Optimal Task Rate Selection in Fixed Priority Systems*

Enrico Bini, **Marco Di Natale**
Scuola Superiore Sant'Anna
{e.bini,marco}@sssup.it

Abstract

The design phase of any real-time system requires balancing the limited computational resources against the functional requirements and the performance of the application. The optimal design solution can be obtained by solving an optimization problem where the system performance is maximized within the schedulability constraints.

In this paper, we provide a procedure that finds the task activation rates maximizing a performance function within the deadline constraints in systems scheduled by fixed priorities. First, we describe the exact feasibility region in the domain of task frequencies. Then, we introduce a procedure that starts by finding an initial solution, and incrementally improves it by using an original branch and bound search, until the global optimum is reached.

Experiments show that our algorithm finds the optimal task periods for practical problems with a remarkable speedup, if compared with existing techniques. When the size of the problem makes the global search intractable, the experiments show that the algorithm can still find a high quality solution in the very early steps.

1. Introduction

It is common practice to develop real-time systems in two stages according to the well-established paradigm of *separation of concerns*. In the first stage, at the *logical level* [16, 15, 18, 3] application developers address the functional requirements and possibly annotate the design with timing constraints. In the second stage, at the *architecture level*, the real-time system designer maps the functions/components developed in the previous stage to real-time threads, defines the hardware (HW) architecture supporting the execution of the software (SW) components and selects the scheduler and the resource managers.

After mapping the functional components onto the HW/SW platform, the computation times of the software threads can be estimated and the designer can check the correctness of the timing requirements upon the target architecture. Schedulability analysis is used in this stage to validate the timing behavior of architecture-level solutions.

The design process is driven by the objective of providing the best possible performance to system functions within the timing constraints. This possibly implies many iterations between the logical and the architectural view, trading implementation complexity for timeliness, and tuning the design parameters, including the use of system resources and the activation rates of tasks. Unfortunately, the separation of the design environments for functionality and for time, together with the limited support (feedback) provided by the existing tools for timing analysis makes the process quite inefficient.

The design trade-offs between quality of implementation and responsiveness have been studied in an integrated, formal framework in the context of control applications [20, 17] and in the context of power-aware applications ([8] for a survey).

Dynamic changes of the design parameters, such as the thread periods, are practically impossible for most embedded controllers, because of the constraints imposed by the existing modeling and code generation tools. In a typical development cycle the controller is designed and validated by simulation (using tools such as Simulink™) assuming each control loop runs at a fixed sampling rate. Automatic code generation tools (such as the Real-Time Workshop™ suite) implement the controller by creating a thread for each sampling rate in the model. In most automotive embedded designs this results in 5 to 20 threads. Furthermore, dynamic changes of task activation rates is not supported by existing RTOS standards, such as OSEK [15].

The design problem can be formally and effectively stated as a static optimization problem. This requires:

- expressing the performance as a function of the *design parameters* (*optimization variables*);
- formalizing the *domain* of the optimization variables.
- formalizing the *objective function(s)*.

Computing the optimal set of design parameters for the system threads is the objective of the following optimization problem:

$$\begin{array}{ll} \text{maximize} & F(\mathbf{x}) \\ \text{subject to} & \mathbf{x} \in \mathbb{D} \end{array} \tag{1}$$

*This work has been partially supported by the European Union under the contract IST-2001-34820 (ARTIST project) and the contract IST-2001-37170 (RECSYS project).

Proceedings of the 26th IEEE International Real-Time Systems Symposium (RTSS'05)
0-7695-2490-7/05 $20.00 © 2005 IEEE

IEEE COMPUTER SOCIETY

where $\mathbf{x} = (x_1, \ldots, x_N)$ represents the vector of all the optimization variables (design parameters), $\mathbb{D}$ is the domain of the allowed values for $\mathbf{x}$, and F is the cost function.

The *optimization variables* $\mathbf{x}$ represent the set of design parameters under the designer's control. In real-time multitask applications each task τ_i may be defined by the simple model (C_i, T_i, D_i): a worst-case computation time, reflecting the complexity of the task implementation; an activation period (or its activation rate $f_i = 1/T_i$), which defines how often the task will run, and a relative deadline, which constrains the time between the activation and the completion of every task instance.

The *function* F provides a measure of the application performance, which represents the design goals. Practical examples are the transient delay, the maximum overshoot and the robustness in control systems or the consumed energy for battery-powered systems. Notice that, by changing the sign of $F(\mathbf{x})$, we can interchangeably have a maximization or minimization problem. In the literature this function is also referred to as utility [10], reward [2] or cost.

Finally, the *set* $\mathbb{D}$ ensures that the application tasks are schedulable (we will often refer to it with the term "schedulability region"). Design variables may also be constrained into intervals (i.e. $x_i \in [x_i^m, x_i^M]$) or forced to be in a discrete set, for example, if the designer has no control over the implementation or when the activation periods depend on environment stimuli that are precisely defined in time.

Because of the difficulty in defining the feasibility region, the design problem can only be solved if the optimization is separately performed in the domains of the computation times only (assuming constant task periods) or, alternatively, in the activation rates domain (constant C_i).

The exact schedulability region in the space of the worst-case computation times C_i has been derived [4] by refining the key contribution due to Lehoczky, Sha, Ding [11]. Performance optimization in the domain of the C_i can be achieved by applying convex optimization techniques, since the feasibility region is the union of convex regions.

