Performance Debugging of Real-Time Systems using Multicriteria Schedulability Analysis

Unmesh D. Bordoloi Samarjit Chakraborty
Department of Computer Science
National University of Singapore
E-mail: {unmeshdu, samarjit}@comp.nus.edu.sg

Abstract

Most of today's real-time embedded systems consist of a heterogeneous mix of fully-programmable processors, fixed-function components or hardware accelerators, and partially-programmable engines. Hence, system designers are faced with an array of implementation possibilities for an application at hand. Such possibilities typically come with different tradeoffs involving cost, power consumption and packaging constraints. As a result, a designer is no longer interested in one *implementation that meets the specified real-time constraints (i.e. is schedulable), but would rather like to identify* all *schedulable implementations that expose the different possible performance tradeoffs. In this paper we formally define this multicriteria schedulability analysis problem and derive a polynomial-time approximation algorithm for solving it. This result is interesting because the problem of optimally computing even one schedulable solution in our setup (and in most common setups) is computationally intractable (NP-hard). Further, our algorithm is reasonably easy to implement, returns good quality (approximate) solutions, and offers significant speedups over optimally computing all schedulable tradeoffs.*

1 Introduction

It is now well-accepted that debugging real-time embedded systems for non-functional or performance constraints occupy a major chunk of their overall design time. Such systems are increasingly becoming heterogeneous and consist of a mix of fully- and partially-programmable processors, fixed-function hardware accelerators and different kinds of buses and memory modules. Applications to be implemented on such systems are partitioned and mapped onto these different processors and hardware components. This results in a large number of implementation possibilities with different performance tradeoffs. As a result, a designer is no longer interested in *one* implementation that meets the real-time constraints associated with a given application (i.e. is schedulable), but would rather like to identify *all* schedulable implementations that expose the different possible performance tradeoffs.

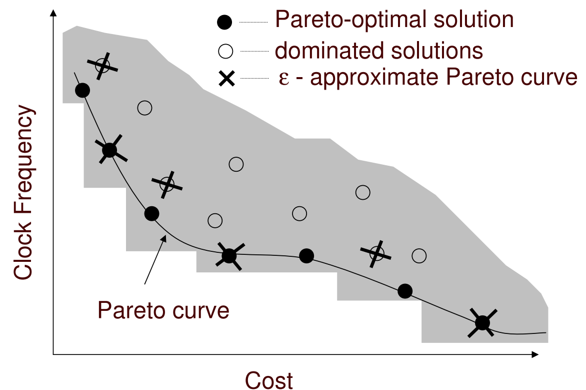


Figure 1. Pareto-optimal solutions.

As a simple example, consider two applications (or tasks) T_1 and T_2 which are required to run concurrently and have predefined deadline constraints. Both T_1 and T_2 can be partially implemented in hardware, with their remaining parts implemented as software running on the same programmable processor P. Such a scheme is in line with coarse-grained FPGA architectures (e.g. Virtex-II PRO from Xilinx), which consist of one or more programmable processors embedded within the FPGA's logic fabric. The portions (or even fractions) of T_1 and T_2 to be implemented in hardware constitute the different implementation options. The two objectives to be optimized are the total hardware cost and the minimum clock frequency of P (which, for example, might influence its power consumption). Clearly, there can be different implementation options which satisfy T_1 and T_2's deadline constraints. If larger fractions of T_1 and T_2 are implemented in hardware, then the hardware cost increases and the required clock frequency of P decreases, and vice versa. For any schedulable implementation, if (c, f) denotes the corresponding hardware *cost* and clock *frequency*, then a designer will be interested in identifying all possible tuples $(c_1, f_1), \ldots, (c_n, f_n)$ which capture the different performance tradeoffs. In the multicriteria optimization parlance, the set $\{(c_1, f_1), \ldots, (c_n, f_n)\}$ is referred to as the *Pareto curve* and each point (c_i, f_i) in this set is called a *Pareto-optimal solution* [9] (see Figure 1).

Each (c_i, f_i) in this set has the property that there does not exist any schedulable implementation of T_1 and T_2 with a performance vector (c, f) such that $c \leq c_i$ and $f \leq f_i$, with at least one of the inequalities being strict. Further, let $\mathcal{S}$ be the set of performance vectors corresponding to all schedulable implementations. Let $\mathcal{P}$ be the set of performance vectors $\{(c_1, f_1), \ldots, (c_n, f_n)\}$ corresponding to all the Pareto-optimal solutions. Then for any $(c, f) \in \mathcal{S} - \mathcal{P}$ there exists a $(c_i, f_i) \in \mathcal{P}$ such that $c_i \leq c$ and $f_i \leq f$, with at least one of these inequalities being strict (i.e. the set $\mathcal{P}$ contains *all* performance tradeoffs). The vectors $(c, f) \in \mathcal{S} - \mathcal{P}$ are referred to as *dominated solutions*, since they are "dominated" by one or more Pareto-optimal solutions as shown in Figure 1.

1.1 Our Contributions

In this paper we present a polynomial-time approximation algorithm for computing the Pareto curve $\mathcal{P} = \{(c_1, f_1), \ldots, (c_n, f_n)\}$. This result is interesting because even the single-criteria version of the problem in very simple settings turns out to be intractable (NP-hard). Given a set of tasks and a processor P running at a predefined clock frequency, the single-criteria version of this problem is to come up with a schedulable (on P) implementation of these tasks with the minimum hardware cost. In other words, the processor has a predefined clock frequency which is provided as an input. Note that the well-studied schedulability analysis problem [3, 7] — where the goal is to decide whether the task set is entirely schedulable on P — is a special case of the single-criteria version of our problem.

The second reason which makes our work interesting is that there can be an exponentially large number of performance vectors (c_i, f_i) in the Pareto curve $\mathcal{P}$, which makes it impossible to compute this entire set in polynomial time. Hence, our polynomial-time approximation algorithm by default also implies approximating the (potentially exponential size) set $\mathcal{P}$ with only a polynomial number of points. In a typical design or performance debugging scenario, a system designer inspects all the tradeoffs in the set $\mathcal{P}$ and then selects one, or at most a few implementations. Hence, from a practical perspective, it is more meaningful if the designer is presented with a reasonably few well-distinguishable tradeoffs in the set $\mathcal{P}$, rather than an exponentially large number of solutions, many of which are very similar to each other. Our approximation algorithm is therefore not only attractive in terms of time-complexity, but also returns more meaningful solutions, as we show later in this paper.

