A $3/2$-Approximation Algorithm for Rate-Monotonic Multiprocessor Scheduling of Implicit-Deadline Tasks

Andreas Karrenbauer[1] and Thomas Rothvoß[2]

[1] Zukunftskolleg, University of Konstanz, Germany
andreas.karrenbauer@uni-konstanz.de
[2] Institute of Mathematics, École Polytechnique Fédérale de Lausanne, Lausanne, Switzerland
thomas.rothvoss@epfl.ch

Abstract We present a new approximation algorithm for rate-monotonic multiprocessor scheduling of periodic tasks with implicit deadlines. We prove that for an arbitrary parameter $k \in \mathbb{N}$ it yields solutions with at most $(\frac{3}{2} + \frac{1}{k})OPT + 9k$ many processors, thus it gives an asymptotic $3/2$-approximation algorithm. This improves over the previously best known ratio of $7/4$. Our algorithm can be implemented to run in time $O(n^2)$, where n is the number of tasks. It is based on custom-tailored weights for the tasks such that a greedy maximal matching and subsequent partitioning by a first-fit strategy yields the result.

1 Introduction

In this paper, we consider the *synchronous rate-monotonic real-time scheduling problem with implicit deadlines*. That is, we are given a set of n tasks $V := \{\tau_1, \dots, \tau_n\}$ attributed with *execution times* $c(\tau_i)$ and *periods* $p(\tau_i)$. Each task releases a job at time 0 and subsequently at each integer multiple of its period (hence *synchronous*). Furthermore, each job of a task has to be finished before the next job of the same task is released. In other words the relative deadlines of jobs are *implicitly* given by the periods. We allow *preemption*, but we require *fixed priorities* to reduce the overhead during runtime. That is, the current job is preempted, if a new job with a higher priority is released. In this context, Liu and Layland [1] have shown that if there are feasible fixed priorities then *rate-monotonic* priorities, which are higher for smaller periods, also define a feasible schedule. See the book of Buttazzo [2] for a comparison of fixed-priority versus dynamic-priority scheduling policies.

Since multi-core and multi-processor environments become more and more popular, we consider the problem of assigning the tasks to a minimum number of processors such that there is a feasible rate-monotonic schedule for each processor. Formally

> Given tasks $V = \{\tau_1, \dots, \tau_n\}$, running times $c : V \to \mathbb{Q}_+$, and periods $p : V \to \mathbb{Q}_+$, where each task τ generates a job of length $c(\tau) \leq p(\tau)$ and relative deadline $p(\tau)$ at $z \cdot p(\tau)$, for all $z \in \mathbb{Z}_{\geq 0}$, find the minimum ℓ such that there is a partition of $V = P_1 \dot\cup \dots \dot\cup P_\ell$ subject to RM-schedulability of P_j for each j.

Here we *forbid migration*, i.e. jobs of the same task must always be processed on the same machine. This scheduling problem has received considerable attention in the real-time and embedded-systems community. This popularity is due to the fact that more and more safety-critical control applications are carried out by microprocessors and in particular by multiprocessor environments. Such scheduling problems are today a major algorithmic challenge in the automotive and aviation industry.

The idea for our algorithm is as follows: Suppose all tasks had utilization larger than $\frac{1}{3}$. Then at most 2 tasks can be assigned to each processor. Define an undirected graph $G = (V, E)$ with the tasks being the nodes and an edge $\{\tau_1, \tau_2\}$ for each pair such that τ_1 and τ_2 can be RM-scheduled on a single processor. Then the size of a maximum matching plus the number of nodes not covered by that matching gives OPT. We incorporate the existence of small tasks by only including an edge $\{\tau_1, \tau_2\} \in E$ if $w(\tau_1)+w(\tau_2)$ exceeds a certain threshold. Here $w(\tau)$ is a proper weight function which is monotonically increasing with the *utilization* $u(\tau) = \frac{c(\tau)}{p(\tau)}$.

1.1 Related work

The famous *Bin Packing* problem is an important special case of our scheduling problem. The objective of Bin Packing is to find a partition of a set of items of different sizes, say $u_i \in (0, 1]$ for $i = 1, \ldots, n$, into a minimum number of bins such that the total size of each bin does not exceed 1. The similarity to our scheduling problem becomes apparent by introducing the notion of the utilization of a task, i.e. $u(\tau) = c(\tau)/p(\tau)$. If all periods are the same, e.g. the common denominator of rational item sizes, then the priorities for the rate-monotonic scheduling problem become irrelevant and a set of tasks is feasible for one processor, if and only if their total utilization does not exceed 1.

Successful heuristics for Bin Packing are *First Fit*, *Next Fit* and *Best Fit*. In all variants the items are assigned in a consecutive manner to a bin, which has enough space (or a new one is opened). For First Fit the current item is assigned to the bin with the smallest index, in Best Fit it is assigned to the bin, whose item sum is maximal. For Next Fit an active bin is maintained. If the current item does not fit into it, a new bin is opened, now being the active one; old bins are never considered again. In *First Fit Decreasing* the items are first sorted by decreasing sizes and then distributed via First Fit. In the worst-case Next Fit produces a 2-approximation, while First Fit needs $\lceil \frac{17}{10} OPT_{\text{BinPacking}} \rceil + 1$ many bins [3]. Asymptotically, Best and First Fit Decreasing have an approximation ratio of $11/9$ [4]. Furthermore, there is an asymptotic PTAS [5] and even an asymptotic FPTAS exists [6]. More on Bin Packing can be found in the excellent survey of Coffman et al. [7].

