A PTAS for Static Priority Real-Time Scheduling with Resource Augmentation

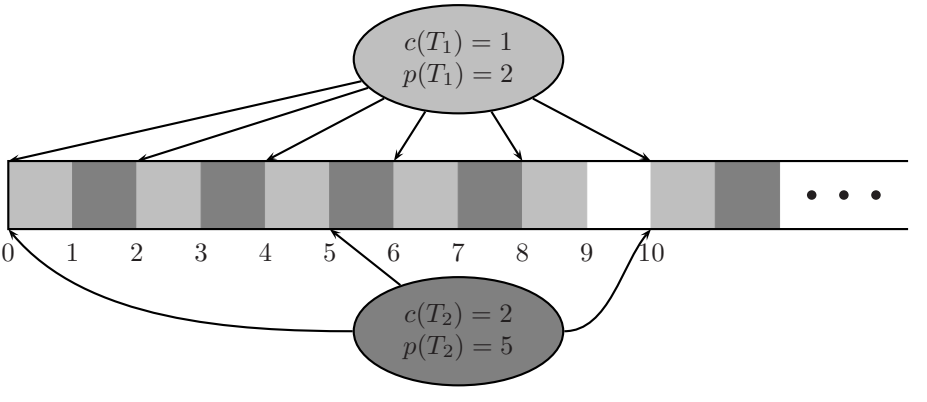
Friedrich Eisenbrand* and Thomas Rothvoß

Institute of Mathematics
École Polytechnique Féderale de Lausanne, 1015 Lausanne, Switzerland
{friedrich.eisenbrand,thomas.rothvoss}@epfl.ch

Abstract. We present a polynomial time approximation scheme for the *real-time scheduling problem with fixed priorities* when resource augmentation is allowed. For a fixed $\varepsilon > 0$, our algorithm computes an assignment using at most $(1+\varepsilon)\cdot OPT+1$ processors in polynomial time, which is feasible if the processors have speed $1+\varepsilon$. We also show that, unless $P = NP$, there does not exist an asymptotic FPTAS for this problem.

1 Introduction

In this paper, we are concerned with a scheduling problem described by Liu and Layland [11], which has received considerable attention in the real-time and embedded-systems community. Here one is given a set of *tasks* $\mathcal{T} = \{T_1, \ldots, T_n\}$, where each task T is characterized by two positive values, its *period* $p(T)$ and its *running time* $c(T)$. The task T releases a *job* requiring running time $c(T)$ at each integer multiple of its period.



If several tasks $\mathcal{T}' \subseteq \mathcal{T}$ are assigned to one processor, then this assignment is *feasible* if each job, being released by some task T at time $i \cdot p(T)$ is finished at time $(i+1)\cdot p(T)$, whereby jobs stemming from tasks with smaller period preempt

* Supported by Deutsche Forschungsgemeinschaft (DFG) within Priority Programme 1307 "Algorithm Engineering".

L. Aceto et al. (Eds.): ICALP 2008, Part I, LNCS 5125, pp. 246–257, 2008.

those stemming from tasks with larger period. Ties are broken in an arbitrary but fixed way. In this case, we also speak about an assignment in which each *task is feasible* itself. Liu and Layland [11] have shown that this *rate-monotonic scheduling* is optimal, meaning if there is a feasible priority assignment, then the one in which the priority of a task T equals $1/p(T)$ is also feasible.

The picture above shows a feasible set $\mathcal{T}' = \{T_1, T_2\}$ of tasks. The arrows indicate the points in time, where the two tasks T_1 and T_2 release jobs. At time 0, the first job of T_1 as well as the first job of T_2 are released. Since the period of T_1 is smaller than the period of T_2, the first job of T_1 is executed, until it is finished at time 1. Now the first job of T_2 is executed, but interrupted by the second job of T_1 at time 2. The execution of the first job of T_2 is resumed at time 3 and finished at time 4. Notice that the processor is idle for one time unit at time 9 and that the schedule repeats at the least common multiple of the periods which is 10. All jobs finish in time. The set $\mathcal{T}'$ is feasible.

The *static-priority real-time scheduling problem* is now to determine a partitioning of a task-set $\mathcal{T}$ into $\mathcal{T}_1, \ldots, \mathcal{T}_k$, such that each $\mathcal{T}_i$ is a feasible set of tasks for one processor and the number k of processors is minimized. In the real-time literature, this problem is also known as the static-priority real-time scheduling problem *with implicit deadlines*, since the deadlines are implicitly given by the periods of the tasks.

Related Work. If the periods $p(T)$ of all tasks in $\mathcal{T}$ are one, then the scheduling problem is simply the well known *bin packing problem*. This is because a set of tasks $\mathcal{T}' \subseteq \mathcal{T}$ would be feasible on one processor if and only if the sum of their running times is bounded by one. Recall that for bin packing an asymptotic PTAS [4] and even an asymptotic FPTAS exists [8].

