Accelerated Simply Periodic Task Sets for RM Scheduling

D. Müller[1]

1: Chemnitz University of Technology, Faculty of Computer Science, D-09107 Chemnitz, GERMANY

Abstract: The article examines rate-monotonic scheduling (RMS). The focus is on efficient schedulability tests of high sensitivity.

Accelerated simply periodic task sets (ASPTSs) are constructed by shortening task periods in order to obtain a transformed – simply periodic – task set where each period is an integer divisor of all longer periods. The article presents a new heuristic for partitioned multiprocessor (MP) scheduling based on Specialization with respect to r (Sr) and Distance-Constrained Tasks (DCT) which use ASPTSs first described by Han and Tyan [9, 10]. They have already shown the advantage of Sr and DCT over the Liu/Layland (LL) and the Burchard (Bu) bound in terms of sensitivity. First, the article compares Sr and DCT as well with other uniprocessor scheduling criteria, both theoretically and empirically. Next, these tests are applied to MP scheduling. Theory is followed by a case study and an empirical investigation with randomised task sets. Related approaches are thoroughly examined and summarised in a scheme where the central role of ASPTSs becomes obvious.

The article shows that Sr and DCT provide a very good trade-off between maximizing the scheduling test sensitivity (no unnecessary hardware) and minimizing the test's computational complexity (towards real-time decisions on schedulability).

Keywords: rate-monotonic scheduling, partitioned multiprocessor scheduling, sufficient schedulability test, Sr, DCT

1. Introduction

Scheduling in computing deals with assigning jobs to processors. Often, equal jobs have to be executed periodically, especially in embedded real-time systems. Then, the periodic task model applies, where jobs are instances of a task. Each task is specified by a period, a (relative) deadline, an execution time and a phase. Jobs are released periodically in accordance to their task's period. All jobs must be completed no later than the respective absolute deadline which is the release time plus the relative deadline. In the most typical case, the (relative) deadline equals the period and the phase is zero. Then, only two parameters characterizing each task remain: period and execution time. A necessary condition for schedulability is that the utilization, the sum of the ratios between execution time and period of all tasks, is not greater than the number of processors. Often it

is much more difficult to provide sufficient or even necessary and sufficient conditions for schedulability. The goal of scheduling is to assign as many tasks as possible to the available processors, or, closer to praxis, to minimise the number of necessary processors for a given task set. So, the economic objective is to minimise necessary CPU resources, and, thus, to minimise hardware costs.

Rate-monotonic scheduling (RMS) is a widely applied scheduling policy for the periodic task model on uniprocessor systems. It enables the scheduling of real-time tasks with a processor utilization of at least 0.69. Often, much higher utilizations of 0.88 [17] or even above 0.90 are possible. RMS is the optimal static-priority scheduling scheme where task priorities are assigned fixedly over the entire scheduling. The rule of RMS is the shorter the task's period the higher the task's priority. Schedulability analysis deals with necessary, sufficient, or necessary and sufficient criteria for the determination whether a task set is schedulable or not.

The only known necessary and sufficient criteria are simulation along the entire hyperperiod of the task set and Time Demand Analysis (TDA) [17]. Both methods involve a computational complexity beyond $O(n^2)$ which is too expensive in many situations. Thus, sufficient criteria have been developed in order to obtain partial results much more quickly. The best-known procedure is the Liu/Layland criterion with only linear complexity but a bad performance. Frequently, a false negative answer is given using such a simple test. In that, the Liu/Layland test is a much too pessimistic one ignoring most of the data provided by the respective task set.

Later on, improved linear computational complexity criteria like the Hyperbolic Bound [2] or the Burchard criterion [3] have been developed. At the other end of the scale, quadratic computational complexity criteria like Critical Task Sets [5] and the Pillai/Shin criterion [21, 13] evolved which are sufficient ones, too. The advantage of them is that they provide high performance and, thus, are good approximations of a TDA in many situations.

In 1997, Han and Tyan [9] presented criteria Sr and DCT with complexities $O(n \log n)$ and $O(n^2)$ with very good performance (sensitivity or true positive rate). They are underrepresented in literature and deserve a thorough investigation. This will be provided by extensive comparisons (tables and charts) both of theoretical and practical (empirical)

nature. Finally, a multiprocessor RM scheduling case study will be delivered.

It will be shown that DCT is the best known $O(n^2)$ sufficient criterion for RM schedulability on a uniprocessor. Sr behaves slightly worse, but has the advantage of an only linear-logarithmic complexity. Hence both procedures are cost-effective. It turns out that DCT as a greedy algorithm to find an accelerated simply periodic task set with a utilization not greater than one is better on the average, while the non-greedy algorithm Sr can outperform DCT in rare cases. Such an example is discussed.

Next, amazing interrelationships between different uniprocessor schedulability criteria will be uncovered. This systematization will stress the central role of ASPTSs in this context. It will be thoroughly explained why using transformations of task sets is the method of choice for a decision on schedulability. In most cases, the used transformation is an acceleration, i.e., to shorten the periods while maintaining the execution times of the individual tasks, respectively. Here, besides simply periodic task sets, roots and fundamental frequencies of the periods of the task play an important role and serve as transformation goals.

Partitioned RM multiprocessor scheduling[1] is strongly based on heuristics since an exhaustive search turns out to be infeasible even for a small number of tasks. A combinatorial explosion is the reason. The approach is characterised by a suitable combination of a Bin Packing heuristic and a uniprocessor schedulability test. A combination of First Fit with Sr or DCT will turn out to be superior to other known RM MP scheduling heuristics in terms of minimizing the number of processors required to schedule a given task set. This is both performed with a task set from a problem posed by Jane Liu as exercise 9.3 in [19], p. 390 and randomly generated task sets. The adequate generation of random task sets is crucial for the informational value of performance ratios. So it would be wrong to take only chains of multiples as periods since this is a very special configuration and not general enough. Thus, special care will be devoted to the random generation of task sets.

As an outlook on future research, the combination of different uniprocessor schedulability tests and the investigation of runtime behaviour of the compared criteria will be suggested. The latter one is important since a purely theoretical comparison of the complexities may not be appropriate to praxis. Here, behaviour with a lower number of tasks and the operations necessary to execute per step are important, too, for runtime evaluations. This goes up to worst-case execution time (WCET) analysis which is a cornerstone of real-time analysis.

---

1   The alternative is *global* scheduling. Here, the optimal algorithms PFAIR and LLREF are better in theory (worst case), ignoring runtime overhead [24, 25, 26].

2. Methodology

2.1 System model

There are given n pre-emptive, independent, periodic real-time tasks $T=\{T_1, T_2, .., T_n\}$ with $T_i=(p_i, e_i)$ where p_i is the period and e_i is the execution time. The deadline is assumed to be equal to the period, i.e., $\forall i : p_i = d_i$. Additionally, $\forall i : 0 < e_i \le p_i$ should hold. The individual utilizations $u_i = e_i / p_i$ sum up to the total utilization

$$u = \sum_{i=1}^{n} u_i$$

. The challenge is to schedule these tasks on a set of m identical processors $P=\{P_1, P_2, .., P_m\}$. The task priorities, denoted by $Prio_i$, shall be assigned statically and in a rate-monotonic way, the shorter the period the higher the priority. Equal period ties are broken arbitrarily, but fixed[2]. Further, we restrict our considerations to the simplest case of partitioned schemes which doesn't allow any migration. The last two restrictions imply that we consider a (1,1)-restricted MP scheduling according to the terminology introduced in [4]. The remaining problem is to map tasks to processors where they can be scheduled following the rate-monotonic uniprocessor scheduling, respectively. We will consider only off-line algorithms where all task parameters are initially known.