The *utilization* of a task set V' is defined as $u(V') = \sum_{\tau \in V'} c(\tau)/p(\tau)$. If V' is feasible (i.e. RM-schedulable on a single machine), then the utilization $u(V')$ is at most 1. However, V' can be infeasible, even if $u(V') < 1$. Liu and Layland [1] have shown that V' is feasible, if $u(V')$ is bounded by $n'(2^{1/n'} - 1)$, where $n' = |V'|$. This bound tends to $\ln(2)$ and the condition is not necessary for feasibility, as the example with equal periods shows. Stronger, but still not necessary conditions for feasibility are given in [8,9,10].

The *response time* of a job is the difference of release time and completion time. The response time of a task is defined as the maximal response time of any of its jobs.

In our synchronous setting, this value is attained for the first job (which is released at time 0), see [1].

If $p(\tau_1) \leq \ldots \leq p(\tau_n)$ then the response time for τ_i in a rate-monotonic, uniprocessor schedule is given by the smallest value $r(\tau_i) \geq 0$ with

$$r(\tau_i) = c(\tau_i) + \sum_{j<i} \left\lceil \frac{r(\tau_i)}{p(\tau_j)} \right\rceil c(\tau_j).$$

Of course $\tau_1, \ldots, \tau_n$ are feasible if and only if $r(\tau_i) \leq p(\tau_i)$ for $i = 1, \ldots, n$. But it was proved in [11] that such response times cannot even be approximated in polynomial time within a constant factor, unless $\mathbf{NP} = \mathbf{P}$. Nevertheless in practice response times can be efficiently computed using a fix-point iteration approach [12]. Furthermore Baruah and Fisher [13] showed that there is an FPTAS for computing the minimum processor speed, which is needed to make a task system RM-schedulable. However, the complexity status of verifying, whether the RM-schedule of a set of implicit deadline tasks on a single machine is feasible, remains an open problem [14]. Fortunately for $n = 2$ there is an simple exact criterion (cf. [15], chapter 32): The task set $\{\tau_1, \tau_2\}$ with $p(\tau_1) \leq p(\tau_2)$ is RM-schedulable if and only if

$$c(\tau_2) \leq \left\lfloor \frac{p(\tau_2)}{p(\tau_1)} \right\rfloor (p(\tau_1) - c(\tau_1)) + \max\left\{0, p(\tau_2) - \left\lfloor \frac{p(\tau_2)}{p(\tau_1)} \right\rfloor p(\tau_1) - c(\tau_1)\right\}. \quad (1)$$

This constant time test will be used in our algorithm.

Most popular algorithms for rate-monotonic multiprocessor scheduling first sort the tasks in a suitable way and then distribute them in a First Fit or Next Fit manner using a sufficient feasibility criterion. See the following table for an overview (with our algorithm in the last row, for the sake of comparability).

algorithm	**references**	**sorting**	**distribution**	**ratio**	**time**
RMNF	[16,17]	inc. $p(\tau)$	Next Fit	2.67	$O(n \log n)$
RMFF	[16,17]	inc. $p(\tau)$	First Fit	2.00	$O(n \log n)$
RRM-FF	[18]	-	First Fit	2.00	$O(n \log n)$
RRM-BF	[18]	inc. $p(\tau)$	Best Fit	2.00	$O(n \log n)$
FFDU	[17]	dec. $u(\tau)$	First Fit	2.00	$O(n \log n)$
RMST	[8]	inc. $S(\tau)$	Next Fit	$\frac{1}{1-\alpha}$	$O(n \log n)$
RMGT	[8]	-	First Fit + RMST	1.75	$O(n^2)$
FFMP	[19]	inc. $S(\tau)$	First Fit	2.00	$O(n \log n)$
k-RMM	-	-	Matching + FFMP	1.50	$O(n^2)$

Here $S(\tau) = \log_2 p(\tau) - \lfloor \log_2 p(\tau) \rfloor$ and $\alpha = \max_{\tau \in V} u(\tau)$. In the table, column "ratio" denotes the best known upper bounds on the asymptotic approximation ratio. The *Rate-monotonic general task* algorithm [8] distributes tasks with utilization at most $1/3$ using RMST and the rest separately with First Fit. Also the algorithms RRM-FF and RRM-BF apply the same grouping strategy. A more detailed description can be found in [17].

Furthermore there is an asymptotic PTAS under resource augmentation, computing for any fixed $\varepsilon > 0$ a solution with $(1 + \varepsilon)OPT + O(1)$ processors, where the tasks on each processor can be feasibly scheduled after increasing the processor speed by a factor of $1 + \varepsilon$ [20]. In the same paper it was proved that unless $\mathbf{P} \neq \mathbf{NP}$, no asymptotic FPTAS can exist for this multiprocessor scheduling problem. But it is still an open question whether an asymptotic PTAS is possible. We refer to the article [21] for an overview on complexity issues of real-time scheduling.