A relevant case of the design problem occurs when all the task computation times C_i are given and the designer must define the task activation rates. This problem is very important, for instance, in the control systems domain, when control tasks are known and their activation rates need to be programmed. From now on, we assume that the (worst-case) task computation time C_i are given.

When the schedulability region $\mathbb{D}$ is the linear constraint

$$\sum C_i \, f_i \leq A \tag{2}$$

the design problem is greatly simplified. Seto et al. [20] solved it for a special cost function (weighted sum of exponential functions). Unfortunately, the schedulability region of Equation (2) is a good model only for Earliest Deadline First (EDF) based systems, when deadlines equal periods.

In reality, most systems of commercial interest use fixed priority schedulers and Rate Monotonic (RM) is the reference priority assignment for most practical applications. When Fixed Priority Scheduling (FPS) is used and deadlines are equal to periods, we can still use the constraint expression of Equation (2) by setting a proper value for the utilization bound A, such as the Liu and Layland bound [13]. Unfortunately, by using this constraint the solution is only a sub-optimum because the test

$$\sum_{i=1}^{n} C_i \, f_i \leq n(\sqrt[n]{2} - 1) \tag{3}$$

is only sufficient and, consequently, the domain $\mathbb{D}$ is a restriction of the exact one. The exact model of $\mathbb{D}$ allows finding the optimal solution of the design problem, rather than a sub-optimum. Unfortunately, to the best of our knowledge, its exact definition has never been explicitly formalized in the n-dimensional space of the task rates. In this paper, we address the case of the domain $\mathbb{D}$ defined as the schedulability region in the case of fixed priorities. Furthermore, we provide a solution to the optimization problem with a generic performance function $F(\mathbf{x})$.

1.1. Related works

Goossens and Richard [6] provided a method to find the optimal priority assignment given the cost function, the task periods and the computation times.

Gerber et al. [5] proposed an algorithm to synthesize the entire task set so that end-to-end deadlines and other timing constraints are satisfied. Task synthesis is based on timing constraints only and application-level performance metrics are neglected.

Seto et al. [20] assumed a linear schedulability region defined by Equation (2) and required the a priori evaluation of a lower bound $f_i \geq f_i^m$ for all task frequencies. The authors also assumed the cost function $F(\mathbf{f}) = \sum \alpha_i \, e^{-\frac{f_i}{f_i^*}}$, which is the result of the first order approximation of the energy spent in a bubble control application. The authors provided a clear advance in the state of art in real-time research, but their approach suffers from a number of weaknesses: it requires each task frequency to be lower bounded by a value f_i^m, it models the schedulability constraint by a linear inequality and it only applies to exponential functions.

In the context of reward-based scheduling, Aydin et al. [2] provided the design solution for a broader class of cost/performance functions, when the schedulability region is approximated by a linear constraint.

In the domain of QoS optimization, a significant work is the QoS-based Resource Allocation Model (Q-RAM), due to Lee et al. [10]. Even though their work provides a general framework dealing with this problem and also with multi-resource constraints, the proposed solution exploits simple linear inequalities, which do underestimate the exact schedulability region.

IEEE COMPUTER SOCIETY

In [14] task periods are changed at run-time in order to improve the performance of critical control loops. The adaptive scheme guarantees that at each time instant the set of periods satisfies the sufficient linear bound in Eq. (2).

All cited works share one common characteristics: tasks schedulability is ensured by a sufficient and simpler condition in order to make the optimization easier. To our best knowledge the only attempt at modeling the necessary and sufficient schedulability region in the space of the task periods has been performed by Seto, Lehoczky and Sha [19]. They presented a period selection algorithm based on integer linear programming (ILP) to evaluate the optimal periods of tasks subject to latency constraints with respect to a system-wide performance measure.

Unfortunately, their method suffers from a major drawback, since it requires the specification of upper bounds for all task periods. These bounds should be as loose as possible in order to explore a broader design space, but, unless an extremely tight bound is provided, the proposed algorithm soon becomes intractable. For very simple task sets consisting of 5–6 tasks (like the example shown in this paper) the method can require the evaluation of more than 10^{15} possible candidate solutions.

2. Assumptions and terminology

Our application consists of a set $\mathcal{T} = \{\tau_1, \tau_2, \ldots, \tau_N\}$ of real time tasks running on a uniprocessor. Each task τ_i is characterized by a period T_i (rate $f_i = 1/T_i$), a worst-case execution time C_i and a relative deadline D_i. In order to shorten the notation we will use bold letters to indicate vectors of parameters. So we have $\mathbf{T} = (T_1, \ldots, T_n)$, $\mathbf{f} = (f_1, \ldots, f_n)$ and $\mathbf{C} = (C_1, \ldots, C_n)$. The task utilization U can be written by the dot product $\mathbf{C} \cdot \mathbf{f}$. The task τ_i executes as an infinite sequence of jobs $\tau_{i,j}$. Every job $\tau_{i,j}$ is activated at $a_{i,j} = (j-1)T_i$ and must be completed by its absolute deadline $d_{i,j} = a_{i,j} + D_i$.

The tasks are scheduled by FPS and they are ordered by a decreasing priority meaning that τ_1 has the highest priority, τ_N the lowest. We also assume deadlines equal to periods meaning that $D_i = T_i$ for all i.

We restrict our study to performance functions F such that $\nabla F \geq 0$ meaning that $\forall i \ \frac{\partial F}{\partial f_i} \geq 0$, and $F(\mathbf{x})$ is convex. The first hypothesis holds for most actual systems since we expect that an increase of any task's rate results in an improvement of the system performance. The second one allows to use efficient optimization techniques. This last assumption holds for practical performance functions considered in the literature [20, 2].