The *utilization* of $\mathcal{T}'$ is defined as $\text{util}(\mathcal{T}') = \sum_{T \in \mathcal{T}'} c(T)/p(T)$. If $\mathcal{T}'$ is feasible, then the utilization $\text{util}(\mathcal{T}')$ is at most 1. However, $\mathcal{T}'$ can be infeasible, even if $\text{util}(\mathcal{T}') < 1$. Consider, for example, again the task system $\mathcal{T}'$ depicted on the cover page. If we increase the running time of T_1 by any $\varepsilon > 0$, then the set $\mathcal{T}'$ is no longer feasible and its utilization is $\text{util}(\mathcal{T}') = (9 + 5 \cdot \varepsilon)/10$. Liu and Layland [11] have shown that $\mathcal{T}'$ is feasible, if $\text{util}(\mathcal{T}')$ is bounded by $n'(2^{1/n'} - 1)$, where $n' = |\mathcal{T}'|$. This bound tends to $\ln 2$ and the condition is not necessary for feasibility, as the example with all periods equal to one shows. Stronger, but still not necessary conditions for feasibility are given in [10,2,12].

It is a longstanding open problem, whether there exists a polynomial time algorithm which decides whether a set $\mathcal{T}'$ of tasks is *feasible on one processor*. A first result in this direction using resource augmentation was presented by Fisher and Baruah [5]. In their paper, the authors show that one can efficiently decide whether a set of tasks is feasible, or infeasible on a faster processor of speed $1 + \varepsilon$. Our approximation scheme can be understood as an extension of their algorithm, which additionally approximates the task-distribution problem.

The sufficient condition $\text{util}(\mathcal{T}') \leqslant n'(2^{1/n'} - 1)$ allows to use first-fit and next-fit algorithms as in the case of bin packing. The currently best ratio for such strategies is 7/4 due to [10]. We refer to [3] for a survey of approximation algorithms based on first-fit and next-fit and to the article [1] for an overview

on complexity issues of real-time scheduling. The literature on approximation schemes, especially in scheduling, is extensive. We refer to [13] for a recent account.

Results. We show that, for each $\varepsilon > 0$ there exists a polynomial time algorithm which computes a partitioning using at most $(1+\varepsilon) \cdot OPT(\mathcal{T}) + 1$ subsets. Each subset is feasible on a processor of speed $1+\varepsilon$. Here $OPT(\mathcal{T})$ denotes the minimum number of processors to feasibly schedule $\mathcal{T}$. Our result is the first PTAS for the real-time scheduling problem with resource augmentation. Furthermore we show that real-time scheduling is harder to approximate than bin packing. Unless $P = NP$, there does not exist an algorithm which has an additive gap of $O(n^{1-\varepsilon})$ for any fixed $\varepsilon > 0$. This implies that there does not exist an asymptotic FPTAS for real-time scheduling without resource augmentation.

The main insights which lead to our PTAS with resource augmentation are twofold.

i) Apart from the standard *rounding of the instance*, we describe the concept of *local feasibility*. The effect of far-scattered periods prevents the application of bin packing techniques. The concept of local feasibility considers these effects only for those tasks, whose periods are close or *local* to the period of the task in consideration. A local feasible schedule is feasible on a slightly faster machine.
ii) In bin packing, small items are first discarded from the instance and then distributed with first-fit. Since the utilization is not a good lower bound for the real-time scheduling problem, a similar approach does not work. We provide a much different technique to treat small tasks. We re-set periods and group small tasks with the same period into one large task. A probabilistic argument shows that the optimum of the modified instance does not grow to much.

2 Preliminaries and Simplifying Assumptions

In [11] it is shown that a set $\mathcal{T}$ of tasks is feasible on one processor, if the first job of each task $T \in \mathcal{T}$ finishes before its period $p(T)$, or in other words, if the *response time* of each task is smaller than its period.

This response time r of a task T is calculated as follows. A task T' with higher priority interrupts T exactly $\lceil r/p(T') \rceil$ many times. Each time, this task consumes its processing time $c(T')$. Therefore r is the smallest fix-point of the *response function*

$$f_T(r) = c(T) + \sum_{T' \in \mathcal{T} \setminus \{T\} : p(T') \leqslant p(T)} \lceil r/p(T') \rceil \cdot c(T'). \tag{1}$$

The task system is feasible if there exists for each T a number $r_T^* \leqslant p(T)$ with $f_T(r_T^*) \leqslant r_T^*$. Notice that one has for each $a \in \mathbb{N}_{>0}$ $f_T(a \cdot r) \leqslant a \cdot f_T(r)$. This shows that the task system $\mathcal{T}$ is feasible if and only if there exists an r_T^* for each $T \in \mathcal{T}$

with $p(T)/2 \leqslant r_T^* \leqslant p(T)$ and $f_T(r_T^*) \leqslant r_T^*$. The vector $r^* = (r_{T_1}^*, \ldots, r_{T_n}^*)$ is a *certificate of feasibility* of the task-system $\mathcal{T} = \{T_1, \ldots, T_n\}$. Similarly, we say that the task-system is *feasible on a processor of speed* $\beta > 0$ if there exists a vector $r^* = (r_{T_1}^*, \ldots, r_{T_n}^*)$ with $p(T)/2 \leqslant r_T^* \leqslant p(T)$ and $f_T(r_T^*) \leqslant \beta \cdot r_T^*$. The next Lemma will be used several times in the sequel. A proof can be found in the full version of this paper.