1.2 Our Contribution

We present a new polynomial time algorithm for rate-monotonic real-time scheduling, which is based on matching techniques and yields solutions of at most $(\frac{3}{2}+\frac{1}{k})OPT+9k$ many processors. The asymptotic approximation ratio tends to $3/2$ (for growing k), improving over the previously best known value of $7/4$. Moreover, we provide experimental evidence that our new algorithm outperforms all other existing algorithms.

2 Preliminaries

During our algorithm it will happen, that we discard a set of (in general small) tasks $V' \subseteq V$ and schedule them using a simple heuristic termed *First Fit Matching Periods* (FFMP), which was introduced in [19]. For a task τ define

$$S(\tau) := \log_2 p(\tau) - \lfloor \log_2 p(\tau) \rfloor \qquad \text{and} \qquad \beta(V) := \max_{\tau \in V} S(\tau) - \min_{\tau \in V} S(\tau)$$

then the FFMP heuristic can be stated as follows

Algorithm 1 FFMP

```
(1) Sort tasks such that $0 \leq S(\tau_i) \leq \ldots \leq S(\tau_n) < 1$
(2) FOR $i = 1, \ldots, n$ DO
    (3) Assign $\tau_i$ to the processor $P_j$ with the least index $j$ such that
            $u(P_j \cup \{\tau_i\}) \leq 1 - \beta(P_j \cup \{\tau_i\}) \cdot \ln(2)$
```

The idea for this ordering of the tasks is that consecutive tasks will have periods that are nearly multiples of each other and hence the bin packing aspect of the problem becomes dominant. Let FFMP(V) denote the value of the solution, which FFMP produces, if applied to V. One can prove the following lemma using well known techniques from [8] (see also [15]).

Lemma 1. *Given periodic tasks $V = \{\tau_1, \ldots, \tau_n\}$ and $k \in \mathbb{N}$. FFMP always produces feasible solutions such that*

- *If $u(\tau_i) \leq \alpha \leq \frac{1}{2}$ for all $i = 1, \ldots, n$, then FFMP$(V) \leq \frac{1}{1-\alpha} u(V) + 3$.*
- *If $u(\tau_i) \leq \frac{1}{2} - \frac{1}{k}$ for all $i = 1, \ldots, n$, then FFMP$(V) \leq \frac{n}{2} + \frac{k}{2}$.*

The RMST algorithm of Liebeherr et al. [8] also fulfills the same properties. But on average the First Fit distribution for FFMP behaves much better than the Next Fit distribution of RMST. However just for a worst-case analysis one could replace FFMP by RMST.

3 Matchings and Schedules

As a powerful tool, we will use matchings in our algorithm. To this end, we define an undirected graph $G = (V, E)$ such that the nodes correspond to the tasks. If there is an edge between the nodes τ_1 and τ_2, then the corresponding tasks can be scheduled on one processor. Suppose for the time being that all tasks have a utilization of more than $\frac{1}{3}$ and thus at most two tasks fit on one processor. Then the maximum cardinality matching in G determines a schedule with a minimum number of processors by reserving one processor for each edge in the matching and one processor for each unmatched node.

For the general setting of tasks with arbitrary utilization, this basic idea for our algorithm persists: Compute a matching in G, schedule each pair of matched tasks together on one processor, and distribute the remaining tasks by FFMP. Of course, the matching should be in such a way that we use the processors efficiently. To this end, we will assign weights to the nodes depending on the utilization of the corresponding tasks. We will later define the weights exactly. For now, let the weights be a function $w : V \to [0, 1]$ and let the *price* of a matching $M \subseteq E$ be

$$price(M) := |M| + w(\overline{M}),$$

where $\overline{M} := \{v \in V \mid \forall e \in M : v \notin e\} \subseteq V$ is the set of unmatched nodes and $w(\overline{M}) := \sum_{v \in \overline{M}} w(v)$. That is, we have to allocate 1 processor for each matched pair of tasks and also some more processors for distributing the remaining unmatched tasks. Note that finding the matching with minimum price is equivalent to computing the maximum weight matching with edge weights $w(e) := w(u) + w(v) - 1$ for each edge $e = \{u, v\}$, since

$$w(M) := \sum_{e \in M} w(e) = \sum_{v \in V} w(v) - \sum_{v \in \overline{M}} w(v) - |M| = w(V) - price(M).$$

While a maximum weight matching in a graph with n nodes and m edges can be found in $O(n(m + n \log n))$ [22], we will see that it is sufficient for our purpose to compute an inclusion-wise maximal matching greedily. That is, we maintain the property, that for all $e \in E \setminus M$ we have $w(e') \geq w(e)$ for all $e' \in M$ or there is an edge $e' \in M$ with $e \cap e' \neq \emptyset$ and $w(e') \geq w(e)$, throughout the algorithm. Furthermore, the algorithm iterates until $\overline{M}$ does not contain an edge, i.e. $|e \cap \overline{M}| < 1$ for all $e \in E$. Note that such a *greedy maximal matching* can be computed in $O(n^2)$ by sorting the tasks by decreasing weight and searching for each task τ_i the first unmatched τ_j with $\{\tau_i, \tau_j\} \in E$. Although we do not have an explicit representation of the edges, the check whether a pair of nodes forms an edge takes only constant time. The interested reader is pointed to [23] or [24] for an extensive account on matchings.