There are two approaches concerning the number of processors. First, this number m can be fixed, and we obtain a decision or feasibility problem whether the task set can be feasibly scheduled. On the other hand, the minimization problem with the objective $m \rightarrow min$ for a given set of n tasks could be raised. We will focus on the minimization problem.

2.2 Two scheduling elements

A strict distinction between the allocation algorithm and the uniprocessor schedulability condition is suggested. Both of them contribute to the performance as well as to the complexity of the heuristics. In general, a trade-off between the complexity and the performance can be expected.

2.3 Allocation algorithm

The problem is related to the Bin Packing Problem. Here, the task is to find an m-partition of a set of n items so that the sum of item sizes in each *bin* is bounded by a constant. The decision problem, cf. 2.1, is NP-Complete while the minimization problem is NP-Hard.

For the decision problem, a brute-force search requires $S(n, m)$ cases to consider. It is the number of ways to partition a set of n items into m non-

---

2   It is convenient to use the task index in order to break the ties.

empty subsets, the Stirling number of the second kind which is exponential in n. The minimization has to check B_n different cases where $B_n = \sum_{m=0}^{n} S(n,m)$ is the n-th Bell number, see [8], p. 258f. and p. 373. Of course, in practice certain odd partitions can be excluded initially because of the size distribution of items, but anyway both problems require an exponential number of cases to check what is impractical or even impossible for a greater n, the number of items. As the notation already suggests, in MP scheduling, the items are the tasks and the bins are the processors.

In order to cope with the minimization, there have been suggested various heuristics. Typical ones are Next Fit (NF), First Fit (FF), Best Fit (BF) and Worst Fit (WF). According to FF, items are allocated to the first bin it fits. In this article, we only use FF, making the allocation algorithm a constant. Our variable is the uniprocessor schedulability condition.

2.4 Uniprocessor Schedulability Condition

Time Demand Analysis (TDA): TDA introduced by Lehoczky *et al.* [17] is beside simulation the only necessary and sufficient condition for RM schedulability. One has to check for a fixed point of the iteration of processing time demand which must be not greater than the deadline of the task (1), a computationally beneficial iterative method given by Audsley *et al.* [1]. Of course, this has to be done for all tasks as the example $\{(5,2),(7,4),(35,1)\}$ shows. It is not sufficient only to check the lowest priority task. Only the probability of a deadline violation is largest for it since it can be disturbed by all other tasks. Sometimes, this fact is not considered correctly. So, Zapata *et al.* claim commenting an example: "Since the lowest priority task is schedulable [..] we conclude that tasks [..] are also schedulable." [23]

$$t_i^{l+1} = e_i + \sum_{j:Prio_j > Prio_i} \left\lceil \frac{t_j^l}{p_j} \right\rceil e_j$$
$$t_i^0 = e_i \qquad (1)$$
$$\forall i \exists l : t_i^{l+1} = t_i^l \leq p_i = d_i$$

TDA has a pseudo-polynomial time complexity. More precisely, it can be given as $O(rn^2)$ where $r = max\ p_i / min\ p_i$ ([19], p. 137).

Pillai/Shin criterion (PS): Pillai and Shin [21, 13] suggest to adapt TDA to obtain a lower-complexity, only sufficient condition. The iteration (1) considers the entire interval $[0; d_i = p_i]$ for each task as it looks for the fixed point which is the (worst case) processing time demand of the task. As a pessimistic approach we consider the maximum number of preemptions by a higher priority task. This is the period ratio rounded up. Multiplying it with the execution

time of the particular higher priority task and summing up over all such tasks gives (2).

$$\forall i : e_i + \sum_{j:Prio_j > Prio_i} \left\lceil \frac{p_i}{p_j} \right\rceil e_j \leq p_i = d_i \qquad (2)$$

Computational complexity is $O(n^2)$.

Liu/Layland criterion (LL): Liu and Layland gave in their classical paper [18] a sufficient upper bound of RM schedulability depending only on the number of tasks to schedule (3).

$$u \leq n(\sqrt[n]{2} - 1) = L(n) \qquad (3)$$

Two minor mistakes in the proof were detected and corrected by Devillers and Goossens [6]. Time complexity is linear since utilization has to be summed up over all tasks.

Liu/Layland limit criterion (LLconst): In order to make this a constant as it is required for a complete analogy to the Bin Packing Problem, cf. 2.3, the infimum limit of L(n) is considered (4).

$$u \leq \ln(2) \qquad (4)$$

Again, time complexity is linear.

Hyperbolic Bound (HB): Instead of considering only the total utilization u, it might be a good idea to include the individual utilizations u_i [2]. This approach resulted in condition (5), which is better than LL. The more heterogeneous the utilizations are distributed, the better is the sensitivity of the criterion.

$$\prod_{i=1}^{n} (u_i + 1) \leq 2 \qquad (5)$$

Computational complexity is linear.

Burchard criterion (Bu): Following [3], we define

$$S_i = \log_2 p_i - \lfloor \log_2 p_i \rfloor$$

for each task and build

$$\beta = max\ S_i - min\ S_i$$

If $\beta < 1 - 1/n$ the condition is

$$u \leq (n-1)(2^{\frac{\beta}{n-1}} - 1) + 2^{1-\beta} - 1 = B(n,\beta) \qquad (6)$$

else, i.e., $\beta \geq 1 - 1/n$, the condition is just (3).

It has been shown that $B(n,\beta) \geq L(n)$ always holds. In that way, Bu is better than LL. Time complexity is linear.

RBound: In 2003, Lauzac *et al.* [16] proposed a new criterion. In order to apply it, the task set has to be transformed first using an algorithm called *ScaleTaskSet* which transforms T into T0 by stretching each period and execution time by an appropriate power of 2. The goal is to have all new periods between half the maximum period and the maximum period. *ScaleTaskSet* includes a sorting by periods in non-decreasing order, but this step is not necessary for the criterion itself. Then, we define

condition	n	u	u_i	p_i	e_i	specifics	complexity	guaranteed equal to or better than
LLconst		x				pure bin packing	$O(n)$	
LL	x	x				most famous	$O(n)$	LLconst
HB			x				$O(n)$	LL
Bu	x	x		x			$O(n)$	LL
RBound	x	x		x		transformation	$O(n)$	LL
PS				x	x	transformation	$O(n^2)$	
CTS		x		x		transformation	$O(n^2)$	LL
Sr				x	x	transformation	$O(n \log n)$	Bu, LL
DCT				x	x	transformation	$O(n^2)$	TDA for $n=2$
TDA				x	x	iteration	$O(rn^2)$	(all)

Table 1: RMS uniprocessor schedulability conditions, r defined in (7)

$$r = max\ p\,'_i - min\ p\,'_i \qquad (7)$$

The condition is then

$$u \le (n-1)(r^{\frac{1}{n-1}} - 1) + \frac{2}{r} - 1 \qquad (8)$$

Time complexity is linear and not log-linear as stated in [16] since the sorting in the original algorithm can be substituted by a min-max search loop. Only these values are evaluated in the further algorithm and the definition of r .