3. The schedulability region in the rate space

As stated in the Introduction, in order to find the optimal task rates, we first need to define the optimization domain $\mathbb{D}$, which ensures schedulability of the task set. Formally, the set $\mathbb{D}$ is defined as follows:

$$\mathbb{D}(\mathbf{C}) = \{\mathbf{f} \geq 0 : \mathcal{T} = (\mathbf{C}, \mathbf{f}) \text{ is sched. by FPS}\}. \quad (4)$$

Since we assume constant (worst-case) computation times, in the rest of the paper we often drop the argument $\mathbf{C}$ and we denote the set simply by $\mathbb{D}$.

To find the frequencies $\mathbf{f}$ of all schedulable task sets, we need a necessary and sufficient schedulability condition. For this purpose, we recall the result provided by Seto et al. [19].

Let R_i be the maximum response time of task τ_i [7, 1]. The task set is schedulable **if and only if**:

$$\forall i \qquad R_i \leq T_i \quad (5)$$

The response time of task τ_i is equal to its computation time C_i plus the interference from higher priority tasks. The interference of any higher priority task τ_j is an integer number of times its computation time C_j. This means that the response time R_i can be written as:

$$R_i = C_i + \sum_{j=1}^{i-1} n_j^{(i)} C_j \quad (6)$$

where $n_j^{(i)} = \left\lceil \frac{R_i}{T_j} \right\rceil$ is the number of jobs of τ_j that interfere in the response time of τ_i. For a fixed number of interfering jobs, we have

$$\forall j = 1, \ldots, i-1 \qquad R_i \leq n_j^{(i)} T_j \quad (7)$$

$$\forall j = 1, \ldots, i-1 \qquad R_i \geq (n_j^{(i)} - 1) T_j. \quad (8)$$

In fact, if Equation (7) is violated then an additional job of τ_j would interfere in the response time R_i. For the same reason, violating Equation (8) would result in one τ_j job less in the interference. Hence, we can extend Proposition 2.1 in [19] by adding the constraint of Eq. (8).

Lemma 1 (Extension of Prop. 2.1 in [19]) *The task set $\mathcal{T}$ is schedulable by FPS **if and only if**:*

$$\forall i = 1, \ldots, N \quad \exists \mathbf{n}^{(i)} \in \mathbb{N}^{i-1}$$

such that:

$$\begin{aligned} &\forall k = 1, \ldots, i-1 \\ &C_i + \sum_{j=1}^{i-1} n_j^{(i)} C_j \leq T_i \\ &(n_k^{(i)} - 1) T_k \leq C_i + \sum_{j=1}^{i-1} n_j^{(i)} C_j \leq n_k^{(i)} T_k \end{aligned} \quad (9)$$

where $\mathbf{n}^{(i)} = (n_1^{(i)}, \ldots, n_{i-1}^{(i)})$ is a tuple of $i-1$ positive integers, and $\mathbb{N}^{i-1}$ is the set of all the possible $\mathbf{n}^{(i)}$.

The schedulability region in the space of the activation rates $f_i = 1/T_i$ can be found by inverting Eq. (9). It follows that

Theorem 1 *The task set $\mathcal{T}$ is schedulable by FPS **if and only if**:*

$$\forall i = 1, \ldots, N \quad \exists \mathbf{n}^{(i)} \in \mathbb{N}^{i-1}$$

IEEE COMPUTER SOCIETY

such that:

$$\forall k = 1, \ldots, i-1$$

$$0 \leq f_i \leq \boxed{\frac{1}{C_i + \sum_{j=1}^{i-1} n_j^{(i)} C_j}}$$

$$\frac{n_k^{(i)} - 1}{C_i + \sum_{j=1}^{i-1} n_j^{(i)} C_j} \leq f_k \leq \boxed{\frac{n_k^{(i)}}{C_i + \sum_{j=1}^{i-1} n_j^{(i)} C_j}} \quad (10)$$

Theorem 1 defines the region $\mathbb{D}$ of the task activation rates $\mathbf{f}$ ensuring schedulability.

Notice that, if i and $\mathbf{n}^{(i)}$ are fixed, Equations (10) determine a constant upper and lower bound on each frequency. The geometry of this constraint is then an *N-dimensional parallelepiped*. In Eq. (10), the upper bounds are boxed, because they have an important role in the optimization algorithm. We refer to these values as *the outermost vertex of the parallelepiped* related to the tuple $\mathbf{n}^{(i)}$.

3.1. The two tasks case

Now, let us spend some time on the simple case of two tasks, when the only available parameters are C_1 and C_2 and the design variables are f_1 and f_2. From Equation (10) task τ_1 is schedulable if and only if

$$0 \leq f_1 \leq \frac{1}{C_1}. \quad (11)$$

By Theorem 1, we also find the necessary and sufficient schedulability condition for task τ_2:

$$\exists n_1^{(2)} \in \mathbb{N} \quad \begin{array}{c} 0 \leq f_2 \leq \dfrac{1}{C_2 + n_1^{(2)} C_1} \\ \dfrac{n_1^{(2)} - 1}{C_2 + n_1^{(2)} C_1} \leq f_1 \leq \dfrac{n_1^{(2)}}{C_2 + n_1^{(2)} C_1} \end{array} \quad (12)$$

It follows that, for a fixed value of $n_1^{(2)}$ (i.e. assuming a specific number of interferences of τ_1 in the job $\tau_{2,1}$), the schedulability region is given by a rectangle bounded by Equations (12). For example, if $n_1^{(2)} = 2$ then $0 \leq f_2 \leq \frac{1}{C_2+2C_1}$ and $\frac{1}{C_2+2C_1} \leq f_1 \leq \frac{2}{C_2+2C_1}$.