Lemma 1. *Let $\mathcal{T}$ be a set of tasks, then the following holds.*

i) If $\text{util}(\mathcal{T}) \leqslant \gamma$ *with* $\gamma > 0$, *then $\mathcal{T}$ is feasible on a processor of speed* $(1/\ln(2)) \cdot \gamma$.
ii) $\text{util}(\mathcal{T}) \leqslant OPT(\mathcal{T}) \leqslant (2/\ln(2)) \cdot \text{util}(\mathcal{T}) + 1$.
iii) If $\mathcal{T}$ is feasible on a processor of speed β and a second set $\mathcal{T}'$ has utilization at most ε, then $\mathcal{T} \cup \mathcal{T}'$ is feasible on a processor of speed $\beta + 2\varepsilon$.

Simplifying Assumptions. The number $1/\varepsilon$ can be assumed to be an integer. Furthermore, we round each period up to the next power of $(1+\varepsilon)$. If a solution of this rounded instance is feasible, then it is also feasible for the original instance on a processor of speed $(1+\varepsilon)$.

Next, choose $k \in \{0, \ldots, (1/\varepsilon)-1\}$ such that the utilization u_k of tasks, having their period in an interval $[(1/\varepsilon)^i, (1/\varepsilon)^{i+1}[$ with $i \equiv k \pmod{1/\varepsilon}$, is minimized. Clearly $u_k \leqslant \varepsilon \cdot \text{util}(\mathcal{T})$. Thus we may remove all tasks, contributing to u_k and schedule them in a first-fit manner on $(2/\ln 2) \cdot OPT + 1$ additional processors, using Lemma 1.ii). This process yields a partitioning of the tasks into *blocks* $\mathcal{B}_1, \ldots, \mathcal{B}_\mu$ with the following properties.

i) If p_i and p_j are periods of tasks in $\mathcal{B}_i$ and $\mathcal{B}_j$ with $i < j$, then $(1/\varepsilon) \cdot p_i \leqslant p_j$.
ii) The number of different periods of tasks in one block $\mathcal{B}_i$ is bounded by $((1/\varepsilon) - 1) \cdot \log_{1+\varepsilon}(1/\varepsilon) \leqslant 1/\varepsilon^3$ which is a constant.

3 Real-Time Scheduling Is Harder Than Bin Packing

Due to its relation to bin packing, a natural question to ask at this point is whether real-time scheduling can be approximated as well as bin packing. The algorithm of Karmarkar and Karp [8] computes a solution to the bin packing problem in polynomial time, which has an *additive* approximation guarantee. More precisely, given an instance I the algorithm computes a solution $APX(I)$ with $APX(I) - OPT(I) \leqslant O(\log^2(OPT(I)))$. An analogous result cannot hold for real-time scheduling unless $P = NP$.

Theorem 2. *If $P \neq NP$, there is no $\varepsilon > 0$ such that there exists a polynomial algorithm which computes an approximate solution $APX(\mathcal{T})$ for each instance $\mathcal{T}$ with*

$$APX(\mathcal{T}) - OPT(\mathcal{T}) \leqslant |\mathcal{T}|^{1-\varepsilon}.$$

Proof. The proof of this theorem is by reduction from 3-PARTITION. An instance of 3-PARTITION is a multiset of $3 \cdot n$ numbers $a_1, \ldots, a_{3n} \in \mathbb{R}_+$. The problem is to decide, whether this set can be partitioned into triples, such that the sum of the numbers of each triple is exactly one. 3-PARTITION is strongly NP-complete see [6]. The idea is now to construct a set of tasks $\mathcal{T} = \mathcal{T}_1 \cup \cdots \cup \mathcal{T}_k$ such that the following holds.

a) All tasks in $\mathcal{T}_j$ have the same period and $\mathcal{T}_j$ consists of $3n$ tasks with utilization $a_1, \ldots . a_{3n}$ respectively.
b) If a subset $\mathcal{T}' \subseteq \mathcal{T}$ contains 3 tasks and the periods of the tasks in $\mathcal{T}'$ are not all equal, then $\mathcal{T}'$ is infeasible.

With such a construction at hand one needs $k \cdot n$ processors if 3-PARTITION has a solution while one needs at least $n \cdot k + k/2$ processors if 3-PARTITION does not have a solution. If there exists an algorithm which computes a solution $APX(\mathcal{T})$ with $APX(\mathcal{T}) - OPT(\mathcal{T}) \leqslant (3 \cdot k \cdot n)^{1-\varepsilon}$ for some $\varepsilon > 0$, then one could use it to test whether 3-PARTITION has a solution, since $(3 \cdot k \cdot n)^{1-\epsilon} < k/2$ for $k = \Omega(n^{1/\varepsilon})$.

What remains to show, is how to construct such an instance $\mathcal{T} = \mathcal{T}_1 \cup \cdots \cup \mathcal{T}_k$ as above. If we define new weights $a_i' = \frac{m/3 + a_i}{m+1}$, then this new instance of 3-PARTITION is equivalent to $a_1, \ldots, a_{3n}$, since three new weights sum up to one if and only if the corresponding old weights sum up to one. This shows that we can assume that the weights $a_1, \ldots, a_{3n}$ are between $1/3 - 1/m$ and $1/3 + 1/m$ for an arbitrary $m \in \mathbb{Z}^+$.