Overview of the proposed scheme: Our proposed scheme takes as an input an error parameter ϵ and returns an *ϵ-approximate Pareto curve* which we denote as ϵ-Pareto curve (or $\mathcal{P}_\epsilon$). Given a Pareto curve $\mathcal{P} = \{(c_1, f_1), \ldots, (c_n, f_n)\}$, an ϵ-approximate Pareto curve is defined as *any* set $\mathcal{P}_\epsilon = \{(c'_1, f'_1), \ldots, (c'_m, f'_m)\}$ such that for any $(c_i, f_i) \in \mathcal{P}$, there exists a $(c'_j, f'_j) \in \mathcal{P}_\epsilon$ for which $c'_j \leq (1 + \epsilon)c_i$ and $f'_j \leq (1 + \epsilon)f_i$. In other words, corresponding to any point on the Pareto curve $\mathcal{P}$, there exists a point on $\mathcal{P}_\epsilon$, each of whose coordinates are at most ϵ distance away from the corresponding coordinates of the point on $\mathcal{P}$. Hence, each "tradeoff" in $\mathcal{P}$ has an "ϵ-approximation" in $\mathcal{P}_\epsilon$, where the semantics of ϵ-approximation are as defined above. In Figure 1, each point on the Pareto curve (denoted by $\bullet$) is approximated by some point (denoted by $\times$) which may be a Pareto-optimal solution or a dominated solution. The set of $\times$ points depend on the value of ϵ, and constitute the set $\mathcal{P}_\epsilon$. Since ϵ is an input provided by the system designer, the error between the approximate and the optimal Pareto curves can be made as small as desired. The running time of our approximation algorithm, as we show later, is polynomial in the size of the problem instance and polynomial in $\frac{1}{\epsilon}$, but exponential in the number of objectives/criteria. However, since the number of objectives is typically small for most real-life problems, this shouldn't pose any problem. Finally, as one might expect, the running time of the algorithm increases as the error parameter ϵ is made smaller.

Our algorithm is made up of the following two parts.

(i) The first part is a polynomial-time approximation algorithm for solving the single-criteria version of the problem. Recall that here we are given a processor P with a predefined clock frequency (or alternatively, a target processor utilization). The goal is to compute a partition of each task such that the portions mapped onto P are schedulable and the total hardware cost is minimized. As we describe later, we assume that each task comes with a specified number of hardware implementation possibilities, i.e. certain subtasks or portions which might be implemented in hardware or in software, and the remaining can only be implemented in software. We believe that this is more realistic than assuming that a task can be arbitrarily partitioned into hardware and software. The approximation algorithm for the single-criteria version takes as an input an error parameter ϵ. It returns a hardware cost which is guaranteed to be no more than $(1 + \epsilon)$ times the minimum cost incurred to schedule the tasks on P with the predefined clock frequency or processor utilization. Alternatively, it says that there does not exist any schedulable implementation of the task set under the possible hardware-implementation options.

(ii) The second part of our algorithm involves imposing a k-dimensional grid on the objective space, where k is the number of objectives being considered. In the case of our bicriteria example (where $k = 2$), this boils down to a rectangular grid. We then (approximately) solve a single-criteria version of our problem

for each grid point by using our approximation algorithm outlined in part (i) and retain only the Pareto-optimal solutions (or rather the "Pareto-optimal grid points"). The crux of this step is in the choice of the grid dimensions, which are also functions of the error parameter ϵ that was used in part (i). By appropriately choosing the grid dimensions, we can guarantee that the approximate Pareto curve is within ϵ distance from the optimal Pareto curve. Further, the number of calls to the approximation algorithm in part (i) is restricted to a polynomial in the problem size and in $\frac{1}{\epsilon}$, but exponential in the number of objectives k.

In summary, both parts (i) and (ii) incur an error in the computation of the Pareto curve. However, the cumulative error is bounded such that the resulting points in the objective space still cover the entire Pareto curve and approximate it with a maximum error of ϵ in all the objectives.

1.2 Related Work

There exists a large body of work on multiobjective optimization [9] and also on multicriteria scheduling and decision making [18]. However, a significant portion of these approaches address the problems from an engineering perspective and relies on heuristics and randomized search techniques such as evolutionary algorithms (e.g. see [8]). Our work in this paper differs from these approaches by taking a classical approximation algorithms standpoint, where the goal is to provide formal guarantees on the quality of the results obtained.

Further, we are also not aware of any work on multicriteria *schedulability analysis* of the form that we present in this paper. Flexible scheduling with multiple concerns is considered to be an important problem in the real-time systems domain (e.g. see [4]). However, to the best of our knowledge, no formal algorithmic solution to this problem is known so far. Our work is also tangentially related to a number of recent papers on performance debugging of real-time and embedded systems from a timing/schedulability analysis perspective. For example, [6, 15, 16, 17, 20] address the problem of *sensitivity analysis* of real-time systems where the goal is to compute permissible changes in certain system parameters that do not result in the required timing/schedulability constraints to be violated. Such changes, especially in a multidimensional setting [16], might be viewed as possible schedulable implementations which are associated with different performance tradeoffs. Similarly, [5] addressed the problem of computing the "schedulable region" or *space* of task periods and worst-case execution times that lead to schedulable systems. The main difference between this line of work and our results is that both sensitivity analysis and schedulable region computation do not explicitly consider schedulable implementation *tradeoffs*, which is our main focus.

The algorithmic techniques presented in this paper have been motivated by [11] and [14]. More specifically, [11] used a partitioning technique to divide a multidimensional objective space into hyper-rectangles – as we do in part (ii) of our scheme – but used it for improving the search quality of a randomized search algorithm. The result that *any* Pareto curve can be ϵ-approximated by a polynomial-size approximate Pareto curve was first proved in [14]. However, for many problems, efficiently (i.e. in polynomial time) computing such approximate Pareto curves might not be possible. Our work in this paper shows that for the multicriteria schedulability analysis problem, such approximate Pareto curves can also be *computed* in polynomial time.

1.3 Organization of this Paper

The rest of the paper is organized as follows. In the next section we introduce our task model and some necessary notations. In Section 3 we then formally define the single-criteria version of the problem, prove that it is NP-hard and derive a polynomial-time approximation scheme for solving it. This is followed by our solution to the multicriteria problem using the approach described in Section 1.1. Some of the experimental results we obtained are described in Section 5. Finally, we conclude in Section 6 by outlining some directions for future work.

For ease of exposition, all the algorithms presented in this paper are for a bicriteria schedulability analysis problem; more specifically, the one we described as an example at the beginning of this paper. However, all our results trivially extend to higher dimensional settings. Similarly, we also considered a simple sporadic task model [3, 12]. Again, it is possible to extend our algorithms to more general task models such as multiframe [13], generalized multiframe [2], and recurring real-time [1] models.

