the task index.

The new concept of a circular similarity measure shall be explained in more detail. Two tasks with power-of-2 period values, say 16 and 8, constitute always a simply periodic task set with an integer period ratio (here 2) due to a rule of exponentiation  $2^a/2^b = 2^{a-b}$  with  $a, b \in \mathbb{N}$ . Then, using w.l.o.g.  $a \geq b$ , we obtain  $a - b \in \mathbb{N}$  and finally  $2^{a-b} \in \mathbb{N}$ . These two tasks have no deviation from the power-of-2 pattern and, thus, a dissimilarity to powers of 2 from above of zero. Assuming a constant deviation of both tasks from this beneficial pattern, say 24 and 12, where both periods are equally scaled (here by 1.5) yields a simply periodic task set, too, since the scaling exactly cancels out in the ratio. When the periods are scaled non-equally, the task set will be no more simply periodic in the general case. But there is a continuous changeover, see Fig. 2 in [12], when increasing w.l.o.g. the first period. Considering e.g. 17 and 8, schedulability will still be high. This case is well explained by a linear similarity measure. But *decreasing* the first period a little bit, e.g. obtaining periods 15 and 8, results as well in a high utilization bound. This is not well reflected by a linear measure since the first period deviates much from a power of 2 and the second is (deviation 0) a power of

2. Thus, the (absolute) difference is high. So, it is much more appropriate to take the difference of dissimilarities to powers of 2 from above *modulo* the maximum deviation as a measure of period similarity.

An analogous example from everyday life is the calculation of spans of time intersecting midnight. From 23:00 to 1:00, one could calculate the span of time to  $23 - 1 = 22[h]$ . This is obviously wrong. The correct calculation and result is:  $1 - 23 \equiv -22 \equiv 2 \pmod{24} = 2[h]$ . For our purpose, this example suggests to represent the similarities to powers of 2 on a circle as we know it from an analog clock. Another common example is the similarity between the months of a year, cf. page 121 in [20].

This enables for better exploiting clusters of tasks arranged on the circle of period similarity to powers of 2 following the explanations above, finally yielding lower counts of necessary processors. Next, the logarithm base for the circular similarity measure could be altered from 2. It will be shown that the last modification yields no significant gain on the average, but in some constructed cases. Thus, it is worth to be mentioned.

Three out of four suggested modifications of RMST yield a lower (and, thus, better) average approximation ratio. This will be shown by theoretical reasoning and empirical tests on synthetic task sets. A new terminology clearly reflecting the four modules in partitioned rate-monotonic scheduling of the *Burchard* kind is introduced.

### III. RMST REVISITED

Rate-Monotonic Small-Tasks (RMST) is a partitioned RM scheduling algorithm for multiprocessors first presented by Burchard *et al.* in 1995 [11]. The cornerstones of every such algorithm are the uniprocessor schedulability criterion and the allocation algorithm [9]. An exact result (necessary and sufficient) on the minimum number of processors necessary for a given task set can be obtained by a *brute force* method. All partitions of the task set have to be considered. The quantity of possibilities is given by the Stirling number of the second kind [9] which is exponential in  $n$ , the number of tasks. For smaller  $n$  values, it could be suitable to prune the search space in order to make the problem manageable. A prominent such technique is Integer Linear Programming (ILP). Note that because of the non-linear terms (e.g. ceiling operator) in an advanced analysis, these techniques do not work [21]. Again, it has to be stressed, that the problem often is oversimplified, cf. Section I. He *et al.* [21] suggest to use Simulated Annealing and Geometric Programming to overcome these problems. Clearly, the problem is intractable for larger  $n$  values, and *heuristics* are required to manage it.

On the level of the uniprocessor schedulability test, an exact test is Time Demand Analysis (TDA) [19]. With its pseudo-polynomial complexity, it is intractable for larger task subsets, too. Thus, a variety of sufficient tests of polynomial complexity have been suggested, e.g. the Liu/Layland

test [2] and the Hyperbolic Bound [16]. These two tests take only utilizations into account. The central idea of RMST is to consider period values as well, both in the uniprocessor test and in the allocation preprocessing.

RMST relies on a linearized version of the Burchard criterion [11] and the Next Fit (NF) allocation strategy. Besides the well-working Burchard criterion, the ingenious idea of [11] is the global presorting of tasks according to increasing  $S_i$  values in order to improve the success rate when it comes to allocation.  $S_i$  quantifies the dissimilarity to a power of 2 from above and is calculated as the decimal fraction of the binary logarithm of the period (1).

$$S_i := \log_2 p_i - \lfloor \log_2 p_i \rfloor \quad (1)$$

This is a good approach since task sets with periods close to simply periodic have higher utilization bounds. Such tasks are close-by in the sorted list what results in improved odds for allocation on a smaller number of processors using NF.