Since τ_2 is schedulable if *there exists* at least one value of $n_1^{(2)}$ such that the rates (f_1, f_2) satisfy Equations (12), then the overall schedulability region $\mathbb{D}(C_1, C_2)$ can be constructed by the *union of an infinite number of rectangles*, each defined by an instance of Equation (12) evaluated for all possible values of $n_1^{(2)}$.

Below, we show an example. Suppose that $C_1 = 1$ and $C_2 = 2$. Table 1 reports some possible values for $n_1^{(2)}$ and the corresponding frequency bounds computed using Equation (12). In the table, the labels loB and upB denote, respectively, the frequency lower and upper bounds. From Equation (10), the lower bound of f_2 is equal to 0.

For this example, we can now depict the schedulability region $\mathbb{D}(1, 2)$ in the space of the activation rates by performing the union of the rectangles defined by the bounds in Table 1. The resulting region is represented in Figure 1.

$n_1^{(2)}$	f_1 loB	f_1 upB	f_2 upB
1	0	1/3	1/3
2	1/4	2/4	1/4
3	2/5	3/5	1/5
4	3/6	4/6	1/6
5	4/7	5/7	1/7
...	...	...	...

Table 1. The rate bounds as function of $n_1^{(2)}$.

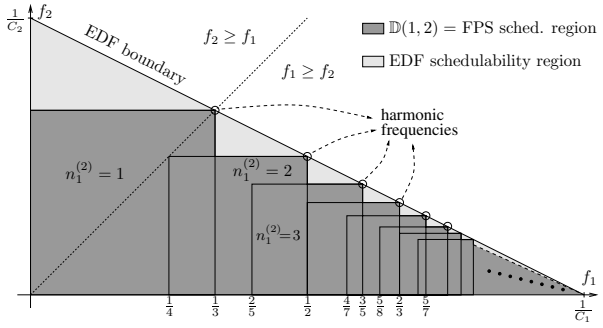

Figure 1. Schedulability region of two tasks.

The EDF bound is also plotted in the figure. In the space of the activation rates it is a linear constraints (see Eq. (2)), which is, as expected, larger than the FPS feasible region.

Figure 1 also depicts the two regions where $f_1 \geq f_2$ and $f_2 \geq f_1$. In the first case FPS coincides with RM. It is interesting to notice that, when FPS $\equiv$ RM, the points where the FPS region touches the EDF bound have harmonic frequencies. In this case, FPS can guarantee all the task deadlines up to 100% utilization [9].

The extension to the more general case of N tasks can be inferred from Theorem 1. In this case, each $\mathbf{n}^{(i)}$ results in an N-dimensional parallelepiped. Similarly to the two tasks case, the space of the activation rates guaranteeing schedulability is given by the union of these parallelepipeds and its boundary consists of surfaces perpendicular to the axis, ending in a set of outermost **vertices**. These vertices have a key role in the optimal design problem.

4. The Optimization Algorithm

As explained in the last section, the space ensuring the schedulability of task τ_i is given by the union of parallelepipeds, each one resulting from a different selection of the tuple $\mathbf{n}^{(i)}$. Each tuple element $n_j^{(i)}$ indicates the number of interferences from higher priority tasks τ_j on the critical job $\tau_{i,1}$. Moreover, the N regions resulting from individual task schedulability conditions, must be intersected in order to ensure the schedulability of all the N tasks.

In this section we focus on the techniques to perform the optimization of the performance/cost function inside the domain $\mathbb{D}$. In 1998, Seto et al. [19] proposed to find the optimal task rates/periods by an ILP algorithm based on branch and bound. In order to explore the domain $\mathbb{D}$, they con-

IEEE COMPUTER SOCIETY

structed a search tree. Unfortunately, following their approach, the number of nodes quickly explodes to impractical numbers. Implementing their algorithm we found that it cannot deal with systems of 6 or more tasks. An approximate solution can still be found by performing the optimization on a simpler-but-sufficient region, so that the algorithm can terminate and produce some result [10, 20]. Instead, we propose an exact algorithm which allows to find the optimum for a task set of practical size in a reasonable time.

The first insight, which allows to simplify the problem, derives from studying the domain together with the performance function. In optimization techniques, from a candidate solution we move to the next one following the direction of the gradient ∇F, which guarantees an increase of the performance F. Due to the property $\nabla F \geq 0$, an increase of any task frequency will always achieve a higher performance value. However, task frequencies cannot increase beyond the domain boundary. When we arrive at the outermost vertex of the parallelepiped, any increase of the performance F would violate the schedulability boundary. This means that **every outermost vertex of the boundary region is a local optimum**, when $\nabla F \geq 0$.

Since *some local optimum is global as well*, a possible global optimization algorithm may consist in checking all the vertices until we find the best performance value.

Unfortunately, this algorithm would never terminate since the number of vertices is infinite, as in Figure 1, for f_1 values approaching to $\frac{1}{C_1}$. The region to be searched must necessarily be bounded, so that the number of candidate vertices is finite. The task of bounding the number of vertices can be accomplished if we know a *first admissible solution*. In Section 4.1 we will assume availability of such an admissible solution, and Section 4.2 explains how to actually find it. Finally, Section 4.3 describes the search procedure.