Next we consider the periods $p_j = 1 + j/(4 \cdot k)$ for $j = 1, \ldots, k$. Those periods are between $1 + 1/(4 \cdot k)$ and $1 + 1/4$. The group $\mathcal{T}_j$ consists now of tasks having period p_j and utilization $a_1, \ldots, a_{3n}$ respectively, which implies a). To show b), consider a set $\mathcal{T}' = \{T_1, T_2, T_3\}$ of three tasks, where the period of T_3 is strictly larger than the period of T_1. We argue that T_3 is infeasible, if $\mathcal{T}'$ is scheduled on one processor. Consider a fix-point r of the response function of T_3

$$r = c(T_3) + \left\lceil \frac{r}{p(T_1)} \right\rceil c(T_1) + \left\lceil \frac{r}{p(T_2)} \right\rceil c(T_2). \tag{2}$$

Since $c(T_3) = \text{util}(T_3) \cdot p(T_3)$ one has $r \geqslant c(T_3)/(1 - \text{util}\{T_1, T_2\})$ which implies that $r \geqslant p(T_3)(1 - 9/(m+6))$. Notice that $p(T_3) \geqslant p(T_1) + 1/(4k)$, thus $p(T_3)/p(T_1) \geqslant 1 + 1/(4k \cdot p(T_1)) \geqslant 1 + 1/(5k)$. If one chooses $m = 90k$, then $r/p(T_1) > 1$ and it follows that $\lceil \frac{r}{p(T_1)} \rceil$ in (2) is at least 2. This means that $r \geqslant c(T_3) + 2c(T_1) + c(T_2) \geqslant 4(1/3 - 1/m)$ which is larger than $5/4 \geqslant p(T_3)$. This implies that T_3 is infeasible. □

Corollary 3. *Unless $P = NP$, there does not exist an asymptotic FPTAS for real-time scheduling.*

Proof. An asymptotic FPTAS [7] is an algorithm, whose running time is polynomial in n and $1/\varepsilon$ and which yields a solution of cost at most $(1+\varepsilon) \cdot OPT + p(1/\varepsilon)$ for some polynomial p. Assume that such an asymptotic FPTAS exists. We

assume w.l.o.g. that $p(1/\varepsilon) = \varepsilon^{-\alpha}$ for a fixed exponent $\alpha > 0$. Then with $\varepsilon = (1/n)^{1/(2\alpha)}$ the algorithm computes a solution with

$$APX - OPT \leqslant \varepsilon \cdot OPT + (1/\varepsilon)^{\alpha} \leqslant n^{1-1/(2\alpha)} + n^{1/2},$$

which is a contradiction to Theorem 2. □

4 Local Feasibility and an Algorithm to Schedule Large Tasks

Consider the response function (1) for a task T. For local feasibility of T, the tasks T' with $p(T') \leqslant \varepsilon \cdot p(T)$ contribute only with their utilization to the response function and the rounding operation in (1) is ignored. Thus the *local response function* $f_T^{local}(r)$ is defined as

$$c(T) + r \cdot \operatorname{util}(\{T' \colon p(T') \leqslant \varepsilon \cdot p(T)\}) + \sum_{\substack{T' \in \mathcal{T} \setminus \{T\} \\ \varepsilon \cdot p(T) < p(T') \leqslant p(T)}} \lceil r/p(T') \rceil \cdot c(T'). \quad (3)$$

The task T is *local feasible*, if there exists a number $p(T)/2 \leqslant r_T^* \leqslant p(T)$ with $f_T^{local}(r_T^*) \leqslant r_T^*$. In other words, the contribution of the rounding operation is only taken into account for tasks which are *close* or *local* to the task in consideration. The other tasks contribute only with their utilization.

We now show that, if an assignment is locally feasible (each task is locally feasible), then it is feasible on processors of speed $1 + 2\varepsilon$. We can therefore relax feasibility to local feasibility, which will later allow us to optimally distribute large tasks.

Lemma 4. *If a set of tasks $\mathcal{T}$ is local feasible on one processor, then it is feasible on a processor of speed $1 + 2\varepsilon$.*

Proof. Let r_T^* be the certificate for local feasibility of $T \in \mathcal{T}$, i.e., one has $p(T)/2 \leqslant r_T^* \leqslant p(T)$ and $f_T^{local}(r_T^*) \leqslant r_T^*$. It is enough to show that $f_T(r_T^*) \leqslant (1 + 2\,\varepsilon) f_T^{local}(r_T^*)$ holds. The difference between $f_T(r_T^*)$ and $f_T^{local}(r_T^*)$ can be bounded by

$$\sum_{T' \colon p(T') \leqslant \varepsilon \cdot p(T)} c(T').$$

Since $1 \leqslant 2\,\varepsilon \cdot r_T^*/p(T')$ this difference is bounded by

$$2\,\varepsilon\, r_T^* \cdot \sum_{T' \in \mathcal{T} : p(T') \leqslant \varepsilon \cdot p(T)} c(T')/p(T') \quad \leqslant \quad 2\,\varepsilon \cdot f_T^{local}(r_T^*)$$

and the result follows. □

4.1 A Dynamic Program to Schedule Large Tasks

We describe now an algorithm which optimally distributes a set of tasks in in polynomial time if we additionally assume that each utilization is bounded from below by the constant ε and an increase of speed by $1 + O(\varepsilon)$ is allowed.