The Burchard criterion (Bu) [11] is based on the  $S_i$  values, too. The range of these values inside a task set characterizes its degree of simplicity of periods in a certain sense. This range is defined as the  $\beta$  value of the task set (2).

$$\beta := \max_{1 \leq i \leq n} S_i - \min_{1 \leq i \leq n} S_i \quad (2)$$

Then, a task set is RM schedulable on one processor if  $\beta < 1 - 1/n$  and (3) or if  $\beta \geq 1 - 1/n$  and (4).

$${}_j u \leq (n-1)(2^{\beta/(n-1)} - 1) + 2^{1-\beta} - 1 \quad (3)$$

$${}_j u \leq n(2^{1/n} - 1) \quad (4)$$

Eliminating  $n$  by a limiting procedure<sup>4</sup> to infinity and linearization for the first case using the equation of a tangent at argument<sup>5</sup>  $\beta = 0$  gives finally a one-equation sufficient RM schedulability test<sup>6</sup> known as the *simplified Burchard criterion* (sBu) (5), cf. Section [11].

$${}_j u \leq \max\{\ln 2, 1 - \beta \ln 2\} \quad (5)$$

The Burchard criterion (in its unsimplified form) turned out to be better than the Liu/Layland criterion (LL) (4) [2], both theoretical [11] and empirical [9]. It belongs to the best classical linear-complexity uniprocessor RM schedulability tests. Only Hyperbolic Bound (HB) [16] and RBound [22] reach similar sensitivities among  $\mathcal{O}(n)$  tests [9].

### IV. IMPROVEMENTS ON RMST

Although using Bu turned out to be a major improvement in comparison to using LL, there are several additional weak points left. These problems shall be tackled. Straight-forward enhancements are the uniprocessor schedulability test (IV-A) and the allocation strategy (IV-B). Less obvious are the

<sup>4</sup>This is possible for bounding from below since the RHSs of both (3) and (4) are strictly decreasing with respect to  $n$ .

<sup>5</sup>This corresponds to the Maclaurin series of degree 1.

<sup>6</sup>By the two cases distinguished, it is a bilinear approximation.

presorting improvement (IV-C) and the base choice (IV-D) which modify the preprocessing for the allocation step.

#### A. Uniprocessor Schedulability Test: DCT instead of sBu

The bilinear utilization bound (5) is attractive due to its simplicity and only linear complexity. On the other hand, sensitivity is only medium. False negative test results are a problem in terms of unnecessary rejection of tasks.

The use of the unsimplified Burchard criterion (3) (4) is definitely an improvement at low costs since complexity remains linear. On the other hand, individual period and execution time values are still not being exploited in such an analysis.

Thus, the application of the principle of Accelerated simply periodic task sets (ASPTSs) is suggested. This approach differs with classical ones in that it is no longer based on utilization bounds. The ASPTS-based schedulability tests have a non-closed form. The basic idea is to search for accelerated task sets which are RMS-schedulable. Then, due to the sustainability property of RMS [23], the original task set with relaxed periods (and deadlines since they are implicit) is schedulable, too. Obviously, it is beneficial to look for accelerated *simply periodic* task sets since they provide the highest possible utilization bound of 1. For each such candidate ASPTS, the utilization which is greater than or equal to the original utilization is calculated. As soon as such a utilization  $u' \leq 1$  is found, the search can be canceled and the test answer is schedulable (first for the transformed task set, but then also for the original task set). In the unfavorable case of not finding any  $u' \leq 1$ , no decision on RMS schedulability is possible. Each such case contributes to a decrease in the sensitivity of the test.

Among the two important algorithms to search for ASPTSs *Specialization with respect to r* (Sr) [24] and *Distance-constrained tasks* (DCT), we recommend DCT because of its higher sensitivity.

Sr is based on a powers-of-2 period ratio pattern. Each task can be taken as pivot, i.e., its period is maintained. Then, every other task is minimally accelerated such that the period ratio becomes a power of 2. At first, complexity seems to be quadratic, but a clever implementation including a sorting routine yields a complexity of only  $\mathcal{O}(n \log n)$ , see [24] for a detailed description.

According to DCT, the original task set is accelerated to a simply periodic one with only integer period ratios. DCT has  $n$  options for taking one task as a pivot. The idea is then to accelerate step-by-step up- and downward in the sorted list of tasks. As soon as an accelerated simply periodic task set of utilization  $ju \leq 1$  is found, the original task set is rendered RM schedulable. Computational complexity is  $\mathcal{O}(n^2)$ . For example, the periods 5, 12 and 27 can be accelerated by DCT to 5, 10, 20 (first is pivot), 4, 12, 24 (second is pivot) or 4.5, 9, 27 (last is pivot). For a more detailed explanation, see [9] and [24].

Other tests like Hyperbolic Bound [16] as done in [15] or even the exact TDA [19], see also Section V, could be used at this stage. But DCT provides a very good trade-off between (test) sensitivity and computational complexity, cf. Section V.

#### B. Allocation Strategy: First Fit instead of Next Fit

Computational complexity of Next Fit (NF) is only linear, compared to the log-linear one of First Fit (FF). On the other hand, performance of NF is worse [25]. NF produces linear waste, cf. [13], p. 46. Thus, NF shall be substituted by FF. This approach was suggested in [26], too. There, FF is also preferred over NF due to its superior task distribution behavior when fault tolerance with multiple task versions is regarded. [17] prefers FF over NF, too.