Critical Task Sets (CTS): Chen *et al.* [5] proposed a transformation of the periods to a critical task set following the ideas of Liu and Layland in their proof of the LL bound [18]. First, the periods are sorted in non-decreasing order. Next, incremental subsets starting from 2 up to n periods are considered. Such a subset is then transformed according to

$$p\,'_j = p_j \left\lfloor \frac{p_i}{p_j} \right\rfloor \qquad (9)$$

with i as the outer loop index and j as the inner loop index. This results in a maximum period ratio no larger than 2 . Then, the construction of a critical task set based on [18] with a calculation of the improved utilization bound is possible.

$$u \le min\left(\left\{\left(\left(\frac{\sum_{j=1}^{i-1} p\,'_{j+1} - p\,'_j}{p\,'_j}\right) + \frac{2p\,'_1 - p\,'_i}{p\,'_i} \Big| i = 2..\text{n}\right)\right\} \cup \{1\}\right)$$
$$(10)$$

The utilization must be no larger than the minimum of the critical task set utilizations of the incremental subsets and (from the necessary condition) 1 . Computational complexity is $O(n^2)$.

Summary and Anticipation: In order to give a summary we provide a comparison of the different uniprocessor schedulability conditions in Table 1 including an anticipation (printed in italics) of our proposed conditions introduced later in Section 3.

3. Special task sets

3.1 Accelerated task sets

A task set $T = \{T_1, T_2, .., T_n\}$ can be transformed into an accelerated task set $T\,'$ by maintaining all execution times $e\,'_i = e_i$ and reducing or maintaining all periods $p\,'_i \le p_i$. Thus, the new utilization $u\,' = \sum_{i=1}^n \frac{e\,'_i}{p\,'_i} \ge \sum_{i=1}^n \frac{e_i}{p_i} = u$ is not less than the original one. Note that the case of equivalence between original and newly constructed periods is included.

This allows us to give the following theorem.

Theorem 1 (RM schedulability by accelerated task sets)
Given a task set T it is schedulable by RM if there is an accelerated task set $T\,'$ which is schedulable by RM.

For a proof, see [9].

3.2 Simply periodic task sets

A task set $T = \{T_1, T_2, .., T_n\}$ is called simply periodic if and only if there is an integer ratio (non-negative) between the periods of each pair of tasks selected from T . The condition can be written as (11).

$$\forall i, j \exists c(i,j) \in \mathbb{N}: \frac{max\{p_i, p_j\}}{min\{p_i, p_j\}} = c \quad (11)$$

The intention behind extracting such special task sets is to fit every task completely into all tasks which have a larger period.

Note that, e.g., the task set $\{(3,1),(6,1),(9,1)\}$ is not simply periodic although there is a great regularity in the periods. Obviously, 9 is not an integer multiple of 6. The pattern behind with a step-wise increase by the minimal period is only appropriate for task sets of size 2. We can repair the fault and obtain $\{(3,1),(6,1),(12,1)\}$ as a valid example for a simply periodic task set.

On the other hand, one could be tempted to restrict the pattern to $\{(a \cdot b^i, e_i)\}$ with $b \in \mathbb{N}$. It is obvious that $a=1.5$ and $b=2$ in the previous valid example. In general, a ratio $c(i,j)=b^{|i-j|}$ must be an integer following that construction. Considering a task set like $\{(3,1),(15,1),(30,1)\}$ which doesn't fit into the above given pattern clarifies that the approach was too restrictive. Relaxing the condition we can state the following lemma:

Lemma 1 (Characterization of simply periodic task sets)

A task set $T=\{T_1, T_2,.., T_n\}$ is simply periodic if and only if an (index-dependent) integer ratio exists for all direct neighbours in the non-decreasingly ordered list of task periods.

A proof in forward direction is easy since the ratio is directly given by definition (11). Assuming a non-integer ratio between two task periods leads immediately to contradiction to definition (11). In backward direction, we have to construct the period ratio between two arbitrary periods and to show that this ratio is always an integer. The ratio is the product of all neighbour ratios along the unique path from minimum to maximum period. Since $\mathbb{N}$ is closed under multiplication, it is indeed a non-negative integer. A test based on Lemma 1 is $O(n \log n)$. Sorting is the dominating step, the check for integer ratios is only of linear complexity.

It turns out that the pattern $\{(a \cdot b^i, e_i)\}$ can be regarded as a special case with a constant neighbouring period ratio of b. The misleading thought in the first attempt was the assumption that there should be a constant integer-additive relationship between neighbouring periods, but it has to be a variable integer-multiplicative one. The second approach is – in that terminology – a constant integer-multiplicative one and too specific.

Intuitively, it might be clear that the zero cut-off property mentioned in the beginning of this subsection enables for a rate-monotonic scheduling without idle processor time which lifts it to the same performance as EDF scheduling under maintenance of its simplicity. This fact is expressed in the following theorem.

Theorem 2 (RMS on a simply periodic task set)
A system consisting of a single simply periodic task set is schedulable by RMS if and only if $u \leq 1$.

A proof sketch is given in [19], p. 129f.

3.3 ASPTSs: Sr and DCT

As the naming already suggests, an ASPTS has to be conform to both conditions (3.1 and 3.2) at the same time. Such a s ways exists to a given task set, but is not unique.

Lemma 2 (Accelerated simply periodic task set)
For an arbitrarily given task set $T=\{T_1, T_2,.., T_n\}$, there always exists an ASPTS. It is not unique.

This can be shown easily by construction. $T'=\{T'_1, T'_2,.., T'_n\}$ with $\forall i: p'_i = inf_i\{p_i\}$ fulfils the conditions since all periods are not increased and a common period means an integer ratio of 1 for all possible pairs.

A straightforward step is it to combine Theorems 1 and 2 to our main theorem:

Theorem 3 (RM schedulability of ASPTSs)
In order to show RM schedulability of a given task set T, it is sufficient to show that there is an accelerated simply periodic task set T' with $u' \leq 1$.

A proof is given in [9]. Theorem 3 as a sufficient condition offers a new way to prove RM schedulability as suggested by [9, 10]. It suffices to look for ASPTSs and to minimise the total utilization. The new test is then given by (12).

$$\exists T'=\{T'_1, T'_2,.., T'_n\}: \sum_{i=1}^{n} \frac{e'_i}{p'_i} \leq 1$$

subject to
$$\forall i: p'_i \leq p_i \qquad (12)$$
$$\forall i: e'_i = e_i$$

and (11)

Note that the test can be cancelled as soon as a transformed task set with a feasible utilization is found. If there is such a $u' \leq 1$, the original task set is schedulable by RM, if not we cannot make a claim on schedulability by RM[3]. For the purpose of finding candidate ASPTSs with low utilizations, two algorithms coined Sr and DCT[4] were proposed in [9, 10]. Sr is a specialization described as well in [19], p. 414ff. and has its origin in pinwheel scheduling [11]. While Sr accelerates to a task sets obeying the pattern $\{(a \cdot b^i, e_i)\}$ with $b=2$ (discussed in 3.2), DCT

---

[3] Of course, this can be checked by TDA or a scheduling simulation both of which are exact (necessary and sufficient) tests.