If we round all running times $c(T)$ down such that the utilization of T becomes the nearest integer multiple of ε^2, then due to the reason that each $c(T)/p(T)$ is at least ε, a feasible schedule for the new task system yields a feasible assignment for the original task system, if the machines have speed $1 + O(\varepsilon)$. Therefore we can also assume that each task T has utilization $c(T)/p(T) \in \varepsilon^2\mathbb{Z}$.

Let $\mathcal{B}_1, \ldots, \mathcal{B}_\mu$ be the block-partitioning of the task system $\mathcal{T} = \{T_1, \ldots, T_n\}$ (see section 2). How many different types of tasks, can be present in one block $\mathcal{B}_i$? The number of different periods of $\mathcal{B}_i$ is bounded by $1/\varepsilon^3$. The number of different utilization-values of tasks in $\mathcal{T}$ is bounded by $1/\varepsilon^2$. Therefore, the number of different types of tasks in each block is bounded by a constant. The tasks are distributed with a dynamic programming algorithm to compute an optimal assignment of $\mathcal{T}$ such that each task is locally feasible. This is done, block-wise.

A vector $a = (a_0, ..., a_{1/\varepsilon^2})$ with $a_i \in \mathbb{Z}$ is called a *configuration*, whereby a_i denotes the number of processors whose utilization is exactly $i \cdot \varepsilon^2$. We require that $\sum_i a_i = n$. Consider the following table entries.

$$A(a, \ell) = \begin{cases} 1 \text{ if tasks in } \mathcal{B}_1, ..., \mathcal{B}_\ell \text{ can be scheduled in a locally feasible way} \\ \quad \text{such that utilization bounds of configuration } a \text{ are met} \\ 0 \text{ otherwise} \end{cases}$$

Note that a has fixed dimension, thus the table has a polynomial number of entries. We now describe, how to compute $A(a, \ell)$ efficiently. Let $b = (b_0, \ldots, b_{1/\varepsilon^2})$ be a processor configuration from a distribution of the tasks $\mathcal{B}_1, \ldots, \mathcal{B}_{\ell-1}$. Then $\text{LocalRTS}(\mathcal{B}_\ell, b, a)$ is defined to be 1, if the tasks in block $\mathcal{B}_\ell$ can be additionally distributed among the processors, such that the bounds of configuration a are met. The base cases are

$$A(a, 1) = \text{LocalRTS}(\mathcal{B}_1, (n, 0, ..., 0), a)$$

For all $\ell > 1$ note that $A(a, \ell) = 1$ if and only if there exists a $b \in \mathbb{Z}^{1/\varepsilon^2+1}$ with $0 \leqslant b_i \leqslant a_i$ for all i and

$$A(b, \ell - 1) = 1 \quad \text{and} \quad \text{LocalRTS}(\mathcal{B}_\ell, b, a) = 1$$

After computing all entries, the optimal number of processors can be read out of the table.

It remains to show, how to determine LocalRTS efficiently. The block $\mathcal{B}_\ell$ has only a constant number of different task-types, each having a utilization, which is lower-bounded by a constant. Suppose that $\mathcal{B}_\ell$ has tasks, whose running-time and period are from the tuples $(c_1, p_1), \ldots, (c_k, p_k)$. A *pattern* is a vector $(x_1, \ldots, x_k) \in \mathbb{N}_0^k$ which represents a set of tasks with these types of total utilization at most 1 (the set, defined by the pattern, contains x_i times task type

(c_i, p_i)). There is only a constant number of patterns, which can be used to distribute the tasks in $\mathcal{B}_\ell$. This shows that LocalRTS can be computed in polynomial time with integer programming in fixed dimension [9]. Details of the model are described in the full version of this paper. We have the following result.

Theorem 5. *Let $\mathcal{T} = \{T_1, \ldots, T_n\}$ be a set of tasks and let $\varepsilon > 0$ be a constant such that $c(T)/p(T) \geqslant \varepsilon$ for all $T \in \mathcal{T}$. Then we can distribute the tasks using $OPT(\mathcal{T})$ many processors in polynomial time, such that the tasks on each processor are feasible if the processors have speed $1 + O(\varepsilon)$.*

5 Small Tasks

The well known approximation algorithms for bin packing [4,8] use the fact that small items of weight at most ε can first be discarded from the instance and then be added in a first-fit way after the larger items have been packed. If a new bin had to be opened to pack the small items, then the weight of each bin, except possibly the last bin, exceeds $1 - \varepsilon$. If m bins are then open, then $(m-1)(1-\varepsilon)$ is a lower bound on $OPT(I)$ which implies that $m \leqslant (1 + 2\varepsilon)\, OPT(I) + 1$.

For the real-time scheduling problem, an analogous approach to deal with tasks having small utilization does not work. This is again because a subset of tasks might be infeasible, even if its utilization only slightly exceeds ln(2). In this section we describe a tailored procedure to eliminate small tasks. It has two steps.

I) In a first step, we discard tasks and re-set periods such that the utilization of each period is at least ε^6. Here, the utilization of a period p is the sum of the utilization of the tasks having period p. The total utilization of the discarded tasks is bounded by $O(\varepsilon) \cdot \text{util}(\mathcal{T})$.
II) In a second step we cluster small tasks of the same period into groups, each of which will be identified into one single task having utilization ε^6.