#### C. Presorting: Index Offset for Start of X Fit

The core idea of Burchard's multiprocessor scheduling approach is the presorting of the tasks according to their  $S_i$  values (1) representing their dissimilarity to a power of 2 *from above* due to the flooring operation in (1). Next, using the difference of maximum and minimum  $S_i$  values as a linear similarity measure, the degree of simplicity of periods is estimated according to (2). NF and FF both start at the task with the lowest  $S_i$  when it comes to task allocation to processors. This is not justified, there are  $n$  different index offsets possible to start with. *Dissimilarity* to a power of 2 *from above* turns into *similarity* to a power of 2 *from below* at the threshold value of 0.5. Thus, the slope of the function has to be inverted at this point. Then, in turn,  $1 - S_i$  represents the dissimilarity to a power of 2, again. As a normalized *circular* similarity measure based on values  $S_i$  and  $S_j$ , we obtain (6).

$$d(i, j) = |2|S_i - S_j| - 1| \quad (6)$$

With that definition, the complementary dissimilarity  $1 - d(i, j)$  fulfills the four conditions of a metric: non-negativity (due to the outer absolute value and the domain of  $S$ ), identity of indiscernibles (by solving), symmetry (absolute value of difference) and triangle inequality.

Note that already Figure 1 in [11], p. 1437, suggests such a circular relationship, but nothing appropriate can be found in the text. Later on, Rothvoß became aware of the problem in [13], p. 23f., but did not tackle it. Plots of the dependence of normalized circular similarity from  $S$  values are given in Figures 1 and 2. In Figure 1, the normalized similarity  $d(i, j)$  (scaled from value 0 as completely different to value 1 as equal) between two task periods is plotted against the absolute value of the difference between their  $S$  values. Clearly, having the same  $S$  values, i.e., difference 0, means equality. Note that a difference of 1 cannot appear according to (1). Only an arbitrarily exact approximation to 1 from below occurs in the border cases of  $S_i = 0$  and  $S_j \rightarrow 1$  or  $S_i \rightarrow 1$  and  $S_j = 0$ . Two task periods are completely

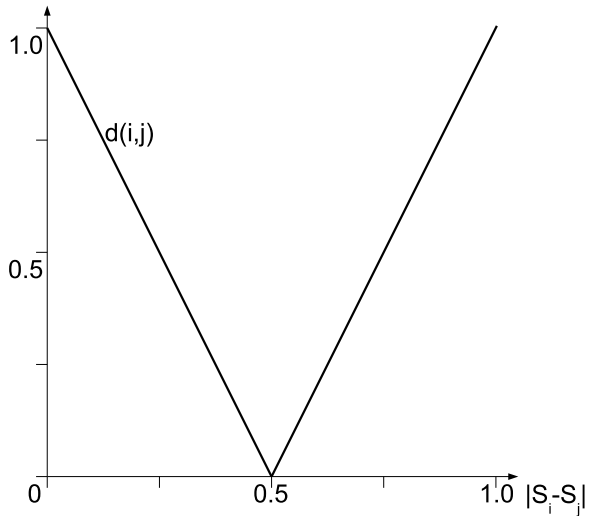


Figure 1. Normalized Circular Similarity as a Function of Absolute  $S$  Value Difference  $|S_i - S_j|$

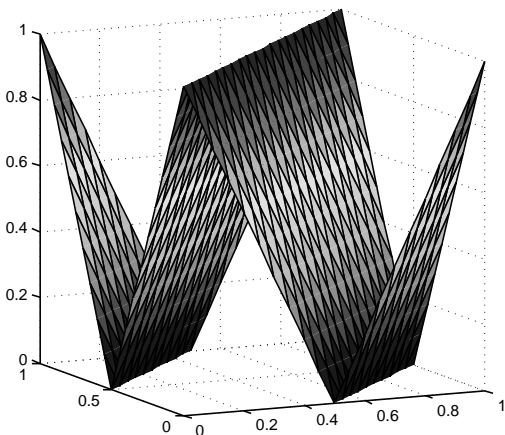


Figure 2. Normalized Circular Similarity as a Function of  $S$  Values  $S_i$  and  $S_j$

different when the absolute difference of their  $S$  values is 0.5. An example for that situation is  $p_i = 1$  and  $p_j = \sqrt{2}$ . This is in accordance with worst-case period ratios, cf. [2] and [27], pp. 85-89.

Then, using (1), we get  $S_i = 0$  and  $S_j = 0.5$ . A period ratio of  $\sqrt{2}$  is the worst case in terms of similarity between two tasks when base 2 is used. The similarity values between the extremes are linearly interpolated. This can be justified by Occam's razor since the straight line is the shortest and simplest connection between two points in a plane.

In Figure 2, the normalized similarity  $d(i, j)$  between two task periods is plotted against their  $S$  values  $S_i$  and  $S_j$ . Thus, it is a function of two variables. Since similarity depends only on the absolute difference between the two  $S$  values,

the function is constant along the *lines of equal difference*  $S_i = S_j + c$  where  $c$  is the difference parameter. Thus, cross sections along  $S_i = a - S_j$  with  $a$  representing a constant sum of  $S_i$  and  $S_j$  are orthogonal to the *lines of equal difference* and thus show parts of the function given in Figure 1.