4 The Algorithm

As indicated in the previous section, we compute a weighted matching to find a good schedule. It remains to define the weights properly. Note that each edge yields a processor in the partition. Hence, we do not want to match two nodes which do not use the processor to some extent. Moreover, each unmatched node is first discarded and later scheduled via FFMP. We are now going to define node weights w in such a way, that a matching with costs γ can be turned into a feasible schedule of roughly γ many processors. Intuitively, the weight $w(\tau) \in [0,1]$ will denote the average number of processors per task, which the FFMP algorithm needs to schedule a large number of tasks, if all tasks have the same utilization as τ. Here we distinguish 3 categories of tasks:

- *Small tasks* ($0 \leq u(\tau) \leq \frac{1}{3}$): Consider tasks $\tau_1, \ldots, \tau_m$ with a small utilization, i.e. $u(\tau_i) \leq \alpha$ for all $i = 1, \ldots, m$ and $\alpha \leq 1/3$. Then we may schedule such tasks with FFMP using $u(\{\tau_1, \ldots, \tau_m\})\frac{1}{1-\alpha} + 3 \leq m \cdot \alpha \frac{1}{1-\alpha} + 3$ many processors (see Lemma 1), thus we choose $w(\tau) := \frac{u(\tau)}{1-u(\tau)}$ for a small task τ.
- *Medium tasks* ($\frac{1}{3} < u(\tau) \leq \frac{1}{2} - \frac{1}{12k}$): Suppose we have tasks $\tau_1, \ldots, \tau_m$ whose utilization is at least $1/3$, but bounded away from $1/2$, say $u(\tau_i) \leq \frac{1}{2} - \frac{1}{12k}$, where k is an integer parameter that we determine later. Then FFMP$(\{\tau_1, \ldots, \tau_m\}) \leq m/2 + O(k)$ (see again Lemma 1), thus we choose $w(\tau) := 1/2$ for medium tasks.
- *Large tasks* ($u(\tau) > \frac{1}{2} - \frac{1}{12k}$): For a large task one processor is sufficient and possibly needed, thus $w(\tau) := 1$ in this case.

Algorithm 2 k-Rate-Monotonic-Matching algorithm (k-RMM)

```
(1) Construct $G = (V, E)$ with edges $e = \{\tau_1, \tau_2\} \in E \Leftrightarrow \{\tau_1, \tau_2\}$ RM-schedulable (according to condition (1)) and $w(e) > 0$.
(2) Sort the edges by decreasing weight (ties are broken arbitrarily) and compute the greedy maximal matching $M$ w.r.t. this order.
(3) For all $\{\tau_1, \tau_2\} \in M$ create a processor with $\{\tau_1, \tau_2\}$
(4) Define
      – $V_i = \{\tau \in \overline{M} : \frac{1}{3} \cdot \frac{i-1}{k} \leq u(\tau) < \frac{1}{3} \cdot \frac{i}{k}\}\ \forall i = 1, \ldots, k$
      – $V_{k+1} = \{\tau \in \overline{M} : \frac{1}{3} \leq u(\tau) \leq \frac{1}{2} - \frac{1}{12k}\}$
      – $V_{k+2} = \{\tau \in \overline{M} : u(\tau) > \frac{1}{2} - \frac{1}{12k}\}$
(5) Distribute $V_{k+2}, V_{k+1}, \ldots, V_1$ via FFMP.
```

The reason to define the weights in this way becomes clear with the proof of the following Theorem, saying that the number of used machines is essentially determined by the price of the matching.

Theorem 1. *Let M be an arbitrary matching in G. The schedule created from M as described in Algorithm 2, uses at most*

$$\left(1 + \frac{1}{2k}\right) \cdot price(M) + 9k$$

many processors.

Proof. We create $|M|$ processors, covering pairs of tasks $\{\tau_1, \tau_2\} \in M$. For scheduling the tasks in V_{k+1} we know that according to Lemma 1

$$\text{FFMP}(V_{k+1}) \leq \frac{|V_{k+1}|}{2} + \frac{12k}{2} = \sum_{\tau \in V_{k+1}} w(\tau) + 6k$$

using that the utilization of all tasks in V_{k+1} lies between $\frac{1}{3}$ and $\frac{1}{2} - \frac{1}{12k}$. Of course FFMP$(V_{k+2}) \leq |V_{k+2}| = \sum_{\tau \in V_{k+2}} w(\tau)$. For each V_i $(i = 1, \ldots, k)$ we know that the utilization of each task is sandwiched by $\frac{1}{3} \cdot \frac{i-1}{k}$ and $\frac{1}{3} \cdot \frac{i}{k}$. Consequently

$$\text{FFMP}(V_i) \leq \frac{1}{1-\frac{i}{3k}} \cdot u(V_i) + 3 \leq \Big(1 + \frac{1}{2k}\Big) \frac{1}{1-\frac{i-1}{3k}} \cdot u(V_i) + 3 \leq \Big(1 + \frac{1}{2k}\Big) w(V_i) + 3$$