4.1. Bounding the vertices

Let us suppose that a first admissible solution $\mathbf{f}^{\text{fi rst}}$ is available. We call the performance value at this point $v^{\text{fi rst}} = F(\mathbf{f}^{\text{fi rst}})$ (see Figure 2).

Any candidate solution $\mathbf{f}$ better than $\mathbf{f}^{\text{fi rst}}$ must: (1) provide a performance better than the current solution, and (2) belong to the schedulability region. If we define the set

$$\mathbb{S}(v) = \{\mathbf{f} \in \mathbb{D} : F(\mathbf{f}) \geq v\}, \tag{13}$$

we can surely assert that any solution $\mathbf{f}$ better than $\mathbf{f}^{\text{fi rst}}$ must be in $\mathbb{S}(v^{\text{fi rst}})$, by definition.

Since we know that the global optimum occurs at some vertex, then it suffices to check all the vertices in $\mathbb{S}(v^{\text{fi rst}})$ to find the optimum. We must then enumerate the vertices in $\mathbb{S}(v^{\text{fi rst}})$, which is equivalent to enumerate the corresponding tuples $\mathbf{n}^{(i)}$ (tuples and vertices are in one-to-one correspondence.)

Suppose we bound $\mathbb{S}(v^{\text{fi rst}})$ by a larger box delimited by a lower and an upper frequency bounds $\mathbf{f}^m$, $\mathbf{f}^M$ (see Figure 2). In this case, enumerating the admissible tuples $\mathbf{n}^{(i)}$ can be easily done. All the possible tuples $\mathbf{n}^{(i)}$ within $\mathbf{f}^m$ and $\mathbf{f}^M$ are computed by imposing a non-empty intersection between the parallelepiped originated by the tuple $\mathbf{n}^{(i)}$ and the box bounded by $\mathbf{f}^m$, $\mathbf{f}^M$

$$\begin{aligned}
\frac{1}{C_i + \sum_{j=1}^{i-1} n_j^{(i)} C_j} &\geq f_i^m \\
\frac{n_k^{(i)}}{C_i + \sum_{j=1}^{i-1} n_j^{(i)} C_j} &\geq f_k^m \\
\frac{n_k^{(i)} - 1}{C_i + \sum_{j=1}^{i-1} n_j^{(i)} C_j} &\leq f_k^M
\end{aligned}$$

By manipulating the equations, we obtain the following linear constraints on the tuples $\mathbf{n}^{(i)}$.

$$\begin{aligned}
& C_i + \sum_{j=1}^{i-1} n_j^{(i)} C_j \leq 1/f_i^m \\
& C_i + \sum_{j=1,\, j\neq k}^{i-1} n_j^{(i)} C_j + n_k^{(i)}(C_k - 1/f_k^m) \leq 0 \\
& C_i + 1/f_k^M + \sum_{j=1,\, j\neq k}^{i-1} n_j^{(i)} C_j + n_k^{(i)}(C_k - 1/f_k^M) \geq 0
\end{aligned} \tag{14}$$

for all $k = 1, \ldots, i-1$.

Equations (14) provide the relationships to enumerate all the tuples $\mathbf{n}^{(i)}$ whose parallelepiped overlaps with the region $\mathbf{f}^m \leq \mathbf{f} \leq \mathbf{f}^M$. Now we need to find the bounds $\mathbf{f}^m$, $\mathbf{f}^M$ such that $\mathbb{S}(v^{\text{fi rst}}) \subseteq \{\mathbf{f} \geq 0 : \mathbf{f}^m \leq \mathbf{f} \leq \mathbf{f}^M\}$. Theoretically, the tightest possible lower and upper bound are

$$f_i^m = \min_{\mathbf{f} \in \mathbb{S}(v^{\text{fi rst}})} f_i \qquad f_i^M = \max_{\mathbf{f} \in \mathbb{S}(v^{\text{fi rst}})} f_i.$$

Unfortunately, it is impossible to calculate these values efficiently due to the complexity of the domain $\mathbb{D}$, which defines $\mathbb{S}(v^{\text{fi rst}})$. For this reason, we first define the convex superset $\overline{\mathbb{S}}(v)$ based on the *necessary and linear* EDF condition, as follows

$$\overline{\mathbb{S}}(v) = \{\mathbf{f} \geq 0 : F(\mathbf{f}) \geq v \quad \wedge \quad \mathbf{C} \cdot \mathbf{f} \leq 1\}. \tag{15}$$

Due to the convexity of F, the region $\overline{\mathbb{S}}(v)$ is convex, differently from $\mathbb{S}(v)$. Hence, lower and upper bounds on task frequencies

$$f_i^m = \min_{\mathbf{f} \in \overline{\mathbb{S}}(v^{\text{fi rst}})} f_i \qquad f_i^M = \max_{\mathbf{f} \in \overline{\mathbb{S}}(v^{\text{fi rst}})} f_i. \tag{16}$$

can be easily computed by solving the convex optimization problems of Equations (16).

A closed form solution can be obtained for some performance functions of interest by using the Lagrange multipliers technique [2]. In general, any convex optimization package (such as the numerical solvers available from Matlab™) can be used to solve this problem. The frequency bounds can be used in Eq. (14) to restrict the search for the optimum to a finite set.