After these discards, re-setting of periods and identification of small tasks, we obtain a new instance $\widetilde{\mathcal{T}}$. If OPT denotes the minimum number of processors to feasibly schedule $\mathcal{T}$, then $\widetilde{\mathcal{T}}$ can be scheduled using $(1+O(\varepsilon))\cdot OPT+1$ processors of speed $1 + O(\varepsilon)$. The next sections describe these two steps in detail.

Periods with Small Utilization

Let p be a period of a task in $\mathcal{T}$. The *utilization* of this period is the sum of the utilizations of tasks, having period p

$$\text{util}(p) = \sum_{T \in \mathcal{T} : p(T) = p} c(T)/p.$$

Suppose now that $\mathcal{B}_1, \ldots, \mathcal{B}_\mu$ is the partitioning of $\mathcal{T}$ into blocks and let $\mathcal{B}_i$ be the first block having utilization $\leqslant \varepsilon^2$. Let j be minimal such that $\text{util}(\mathcal{B}_i \cup \cdots \cup \mathcal{B}_j) \geqslant$

ε^2. If this utilization is larger than ε, then we discard $\mathcal{B}_i, \ldots, \mathcal{B}_{j-1}$ from $\mathcal{T}$. Otherwise, we re-set the period of each task to an arbitrary value sandwiched between the smallest and the largest period of a task in $\mathcal{B}_i \cup \cdots \cup \mathcal{B}_j$. Thereby the utilization of this period is at least ε^2. We repeat this procedure until such a block $\mathcal{B}_i$ having utilization ε^2 cannot be found anymore. The utilization of the tasks which are discarded with this procedure is bounded by $\varepsilon \cdot \text{util}(\mathcal{T})$. With first-fit, these tasks can be scheduled on $O(\varepsilon) \cdot OPT + 1$ additional processors.

Define $p_{\min}(\mathcal{T}) = \min\{p(T) \mid T \in \mathcal{T}\}$ and $p_{\max}(\mathcal{T}) = \max\{p(T) \mid T \in \mathcal{T}\}$. The next lemma shows that re-setting the periods of the tasks in $\mathcal{B}_i \cup \cdots \cup \mathcal{B}_j$ to an arbitrary period in $[p_{\min}(\mathcal{B}_i \cup \cdots \cup \mathcal{B}_j), p_{\max}(\mathcal{B}_i \cup \cdots \cup \mathcal{B}_j)]$ is a feasible operation, if we are to run the tasks on machines of speed $1 + O(\varepsilon)$. More precisely, the lemma implies that, if the tasks could be scheduled on k machines before the re-setting operation, then they can be scheduled on k machines of speed $1+O(\varepsilon)$ after the re-setting operation.

Lemma 6. *Suppose that $\mathcal{T}_1 \cup \cdots \cup \mathcal{T}_k$ is a feasible task system with the property that $p_{\max}(\mathcal{T}_i) \leqslant \varepsilon \cdot p_{\min}(\mathcal{T}_j)$ whenever $i < j$. Let $I \subseteq \{1, \ldots, k\}$ be a set of indices i with $\text{util}(\mathcal{T}_i) \leqslant \varepsilon$ and let $\mathcal{T}^*$ be an instance emerging from $\mathcal{T}_1 \cup \cdots \cup \mathcal{T}_k$ by assigning for each $i \in I$ to each task $T \in \mathcal{T}_i$ an arbitrary period in $[p_{\min}(\mathcal{T}_i), p_{\max}(\mathcal{T}_i)]$ while keeping the utilization of the tasks invariant. The tasks $\mathcal{T}^*$ are feasible on a processor with speed $1 + O(\varepsilon)$.*

Proof. By Lemma 4 it is enough to show that each such changed task is locally feasible on a processor of speed $1 + O(\varepsilon)$. For this purpose, suppose that $T \in \mathcal{T}_i$ and let T^* be the task stemming from T by changing its period. Furthermore let $\mathcal{T}_i^*$ be the changed tasks $\mathcal{T}_i$. Lemma 1.iii) shows that $(\mathcal{T} \setminus \mathcal{T}_i) \cup \mathcal{T}_i^*$ is feasible on a processor of speed $1 + O(\varepsilon)$. Thus, after changing the periods in $\mathcal{T}_i$ only, the system is feasible on a processor of speed $1 + O(\varepsilon)$.

In particular T^* is local feasible on a processor of speed $1 + O(\varepsilon)$. Scaling the periods in the other sets $\mathcal{T}_j$, $j \neq i$ leaves the local response function for T^* unchanged. This shows the claim. □

After applying the procedure described above, the situation is now as follows. Each block of the task system has utilization at least ε^2. Choose $\gamma = \varepsilon^6$ and remove the tasks of all periods having utilization less than γ. Recall that the number of periods in each block is bounded by $1/\varepsilon^3$, thus we remove a utilization of at most $\gamma/\varepsilon^3 = \varepsilon^3$ from each block. Comparing this to the total utilization of each block, one observes that this removed tasks can be scheduled, using again $O(\varepsilon) \cdot OPT + 1$ many extra processors.

Periods with Large Utilization

Each period has now a utilization of at least $\gamma = \varepsilon^6$. Next, we partition $\mathcal{T}$ into $\mathcal{T}_{large}, \mathcal{T}_1, \ldots, \mathcal{T}_q$ such that $\mathcal{T}_{large}$ contains all tasks with utilization at least γ, the tasks in $\mathcal{T}_i$ have the same period p_i and $\gamma \leqslant \text{util}(\mathcal{T}_i) \leqslant 3 \cdot \gamma$.