[4] Sr means Specialization with respect to r which is further explained in [10]. DCT probably means Distance-Constrained Tasks. Definitely, it has nothing to do with the Discrete Cosine Transform known from signal and image processing.

uses individual ratios according to $p'_j = p_i / \left\lceil \dfrac{p_i}{p_j} \right\rceil$

and $p'_i = \left\lfloor \dfrac{p_i}{p_j} \right\rfloor p_j$ for $p_i > p_j$ which are the locally optimal transformations.

This *greedy* approach will be discussed in 3.4. The computational complexity of tests based on these two algorithms is $O(n \log n)$ and $O(n^2)$. Sr uses a sorting[5] and precalculation of partial sums which enables for an iterative calculation of utilizations avoiding a double loop with complexity $O(n^2)$. This is further explained in [9] and [10]. For a better understanding of the two algorithms, see example in Table 2. The best results can be obtained using a combination of both algorithms. A task set is marked RM schedulable if one of both algorithms Sr and DCT is able to find a task set T' with $u' \leq 1$. The complexity of this approach, coined $Sr \vee DCT$, is $O(n^2)$.

3.4 Theoretical Performance of Sr and DCT

Theorem 4 (Theoretical Performance Sr/DCT)
A test based on Sr is better than LL and better than Bu since each task found to be RM schedulable by LL or by Bu can also be found to be schedulable by Sr [9, 10]. DCT is necessary and sufficient for the case of $n=2$ [9].

There are task sets not passing LL or Bu tests but Sr as will turn out by example in Section 4, so the first part describes a proper relationship. The second part of the theorem makes DCT equivalent to TDA in the case of $n=2$. Note that this results in another advantage over the compound algorithm Rate Monotonic General Tasks (RMGT) [3] which uses Rate Monotonic Small Tasks (RMST) [3] based on a simplification of Bu for small tasks and TDA for large tasks. Such a quite complicated heterogeneity in testing is no longer necessary by using DCT. It can be shown that DCT is equivalent to the necessary and sufficient test given as Theorem 32.9 in [7]. The condition for schedulability with $n=2$ and $p_1 \leq p_2$ is given there as

$$e_2 \leq \left\lfloor \frac{p_2}{P_1} \right\rfloor (p_1 - e_1) + max\left\{ 0, p_2 - \left\lfloor \frac{p_2}{P_1} \right\rfloor p_1 - e_1 \right\}$$

(13)

This interrelationship stresses the prime importance of the given approach. As an example, we test the schedulability of task sets of the kind $\{(5, e_1), (7, e_2)\}$. The possible pairs of e_1 and e_2 resulting in a schedulable task set can be visualised graphically, see Figure 1.

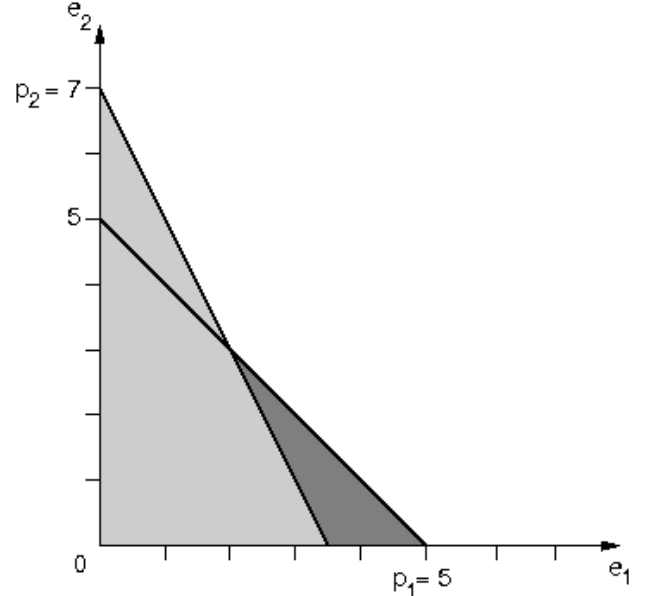5   Sorting is $O(n \log n)$.

Figure 1: Task sets of the kind $\{(5, e_1), (7, e_2)\}$ detectable by DCT (light and dark grey) and PS (only light grey) in the first quadrant of the $e_1 e_2$ plane

The filled region denotes the solution set. The linear function $e_2 = 5 - e_1$ represents the utilization limit if the second task is accelerated to a period of 5. Second, the first task could be accelerated to a period of 3.5 resulting in an acceleration limit of $e_2 = 7 - 2 e_1$. Since the existence of one ASPTS suffices according to Theorem 3, a union of both triangles gives the solution set. Note further that the PS bound (2) only considers $e_2 \leq 7 - 2 e_1$ as constraint omitting the other possibility $e_2 \leq 5 - e_1$ in the OR term. Thus, the dark grey region represents task sets detected schedulable by DCT but not by PS, characterizing DCT superior to PS. For $n=2$, DCT is always equal to or better than PS since the PS constraint is relaxed by an additional OR term.

Finally, we want to point out that DCT pursues a greedy strategy when constructing the accelerated task sets. It will turn out that this approach is (in the average case) superior to the more restricted one only using the simply periodic task pattern $\{(a \cdot b^i, e_i)\}$ introduced in 3.2 where, additionally, $b=2$, which corresponds to Sr. Anyway, there are rare cases where the greedy DCT is worse than Sr. An example is the task set $\{(2,1),(11,2),(17,4)\}$, see Table 2. DCT accelerates it to $\{(2,1),(10,2),(10,4)\}$, $\{(\frac{11}{6},1),(11,2),(11,4)\}$ and $\{(1.7,1),(8.5,2),(17,4)\}$, all of them *not* meeting the utilization bound of 1. Sr transforms the task set to $\{(2,1),(8,2),(16,4)\}$,

$\{(\frac{11}{8},1),(11,2),(11,4)\}$ and

$\{(\frac{17}{16},1),(\frac{17}{2},2),(17,4)\}$. Choosing the first task as pivot results in a utilization of exactly 1 . Thus, the original task set is RM schedulable[6]. In this particular case, it is better to accelerate T_2 more because the so enabled lower acceleration of T_3 results in a lower total utilization.

	original		Sr			DCT		
i	p_i	e_i	p'_i	p'_i	p'_i	p'_i	p'_i	p'_i
1	2	1	2	11/8	17/16	2	11/6	1.7
2	11	2	8	11	8.5	10	11	8.5
3	17	4	16	11	17	10	11	17
u	0.917	1	1.273	1.925		1.1	1.091	1.059

Table 2: A task set detected schedulable by Sr, but not by DCT; pivot periods are written in coloured background, utilizations are rounded to 3 decimal places

4. Case study

As a first test, we want to apply the suggested approach to a problem raised as exercise 9.3 in [19], p. 390. Ten asks with parameters given in Table 3 shall be scheduled on a multiprocessor system. The obvious question is for the minimal number of processors necessary. Following the sub-tasks given by [19], p. 390 both with FF-LL and RMST, a number of processors of $m=4$ is obtained. This is not the optimal number of $m=3$. Clearly, with a total utilization of $u\approx 2.47$, a schedule with only two processors is impossible. By running the heuristics, it turns out that only FF-TDA, FF-DCT, FF-PS, FF-CTS and FF-Bu[7] achieve the optimal number of 3. All other approaches find the sub-optimal solution of $m=4$. Here, the border line runs almost exactly between the $O(n)$ and the higher complexity methods. Only the more complex criteria and Bu are able to obtain the optimal value of $m=3$. This suggests strongly

---

6 We can also consider other bases than $b=2$ for the specialization. The average performance becomes worse [7], but in rare cases, they can be better. Note that, e.g., $\{(2,1),(20,2),(55,20)\}$ can be detected schedulable by Sr with $b=3$, but neither by Sr nor by DCT. So, consecutive tests with different bases can improve the result. On the other hand, the gain might be low and not worth the effort in most cases.