2 Task Model

In this paper, we use the sporadic task model in a preemptive uniprocessor environment to illustrate our approximation scheme. Thus, we are interested in the schedulability analysis of a task set $\tau = \{T_1, T_2, \ldots, T_m\}$ consisting of m hard real-time tasks. Any task T_i can get triggered independently of other tasks in τ. Each task T_i generates a sequence of jobs; each job is characterized by the following parameters:

- *Release Time*: the release time of two successive jobs of the task T_i is separated by a minimum time interval of P_i time units.
- *Deadline*: each job generated by T_i must complete by D_i time units since its release time.
- *Workload*: the worst case execution requirement of any job generated by T_i is denoted by E_i.

Tasks in the Task Set	Workload	Cost
# choices for task $T_1 = 3$ $E_1 = 12, P_1 = 40$	10	15
	8	45
	4	90
# choices for task $T_2 = 2$ $E_2 = 6, P_2 = 16$	5	24
	2	42
# choices for task $T_3 = 3$ $E_2 = 6, P_3 = 25$	8	11
	6	26
	5	82

Table 1. Implementation choices for three different tasks in a task set. Each row of this table shows the new execution requirement (on a programmable processor) because of a part of the task being implemented in hardware, along with the incurred hardware cost.

Throughout this paper, we assume the underlying scheduling policy to be the earliest deadline first (EDF). Again, our algorithm can be suitably modified to handle other scheduling policies as well. Assuming that for all tasks T_i, $D_i \geq P_i$, the schedulability of the task set τ can be given by the following condition.

Theorem 1 *A set of sporadic tasks τ is schedulable under EDF if and only if*

$$(U = \sum_{i=1}^{m} \frac{E_i}{P_i}) \leq 1$$

where U is the processor utilization due to τ [3, 12].

3 The Single-Criteria Problem

In this section, we formally state the single-criteria version of the problem along with an illustrative example. We then show that this problem is intractable even for the simple sporadic task model described in Section 2. Finally, we derive a fully polynomial-time approximation scheme (FPTAS) [10] for solving it.

Recall that we are given a processor P with a predefined clock frequency, and a specified number of subtasks of each task T_i which can be implemented in hardware. Our goal is to identify the implementation choices that lead to the minimum hardware cost, provided the portions of the tasks mapped onto P are schedulable.

If the task set is entirely schedulable on the processor P (i.e. $U \leq 1$), then the problem is trivial and we need not incur any hardware costs. However, if the entire task set is not schedulable on P, then certain portions of some of the tasks in the set will have to be implemented in hardware to reduce the load on P. The problem is then that of identifying which portions or subtasks of each task should be mapped onto hardware such that the minimum hardware cost is incurred.

For each task T_i, let there be n_i hardware implementation choices. Each of these n_i choices is associated with a certain hardware cost. Choosing the jth implementation choice for the task T_i lowers its execution requirement on P from E_i to $e_{i,j}$. Equivalently, the amount by which the execution requirement of T_i gets lowered on P is $\delta_{i,j} = E_i - e_{i,j}$. Hence, for each task T_i we have a set of choices $S_i = \{(\delta_{i,1}, c_{i,1}), \ldots, (\delta_{i,n_1}, c_{i,n_1})\}$, where $c_{i,j}$ is the hardware cost associated with the jth implementation choice. The goal is to identify one choice for each task, which would lower the processor utilization to less than or equal to one, and minimize the total hardware cost. In what follows, we shall refer to this as the *minimum cost schedulability analysis* problem.

We now illustrate this problem with the help of an example. A task set τ has three tasks $\{T_1, T_2, T_3\}$ with $\{E_1 = 12, P_1 = 40\}$, $\{E_2 = 6, P_2 = 16\}$, and $\{E_3 = 11, P_3 = 25\}$. Clearly the processor utilization $U > 1$ and hence this task set is not schedulable, without some of the subtasks being mapped onto hardware. The different possible hardware implementation choices for each task in this set is shown in Table 1. Each row of this table shows the new execution requirement of a task on P after a part of this task is implemented in hardware, and the associated hardware cost. Note that as the execution requirement or workload of a task decreases, its associated hardware cost increases.

Following the notation we introduced above, for T_1 we have $e_{1,1} = 10$, $e_{1,2} = 8$ and $e_{1,3} = 4$. The corresponding hardware costs are $c_{1,1} = 15$, $c_{1,2} = 45$ and $c_{1,3} = 90$. Hence, the implementation choices for T_1 are given by the set $S_1 = \{(2, 15), (4, 45), (8, 90)\}$. The choices for T_2 and T_3 can be similarly computed from this table. Note that while T_1 and T_3 have three choices each, T_2 has only two choices. Thus, $n_1 = n_3 = 3$ and $n_2 = 2$. The goal is to select one choice from each set S_1, S_2 and S_3, such that we obtain a minimum-cost schedulable system.

3.1 NP-hardness

We show that the minimum cost schedulability analysis problem is NP-hard using a polynomial-time transformation from the 0-1 knapsack problem [10].

Theorem 2 *The minimum cost schedulability analysis problem is NP-hard.*

Proof: The decision version of the minimum cost schedulability analysis problem asks whether there is a set of choices of the execution requirements such that the condition $U \leq 1$ is satisfied, and the total cost is $\leq C$.

The knapsack problem specifies m items with integral weights w_i and profits p_i, $i = 1, 2, \ldots, m$, an integral weight constraint W and a profit goal G. Let the m binary variables $x_i \in \{0, 1\}$ correspond to the selection of the ith item. The knapsack decision problem asks if there exists a

subset of items, the sum of whose profits $\sum_{i=1}^{m} p_i x_i \geq G$ and the sum their weights is $\sum_{i=1}^{m} w_i x_i \leq W$.

We transform the knapsack problem into a special instance of our problem which is obtained by setting $n_i = 1$, for $i = 1, 2, \ldots, m$. Towards this, let $\delta_{i,1} = p_i$ and $c_{i,1} = w_i$. Hence, corresponding to each item i in the knapsack problem with weight w_i and profit p_i, there is a task T_i with $\delta_{i,1} = p_i$ and cost $c_{i,1} = w_i$. For this problem instance, let all the m tasks in the task set τ have the same deadline D. Further, let all the periods be equal to their deadlines, i.e. $P_i = D$ for all $\{i = 1, 2, \ldots, m\}$. The values D and E_i are chosen such that the $\sum E_i - D = G$. Our claim is that the minimum cost schedulability analysis decision problem returns a *Yes* answer if and only if the knapsack problem returns a *Yes*. To verify this, let us first consider the *if* direction. This immediately implies that $\sum_{i=1}^{m} c_i x_i \leq C$ for the problem instance we constructed. For our special instance, where $n_i = 1$, the binary variable $x_i \in \{0, 1\}$ corresponds to the selection of the $\{i, 1\}$th choice. Again, a solution to the knapsack problem also implies that:

$$
\begin{aligned}
&\sum_{i=1}^{m} p_i x_i \geq G \\
\Rightarrow &\sum \delta_{i,1} x_i \geq \sum E_i - D \\
\Rightarrow &\sum E_i - \sum \delta_{i,1} x_i \leq D \\
\Rightarrow &(E_1 - \delta_{1,1} x_1) + (E_2 - \delta_{2,1} x_2) + \cdots + (E_m - \delta_{m,1} x_m) \leq D \\
\Rightarrow &U \leq 1
\end{aligned}
$$

The claim can be similarly verified in the other direction. Thus, the special case of the minimum cost schedulability analysis problem is NP-hard and the theorem follows. □

3.2 Approximating the Minimum Cost Schedulable Solution

In this section we first present a dynamic programming algorithm (Algorithm 1) to compute the minimum cost that must be incurred to obtain a schedulable task set. This algorithm runs in pseudo-polynomial time. We then use this algorithm to derive a *fully polynomial-time approximation scheme* (FPTAS) for the same problem.