IEEE COMPUTER SOCIETY

Figure 2 depicts the procedure explained in this section. Suppose the availability of an admissible solution $\mathbf{f}^{\text{first}}$ (a

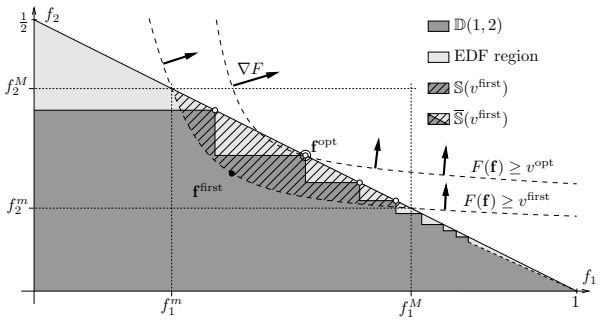

Figure 2. Bounding the candidate vertices.

black dot in the figure) whose performance is v^{first}. The candidate solutions (represented by white dots) which can improve the performance $F(\mathbf{f})$ are in the set $\overline{\mathbb{S}}(v^{\text{first}})$. These points are all at the outermost vertices of some rectangle, because $\nabla F \geq 0$ (the gradient is indicated by thick black arrows). The possible tuples are given by Equation (14), where the frequency lower and upper bounds $\mathbf{f}^m$, $\mathbf{f}^M$ are found by solving the problem in Eq. (16). From the candidate tuples we directly find the vertices and by enumerating them we can find the optimum, indicated in Figure 2 by $\mathbf{f}^{\text{opt}}$.

At this point there are still two open problem: (1) how do we find a first admissible solution and (2) how do we enumerate the vertices to search for the optimum. Next sections 4.2 and 4.3 will cover these steps.

4.2. First admissible solution

The previous section shows how to bound the search region starting from an admissible solution $\mathbf{f}^{\text{first}}$. This section explains how to find an admissible solution *with a performance close to the global optimum*. This last property is very important, since (as illustrated by Figure 2) a good performance value of the solution $\mathbf{f}^{\text{first}}$ can significantly restrict $\overline{\mathbb{S}}(v^{\text{first}})$ and speed up the search procedure.

The procedure for finding the first admissible solution $\mathbf{f}^{\text{first}}$ is the following (see Figure 3):

1. we find the optimum on a looser linear constraint. The EDF schedulability condition, which is also necessary for FPS, provides a suitable choice. Hence, we label this solution $\mathbf{f}^{\text{EDF}}$;
2. from $\mathbf{f}^{\text{EDF}}$, we scale down the task frequencies until the intersection with the FPS boundary is found. $\mathbf{f}^{\text{hit}}$ are the frequencies at this point;
3. thanks to the knowledge of the geometry of the boundary, we know $\mathbf{f}^{\text{hit}}$ lays onto a parallelepiped. We can then move to the outer vertex and find $\mathbf{f}^{\text{vert}}$. Due to the property $\nabla F \geq 0$ the vertex $\mathbf{f}^{\text{vert}}$ has a better performance value than $\mathbf{f}^{\text{hit}}$.

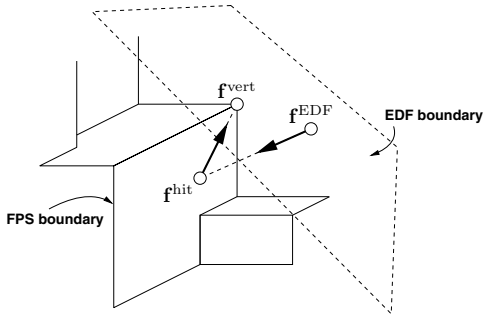

Figure 3. Finding the first admissible solution.

The first step can easily be solved by convex optimization techniques. In fact, the EDF domain is linear and the function F is convex. Closed solutions exist in the literature [2, 20] for this problem.

If the task set $(\mathbf{C}, \mathbf{f}^{\text{EDF}})$ is schedulable by FPS, then the entire optimization routine is finished and $\mathbf{f}^{\text{EDF}}$ is the optimum for FPS as well. Then, in the following paragraphs we assume that the task set with activation frequencies $\mathbf{f}^{\text{EDF}}$ is not schedulable by FPS.

In the second step, we want to find the maximum $\alpha < 1$ such that the task set $(\mathbf{C}, \alpha\, \mathbf{f}^{\text{EDF}})$ is schedulable by FPS. The problem of scaling the task parameters such that the resulting task set is schedulable, has already been solved in the domain of the computation times $\mathbf{C}$ [12].

The following theorem establishes a relationship between the scaling operations in the two domains.

Theorem 2 *Given a task set $\mathcal{T} = \{\tau_1, \tau_2, \ldots, \tau_N\}$, $\tau_i = (C_i, T_i)$. Let $\mathcal{T}^C$ be the task set where each task computation time is scaled by α and $\mathcal{T}^T$ be the task set where each task period in scaled by $1/\alpha$. Formally:*

$$\mathcal{T}^C = \{(\alpha C_1, T_1), \ldots, (\alpha C_N, T_N)\}$$
$$\mathcal{T}^T = \{(C_1, T_1/\alpha), \ldots, (C_N, T_N/\alpha)\}.$$

*The task set $\mathcal{T}^C$ is schedulable **if and only if** the task set $\mathcal{T}^T$ is schedulable.*

Proof. We start from the schedulability of $\mathcal{T}^C$ provided by Lehoczky et al. [11], and we prove the schedulability of $\mathcal{T}^T$ by means of an "if and only if" chain.