7 Note that FF-Bu is an improved version of RMST since it uses the Bu criterion in its original form instead of the simplified one used by RMST.

that it is worth not only to use linear complexity heuristics in MP scheduling.

i	1	2	3	4	5	6	7	8	9	10
p_i	7	21	29	49	64	66	160	235	260	450
e_i	2	3	9	15	20	16	32	72	25	120

Table 3: Parameters of 10 periodic tasks for multiprocessor scheduling

Further investigations shall consider the number of partitions leading to a schedulable configuration found by the different heuristics. The most general approach giving exactly the possible partitions and so the optimal number of processors, too, is to combine the exact methods of both scheduling elements, cf. 2.2. Thus, a brute-force search with TDA will be the reference. The number of cases to consider is the 10th Bell number which is 115975 , cf. 2.3. This is possible with a today's computer in a reasonable amount of time. Anyway, we want to limit the search space and then compare the performance of the different algorithms in the specified parts of the search space. Since we already know that the optimal value is $m=3$, this part close to the border of impossibility to schedule is the most interesting one. There are still $S(10,3)=9330$ possibilities, cf. 2.3. Among them, we want to consider three combinations of numbers of tasks assigned to each of the three processors: $4-3-3$, $4-4-2$ and $5-3-2$. They contain 2100 , 1575 and 2520 cases, respectively. In Table 4, a comparison of the different uniprocessor schedulability conditions in relationship to TDA as the reference is given. Performance has to be seen as sensitivity or true positive rate.

	$4-3-3$		$4-4-2$		$5-3-2$	
TDA	763		70		9	
$Sr\vee DCT$	470	61.6%	12	17.1%	0	0.0%
DCT	462	60.6%	11	15.7%	0	0.0%
Sr	268	35.1%	2	2.9%	0	0.0%
CTS	385	50.5%	22	31.4%	0	0.0%
PS	433	56.7%	17	24.3%	7	77.8%
HB	0	0.0%	0	0.0%	0	0.0%
Bu	2	0.3%	0	0.0%	0	0.0%
RBound	1	0.1%	0	0.0%	0	0.0%
LL	0	0.0%	0	0.0%	0	0.0%
LLconst	0	0.0%	0	0.0%	0	0.0%

Table 4: Performance of different uniprocessor schedulability conditions on a set of ten tasks

Clearly, the conditions $Sr \lor DCT$, DCT, Sr, CTS and PS outperform all other conditions significantly. Among them, the combination of Sr and DCT, which requires the highest effort, delivers the best performance in the most typical $4-3-3$ case. But the advantage is only marginal, so using DCT might be sufficient. In the exotic cases $4-4-2$ and $5-3-2$, CTS and PS outperform the suggested approaches. This is interesting, but should not be overvalued since the sample size is very small. Remarkable is the breakdown of almost all known only sufficient conditions in the particularly difficult case $5-3-2$. An exception is PS which detects 7 of 9 correct partitions. It can be seen as an outlier. Results in Section 5 will deliver more evidence.

5. Simulation with randomised task sets

In the following, we will present two simulation results, one for uniprocessor scheduling and one for multiprocessor scheduling. An important step in both cases is the generation of random task sets. The core of such a function is pseudo random number generation. In order to obtain a uniform distribution it is wrong to use a modulo operator since this operation restricts the result to only the lower value bits. Instead, scaling by division should be used.

5.1 Uniprocessor

Tasks are randomly given an integer execution time in $[1,10]$ (uniform). The period results from adding an integer uniform distribution in $[1,100]$ to the execution time. The target is $n=10$ and $u=0.70..0.96$. So, the total utilization is incremented in steps of 0.02. If the target utilization is reached or surpassed with less than 10 tasks this set will be omitted. Otherwise, the execution time of the 10th task is reduced in order to reach the respective target utilization exactly. $10,000$ task sets exactly meeting the requirements for n and u are checked for each u, where for $u=0.96$ only 100 task sets are checked. The reason is that it is much more unlikely that they are RM schedulable. The results can be seen in Figure 2.

We can see that higher utilization task sets are more difficult to recognise as schedulable. This is true for all investigated uniprocessor schedulability conditions. The slight increase from 0.94 to 0.96 is not significant since the sample size of 100, see above, is not convincing. Note that LL breaks down already right above $u=0.70$ since the limit $10(2^{1/10}-1)$ is between 0.70 and 0.72. All three newly proposed conditions are better than the established ones in all
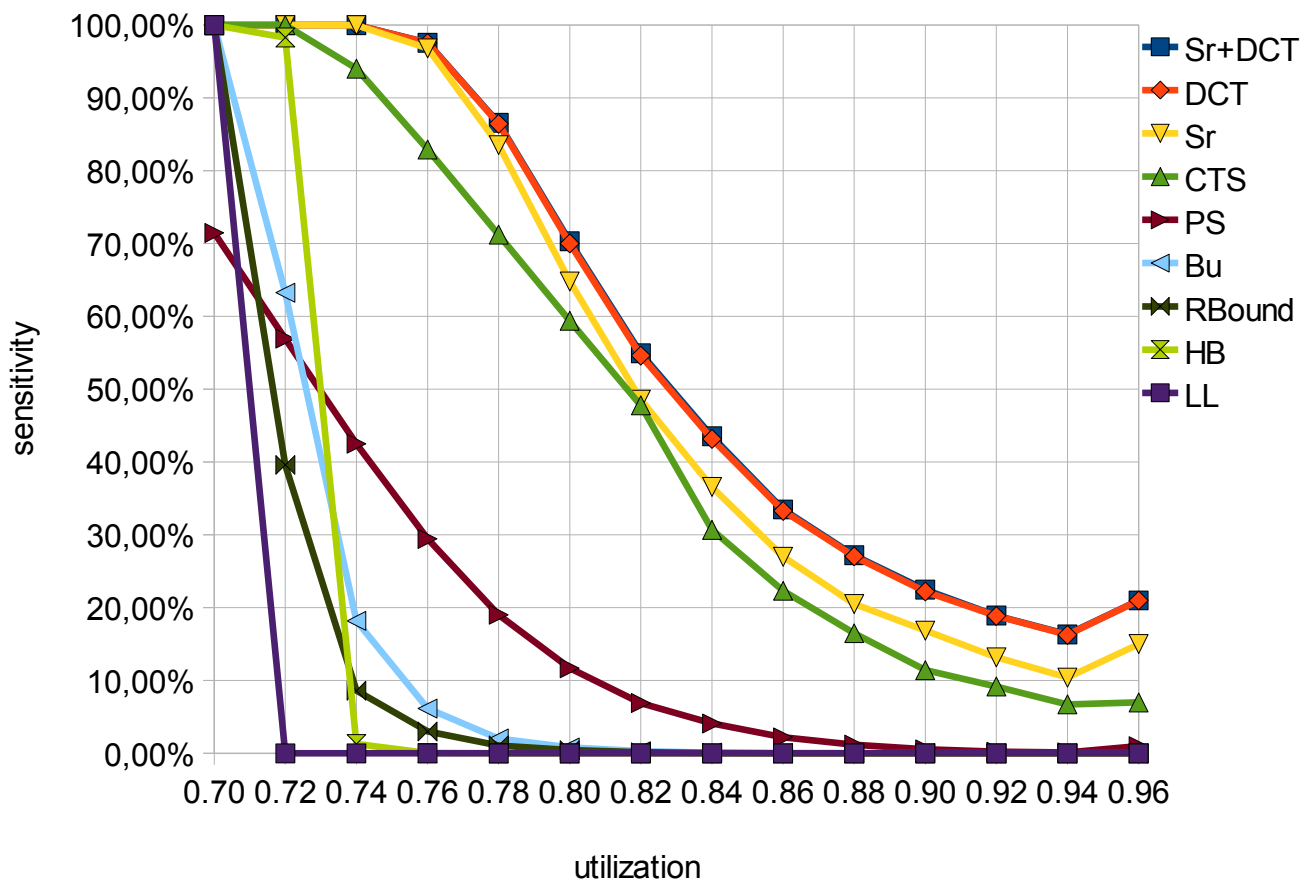