Let $U_{i,j}$ be the minimum utilization that might be achieved by considering only a subset of tasks from $\{1, 2, \ldots, i\}$ when the cost is exactly j. If no such subset exists we set $U_{i,j} = \infty$. Let the maximum cost be C i.e. $C = max_{(i=1,2,\ldots,n; j=1,2,\ldots,n_i)} c_{i,j}$. Clearly, mC is an upper bound on the total cost that might be incurred. All other notations used are as introduced in Section 3.

Lines 1 to 5 of Algorithm 1 initialize $U_{0,0}$ to $\sum_{i=1}^{m} E_i/P_i$, and $U_{0,j}$ to ∞ for $j = \{1, 2, \ldots, mC\}$. The

Algorithm 1 Minimum-cost schedulability analysis

Input: The task set τ, and a set S_i for each task T_i.

```
1:  U_{0,0} ← Σ_{i=1}^{m} E_i/P_i
2:  tag_j ← 1
3:  for j ← 1 to mC do
4:     U_{0,j} ← ∞
5:     tag_j ← 1
6:  end for
7:  for i ← 1 to m do
8:     for j ← 0 to mC do
9:        if tag_j = 1 then
10:          U_{i,j} = U_{i-1,j}
11:       else
12:          tag_j = 1
13:       end if
14:       For each pair (δ_{i,k}, c_{i,k}) that belongs to the set S_i
15:       if (j + c_{i,k}) ≤ mC then
16:          if tag_{j+c_{i,k}} = 1 then
17:             U_{i,j+c_{i,k}} ← min{U_{i-1,j+c_{i,k}}, U_{i-1,j} − δ_{i,k}/P_i}
18:             if U_{i,j+c_{i,k}} = U_{i-1,j} − δ_{i,k}/P_i then
19:                tag_{j+c_{i,k}} = 0
20:             end if
21:          else
22:             U_{i,j+c_{i,k}} ← min{U_{i,j+c_{i,k}}, U_{i-1,j} − δ_{i,k}/P_i}
23:          end if
24:       end if
25:    end for
26: end for
27: MinCost ← min{j | U_{n,j} ≤ 1}
```

values $U_{i,j}$ for $i = 1$ to $i = m$ are computed using the iterative procedure in lines 7 to 26. For an iteration where $(i = i')$ and $(j = j')$, we say that $U_{i',j'+c_{i,k}}$ is *updated* using the recursive computation in lines 16 to 23 where $U_{i',j'+c_{i,k}}$ is assigned the value $\{U_{i'-1,j} - \delta_{i',k}/P_{i'}\}$. Thus, $U_{i',j'+c_{i,k}} \neq U_{i'-1,j'+c_{i,k}}$, i.e. $U_{i',j'+c_{i,k}}$ does not carry the value from the previous iteration but is updated with a new value. When such an updated entry is accessed after a few iterations (i.e. when $j = j' + c_{i,k}$), this updated value should not get re-initialized to its previous value (line 10). This is taken care of in lines 9 to 12 with the help of the *if-else* conditional statements on the variable tag_j . Towards this, the value tag_j is set to 0 for updated entries in lines 18 to 19. It can be easily verified that the running time of Algorithm 1 is $O(nmC)$, where $n = \sum_{i=1}^{m} n_i$, and its space complexity is $O(m^2 C)$.

Next, we present an FPTAS for the minimum cost schedulability analysis problem. Towards this, we divide the cost space between 1 and mC into $O(n \log_{1+\epsilon} mC)$ intervals as $(1, (1+\epsilon)^{1/n}], ((1+\epsilon)^{1/n}, (1+\epsilon)^{2/n}], \ldots$.

Our FPTAS is based on Algorithm 1. But instead of running it for all possible cost values, from 0 to mC, we only consider the value 0 and the upper end points of the par-

titioned intervals we described above. Let $\widetilde{U}_{i,\widetilde{j}}$, represent the utilization value with the cost *at most* $\widetilde{j}$, where $\widetilde{j}$ always takes the value 0 or the value of one of the upper endpoints of the abovementioned intervals. During the iteration for the current entry $\widetilde{U}_{i,\widetilde{j}}$, the following procedure is executed for the recursive equations in lines 17 and 22 of Algorithm 1. The cost of $[\widetilde{j} + c_{i,j}]$ is rounded up to the next upper endpoint $\widetilde{u}$. The value in this entry i.e. $\widetilde{U}_{i-1,\widetilde{u}}$ is compared with $\widetilde{U}_{i,\widetilde{j}} - \delta_{i,j}/P_i$, and the minimum of the two is stored in $\widetilde{U}_{i,\widetilde{u}}$. This explains how to update the main recursive equation in lines 15 and 17. The value for tag_j can be updated in a similar way (lines 19 and 20). The running time of the resulting algorithm is $O(n^2 \log_{1+\epsilon} mC)$ (which is bounded by a polynomial in the problem size and in $\frac{1}{\epsilon}$).

Theorem 3 *Our approximation algorithm for the minimum cost schedulability analysis problem is a* $(1+\epsilon)$*-approximation scheme.*

Proof: The algorithm can be proved to be a $(1+\epsilon)$-approximation scheme if we can show that $\widetilde{j} \leq (1+\epsilon)j$. This is achieved by proving the following two properties for all values of i and j.

Property 1: $\widetilde{U}_{i,\widetilde{j}} \leq U_{i,j}$
Property 2: $\widetilde{j} \leq (1+\epsilon)^{i/m} j$

We will prove these two properties using induction on i. First, consider the items only from the task T_1, i.e. $i = 1$. This implies that in the exact algorithm, there will be an update for $U_{i,p_{1,k}}$ corresponding to all $k = \{1, 2, \ldots, n_1\}$. Property 1 holds by equality. We can easily verify that $\widetilde{j}/(1+\epsilon)^{1/n} \leq c_{i,k}$ (Property 2) follows from the fact that $\widetilde{j}$ is the upper bound in the same interval as j, and hence $\widetilde{j}/(1+\epsilon)$ will definitely be in the preceding interval.