by applying again Lemma 1 together with the fact that $w(\tau) \geq u(\tau) \cdot \frac{1}{1-(i-1)/(3k)}$ for all $\tau \in V_i$. We conclude that the total number of processors in the produced solution is

$$\begin{aligned} |M| + \sum_{i=1}^{k+2} \text{FFMP}(V_i) &\leq |M| + \sum_{\tau \in V_{k+1} \cup V_{k+2}} w(\tau) + 6k \\ &\quad + \Big(1 + \frac{1}{2k}\Big) \sum_{\tau \in V_1 \cup \ldots \cup V_k} w(\tau) + 3k \\ &\leq |M| + \Big(1 + \frac{1}{2k}\Big) \sum_{\tau \in V_1 \cup \ldots \cup V_{k+2}} w(\tau) + 9k \\ &\leq \Big(1 + \frac{1}{2k}\Big) \cdot \mathit{price}(M) + 9k. \end{aligned}$$

□

It remains to show that the price of the matching computed by Algorithm 2 is at most roughly $\frac{3}{2}$ times the number of necessary processors. To this end, we first show that for any partition, there is a matching with the appropriate price.

Theorem 2. *For any feasible partition $\mathcal{P} = \{P_1, \ldots, P_\ell\}$ of the tasks, there is a matching $M_{\mathcal{P}}$ with*

$$\mathit{price}(M_{\mathcal{P}}) \leq \Big(\frac{3}{2} + \frac{1}{12k}\Big) \cdot |\mathcal{P}|$$

such that no $e \in M_{\mathcal{P}}$ crosses a $P_i \in \mathcal{P}$, i.e. either $e \subseteq P_i$ or $e \cap P_i = \emptyset$.

Proof. Consider a processor P_i. After reordering let $\tau_1, \ldots, \tau_q$ be the tasks on P_i, sorted such that $u(\tau_1) \geq \ldots \geq u(\tau_q)$. First suppose that $q \geq 2$. We will either cover two tasks in P_i by a matching edge or leave all tasks uncovered. But in any case we guarantee, that the tasks in P_i contribute at most $(\frac{3}{2} + \frac{1}{12k})$ to $\mathit{price}(M_{\mathcal{P}})$. We distinguish two cases, depending on whether P_i contains a large task or not.

Case τ_1 not large: We leave all tasks in P_i uncovered. Note that all tasks in P_i are either of small or medium size, hence $w(\tau_j) \leq \frac{3}{2} u(\tau_j)$ for $j = 1, \ldots, q$. The contribution of P_i is $\sum_{j=1}^{q} w(\tau_j) \leq \frac{3}{2} \sum_{j=1}^{q} u(\tau_j) \leq \frac{3}{2}$.

Case τ_1 large: We add $\{\tau_1, \tau_2\}$ to the matching. We may do so since both tasks are RM-schedulable, the weight of the edge is positive because τ_1 is large, and hence

$\{\tau_1, \tau_2\} \in E$. The contribution is

$$1 + \sum_{j=3}^{q} w(\tau_j) \leq 1 + \underbrace{\sum_{j=3}^{q} u(\tau_j)}_{\leq 1 - u(\{\tau_1,\tau_2\})} \cdot \frac{1}{1 - \underbrace{u(\tau_j)}_{\leq u(\tau_2)}} \leq 1 + \frac{1 - u(\tau_1) - u(\tau_2)}{1 - u(\tau_2)} \qquad (2)$$

$$\leq 1 + \frac{\frac{1}{2} + \frac{1}{12k} - u(\tau_2)}{1 - u(\tau_2)} \leq \frac{3}{2} + \frac{1}{12k}$$

using that $\tau_3, \dots, \tau_q$ are small and $\frac{a-x}{1-x}$ is monotone decreasing if $a < 1$.

If $q = 1$, then we do not cover τ_1. The contribution is at most 1. Moreover, the above construction guarantees that no edge in $M_{\mathcal{P}}$ crosses a processor P_i. $\square$

If we compute a maximum weight matching in our algorithm (say in running time in $O(n^3)$), by simply combining Theorems 1 and 2, we can already obtain a bound of

$$\left(\frac{3}{2} + \frac{1}{12k}\right) \cdot \left(1 + \frac{1}{2k}\right) \cdot OPT + 9k \leq \left(\frac{3}{2} + \frac{1}{k}\right) OPT + 9k$$

on the number of used processor. However, we do not want to fall short of the running time of $O(n^2)$ of the $7/4$-approximation algorithm of Liebeherr et al. [8]. Hence, we use a greedy matching instead, which can be computed in $O(n^2)$. Observe that in the previous proof, in particular for the second case, we left some slack to the approximation ratio. This will become useful in the proof of the next theorem, saying that for any feasible partition it is sufficient to consider a greedy maximal matching.