On a circle, there is no outstanding point. Thus, to start with the task with the lowest  $S_i$  value, is just *one* out of  $n$  equal possibilities. The systematic approach is to consider all of them and taking the one with the best result. For a full search, all offsets have to be included, increasing the computational complexity by a factor of  $n$ . Note that this search on a soundly sorted ring with  $n$  different initializations is still much less complex than an exhaustive search (in terms of input for the allocation step itself) with  $n!$  possibilities.

#### D. Base for Period Similarity: Change from 2 to 3

In order to obtain a sorted ring as a basis for the allocation step, the choice of the logarithm base in (1) is influential. One could take 3 instead of 2 as base. In [13], p. 45, where FF-Bu is described, the logarithm base is omitted what is a cue that the author was aware of the influence of base choice. It will turn out that the widely used base 2 is indeed the most reasonable one, but there are cases where other bases are better choices. The problem of only considering base-2 approaches is not seldom occurring in computer science. E.g., specialization algorithms first did only regard *binary* systems. Later on, other bases were included in research, see [28].

We define  $S_3$  values similar to (1) for a base-3 period dissimilarity measure from above in (7).

$$S_3 := \log_3 p_i - \lfloor \log_3 p_i \rfloor \quad (7)$$

Consider the task set  $\{T_1(5, 2), T_2(15, 9)\}$ . It has a utilization of  $u = 1$  and is RM schedulable since it is simply periodic. While the  $S$  values are  $S_1 \approx 0.322$  and  $S_2 \approx 0.907$ , the two  $S_3$  values are the same:  $S_{31} = S_{32} \approx 0.465$ . Thus, a presorting according to the classical method tends to separate  $T_1$  and  $T_2$  in the ring while the newly suggested base-3 method will group them together in the ring. Thus, an assignment to one and the same processor is likely in the latter case, especially when combining it with the offset method, cf. Subsection IV-C. The more reasonable procedure for this example is the base-3 method since it does not separate perfectly fitting tasks. Hence, there are cases where the conventional base-2 ordering is not appropriate. Similarity to a power of 2 does not completely cover intra-set period similarity since the period ratios can be dominated by other prime factors.

In the general case, other prime numbers are base candidates, too. Thus, a search starting with bases 2, 3, 5, 7, 11, ... would be the best. On the other hand, improvements with other bases than 2 seem to be restricted to exotic examples. Thus, it might not pay off since success of variations

in this early processing step can only be evaluated after the allocation step, leading to an expensive trial-and-error procedure.

### E. Discussion

After having introduced four improvement suggestions for RMST, we return to the motivating example from Section II. We iterate through the improvements given and evaluate their effect.

Concerning the base choice (IV-D), the sequence obtained by sorting the tasks according to their  $S3$  values is already very close<sup>7</sup> to the optimal partitioning in Table II. The here inappropriate shuffling as done by a  $S$  sorting impairs the sequence so seriously that FF cannot find the optimal task clusters.

Second (IV-C), it is necessary to split the task set between  $T_7$  and  $T_8$  since otherwise all three processors would be too crowded when it comes to allocation of  $T_8$ . Hence, the offset method as an implementation of a circular similarity measure is necessary.

Third (IV-A), the Burchard test cannot declare a single partition on the three processors as schedulable. On processor 1, the Bu bound is  $u_{Bu} \approx 0.761$ , on processors 2 and 3, there is a fallback to LL since it performs better, cf. (3) and (4). The utilization bounds are  $u_{LL}(2) \approx 0.828$  and  $u_{LL}(4) \approx 0.757$ .

Due to the good coverage by the offset method, FF, cf. IV-B, is not superior to NF in this example when applying all other improvements. With the right offset chosen, only the separators have to be placed correctly what is well possible with the lower-complexity NF. In other cases, FF can behave better than NF. This observation suggests that the improvements are not completely orthogonal, but almost.

For a summary on the partitioning of the example task set, see Figure 3. Except for the allocation strategy change from NF to FF, all other three improvements to RMST are necessary for decreasing the number of required processors from 4 to 3. In Figure 3, a circle in clockwise orientation represents  $S3$  values from 0 to 1. The result is that 3 processors are required for scheduling 10 tasks  $S3_1$  to  $S3_{10}$ . Simply starting a clustering according to First Fit or Next Fit with the smallest  $S3$  value ( $S3_4 = S3_9$ ) would result in 4 required processors. But the similarity relationship is circular. Thus, a circular line is an appropriate geometrical representation since it has no outstanding point. Further, we obtain that starting with Task 8 characterized by  $S3_8$  results in an optimal clustering onto just 3 processors, cf. Table II. Thus, a less restricted choice, different from the  $S3 = 0$  line, of the first cut between tasks to allocate them to processors is essential to improve the approximation in terms of the number of required processors. This freedom is appropriate

<sup>7</sup>Note that permutations on one and the same processor do not make a difference.