Let us now consider the induction step for any $i > 1$, assuming that both the properties hold true for $i-1$, i.e. we have dealt with the tasks T_1 to T_{i-1}. This step considers the pairs $(\delta_{i,k}, c_{i,k})$ in the set S_i. The entries in the array $U_{i,j}$ which are not updated definitely satisfy both the properties.

Now consider entries which are updated i.e. an item $(\delta_{i,k}, c_{i,k})$ was added to $U_{i,j}$ such that $U_{i,j+c_{i,k}}$ was updated. Since the claim is true for $i-1$, there exists $\widetilde{U}_{i-1,\widetilde{j}}$ such that $\widetilde{j} \leq (1+\epsilon)^{i-1/n} j$. Now, we consider $c_{i,k}$ which will be added to $\widetilde{j}$, and is rounded up to $\widetilde{u}$. Given the manner in which we constructed our intervals, we have

$$
\begin{aligned}
\widetilde{u}/(1+\epsilon)^{1/n} &\leq \widetilde{j} + c_{i,k} \leq \widetilde{j}(1+\epsilon)^{i-1/n} + c_{i,k} \\
\widetilde{u}/(1+\epsilon)^{1/n} &\leq (\widetilde{j} + c_{i,k})(1+\epsilon)^{i-1/n} \\
&\Rightarrow \widetilde{u} \leq (\widetilde{j} + c_{i,k})(1+\epsilon)^{i-1/n}
\end{aligned}
$$

Property 1 can be verified using the following steps, where the inequality holds by induction.

$$
\begin{aligned}
\widetilde{U}_{i,\widetilde{j}} &= min\{\widetilde{U}_{i,\widetilde{j}}, U_{i-1,\widetilde{j}} - \delta_{i,\widetilde{j}}/P_i\} \\
&\leq \{U_{i-1,j} - \delta_{i,j}/P_i\} = U_{i-1,j+c_{i,k}}
\end{aligned}
$$

□

4 Multicriteria Schedulability Analysis

In the previous section we addressed the single-criteria version of the problem, namely we assumed the processor's clock frequency to be prespecified. In this section, we relax this assumption and present a scheme to compute the *Pareto curve* containing the *Pareto-optimal* set of performance vectors $\{(c_1, f_1), \ldots, (c_n, f_n)\}$, where (c_i, f_i) denotes the hardware *cost* and the clock *frequency* for a particular schedulable implementation (recall the definition of Pareto optimality from Section 1).

For simplicity of exposition, we will henceforth assume that the processor P's clock frequency is constant and all the execution times of the tasks are specified with respect to this clock frequency. Our objective will be to minimize P's utilization (by mapping certain subtasks onto hardware) and at the same time also minimize the total hardware cost. In other words, our goal is to compute the *cost-utilization* Pareto curve $\{(c_1, u_1), \ldots, (c_n, u_n)\}$ for a prespecified clock frequency of P. It is straightforward to see that such a Pareto curve can be easily transformed into a *cost-frequency* Pareto curve with P's utilization being ≤ 1 for the different frequency values.

Unfortunately, computing the exact *cost-utilization* Pareto curve is computationally intractable. This can be easily verified from the following two facts. First, the Pareto curve would typically contain an exponential number of points (which obviously cannot be computed in polynomial time). Second, computing any one point on the Pareto curve is NP-hard, as we showed in Section 3. Hence, our goal is to *approximately* compute this curve in polynomial time.

Recent work by Papadimitriou and Yannakakis [14] has shown that for any multiobjective optimization problem, there exists a polynomial-sized ϵ-approximate Pareto curve $\mathcal{P}_\epsilon$ for any given ϵ. Further, [14] showed that a necessary and sufficient condition for computing such a $\mathcal{P}_\epsilon$ in polynomial time is the existence of a polynomial-time algorithm for solving, what was referred to as the *GAP problem.* In what follows, we state the version of the GAP problem that arises in our setting and show that it can be solved in polynomial time.

4.1 The GAP Problem

For a two-dimensional multiobjective optimization problem, the GAP problem can be stated as follows: Given a vector $b = (b_1, b_2)$, either return a solution whose vector dominates b, or report that there is no solution whose

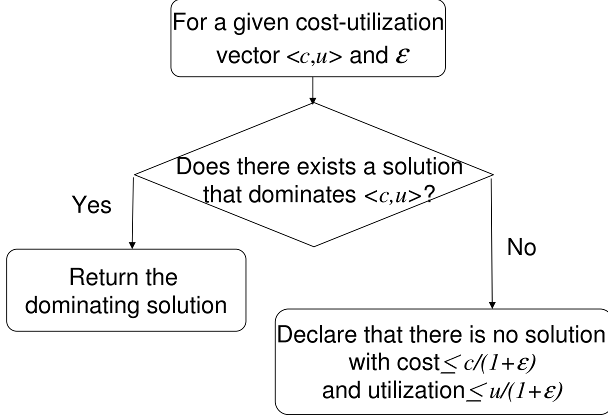


Figure 2. The GAP problem corresponding to our *cost-utilization* tradeoff problem.

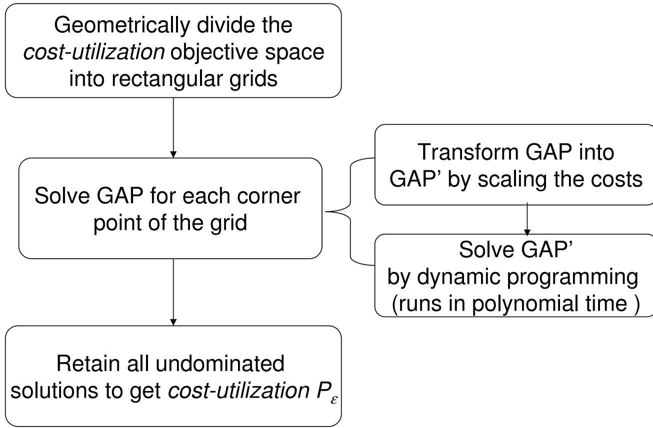


Figure 3. An FPTAS for computing $\mathcal{P}_\epsilon$ using an algorithm for solving GAP.

vector is better than b by at least a factor of $1 + \epsilon$ in both dimensions. In our setup, the objective is to minimize the utilization $U(S) = \sum_{i=1}^{m} \frac{E_i - x_{i,j}\delta_{i,j}}{P_i}$ and the cost $C(S) = \sum_{i=1}^{m} c_{i,j}x_{i,j}$, where S is the chosen implementation among the various available options (see Table 1). Hence, the corresponding GAP problem can be stated as: Given a cost c, a utilization u and an $\epsilon \geq 0$, either return a solution S such that $C(S) \leq c$ and $U(S) \leq u$, or else declare that there is no solution S such that $C(S) \leq \frac{c}{1+\epsilon}$ and $U(S) \leq \frac{u}{1+\epsilon}$ (see Figure 2). In this section, we will show that there exists a polynomial-time algorithm to solve this GAP problem.