Figure 2: Performance of different uniprocessor schedulability conditions on randomised sets of ten tasks; 10,000 simulations per u value; Sr + DCT curve is only slightly above the DCT curve

cases. Best sensitivity delivers $Sr \vee DCT$. But the improvement compared to pure DCT is marginal and doesn't pay off the higher effort. So, DCT turns out to be the condition of choice. Note that both CTS and PS as the other conditions with $O(n^2)$ complexity besides DCT reach medium sensitivity. Anyway, a serious drawback of PS is its lack of a guarantee, cf. Table 1, compared to LL. So, for low utilizations PS is worse than the linear complexity LL condition. On the other hand, CTS includes a guarantee, but exhibits worse behaviour compared to DCT or Sr. Sr shows a good performance related to its computational complexity of $O(n \log n)$.

5.2 Multiprocessor

For the multiprocessor case, the target utilization is 2.5. In order to enable this, the period results here from adding an integer uniform distribution in $[1,30]$ to the execution time. For all criteria, FF was used as allocation algorithm. In the cases of Bu and RBound, a processing of the task set has been conducted before applying FF. For Bu, tasks are sorted by the S values, cf. 2.4.6, in non-decreasing order. For RBound, the procedure ScaleTaskSet, cf. 2.4.7, including a sorting of the transformed set with non-decreasing periods is applied. Such a preprocessing further improves the performance of the algorithms being opponents to the new ones, and, thus, makes the competition even harder.

The value m is the result of the minimization problem. $100,000$ task sets have been checked by each heuristic, respectively. Figure 3 shows the results.
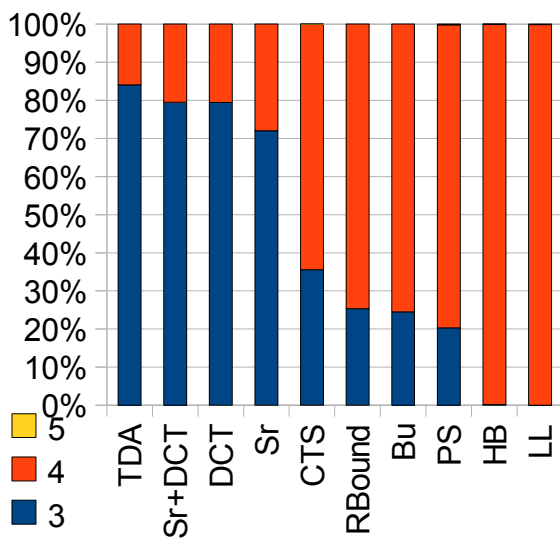

Figure 3: Results of the MP scheduling minimization problem (value of m) for randomised task sets with u = 2.5 and n = 10 using FF with different uniprocessor schedulability conditions; Bu and RBound apply a preprocessing concerning the order of the tasks

Note that the m value of 5 occurs so rarely that it is almost invisible in the bars CTS, PS, HB and LL. Clearly, again the three new heuristics based on the new criteria outperform the established ones significantly. While they peak, together with the exact uniprocessor schedulability condition, at $m = 3$, the other ones peak at $m = 4$. This means that they would waste an entire processor in most cases. Again, $Sr \vee DCT$ has no great advantage compared to DCT itself. Thus, we recommend to use DCT.

6. Related Work

Sr and DCT are underrepresented in literature. While the article by Zapata and Alvarez [22] not even mentions the great work of Han and Tyan, Lauzac *et al.* [16] shortly discuss their approach, but claim that "[..]SRFF is always between that of RMFF and RBound[..]" [16] and then no further consider it. Zapata *et al.* came in [23] to the result that DCT and Sr yield the best performance among all sufficient tests they investigated.

In [15], Kuo *et al.* claim to have found a polynomial-time schedulability test with higher performance and efficiency. The idea is to use the number of roots instead of the number of tasks in the LL formula. A task is called a root if there is no other process with a larger period that is an integer multiple of the original period. There are task sets for which their so called Root-Based test recognises schedulability where all other tests collapse. But the problem is that this is only possible for very artificial task sets as they have been used in their "simulations". Only in case of task sets with a single root the bound becomes 1.0. In all other cases, it is almost as bad as LL since the most typical number of roots is close to the number of tasks itself. E.g., for the task set given in Section 4, there are 9 roots. Thus, the Root-Based test with $O(n^2)$ complexity [15] is in the average case much worse than the proposed tests. An illustrative example is the task set $[\![(5,2),(7,4),(35,1)]\!]$. At first, it seems to be RM schedulable since there is one root and utilization is exactly one. The problem is that the precondition in Theorem 4 in [15] is not fulfilled. The algorithm has to be done in an iterative or recursive manner. In the example, the middle priority task doesn't meet its deadline. On the other hand, this property makes the approach suitable for on-line schedulability [15].

Similarly, Kuo and Mok [14] suggested a schedulability condition based on the number of fundamental frequencies. In the division graph, which is the *Hasse* diagram induced by the half order relationship $p_i | p_j$, it coincides with the number of fundamental chains. Again, this number is then put into the LL formula. An algorithm finding the number of fundamental frequencies with a complexity of $O(n^{5/2})$ is given in [14] by Kuo and Mok. Chen *et al.* gave an im-

provement to $O(n^2)$ complexity in [5] with the additional advantage that their *Algorithm 2* can obtain a smaller k value if a root is smaller than a leaf. The problem is that these algorithms result in a high sensitivity only for sets with just one fundamental frequency. Note that this means a simply periodic task set which corresponds to a very restricted part of the parameter space.