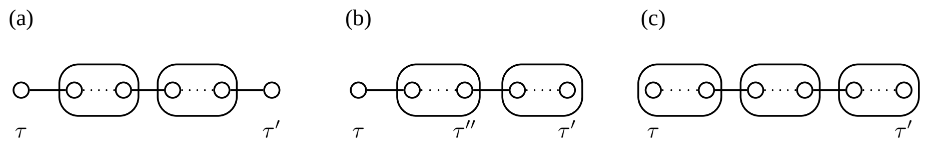
Theorem 3. *If $\mathcal{P} = \{P_1, \dots, P_\ell\}$ be a feasible partition, then we have for a greedy matching M that*

$$\mathit{price}(M) \leq \left(\frac{3}{2} + \frac{1}{6k}\right) |\mathcal{P}|.$$

Proof. This proof is based on a comparison of M with the matching $M_{\mathcal{P}}$, constructed in Theorem 2. To this end, we consider the symmetric difference of the two matchings, i.e. let $E' := M \Delta M_{\mathcal{P}}$. Note that E' is a collection of disjoint paths and cycles, i.e. for all $v \in V$, we have $|\{e \in E' : v \in e\}| \leq 2$. First, we consider a cycle $C \subseteq E'$. Observe that $|C \cap M| = |C \cap M_{\mathcal{P}}|$ by the fundamentals of matching theory. Let $q := |C \cap M|$ and let $P_1, \dots, P_q$ be the processors that contain edges from $C \cap M_{\mathcal{P}}$. Note that each edge in $M_{\mathcal{P}}$ is contained in exactly one processor and moreover that M matches all nodes in $P_1 \cup \dots \cup P_q$ that $M_{\mathcal{P}}$ does. Hence, we have

$$|C \cap M| + \sum_{i=1}^{q} w(P_i \cap \overline{M}) = |C \cap M_{\mathcal{P}}| + \sum_{i=1}^{q} w(P_i \cap \overline{M}_{\mathcal{P}}) \leq \left(\frac{1}{2} + \frac{1}{12k}\right) q.$$

Next, we consider a path $Q \subseteq E'$. Again let $P_1, \dots, P_q$ be the processors containing edges from $M_{\mathcal{P}} \cap Q$. We distinguish the three cases, when both, one, or none of the end-nodes of the path Q are matched in M as illustrated below. The solid edges belong to M and the dashed ones belong to $M_{\mathcal{P}}$. The boxes represent the processors of



Case (a). If both ends of Q are matched in M, then $|M \cap Q| - 1 = |M_{\mathcal{P}} \cap Q| = q$. Hence,

$$|M \cap Q| + \sum_{i=1}^{q} w(P_i \cap \overline{M}) \leq |M_{\mathcal{P}} \cap Q| + \sum_{i=1}^{q} w(P_i \cap \overline{M}_{\mathcal{P}}) + 1 - w(\tau) - w(\tau')$$

where τ, τ' are the both ends of Q. If one of τ, τ' is large, then there is nothing to show. Suppose that none of them is large. Then there is at least one processor that contains two large tasks, since Q has an odd number of edges and since by definition each edge contains at least one large task. Furthermore by the greedy selection, there is at least one large neighboring task in this path, and by the same parity argument, there is a further processor with two large tasks. Note that $q \geq 2$ if neither τ nor τ' is large. If $q = 2$ like in the above example, then all unmatched tasks on the two processors have a smaller weight than τ or τ', respectively. Since this yields the claim, we suppose that $q > 2$ in the following.

$$|M \cap Q| + \sum_{i=1}^{q} w(P_i \cap \overline{M}) \leq q + 1 + \Big(\frac{1}{2} + \frac{1}{12k}\Big)(q-2) + \frac{2}{6k-1} \leq \Big(\frac{3}{2} + \frac{1}{6k}\Big)q$$

Case (b). If exactly one of the endpoints of Q is matched in M, say τ, and the other endpoint, say τ' is matched on processor P_q, then

$$|M \cap Q| + \sum_{i=1}^{q} w(P_i \cap \overline{M}) \leq |M_{\mathcal{P}} \cap Q| - w(\tau) + w(\tau') + \sum_{i=1}^{q} w(P_i \cap \overline{M}).$$

If $q = 1$, then the greedy selection implies that $w(\tau) \geq w(\tau')$. Hence, we assume that $q \geq 2$. Let τ'' be as in the illustration. By the greedy selection, we have $u(\tau'') \geq u(\tau')$. If τ'' is small, then

$$w(P_{q-1} \cap \overline{M}) + w(P_q \cap \overline{M}) \leq \frac{\frac{1}{2} + \frac{1}{12k} - u(\tau'')}{1 - u(\tau'')} + \frac{\frac{1}{2} + \frac{1}{12k}}{1 - u(\tau')} \leq 1 + \frac{1}{4k}$$

as in Ineq. (2) in the proof of Theorem 2. By a similar argument, the same bound holds if τ'' is medium. If τ'' is large, then either τ is large itself or there is a processor P_j with $j \in \{1, \ldots, q-1\}$ with two large tasks, since each edge contains at least one large task. In the former case, there is nothing to show, whereas in the latter case, we may assume w.l.o.g. that $j = q - 1$ and hence the bound of $1 + \frac{1}{4k}$ also holds. Note that if $w(\tau') = 1$, then no further unmatched task can be on P_q, and hence $w(P_q \cap \overline{M}) = 1$, because they would have been matched by the algorithm. Altogether, this yields

$$|M \cap Q| + \sum_{i=1}^{q} w(P_i \cap \overline{M}) \leq q + \Big(\frac{1}{2} + \frac{1}{12k}\Big)(q-2) + 1 + \frac{1}{4k} \leq \Big(\frac{3}{2} + \frac{1}{6k}\Big)q.$$