The idea is now to treat the tasks in the sets $\mathcal{T}_i$ with period p_i as one single task having period p_i and utilization $\text{util}(\mathcal{T}_i)$. By doing so, we lose some flexibility

to distribute small tasks. Those belonging to one group must be assigned to the same processor. The next theorem establishes that we do not loose too much in terms of optimality, if we again allow processors of speed $1+O(\varepsilon)$. The theorem is proved by applying a Chernoff-type argument.

Theorem 7. *Let $\gamma = \varepsilon^6$ and let $\mathcal{T}$ be a set of tasks which can be partitioned into subsets $\mathcal{T}_{large}, \mathcal{T}_1, \ldots, \mathcal{T}_q$ such that the following conditions hold.*

a) $\mathcal{T}_{large}$ contains all tasks with utilization at least γ.
b) The tasks in $\mathcal{T}_i$ have the same period p_i and $\gamma \leqslant \text{util}(\mathcal{T}_i) \leqslant 3 \cdot \gamma$.

If $\mathcal{T}'$ denotes the instance stemming from identifying each set $\mathcal{T}_i$ as one task with period p_i and running time $\sum_{T \in \mathcal{T}_i} c(T)$, then for $\varepsilon \leqslant \frac{1}{3}$ one can schedule $\mathcal{T}'$ on

$$(1 + O(\varepsilon)) \cdot OPT(\mathcal{T}) + 1$$

machines of speed $1 + O(\varepsilon)$.

Proof. We have to show that there exists a solution which uses at most $(1 + O(\varepsilon))OPT(\mathcal{T}) + 1$ processors of speed $1 + O(\varepsilon)$, in which the tasks of each $\mathcal{T}_i$ are scheduled together on one processor. To do so, consider an optimal schedule for $\mathcal{T}$ which uses the processors $P_1, \ldots, P_k$. Clearly, we can identify the tasks $S \subseteq \mathcal{T}_i$ which are assigned to the same processor P_j into one task with period p_i and processing time $\sum_{T \in S} c(T)$. Therefore, we can assume that each processor contains at most one task from each set $\mathcal{T}_i$. In the new solution the tasks in each set $\mathcal{T}_i$ are scheduled on one processor. This is done using randomization. If a processor does not contain a task from $\mathcal{T}_i$, then $\mathcal{T}_i$ will not be assigned to it. Otherwise suppose that $T \in \mathcal{T}_i$ is assigned to the processor P. The probability that all tasks in $\mathcal{T}_i$ are assigned to P will be $\text{util}(T)/\text{util}(\mathcal{T}_i)$.

For a task $T \in \mathcal{T}$ let E_T be event that $f_T(r_T^*)$ exceeds $(1 + 2\varepsilon) \cdot r_T^*$. We next show that the probability of E_T is bounded by ε. This means that the expected utilization of the tasks which exceed their deadline even on the faster processors is bounded by $\varepsilon \cdot \text{util}(\mathcal{T})$. By removing those tasks and by scheduling them on a set of new processors in a first-fit manner, we only require an additional number of at most $(2/\ln 2) \cdot \varepsilon \cdot OPT(\mathcal{T}) + 1$ processors and the result follows.

We show this first for a task $T \in \mathcal{T}_{large}$. Suppose $T \in \mathcal{T}_{large}$ is assigned to processor P. Let $I \subseteq \{1, \ldots, q\}$ be the set of indices corresponding to the sets $\mathcal{T}_i$ whose tasks T_i on P have higher priority than T. Let $\mathcal{T}'_{large}$ be the set of tasks in $\mathcal{T}_{large}$ that lie on P and have higher priority than T. To bound $\Pr[E_T]$ we inspect the response function

$$f_T(r) = c(T) + \sum_{T' \in \mathcal{T}'_{large}} \left\lceil \frac{r}{p(T')} \right\rceil \cdot c(T') + r \cdot \sum_{i \in I} \frac{p_i}{r} \cdot \left\lceil \frac{r}{p_i} \right\rceil \cdot c(T_i)/p_i.$$

Since T meets its deadline, there exists a number r_T^* with $p(T)/2 \leqslant r_T^* \leqslant p(T)$ and $f_T(r_T^*) \leqslant r_T^*$. From this, we can conclude that the number $a_i = \frac{p_i}{r_T^*} \cdot \lceil r_T^*/p_i \rceil$ satisfies $1 \leqslant a_i \leqslant 2$.

After randomly redistributing the tasks in $\mathcal{T}_1, \ldots, \mathcal{T}_q$ the evaluation of the response function at r_T^* is a random variable of the form

$$c(T) + \sum_{T' \in \mathcal{T}'_{large}} \left\lceil \frac{r_T^*}{p(T')} \right\rceil \cdot c(T') + r_T^* \cdot \sum_{i \in I} a_i \cdot X_i$$

where the $X_i \in \{0, \text{util}(\mathcal{T}_i)\}$ are independent random variables. For $X := \sum a_j X_j$ one has $E[X] \leqslant 1$. It is sufficient to show that $\Pr[X \geqslant E[x] + \varepsilon] \leqslant \varepsilon$. This can be done with a variant of the Chernoff bound. Choose

$$\alpha := \max_i \{a_i \cdot \text{util}(\mathcal{T}_i)\} \leqslant 2 \cdot 3\varepsilon^6 = 6\varepsilon^6.$$

A Chernoff-type argument yields, that

$$\Pr[X \geqslant E[X] + \varepsilon] = \Pr\left[X \geqslant \left(1 + \frac{\varepsilon}{E[X]}\right) E[X]\right] \leqslant e^{-\frac{1}{6\varepsilon^6} \frac{\varepsilon^2}{3E[X]^2} E[X]} \leqslant \varepsilon,$$

where the last inequality follows from $E[X] \leqslant 1$ and $\varepsilon \leqslant 1/3$.