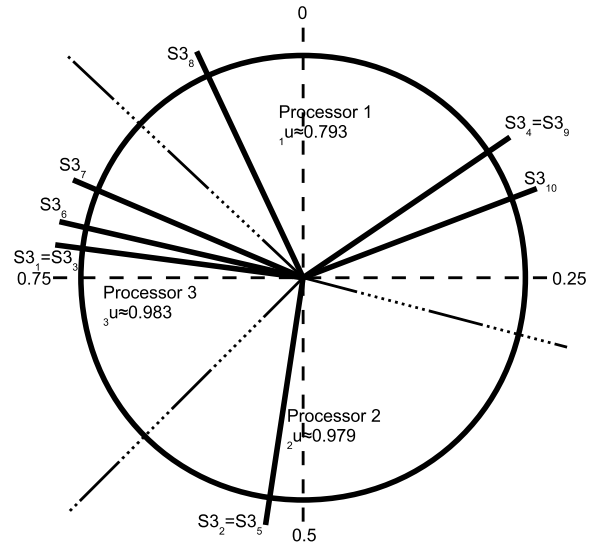


Figure 3. Partitioned Example Task Set Using all Improvements: Circular Similarity of Tasks Based on Their  $S3$  Values; Processor-1-Tasks are Intersecting the 0 Point Which is an Outstanding Point on a Straight-Line Segment, but not on a Circle

to the circular nature of the similarity measure of  $S3$  (and  $S$ ) values.

## V. EXPERIMENTS ON SYNTHETIC TASK SETS

For a stringent naming of the algorithms possible with these RMST modifications, we suggest the following scheme given in Table III. E.g., RMST is NF-sBu-noOffset-Base2

NF	sBu	noOffset	Base2
FF	Bu	Offset	Base3
	DCT		
	TDA		

Table III  
NAMING SCHEME FOR PARTITIONED MULTIPROCESSOR SCHEDULING OF THE Burchard Kind

in this terminology. The most promising combination in terms of average processor minimization performance is FF-DCT-Offset-Base2. Since all four dimensions can be combined freely, there are 32 combinations. In the following simulation, only 12 of them will be evaluated since FF-X-noOffset-Base2 algorithms have already been discussed in [9], and trends of the usefulness of each modification can be concluded from single-dimension modifications. The exact test TDA has been added to the uniprocessor scheduling criteria in order to have a reference. Note that a promising combination like FF-TDA-Offset-Base2 is not optimal for the minimization of  $m$ , it is just a very good heuristic. Just one optimal stage does not make the entire procedure optimal. Only an exhaustive search would find the global optimum, cf. Section II.

100,000 task sets of exactly 10 tasks each are investigated. The target *total* utilization is 2.5. Task utilizations follow a uniform distribution and are generated using the *UUniFast-Discard* algorithm [29] which is a multiprocessor extension of the *UUniFast* algorithm suggested in [30]. Period values follow an integer (i.e., granularity equals to one) log-uniform distribution in the interval  $[10, 10^5]$  which was suggested to be more appropriate than a uniform distribution [31]. WCETs are fully determined by utilization and period values and can be calculated easily as products of these two values. The results of obtained percentages

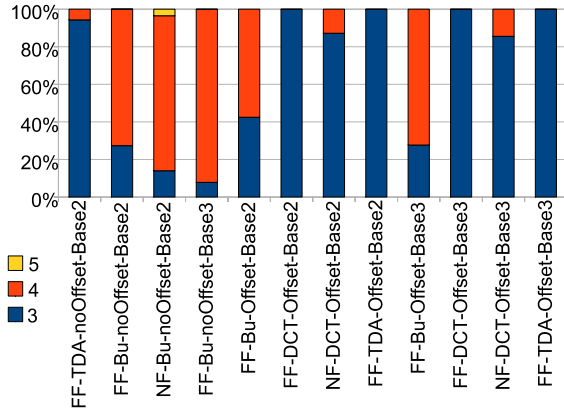


Figure 4. Performance of Suggested Algorithms on Synthetic Task Sets: Number  $m$  of Processors Required for Partitioned Scheduling, Percentages of Synthetic Task Sets;  $n = 10$ ,  $u = 2.5$ ,  $u_{max} = 1$

Algorithm	m=3	m=4	m=5
FF-DCT-Offset-Base2	99,908	92	0
FF-DCT-Offset-Base3	99,910	90	0
FF-TDA-Offset-Base2	99,918	82	0
FF-TDA-Offset-Base3	99,917	83	0

Table IV  
PERFORMANCE OF SUGGESTED ALGORITHMS ON SYNTHETIC TASK SETS: NUMBER  $m$  OF PROCESSORS REQUIRED FOR PARTITIONED SCHEDULING, PERCENTAGES OF SYNTHETIC TASK SETS;  $n = 10$ ,  $u = 2.5$ ,  $u_{max} = 1$ ; ZOOM INTO THE BEST ALGORITHMS

of  $m$  values are given in Figure 4. Note that  $m$  values of 4 and 3 are predominant. The rare  $m$  value 5 occurs at a significant portion only with the algorithm NF-Bu-noOffset-Base2. Since Figure 4 shows no clear difference between the four algorithms FF-DCT-Offset-Base2, FF-DCT-Offset-Base3, FF-TDA-Offset-Base2 and FF-TDA-Offset-Base3, the results for them are given in Table IV.