Note that a polynomial-time algorithm to solve the GAP problem implies an FPTAS for computing P_ϵ. This is because the following FPTAS can be devised using the algorithm for solving GAP (shown schematically in Figure 3). First, geometrically partition the objective space along all dimensions with a ratio $1+\epsilon'$, where $\epsilon' = (1+\epsilon)^{1/2} - 1$. For each corner point of this grid, call the GAP routine (i.e. the

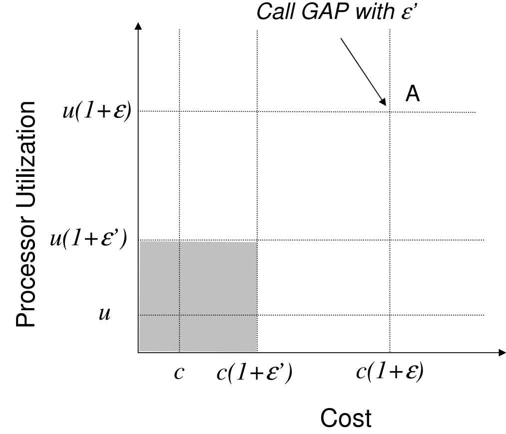


Figure 4. Solving the GAP problem for the corner point A will either return a dominating solution or declare that there is no solution in the shaded area.

algorithm for solving GAP) with the parameter ϵ', and keep all the undominated solutions (see Figure 4 for an illustration of this procedure). This implies that for each rectangle which contains a solution in the exact Pareto curve, there will also be a solution within the same rectangle which belongs to $\mathcal{P}_\epsilon$. The *distance* between these two solutions can be bounded using the dimensions of the rectangle. Hence, for every solution s in the Pareto curve, there exists a solution q in $\mathcal{P}_\epsilon$ such that $\frac{q}{(1+\epsilon)} \leq s$. Moreover, because the number of rectangles is polynomially bounded, it follows that the number of points in $\mathcal{P}_\epsilon$ will also be a polynomial.

Theorem 4 *There exists an algorithm for constructing the cost-utilization ϵ-Pareto curve, which runs in time polynomial in the size of the input and in $\frac{1}{\epsilon}$.*

Proof: As discussed above, a necessary and sufficient condition for the existence of an FPTAS for computing the approximate cost-utilization Pareto curve $\mathcal{P}_\epsilon$ is that the following GAP problem should be solvable in time, which is polynomial in the input size and in $1/\epsilon$.

Problem Statement: Given a cost c, utilization u and an $\epsilon \geq 0$ either return a solution S such that $C(S) \leq c$ and $U(S) \leq u$, or else declare that there is no solution S such that $C(S) \leq \frac{c}{1+\epsilon}$ and $U(S) \leq \frac{u}{1+\epsilon}$.

Solution to the GAP Problem: We now present a polynomial-time algorithm to solve this GAP problem. It involves the following two steps:

- *Transforming Costs*
  Let $r = \lceil \frac{m}{\epsilon} \rceil$. Modify each cost $c_{i,j}$ to $c'_{i,j}$ such that $c'_{i,j} = \lceil \frac{c_{i,j}r}{c} \rceil$. This leads to the following properties:

  (a) If a solution with the transformed costs satisfies $C'(S) \leq r$, then $C(S) \leq c$.

Proof of Property (a):

$$\sum c'_{i,j}x_{i,j} = \sum \left\lceil \frac{c_{i,j}x_{i,j}r}{c} \right\rceil \geq \frac{r}{c}\sum c_{i,j}x_{i,j}$$

Hence, $C'(S) \leq r \Rightarrow \frac{r}{c}\sum c_{i,j}x_{i,j} \leq r$

This implies that $C(S) \leq c$.

(b) If the solution satisfies $C(S) \leq \frac{c}{1+\epsilon}$, then $C'(S) \leq r$.

Proof of Property (b):

$$C(S) \leq \frac{c}{1+\epsilon} \Rightarrow \sum c_{i,j}x_{i,j} \leq \frac{c}{1+\epsilon}$$
$$\Rightarrow \sum \frac{c_{i,j}x_{i,j}}{c}\frac{m}{\epsilon} \leq \frac{m}{\epsilon(1+\epsilon)}$$
$$\Rightarrow \sum \left\lceil \frac{c_{i,j}x_{i,j}r}{c} \right\rceil \leq \left\lceil \frac{m}{\epsilon(1+\epsilon)} \right\rceil$$
$$\Rightarrow C'(S) \leq \left\lceil \frac{m}{\epsilon} \right\rceil = r \Rightarrow C'(S) \leq r$$

Consider the problem of determining if there exists a solution with the modified costs such that $C'(S) \leq r$. Let us call this problem GAP′. From property (a), we know that if this problem returns an affirmative answer then the GAP problem would also return a dominating solution. On the other hand, if GAP′ returns a negative answer then property (b) leads to the conclusion that there is no solution with cost $\leq c/(1+\epsilon)$. Hence, from the above properties we can infer that solving GAP′ is equivalent to solving the original GAP problem.

- *Solving GAP′*

  We present a dynamic programming algorithm to solve the GAP′ problem. This algorithm can be constructed with the following adjustments to Algorithm 1.

  1. Run Algorithm 1 with the modified costs $c'_{i,j}$.
  2. Instead of iterating over all the cost values up to mC, iterate only up to a cost value of at most r.
  3. Finally, if the minimum value in the final array $U_{n,\{1,\ldots,r\}}$ is such that it is $\leq u$, then return the solution otherwise declare that there is no solution.

  Computing each row of the table built by this dynamic programming algorithm requires $O(n_i r)$ running time. Hence, this algorithm runs in time $O(nm/\epsilon)$, where $n = \sum n_i$.

Hence, a polynomial-time algorithm exists for solving the GAP problem, which in turn proves our theorem. □

Now that we have presented the GAP subroutine for our problem, we can present the full algorithmic details for computing the *cost-utilization* Pareto curve. Recall that we already outlined this scheme in Figure 3. Algorithm 2 specifies the steps to compute the ϵ-approximate *cost-utilization* Pareto curve in some more detail. Note, that in step 1 of Algorithm 2 we partition only the cost space (and not both utilization and cost space). This is because if a point (c, u) dominates the corner (c_1, u_1) and $u_1 < u_2$, then (c, u) definitely dominates (c_1, u_2) . In steps 2 and 3, we scale the costs, run Algorithm 1 for every co-ordinate in the partitioned cost space and retain the minimum utilization at each co-ordinate. The runtime complexity of this algorithm is $O(\frac{nm}{\epsilon} log_{1+\epsilon} mC)$.