Case (c). If none of the endpoints of Q are matched in M, then

$$|M \cap Q| + \sum_{i=1}^{q} w(P_i \cap \overline{M}) \leq |M \cap Q| + \sum_{i=1}^{q} w(P_i \cap \overline{M}_{\mathcal{P}}) - 1 + w(\tau) + w(\tau')$$

where τ, τ' are the both ends of Q. If neither τ nor τ' is large, then there is nothing to show. Hence, we assume w.l.o.g. that $w(\tau) = 1$. Since τ is not matched in M, there is no further task on the same processor that is also unmatched in M. Hence,

$$|M \cap Q| + \sum_{i=1}^{q} w(P_i \cap \overline{M}) \leq \Big(\frac{3}{2} + \frac{1}{12k}\Big)(q-1) + w(\tau') \leq \Big(\frac{3}{2} + \frac{1}{12k}\Big)q.$$

□

Corollary 1. *Algorithm 2 produces a solution of cost* $(\frac{3}{2} + \frac{1}{k})OPT + 9k$ *in time* $O(n^2)$.

Proof. Note that for each set $\{\tau_1, \tau_2\}$ RM-schedulability can be tested in constant time using condition (1). Sorting the tasks by decreasing utilization takes $O(n \log n)$ time and is subsumed by the time necessary to create G, which is $O(n^2)$. In fact, it is only necessary to scan each large task and check with every other task with smaller utilization whether they can be scheduled together. If so both tasks are marked as matched provided that none of them has been matched before. However, this procedure still requires quadratic running time since all tasks might be large in the worst case. The running time of FFMP is $O(n' \log n')$ for scheduling n' tasks, thus the total running time is $O(n^2)$.

The approximation guarantee follows from Theorem 3 and Theorem 1, since we may combine them to show that the number of processors produced does not exceed

$$\Big(1 + \frac{1}{2k}\Big) \cdot \Big(\frac{3}{2} + \frac{1}{6k}\Big)OPT + 9k \leq \Big(\frac{3}{2} + \frac{1}{k}\Big) \cdot OPT + 9k.$$

□

5 Experimental results

We have implemented and compared our k-RMM algorithm experimentally with the ones, which are known from the literature and have already been mentioned in Sect. 1.1. To this end, we have randomly generated instances with the number of tasks n ranging from 10 to 10^5. That is, for each given n, we have generated 100 samples, where integer periods have been chosen out of $(0, 500)$ uniformly at random and independently utilizations from $(0, 1)$ u.a.r. All algorithms have been tested on the same instances to allow also a direct comparison. With a choice of $k = \lfloor \sqrt{n} \rfloor$, our new algorithm has outperformed the others on almost all instances (in fact it has been 1 processor worse on only 4 instances). For $n = 10$ and $n = 20$, we have also computed the optimum solutions by a configuration-based ILP solved with CPLEX. For 82% of the instances with 10 tasks and 76% of the instances with $n = 20$, our k-RMM has found the optimum solution, and in the remaining cases it only fell short by one processor. Looking at

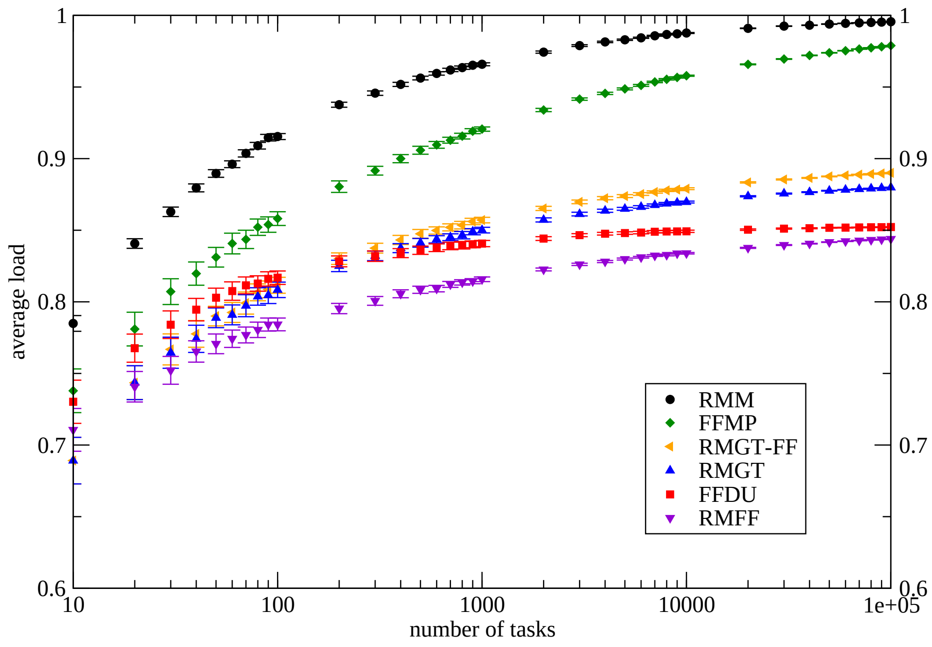


Figure 1. A comparison of our algorithm with the ones known from the literature w.r.t. the average processor load.

the average processor load, i.e. the total utilization divided by the number of allocated processors, in Fig. 1, one can see that our k-RMM algorithm uses the processor much more efficiently than the other approximation algorithms.