The common problem of the criteria based on the number of roots or the number of fundamental frequencies [14, 15] is that they do not transform the task set and so remain too specific. Thus, they are not able to reach good average sensitivity on random task sets. The argument of the authors that harmonic rates are typical e.g. for multimedia processes [15] trivialises the problem. A configuration with just one harmonic chain can be detected without any advanced algorithm in most cases.

As a consequence, it might be a good idea to combine the basic idea of the Root-Based test with a transformation to accelerated task sets. But it turns out that in case of assuming only one root, which is then the task with the longest period, such a "test" would not be sufficient. The reason is that deadline violations in RMS can not only occur in the lowest priority task. Instead, all tasks[8] have to be checked for possible deadline violations in the accelerated task set with only one root. This leads directly to PS, see (14). T_i corresponds to a root task. The denominator $p_i / \left\lceil \dfrac{p_i}{p_j} \right\rceil$ is the reduced period of the appropriate accelerated task T_j and is closely related to the term used in DCT for shorter period tasks. The difference is that in DCT, the shortening is applied iteratively while here, it is applied related to the root task. This becomes apparent by again considering the task set $\{(2,1),(11,2),(17,4)\}$ given in 3.4. It will be accelerated to $\{(17/9,1),(8.5,2),(17,4)\}$ with a utilization of exactly 1 and, thus, T_3 experiences no deadline violation. Here, T_2 is not distorted by T_1 since it can be accelerated to $\{(11/6,1),(11,2)\}$ with a utilization less than 1. Thus, not only Sr, see 3.4, but also PS is able to detect $\{(2,1),(11,2),(17,4)\}$ as RM schedulable.

So, PS is closely related to DCT which highlights the central role of ASPTSs.

$$\forall i : e_i + \sum_{j:Prio_j > Prio_i} \frac{e_j}{p_i} \leq 1 \Leftrightarrow$$

$$\left\lceil \frac{p_i}{p_j} \right\rceil \quad (14)$$

$$\forall i : e_i + \sum_{j:Prio_j > Prio_i} \left\lceil \frac{p_i}{p_j} \right\rceil e_j \leq p_i = d_i$$

The so called *Algorithm 3* by Chen *et al.* in [5] with a complexity of $O(n^3)$ is an improvement of CTS. As a preprocessing, periods which are integer divisors of other periods and other ones fulfilling an advanced condition are eliminated. According to [23], the performance of this algorithm is in the range of DCT and Sr, but it can outperform them only in case of small maximum task utilizations $\alpha \leq 0.5$. A serious drawback is the high complexity. Often, a TDA will be preferable.

Similar to CTS, Park *et al.* [20] gave a linear programming (LP) approach in order to obtain a task set specific utilization bound, too. A total utilization below the LP bound means that at least one of the constraints must be violated. Hence, for a single decision on schedulability, it is sufficient to test all given constraints. A sole violation entails schedulability of the task set. By not finding such a violation, nothing can be said on feasibility. The criterion is sufficient but not necessary. The approach is closely related to PS. The set of constraints could be reduced by Hu and Quan [12]. They claim that their best algorithm
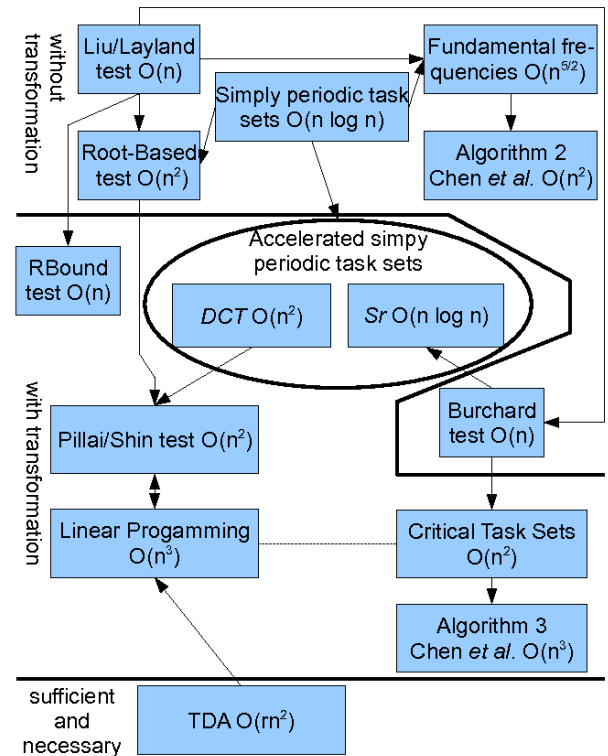

Figure 4: Relationships between different uniprocessor scheduling criteria

---

8   The only exception is the highest priority task which always will be scheduled at the beginning of its period.

can outperform DCT for larger values of n. But they have to admit that DCT is much better in terms of running time, ca. one order of magnitude in their tests. Their constraints correspond to a generalization of PS. Instead of only checking with the period p_i of the incremental lowest priority task, accelerated periods $\left\lfloor \dfrac{p_i}{p_j} \right\rfloor p_j$ are tested, too. Finally, all other tasks are accelerated related to the accelerated lowest priority task. The idea of the Root-Based test is exploited as done in PS. Since PS is $O(n^2)$ and the extension is an additional loop over tasks, this approach has a computational complexity of $O(n^3)$, and is, thus, not a direct competitor of DCT or Sr. E.g., the schedulability of the task set $\{(2,1),(11,2),(17,4)\}$ given in 3.4 can be detected with their LP approach. It accelerates the set to $\{(2,1),(8,2),(16,4)\}$ as Sr does. Thus, the approach seems to be conceptually interesting and reaches a very good performance at the price of much higher complexity and running times than Sr/DCT. In the case of $n=2$, it turns out to be equivalent to DCT and, thus, to be a necessary and sufficient criterion. LP can be seen as filling a gap between PS and TDA. Summing up, see Figure 4.

7. Conclusions

We came to the result that DCT is the best known $O(n^2)$ sufficient criterion for RM schedulability on a uniprocessor. Our simulations suggest that multiprocessor scheduling heuristics based on Sr or DCT outperform all other heuristics under investigation. With a computational complexity of $O(n \log n)$ and $O(n^2)$, the criteria are less complex than TDA which is pseudo-polynomial. On the other hand, they are more complex than classical criteria with linear complexity. A serious competitor is the Critical Task Set test. Anyway, it turned out to be worse than Sr and worse than DCT. The Pillai/Shin test with a complexity of $O(n^2)$ like DCT provides no guarantees at all and exhibits in most cases a worse sensitivity than DCT. In a concrete run-time comparison, DCT is often faster than the linear complexity criteria RBound or the Burchard criterion since they include several loop cycles, making the constant belonging to $O(n)$ high, and use expensive operations like power and logarithm extensively. This is also true for Sr. Thorough investigation of runtime behaviour could be dealt with in future research.

Interrelationships between ASPTSs and the concepts of the Root-Based test and the Pillai/Shin test have been pointed out. The connection between the LP approach of Hu and Quan, DCT, and the Root-Based test has been given explicitly. An extensive study of related work has shown that a lot of approaches are closely related to ASPTSs.