Algorithm 2 Approximating the Pareto curve.

1: Partition the range of costs from 1 to mC geometrically with a ratio $1 + \epsilon' = (1 + \epsilon)^{1/2}$, thus dividing the cost space into $O(log_{1+\epsilon} mC)$ coordinates.

2: For each coordinate b, call *Algorithm 1* with transformed costs $c'_{i,j} = \lceil \frac{c_{i,j}r}{b} \rceil$, where $r = \lceil \frac{m}{\epsilon'} \rceil$.

3: For each run of Step 2, find the solution with the minimum utilization.

4: Retain all the undominated solutions from the solutions found in Step 3. This will represent a ϵ-Pareto curve.

5 Experimental Results

In this section we report some of the experimental results that we obtained by running our approximation algorithm on a set of synthetic task sets. (Note that to compute the exact Pareto curve, we need to run the Algorithm 1 and then retain all the undominated solutions.) We also compared these results with those obtained by running the optimal algorithm. In Section 5.1 we show the running times of the optimal and the approximation algorithms. In Section 5.2 we illustrate the difference in the sizes of $\mathcal{P}_\epsilon$ and the exact Pareto curve.

For our experiments we randomly generated tasks with execution requirements between 200 and 600 time units; the periods were between 600 and 20000 time units. The number of hardware implementation choices associated with any task was varied between 1 and 10, i.e. $1 \leq n_i \leq 10$. The parameter C, which is the maximum cost associated with any implementation choice was set to 5000 for our experiments. For each choice, the maximum value associated with any $\delta_{i,j}$ was set to E_i.

All the CPU times reported below were measured on a machine with Windows XP, running on a 3.0 GHz CPU with 1 GB RAM. All the implementations were done in C++.

5.1 Running Times

Figure 5 shows the running times involved in computing the exact Pareto curve and the FPTAS for three different values of ϵ when the number of tasks in the task set is progressively increased from 10 to 50. These task sets were

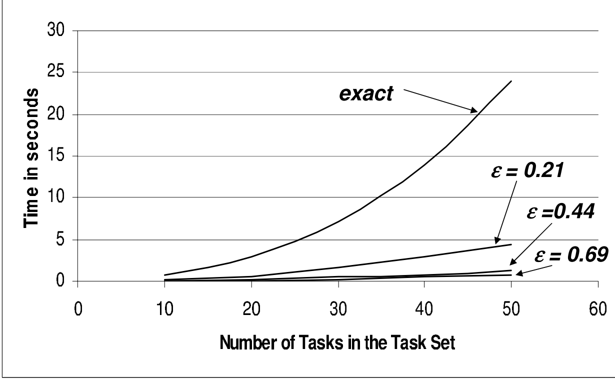


Figure 5. Graph comparing the running times of the exact and the approximate algorithms for various task sets with $C = 5000$.

generated with the parameter $C = 5000$. It can be seen that even for small values of ϵ (e.g. when $\epsilon = 0.69$) the approximate algorithm runs about 20 times faster than the exact algorithm. For larger values of ϵ (e.g. $\epsilon = 3$), the speedups are even more significant (note that ϵ need not be ≤ 1).

The reason behind choosing the values 0.21, 0.44, and 0.69 for ϵ is as follows. Our approximation algorithm involves the computation of the value $(1+\epsilon)^{1/2}$. This value might turn out to be an irrational number if ϵ is not carefully chosen. Hence, to avoid any possible rounding-off errors in our implementation, the above values were chosen for ϵ.

5.2 Size of the Pareto Curves

As discussed in Section 4, the *cost-utilization* Pareto curve typically contains an exponential number of points. The approximation algorithm generates a polynomial-sized ϵ-approximate Pareto curve. In this section, we compare the number of points in the exact Pareto curve and in $\mathcal{P}_\epsilon$. To help visualize the difference in their sizes, Figure 6 shows the exact Pareto curve and the $\mathcal{P}_\epsilon$ generated by our algorithm for task sets with 10 tasks in each set and $C = 5000$.

The following two observations can be easily visualized from these graphs: (i) the number of points in $\mathcal{P}_\epsilon$ decrease with a corresponding increase in the value of ϵ, and (ii) the gap between the exact and approximate curves widens with larger values of ϵ, implying that the relative error indeed increases.

These graphs show the Pareto curves for a task set with 10 tasks. Table 2 lists the number of points in the exact Pareto curve and in $\mathcal{P}_\epsilon$ for task sets with 10, 20, 30, 40 and 50 tasks. From this table it can once again be seen that as the relative error is allowed to increase, the size of the approximate Pareto curve decreases. Note from Table 2 that for small values of ϵ (e.g. $\epsilon = 0.21$), the size of the $\mathcal{P}_\epsilon$ contains up to 96% less points compared to the optimal Pareto curve.

Task Sets	ϵ	# Points on P_ϵ
# tasks in the task set $\tau_1 = 10$ # points in the exact Pareto curve $= 62$	0.21	40
	0.44	26
	0.69	22
	3	9
# tasks in the task set $\tau_2 = 20$ # points in the exact Pareto curve $= 239$	0.21	60
	0.44	38
	0.69	26
	3	11
# tasks in the task set $\tau_3 = 30$ # points in the exact Pareto curve $= 828$	0.21	63
	0.44	37
	0.69	27
	3	12
# tasks in the task set $\tau_4 = 40$ # points in the exact Pareto curve $= 1061$	0.21	76
	0.44	44
	0.69	31
	3	12
# tasks in the task set $\tau_5 = 50$ # points in the exact Pareto curve $= 2033$	0.21	72
	0.44	42
	0.69	30
	3	12

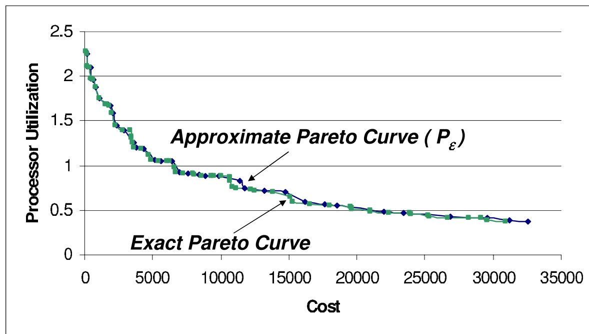
Table 2. Number of points in $\mathcal{P}_\epsilon$ generated by our proposed approximation algorithm, versus the number of points in the optimal Pareto curve. This table shows the results for five task sets generated with $C = 5000$, where each set contains between 10-50 tasks. The numbers in the rightmost column are the number of points in $\mathcal{P}_\epsilon$ when the value of ϵ is set to 0.21, 0.44, 0.69, and 3.