$$\mathcal{T}^C \text{ is schedulable } \iff$$
$$\forall i \quad \exists t \in [0, T_i] \quad : \quad \alpha C_i + \sum_{j=1}^{i-1} \left\lceil \frac{t}{T_j} \right\rceil \alpha C_j \leq t \iff$$
$$\forall i \quad \exists t \in [0, T_i] \quad : \quad C_i + \sum_{j=1}^{i-1} \left\lceil \frac{t}{T_j} \right\rceil C_j \leq \frac{t}{\alpha} \iff$$
$$\forall i \quad \exists t^* \in \left[0, \frac{T_i}{\alpha}\right] \quad : \quad C_i + \sum_{j=1}^{i-1} \left\lceil \frac{t^*}{(T_j/\alpha)} \right\rceil C_j \leq t^*$$
$$\iff \mathcal{T}^T \text{ is schedulable}$$

IEEE COMPUTER SOCIETY

as required. □

Thanks to Theorem 2 and to previous results on the minimum processor speed which can schedule a task set [12], we can safely assert that the scaling factor to go from the EDF boundary to the FPS one is

$$\alpha_{\max} = \min_{i=1,\ldots,N} \max_{t \in \mathsf{schedP}_i} \frac{t}{C_i + \sum_{j=1}^{i-1} \lceil t\, f_j^{\text{EDF}} \rceil\, C_j} \qquad (17)$$

where schedP_i is the appropriate set of schedulability points [11, 4] for the task τ_i. Hence the task frequencies $\mathbf{f}^{\text{hit}}$ where we hit the FPS boundary after the scaling process are

$$\mathbf{f}^{\text{hit}} = \alpha_{\max}\, \mathbf{f}^{\text{EDF}}. \qquad (18)$$

Since the scaling operation starts from a 100% utilization task set, $\alpha_{\max}$ is also the utilization of the task set at the boundary. This utilization is strictly less than 1 because the set $\mathbf{f}^{\text{EDF}}$ is not schedulable.

Finally, from the property $\nabla F \geq 0$ of the performance function, we know that by increasing any frequency we also increase the performance. Hence, the solution $\mathbf{f}^{\text{hit}}$ can be further improved moving to the outermost vertex of the current parallelepiped. Theorem 1 provides an effective means to compute the outermost vertex of the parallelepiped, once the integers $\mathbf{n}^{(i)}$ are known. The best way to find the number of interferences $\mathbf{n}^{(i)}$ is to compute the response times [7, 1, 21] at the current point $\mathbf{f}^{\text{hit}}$. Since the task set $(\mathbf{C}, \mathbf{f}^{\text{hit}})$ is schedulable by costruction, the response time iterative routine converges. For each task we have:

$$R_i^{\text{hit}} = C_i + \sum_{j=1}^{i-1} n_j^{(i)} C_j \qquad (19)$$

where $n_j^{(i)} = \lceil R_i^{\text{hit}}\, f_j^{\text{hit}} \rceil$. In this way, we find the $N-1$ tuples $\mathbf{n}^{(i)}$ for all task indices i from 2 to N (trivially $R_1 = C_1$).

The outermost vertex of the parallelepiped originating from tuple $\mathbf{n}^{(i)}$ can be calculated from Equations (10). Hence, in order to compute $\mathbf{f}^{\text{vert}}$, the N parallelepipeds computed from tuples $\mathbf{n}^{(i)}$ must be intersected to ensure the schedulability of the N tasks. Consequently we have:

$$f_j^{\text{vert}} = \min_{i=j,\ldots,N} \frac{\lceil R_i^{\text{hit}}\, f_j^{\text{hit}} \rceil}{R_i^{\text{hit}}}. \qquad (20)$$

The point $\mathbf{f}^{\text{vert}}$ is then used as initial solution for the optimization algorithm.

The procedure for obtaining the local optimum $\mathbf{f}^{\text{vert}}$ should not be overlooked and it is rightfully among the main contributions of this work for its practical implications. It not only allows reducing the size of the search tree, but the quality of the $\mathbf{f}^{\text{vert}}$ solution is very close to the global optimum, as demonstrated by our experiments in Section 6. $\mathbf{f}^{\text{vert}}$ is by all means a practically acceptable solution to the optimization problem whenever the number of tasks makes the exhaustive search impossible.

```
1:  procedure OPTIMFPSRATES(C)
2:      f^EDF ← solveOnEDF(C)
3:      f^hit ← α_max f^EDF
4:      f^vert ← findVertex(C, f^hit)
5:      f^cur ← f^vert
6:      v^cur ← F(f^cur)
7:      f^m ← findMinBound(C, v^cur)
8:      f^M ← findMaxBound(C, v^cur)
9:      VISITTREE(1, f^m, f^M)
10:     f^opt ← f^cur
11: end procedure

12: procedure VISITTREE(i, f^m, f^M)
13:     if i = N then
14:         if F(f^M) > v^cur then
15:             f^cur ← f^M
16:             v^cur ← F(f^cur)
17:         end if
18:     else
19:         f^bound ← solveOnConstrEDF(C, f^m, f^M)
20:         v^bound ← F(f^bound)
21:         if v^cur ≥ v^bound then
22:             prune this node
23:         else
24:             T_i ← allTuples(i + 1, C, f^m, f^M)
25:             for each n_i ∈ T_i do
26:                 f^m_child ← lowBound(C, n_i)
27:                 f^M_child ← upBound(C, n_i)
28:                 f^m_child = max(f^m_child, f^m)
29:                 f^M_child = min(f^M_child, f^M)
30:                 VISITTREE(i + 1, f^m_child, f^M_child)
31:             end for
32:         end if
33:     end if
34: end procedure
```