The substitution of Bu by DCT (IV-A) reduces the average number of processors needed significantly as does the replacement of NF by FF (IV-B). Third, the offset search (IV-C) is reasonable in terms of decreasing the average  $m$  value.

The situation is different for taking 3 instead of 2 as the logarithmic base for period similarity ordering (IV-D). This new approach entails consistently worse results. But the differences become of less importance when offset search is applied. In a direct comparison between FF-DCT-Offset-Base2 and FF-DCT-Offset-Base3, the base-3 approach is even marginally better, see Table IV. Anyway, for this component, it is better to keep the established base-2 option.

In total, the algorithm with the best performance in terms of minimizing the expected number of processors required is FF-TDA-Offset-Base2. But FF-TDA-Offset-Base3, FF-DCT-Offset-Base3 and FF-DCT-Offset-Base3 show results only marginally worse. The differences between FF-DCT-Offset-X and FF-TDA-Offset-X are so small that the polynomial complexity of FF-DCT-Offset-X compared to the pseudo-polynomial one of FF-TDA-Offset-X makes FF-DCT-Offset-Base2 to the overall winner both considering performance and complexity. Similar results have been

Algorithm	m=6	m=7	m=8	m=9
FF-TDA-noOffset-Base2	99,584	416	0	0
FF-Bu-noOffset-Base2	64,409	35,387	4	0
NF-Bu-noOffset-Base2	14,480	78,672	6,843	5
FF-Bu-noOffset-Base3	1,804	96,672	1,523	1
FF-Bu-Offset-Base2	87,475	12,525	0	0
FF-DCT-Offset-Base2	100,000	0	0	0
NF-DCT-Offset-Base2	79,653	20,335	12	0
FF-TDA-Offset-Base2	100,000	0	0	0
FF-Bu-Offset-Base3	15,230	84,770	0	0
FF-DCT-Offset-Base3	100,000	0	0	0
NF-DCT-Offset-Base3	73,461	26,512	27	0
FF-TDA-Offset-Base3	100,000	0	0	0

Table V  
PERFORMANCE OF SUGGESTED ALGORITHMS ON SYNTHETIC TASK SETS: NUMBER  $m$  OF PROCESSORS REQUIRED FOR PARTITIONED SCHEDULING,  $n = 20$ ,  $u = 5$ ,  $u_{max} = 0.5$

obtained under variation of the number of tasks  $n = 5, 10, 20, 40$ , of the total utilization  $u = 1, 2, 2.5, 5, 10$ , and of the maximum utilization  $u_{max} = 0.5, 1$ . The advantage of base-2 approaches compared to base-3 approaches seems to increase with an increasing number of tasks, see Table VI. With a restricted maximum utilization  $u_{max} = 0.5$ , FF-DCT-Offset-Base2 (and its TDA and base-3 relatives) can sometimes reach the optimal number of processors even for all synthetic task sets, see Table V. These results suggest that the favored algorithm FF-DCT-Offset-Base2 even increases its winning margin when it comes to more realistic situations with a higher number of tasks and a restricted maximum utilization.

## VI. CONCLUSIONS

The already successful RMST algorithm has been further improved in terms of minimization of the average number of processors required for partitioned RM scheduling of a given task set. Apparently, partitioned scheduling based on RMS has been oversimplified too often. It is important to

Algorithm	m=11	m=12	m=13	m=14	m=15	m=16	m=17	m=18
FF-TDA-noOffset-Base2	15,430	67,550	16,102	902	16	0	0	0
FF-Bu-noOffset-Base2	303	43,296	49,502	6,608	289	2	0	0
NF-Bu-noOffset-Base2	9	2,645	27,116	47,523	20,395	2,256	55	1
FF-Bu-noOffset-Base3	21	23,126	61,692	14,369	771	21	0	0
FF-Bu-Offset-Base2	445	59,741	36,725	2,974	113	2	0	0
FF-DCT-Offset-Base2	46,384	49,432	3,978	204	2	0	0	0
NF-DCT-Offset-Base2	101	8,214	43,107	39,987	8,250	338	3	0
FF-TDA-Offset-Base2	48,287	47,573	3,935	203	2	0	0	0
FF-Bu-Offset-Base3	152	55,252	41,299	3,184	111	2	0	0
FF-DCT-Offset-Base3	45,248	50,574	3,970	206	2	0	0	0
NF-DCT-Offset-Base3	58	7,716	42,305	40,648	8,896	374	3	0
FF-TDA-Offset-Base3	47,441	48,417	3,935	205	2	0	0	0

Table VI

PERFORMANCE OF SUGGESTED ALGORITHMS ON SYNTHETIC TASK SETS: NUMBER  $m$  OF PROCESSORS REQUIRED FOR PARTITIONED SCHEDULING,  $n = 20$ ,  $u = 10$ ,  $u_{max} = 1$

account for period values when using RMS since simple utilization bounds like Liu/Layland are too pessimistic. Such an oversimplification can result in a significant waste of processors. Thoroughly examined, partitioned RMS is not simple Bin Packing. Our advanced methods reduce the waste of CPU time in the average case.