6 Concluding Remarks

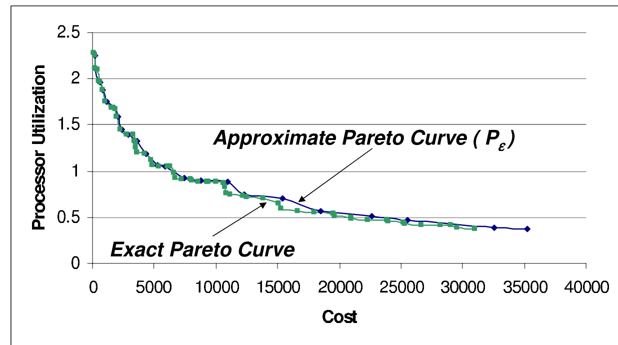
In this paper we introduced a multicriteria version of the classical schedulability analysis problem. We showed that this problem is NP-hard even for simple task models and presented an approximation algorithm for solving it. Our approximation algorithm is not only computationally efficient, but also returns more meaningful results from a practical performance-debugging perspective (as discussed in Section 1.1).

There are a number of directions in which our work can be extended. The most notable among these being a possible extension of our scheme to account for communication costs and dependencies between parts of a task, some of which are implemented in hardware and the remaining in software. Such details were abstracted away in this paper for the sake of a clean theoretical formulation.

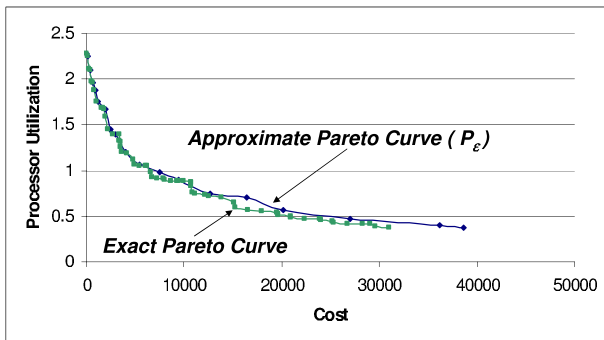
Lastly, it may be noted that although our algorithm generated polynomial-sized $\mathcal{P}_\epsilon$ curves, they need not necessarily contain the fewest possible points required to represent an ϵ-approximate Pareto curve. It would be interesting to see whether it is possible to generate the smallest sized $\mathcal{P}_\epsilon$ in our setting, based on the recent results from [19].



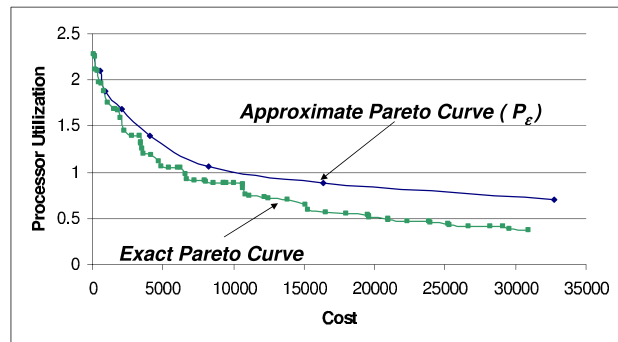
(a) $\epsilon = 0.21, C = 5000$



(b) $\epsilon = 0.44, C = 5000$



(c) $\epsilon = 0.69, C = 5000$



(d) $\epsilon = 3, C = 5000$

Figure 6. The exact and approximate Pareto curves for a task set with 10 tasks and $C = 5000$, for different values of ϵ.

References

[1] S. Baruah. Dynamic- and static-priority scheduling of recurring real-time tasks. *Real-Time Systems*, 24(1):93–128, 2003.

[2] S. Baruah, D. Chen, S. Gorinsky, and A. Mok. Generalized multiframe tasks. *Real-Time Systems*, 17(1):5–22, 1999.

[3] S. Baruah, A. Mok, and L. Rosier. Preemptively scheduling hard-real-time sporadic tasks on one processor. In *IEEE Real-Time Systems Symposium (RTSS)*, 1990.

[4] G. Bernat and A. Burns. Three obstacles to flexible scheduling. In *Euromicro Conference on Real-Time Systems (ECRTS)*, 2001.

[5] E. Bini and M. D. Natale. Optimal task rate selection in fixed priority systems. In *IEEE Real-Time Systems Symposium (RTSS)*, 2005.

[6] E. Bini, M. D. Natale, and G. C. Buttazzo. Sensitivity analysis for fixed priority real-time systems. In *Euromicro Conference on Real-Time Systems (ECRTS)*, 2006.

[7] G. Buttazzo. *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*. Kluwer Academic Publishers, Boston, 1997.

[8] C. A. C. Coello, A. H. Aguirre, and E. Zitzler, editors. *Proc. Third International Conference on Evolutionary Multi-Criterion Optimization*. Lecture Notes in Computer Science 3410, Springer-Verlag, 2005.

[9] K. Deb. *Multi-Objective Optimization Using Evolutionary Algorithms*. John Wiley & Sons, 2001.

[10] D. Hochbaum, editor. *Approximation Algorithms for NP-Hard Problems*. PWS Publishing Company, Boston, 1997.

[11] M. Laumanns, L. Thiele, K. Deb, and E. Zitzler. Combining convergence and diversity in evolutionary multiobjective optimization. *Evolutionary Computation*, 10(3):263–282, 2002.

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

[13] A. Mok and D. Chen. A multiframe model for real-time tasks. *IEEE Transactions on Software Engineering*, 23(10):635–645, 1997.

[14] C. H. Papadimitriou and M. Yannakakis. On the approximability of trade-offs and optimal access of web sources. In *Foundations of Computer Science (FOCS)*, 2000.

[15] S. Punnekkat, R. Davis, and A. Burns. Sensitivity analysis of real-time task sets. In *Asian Computing Science Conference on Advances in Computing Science (ASIAN)*, 1997.

[16] R. Racu, A. Hamann, and R. Ernst. A formal approach to multi-dimensional sensitivity analysis of embedded real-time systems. In *Euromicro Conference on Real-Time Systems (ECRTS)*, 2006.

[17] R. Racu, M. Jersak, and R. Ernst. Applying sensitivity analysis in real-time distributed systems. In *Real-Time and Embedded Technology and Appl. Symp. (RTAS)*, 2005.

[18] V. T'kindt, J. C. Billaut, and H. Scott. *Multicriteria Scheduling : Theory, Models and Algorithms*. Springer, 2006.

[19] S. Vassilvitskii and M. Yannakakis. Efficiently computing succinct trade-off curves. *Theoritical Computer Science*, 348(2), 2005.

[20] S. Vestal. Fixed-priority sensitivity analysis for linear compute time models. *IEEE Trans. Softw. Eng.*, 20(4):308–317, 1994.