DCT delivers an outstanding performance at a reasonable practical run-time. Additionally, DCT provides a necessary and sufficient criterion for task sets of size $n=2$ making it as powerful as TDA in this particular case. Last but not least, the principle behind DCT, taking one period as a pivot and constructing a harmonic set by shortening (if necessary) all other periods, is well understandable and can be applied even by humans and not only by computers for smaller values of n. Its elegance strongly suggests a higher influence on teaching.

A further improvement can be achieved by combining different tests. Here, $Sr \lor DCT$ lead to only very little´gain, not paying off the increased effort. The combinations $PS \lor DCT$ (similar to the LP approach by Hu and Quan [12]) and $CTS \lor DCT$ might be more promising since the concepts behind are less similar. This could be a matter of future research.

A combination of the new tests with the FF policy leads to new powerful MP scheduling heuristics. They can reach results close to the optimum. This suggests that the algorithms Sr and DCT are still better suitable for multiprocessor scheduling.

8. Acknowledgement

9. References

[1]   N. Audsley, A. Burns, M. Richardson, K. Tindell, and A.J. Wellings. Applying new scheduling theory to static priority pre-emptive scheduling. Software Engineering Journal, 8(5):284–292, Sep 1993.

[2]   Enrico Bini, Giorgio Buttazzo, and Giuseppe Buttazzo. A hyperbolic bound for the rate monotonic algorithm. In ECRTS '01: Proceedings of the 13th Euromicro Conference on Real-Time Systems, pages 59–66, Washington, DC, USA, 2001. IEEE Computer Society.

[3]   A. Burchard, J. Liebeherr, Yingfeng Oh, and S.H. Son. New strategies for assigning real-time tasks to multiprocessor systems. Computers, IEEE Transactions on, 44(12):1429–1442, Dec 1995.

[4]   John Carpenter, Shelby Funk, Philip Holman, James Anderson, and Sanjoy Baruah. A categorization of real-time multiprocessor scheduling problems and algorithms. In Handbook on Scheduling Algorithms, Methods, and Models, pages 30–1–30–19. Chapman Hall/CRC, Boca, 2004.

[5]   Deji Chen, Aloysius K. Mok, and Tei-Wei Kuo. Utilization bound revisited. IEEE Transactions on Computers, 52(3):351–361, 2003.

[6]   R. Devillers and J. Goossens. Liu and layland's schedulability test revisited. Information Processing Letters, 73(5-6):157–161, March 2000.

[7]   Sudarshan K. Dhall. Approximation algorithms for scheduling time-critical jobs on multiprocessor systems. In Handbook on Scheduling Algorithms,

Methods, and Models, pages 32–1–32–30. Chapman Hall/CRC, Boca, 2004.

[8] Ronald L. Graham, Donald E. Knuth, and Oren Patashnik. Concrete Mathematics: A Foundation for Computer Science. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1994.

[9] C.-C. Han and H.-Y. Tyan. A better polynomialtime schedulability test for real-time fixed-priority scheduling algorithms. Real-Time Systems Symposium, IEEE International, 0:36–45, 1997.

[10] Ching-Chih Han, Kwei-Jay Lin, and Chao-Ju Hou. Distance-constrained scheduling and its applications to real-time systems. Computers, IEEE Transactions on, 45(7):814–826, Jul 1996.

[11] R. Holte, A. Mok, L. Rosier, I. Tulchinsky, and D. Varvel. The pinwheel: a real-time scheduling problem. System Sciences, 1989. Vol.II: Software Track, Proceedings of the Twenty-Second Annual Hawaii International Conference on, 2:693–702 vol.2, Jan 1989.

[12] Xiaobo (Sharon) Hu and Gang Quan. Fast performance prediction for periodic task systems. Hardware/Software Co-Design, International Workshop on, 0:72, 2000.

[13] C. M. Krishna and K. G. Shin. Real-Time Systems. McGraw-Hill, 1997.

[14] T.-W. Kuo and A.K. Mok. Load adjustment in adaptive real-time systems. Real-Time Systems Symposium, 1991. Proceedings., Twelfth, pages 160–170, Dec 1991.

[15] Tei-Wei Kuo, Li-Pin Chang, Yu-Hua Liu, and Kwei-Jay Lin. Efficient online schedulability tests for real-time systems. IEEE Trans. Softw. Eng., 29(8):734–751, 2003.

[16] Sylvain Lauzac, Rami Melhem, and Daniel Mossé. An improved rate-monotonic admission control and its applications. IEEE Trans. Comput., 52(3):337–350, 2003.

[17] John P. Lehoczky, Lui Sha, and Y. Ding. The rate monotonic scheduling algorithm: Exact characterization and average case behavior. In RTSS, pages 166–171, 1989.

[18] C. L. Liu and James W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. J. ACM, 20(1):46–61, January 1973.

[19] Jane W. S. Liu. Real-Time Systems. Prentice Hall, 2000.

[20] Dong-Won Park, S. Natarajan, A. Kanevsky, and Myung Jun Kim. A generalized utilization bound test for fixed-priority real-time scheduling. Real-Time Computing Systems and Applications, International Workshop on, 0:73, 1995.

[21] Padmanabhan Pillai and Kang G. Shin. Realtime dynamic voltage scaling for low-power embedded operating systems. SIGOPS Oper. Syst. Rev., 35(5):89–102, 2001.

[22] Omar U. Pereira Zapata and Pedro Mejia Alvarez. EDF and RM multiprocessor scheduling algorithms: Survey and performance evaluation. Technical report, Departamento de Computación, CINVESTA-VIPN, Mexico, 2005.

[23] Omar U. Pereira Zapata, Pedro Mejia Alvarez, and Luis E. Leyva del Foyo. Comparative analysis of real-time scheduling algorithms on one processor under rate monotonic. Technical report, Departamento de Computación, CINVESTAV-IPN, Mexico, 2005.

[24] Cho, H.; Ravindran, B. & Jensen, E. D. An Optimal Real-Time Scheduling Algorithm for Multiprocessors, Real-Time Systems Symposium, IEEE International, 0: 101-110, 2006

[25] Baker, T. P. Comparison of empirical success rates of global vs. partitioned fixed-priority and EDF scheduling for hard real time, Technical Report, Florida State University, 2005

[26] Kato, S.; Yamasaki, N. & Ishikawa, Y. Semi-partitioned Scheduling of Sporadic Task Systems on Multiprocessors ECRTS '09: Proc. of the 2009 21st Euromicro Conference on Real-Time Systems, IEEE Computer Society, 249-258, 2009

10. Glossary

ASPTS: Accelerated Simply Periodic Task Set

BF: Best Fit

Bu: Burchard (criterion)

CTS: Critical Task Sets

DCT: Distance-Constrained Tasks

FF: First Fit

HB: Hyperbolic Bound

LL: Liu/Layland (criterion)

LLconst: Liu/Layland limit criterion

LP Linear Programming

MP: Multiprocessor

NF: Next Fit

PS: Pillai/Shin (criterion)

RBound RBound criterion

RM(S): Rate-monotonic (scheduling)

RMFF Rate Monotonic First Fit

RMGT Rate Monotonic General Tasks

RMST Rate Monotonic Small Tasks

Sr: Specialization with respect to r

TDA: Time Demand Analysis

WCET: Worst Case Execution Time

WF: Worst Fit