Four aspects of RMST have been investigated: uniprocessor scheduling algorithm, allocation strategy, mode of similarity measure on the pre-sorted task list, and the logarithmic base for presorting tasks. It turned out that these four approaches are orthogonal with the exception of an interdependence between the allocation strategy and the allowance of different index offsets. Anyway, the best option here is to choose First Fit and Offset.

For the motivating example, three out of four modifications are necessary for requiring only  $m = 3$  instead of  $m = 4$  processors. Only the change from NF to FF does not contribute in this case. The situation is different for synthetic task sets. The exception not decreasing the average number of processors needed is the change of the logarithmic base from 2 to 3. But the motivating example has shown that base-2 approaches are *not* dominating base-3 approaches, i.e., base 2 is not always better than other ones.

A clear terminology based on the four components of the algorithms was introduced. The algorithm FF-DCT-Offset-Base2 is most promising since it applies exactly the three proven and tested modifications while maintaining the more successful logarithmic base 2. Excluding the even slightly better algorithms FF-TDA-Offset-Base2 and FF-TDA-Offset-Base3 with the drawback of a pseudo-polynomial complexity, FF-DCT-Offset-Base2 is also in the empirical evaluation the winner.

For the future, we plan to further verify our results on the improvements compared to classical RMST both with more sophisticated analytical and empirical methods. Computational complexities of the new approaches shall be investigated. Clearly, the improved  $m$  values come at a price of more effort. While this effort may be tolerable for FF instead of NF ( $\mathcal{O}(n \log n)$  instead of  $\mathcal{O}(n)$ ), applying DCT

instead of Bu (from  $\mathcal{O}(n)$  to  $\mathcal{O}(n^2)$ , see [9]) and the offset search (factor  $n$ ) might be too expensive, particularly in admission control systems, cf. [32], where online scheduling is required. Clever solutions are to substitute DCT by the only slightly lower-sensitivity Sr with only  $\mathcal{O}(n \log n)$  complexity, cf. [9], and looking for reasonable constraints on the offset value in the offset search. A key parameter is probably the maximum number of tasks on a processor which can be bounded from above by the number of smallest tasks until reaching the threshold value of 1.

## VII. ACKNOWLEDGMENTS

We would like to thank Alejandro Masrur for his useful comments on a preliminary version of this paper and Jai Dhariwal for proofreading.

## REFERENCES

- [1] S. K. Dhall and C. Liu, "On a real-time scheduling problem," *Operations Research*, vol. 26, no. 1, pp. 127–140, 1978.
- [2] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *J. ACM*, vol. 20, no. 1, pp. 46–61, January 1973. [Online]. Available: <http://dx.doi.org/10.1145/321738.321743>
- [3] M. L. Dertouzos, "Control robotics: The procedural control of physical processes," in *IFIP Congress*, 1974, pp. 807–813.
- [4] G. Levin, S. Funk, C. Sadowski, I. Pye, and S. Brandt, "DP-FAIR: A simple model for understanding optimal multiprocessor scheduling," in *Proc. of the 22nd Euromicro Conference on Real-Time Systems*, 2010, pp. 3–13.
- [5] R. I. Davis and A. Burns, "A survey of hard real-time scheduling algorithms and schedulability analysis techniques for multiprocessor systems," *ACM Comput. Surv.*, vol. 43, no. 4, 2011.
- [6] B. Andersson, "Static-priority scheduling on multiprocessors," Ph.D. dissertation, Chalmers University of Technology, 2003.