If $T \in \mathcal{T}_i$ for some i, then the above analysis can be applied after the observation that $c(T)$ grows at most up to $3 \cdot \gamma \cdot p(T)$. This can be bounded by $6 \cdot \gamma \cdot r_T^*$ which is bounded by $\varepsilon \cdot r_T^*$. □

By combining the treatment of periods with small utilization and periods with large utilization, we obtain the main result of this section.

Theorem 8. *Let $\mathcal{T}$ be a set of tasks and $\varepsilon < 1/3$. There is a polynomial time algorithm which discards a subset $\mathcal{T}'$ with* $\text{util}(\mathcal{T}') \leqslant O(\varepsilon) \cdot \text{util}(\mathcal{T})$*, constructs an instance $\widetilde{\mathcal{T}}$ such that each task of $\widetilde{\mathcal{T}}$ has utilization of at least ε^6 and $\widetilde{\mathcal{T}}$ can be scheduled on $(1 + O(\varepsilon)) \cdot OPT(\mathcal{T}) + 1$ processors of speed $1 + O(\varepsilon)$. Furthermore, each feasible packing of $\widetilde{\mathcal{T}} \cup \mathcal{T}'$ on k processors of speed $1 + O(\varepsilon)$ yields a feasible packing of the original task set $\mathcal{T}$ on k processors of speed $1 + O(\varepsilon)$.*

Notice that we had to discard tasks of utilization $O(\varepsilon) \cdot \text{util}(\mathcal{T})$ processors three times in this paper. If we collect all discarded tasks and then schedule this tasks once in a first-fit manner, this requires at most $O(\varepsilon) \cdot OPT + 1$ processors. Thus by combining Theorem 8 with Theorem 5, we obtain the main result of this paper.

Theorem 9. *For each $\varepsilon > 0$ there exists a polynomial time algorithm, which partitions a set of tasks $\mathcal{T}$ into $\mathcal{T}_1, \ldots, \mathcal{T}_k$ such that each $\mathcal{T}_i$ is feasible on a processor of speed $1 + O(\varepsilon)$ and $k \leqslant (1 + O(\varepsilon)) \cdot OPT + 1$.*

References

1. Baruah, S., Goossens, J.: Scheduling real-time tasks: Algorithms and complexity. In: Leung, J.Y.-T. (ed.) Handbook of Scheduling — Algorithms, Models, and Performance Analysis, ch. 28. Chapman & Hall/CRC, Boca Raton (2004)

2. Davari, S., Dhall, S.K.: On-line algorithms for allocating periodic-time-critical tasks on multiprocessor systems. Informatica (Slovenia) 19(1) (1995)
3. 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, ch. 32, Chapman & Hall/CRC, Boca Raton (2004)
4. Fernandez de la Vega, W., Lueker, G.S.: Bin packing can be solved within $1 + \varepsilon$ in linear time. Combinatorica 1(4), 349–355 (1981)
5. Fisher, N., Baruah, S.: A fully polynomial-time approximation scheme for feasibility analysis in static-priority systems with arbitrary relative deadlines. In: ECRTS 2005: Proceedings of the 17th Euromicro Conference on Real-Time Systems (ECRTS 2005), pp. 117–126. IEEE Computer Society, Los Alamitos (2005)
6. Garey, M.R., Johnson, D.S.: Computers and Intractability. Freeman, San Francisco (1979)
7. Johnson, D.S.: The NP-completeness column: an ongoing guide. J. Algorithms 13(3), 502–524 (1992)
8. Karmarkar, N., Karp, R.M.: An efficient approximation scheme for the one-dimensional bin-packing problem. In: 23rd annual symposium on foundations of computer science, Chicago, Ill., pp. 312–320. IEEE, New York (1982)
9. Lenstra, H.W.: Integer programming with a fixed number of variables. Mathematics of Operations Research 8(4), 538–548 (1983)
10. Liebeherr, J., Burchard, A., Oh, Y., Son, S.H.: New strategies for assigning real-time tasks to multiprocessor systems. IEEE Trans. Comput. 44(12), 1429–1442 (1995)
11. Liu, C.L., Layland, J.W.: Scheduling algorithms for multiprogramming in a hard-real-time environment. J. ACM 20(1), 46–61 (1973)
12. Oh, Y., Son, S.H.: Allocating fixed-priority periodic tasks on multiprocessor systems. Real-Time Syst. 9(3), 207–239 (1995)
13. Schuurman, P., Woeginger, G.: Approximation schemes – a tutorial. In: Möhring, R.H., Potts, C.N., Schulz, A.S., Woeginger, G.J., Wolsey, L.A. (eds.) Lectures on Scheduling (to appear, 2007)