Figure 1 suggests that the average load for k-RMM converges to 1 as n goes to infinity. In fact, it is not hard to prove that the *waste* of k-RMM, i.e. the difference between the allocated processors and the total utilization, scales sub-linearly with the number of tasks on random instances. More precisely, the same bound of $O(n^{3/4} \log^{3/8} n)$ for the waste of FFMP, which has been shown in [19], also holds for our new algorithm. However, experiments suggest that this bound for k-RMM might be something closer to $\sqrt{n}$. A further interesting open question is whether there exists an asymptotic PTAS for the rate-monotonic multiprocessor scheduling problem.

References

1. Liu, C.L., Layland, J.W.: Scheduling algorithms for multiprogramming in a hard-real-time environment. J. ACM **20**(1) (1973) 46–61
2. Buttazzo, G.: Hard Real-time Computing Systems: Predictable Scheduling Algorithms And Applications (Real-Time Systems Series). (2004)
3. Garey, M.R., Graham, R.L., Johnson, D.S., Yao, A.C.C.: Resource constrained scheduling as generalized bin packing. J. Combin. Theory Ser. A **21** (1976) 257–298
4. Johnson, D.S.: Near-optimal bin packing algorithms. PhD thesis, MIT, Cambridge, MA (1973)

5. Fernandez de la Vega, W., Lueker, G.S.: Bin packing can be solved within $1 + \varepsilon$ in linear time. Combinatorica **1**(4) (1981) 349–355
6. Karmarkar, N., Karp, R.M.: An efficient approximation scheme for the one-dimensional bin-packing problem. In: FOCS'82. IEEE (1982) 312–320
7. Coffman, Jr., E.G., Garey, M.R., Johnson, D.S.: Approximation algorithms for bin-packing—an updated survey. In: Algorithm design for computer system design. Springer (1984)
8. Liebeherr, J., Burchard, A., Oh, Y., Son, S.H.: New strategies for assigning real-time tasks to multiprocessor systems. IEEE Trans. Comput. **44**(12) (1995) 1429–1442
9. Liu, J.: Real-Time Systems. Prentice Hall PTR, Upper Saddle River, NJ, USA (2000)
10. Bini, E., Buttazzo, G., Buttazzo, G.: A hyperbolic bound for the rate monotonic algorithm. In: ECRTS '01. (2001) 59
11. Eisenbrand, F., Rothvoß, T.: Static-priority Real-time Scheduling: Response Time Computation is NP-hard. In: RTSS. (2008)
12. Audsley, A.N., Burns, A., Richardson, M., Tindell, K.: Applying new scheduling theory to static priority pre-emptive scheduling. Software Engineering Journal (1993) 284–292
13. Fisher, N., Baruah, S.: A fully polynomial-time approximation scheme for feasibility analysis in static-priority systems with arbitrary relative deadlines. In: ECRTS '05. (2005) 117–126
14. Baruah, S.K., Pruhs, K.: Open problems in real-time scheduling. Journal of Scheduling (2009)
15. Leung, J.: Handbook of Scheduling: Algorithms, Models, and Performance Analysis. CRC Press, Inc., Boca Raton, FL, USA (2004)
16. Dhall, S.K., Liu, C.L.: On a real-time scheduling problem. Operations Research **26**(1) (January-February 1978) 127–140
17. Dhall, S.K.: Approximation algorithms for scheduling time-critical jobs on multiprocessor systems. In Leung, J.Y.T., ed.: Handbook of Scheduling — Algorithms, Models, and Performance Analysis. Chapman & Hall/CRC (2004)
18. Oh, Y., Son, S.H.: Allocating fixed-priority periodic tasks on multiprocessor systems. Real-Time Syst. **9**(3) (1995) 207–239
19. Karrenbauer, A., Rothvoß, T.: An average-case analysis for rate-monotonic multiprocessor real-time scheduling. In: ESA'09. (2009) 432–443
20. Eisenbrand, F., Rothvoß, T.: A PTAS for static priority real-time scheduling with resource augmentation. In: ICALP (1). (2008) 246–257
21. Baruah, S., Goossens, J.: Scheduling real-time tasks: Algorithms and complexity. In: Handbook of Scheduling — Algorithms, Models, and Performance Analysis. (2004)
22. Gabow, H.N.: Data structures for weighted matching and nearest common ancestors with linking. In: SODA'90. (1990)
23. Cook, W.J., Cunningham, W.H., Pulleyblank, W.R., Schrijver, A.: Combinatorial Optimization. John Wiley, New York (1997)
24. Schrijver, A.: Combinatorial Optimization: Polyhedra and Efficiency. Volume 24 of Algorithms and Combinatorics. Springer (2003)