- [7] A. Bastoni, B. B. Brandenburg, and J. H. Anderson, "An empirical comparison of global, partitioned, and clustered multiprocessor EDF schedulers," *Real-Time Systems Symposium, IEEE International*, vol. 0, pp. 14–24, 2010.
- [8] G. C. Buttazzo, "Rate monotonic vs. EDF: judgment day," *Real-Time Syst.*, vol. 29, pp. 5–26, January 2005.
- [9] D. Müller, "Accelerated simply periodic task sets for RM scheduling," in *Proc. of Embedded Real Time Software and Systems (ERTS<sup>2</sup>)*, Toulouse, 2010, p. 46. [Online]. Available: [http://www.erts2010.org/Site/0ANDGY78/Fichier/PAPIERS%20ERTS%202010%202/ERTS2010\\_0140\\_final.pdf](http://www.erts2010.org/Site/0ANDGY78/Fichier/PAPIERS%20ERTS%202010%202/ERTS2010_0140_final.pdf)
- [10] S. Davari and S. K. Dhall, "On a real-time task allocation problem," in *Proc. of the 19th Hawaii International Conference on System Science*, 1985.
- [11] A. Burchard, J. Liebeherr, Y. Oh, and S. Son, "New strategies for assigning real-time tasks to multiprocessor systems," *Computers, IEEE Transactions on*, vol. 44, no. 12, pp. 1429–1442, Dec 1995.
- [12] S. Lauzac, R. Melhem, and D. Mosse, "An efficient RMS admission control and its application to multiprocessor scheduling," in *Parallel Processing Symposium, 1998. IPPS/SPDP 1998. Proceedings of the First Merged International ... and Symposium on Parallel and Distributed Processing 1998*, Apr. 1998, pp. 511–518.
- [13] T. Rothvoß, "On the computational complexity of periodic scheduling," Ph.D. dissertation, École Polytechnique Fédérale de Lausanne (EPFL), Lausanne, 2009. [Online]. Available: <http://library.epfl.ch/theses/?nr=4513>
- [14] A. Karrenbauer and T. Rothvoß, "A 3/2-approximation algorithm for rate-monotonic multiprocessor scheduling of implicit-deadline tasks," in *Proceedings of the 8th international conference on Approximation and online algorithms*, ser. WAOA'10. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 166–177. [Online]. Available: <http://portal.acm.org/citation.cfm?id=1946240.1946255>
- [15] Y. Oh and S. H. Son, "Allocating fixed-priority periodic tasks on multiprocessor systems," *Real-Time Syst.*, vol. 9, pp. 207–239, November 1995.
- [16] E. Bini, G. Buttazzo, and G. Buttazzo, "A hyperbolic bound for the rate monotonic algorithm," in *ECRTS '01: Proceedings of the 13th Euromicro Conference on Real-Time Systems*. Delft, The Netherlands: IEEE Computer Society, 2001, pp. 59–66.
- [17] A. Karrenbauer and T. Rothvoß, "An Average-Case Analysis for Rate-Monotonic Multiprocessor Real-time Scheduling," in *17th Annual European Symposium on Algorithms (ESA'09)*, ser. Lecture Notes in Computer Science. Berlin: Springer, 2009. [Online]. Available: <http://algo2009.itu.dk/esa-2009>
- [18] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co., 1990.
- [19] J. P. Lehoczky, L. Sha, and Y. Ding, "The rate monotonic scheduling algorithm: Exact characterization and average case behavior," in *RTSS*, 1989, pp. 166–171.
- [20] K.-D. Althoff, K. Bach, and M. Reichle, "Realizing modularized knowledge models for heterogeneous application domains," in *Proceedings of the 8th industrial conference on Advances in Data Mining: Medical Applications, E-Commerce, Marketing, and Theoretical Aspects*, ser. ICDM '08. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 114–128. [Online]. Available: [http://dx.doi.org/10.1007/978-3-540-70720-2\\_9](http://dx.doi.org/10.1007/978-3-540-70720-2_9)
- [21] X. He, Z. Gu, and Y. Zhu, "Task allocation and optimization of distributed embedded systems with simulated annealing and geometric programming," *Comput. J.*, vol. 53, pp. 1071–1091, September 2010.
- [22] S. Lauzac, R. Melhem, and D. Mossé, "An improved rate-monotonic admission control and its applications," *IEEE Trans. Comput.*, vol. 52, no. 3, pp. 337–350, 2003.
- [23] S. Baruah and A. Burns, "Sustainable scheduling analysis," in *Real-Time Systems Symposium, 2006. RTSS '06. 27th IEEE International*, 2006, pp. 159–168.
- [24] C.-C. Han and H.-Y. Tyan, "A better polynomial-time schedulability test for real-time fixed-priority scheduling algorithms," *Real-Time Systems Symposium, IEEE International*, vol. 0, pp. 36–45, 1997.
- [25] C. Bays, "A comparison of Next-fit, First-fit, and Best-fit," *Commun. ACM*, vol. 20, no. 3, pp. 191–192, 1977.
- [26] H. Beitollahi and G. Deconinck, "Fault-tolerant partitioning scheduling algorithms in real-time multiprocessor systems," in *PRDC '06: Proceedings of the 12th Pacific Rim International Symposium on Dependable Computing*. Riverside, USA: IEEE Computer Society, 2006, pp. 296–304.
- [27] G. C. Buttazzo, *Hard real-time computing systems: predictable scheduling algorithms and applications*. Kluwer Academic Publishers, 1997.
- [28] C.-W. Hsueh and K.-J. Lin, "An optimal pinwheel scheduler using the single-number reduction technique," in *RTSS '96: Proceedings of the 17th IEEE Real-Time Systems Symposium*. Washington, DC, USA: IEEE Computer Society, 1996, p. 196.
- [29] R. I. Davis and A. Burns, "Priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems," in *IEEE Real-Time Systems Symposium*, T. P. Baker, Ed. Washington, DC, USA: IEEE Computer Society, 2009, pp. 398–409.
- [30] E. Bini and G. C. Buttazzo, "Measuring the performance of schedulability tests," *Real-Time Syst.*, vol. 30, pp. 129–154, May 2005.
- [31] P. Emberson, R. Stafford, and R. I. Davis, "Techniques for the synthesis of multiprocessor tasksets," in *Proc. of the 1st Int'l Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS 2010) In conjunction with the 22nd Euromicro Conference on Real-Time Systems (ECRTS10)*, G. Lipari and T. Cucinotta, Eds., Brussels, Belgium, 2010, pp. 6–11.
- [32] A. Masrur, S. Chakraborty, and G. Färber, "Constant-time admission control for deadline monotonic tasks," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2010*, 2010, pp. 220–225.