Figure 4. The Optimization Algorithm

4.3. Search Procedure

A common technique to find the optimum onto a finite set consists of enumerating all the possible solutions. The most effective enumeration technique is the *branch and bound*. By this technique, the set of candidate solutions is split into smaller subsets. Each subset is further divided in subsets until individual solutions are obtained. This recursive partitioning originates a tree structure, where each node represents a different subset depending on the path from the root node. This scheme requires a *branching rule*, which defines how a node is decomposed in subnodes and a *bounding rule* which provides an upper bound (or a lower bound for minimization problems) of the optimum on a subtree. If the current solution is better than the upper bound, then the subtree can be pruned because no better solution will be obtained from visiting it. In Figure 4 we report the pseudocode of the routines implementing the search.

IEEE COMPUTER SOCIETY

The first admissible solution is found in lines 2–4, following the procedure outlined in Section 4.2. From this first solution we evaluate the region to be explored in order to find the optimum (lines 7–8), then the core of the optimization search is entered by calling VISITTREE. **Important**: in the pseudocode we assume that the number of tasks n, the computation times $\mathbf{C}$, the current solution $\mathbf{f}^{\text{cur}}$ and the current performance v^{cur} are implemented as global variables.

The routine VISITTREE implements the visit into the search tree. It can be subdivided into three parts: the *update rule* (lines 13–17), the *bounding rule* (lines 19–22) and the *branching rule* (lines 24–31). We first explain the branching rule because it describes the tree structure used in the optimization.

Branching Rule In our problem formulation the depth of the tree is $N-1$. The root node has depth 1, whereas the leaf nodes have depth N (see Figure 5 for the tree representation). At level i, based on the current frequency bounds $\mathbf{f}^m$, $\mathbf{f}^M$, all the admissible tuples $\mathbf{n}^{(i)}$ guaranteeing the schedulability of task τ_i are computed by means of Eq. (14) (line 24). Each tuple value constitutes a child of the current node. Tight frequency bounds result in few tuples and, consequently, in a small number of branches. This remark explains why **tight frequency bounds reduce the run-time of the algorithm**.

For each child node, the selection of a tuple $\mathbf{n}^{(i)}$ imposes more stringent frequency bounds given by Equations (10) (see lines 26– 27). These bounds must be intersected with the bounds of the father node (lines 28–29 in the pseudocode) in order to obtain the frequency bounds at the child node. Notice that this restriction contributes significantly to speed the algorithm up. After the new bounds are computed, the recursive routine is called for each child node (line 30).

Update Rule The update rule (lines 13–17) takes place at leaf nodes. For each leaf node we evaluate the corresponding vertex solution. We compare the solution with the current one and we update $\mathbf{f}^{\text{cur}}$ and v^{cur} if the performance is improved. $\mathbf{f}^{\text{vert}}$ is the initial value for the current solution.

Bounding Rule The bounding rule (lines 19–22) is a key quality factor of any branch and bound algorithm. In fact, if the current solution is better than an upper bound of some node, the node can be safely pruned. Technically speaking, the pruned subtree is *implicitly visited*. We propose to bound the solution of a node by solving the convex optimization problem on the EDF feasibility region. This means that the function $\mathsf{solveOnConstrEDF}(\mathbf{C}, \mathbf{f}^m, \mathbf{f}^M)$ used at line 19 is the solution of the problem

$$\text{maximize } F(\mathbf{f}) \quad \text{subject to } \begin{cases} \mathbf{C}\cdot\mathbf{f} \le 1 \\ \mathbf{f}^m \le \mathbf{f} \le \mathbf{f}^M \end{cases}, \tag{21}$$

which can be obtained by convex optimization techniques.

The upper bound at a node also provides an estimate of the quality of the solution that we expect to find within the node itself. It is then very convenient to visit first the child node that shows a superior value of the bound. By doing so, we move faster toward higher performance solutions and, consequently, we can prune a higher number of nodes. This higher-bound-first node selection rule is hidden in line 25, where we suppose *to loop on the nodes in descending order of the bound value*.

Matlab code implementing the search procedure is available on-line at `http://feanor.sssup.it/~bini/publications/2005BinDiN.html`.

5. Example of Application

In order to explain the algorithm more clearly, we solve the problem proposed in [19], where $C_1 = 10$, $C_2 = 15$, $C_3 = 20$, $C_4 = 25$, $C_5 = 30$ and $C_6 = 35$. The objective is finding the feasible task frequencies that *minimize* the cost function

$$F(\mathbf{T}) = \sum e^{-\beta_i/T_i} \text{ with } \beta = (20.4, 31, 40, 48, 54, 55). \tag{22}$$

In this example, to let the reader better visualize the results, we will use the period notation, since $T_i = 1/f_i$.

As explained before, the first step is to compute the optimal solution onto the EDF linear constraint. We find the following periods

$$\mathbf{T}^{\text{EDF}} = (51.2,\ 75.3,\ 105.6,\ 142.0,\ 197.3,\ 399.1) \tag{23}$$

and the cost of this solution is $v^{\text{EDF}} = F(\mathbf{T}^{\text{EDF}}) = 4.36321$.

The required scaling to bring the point $\mathbf{T}^{\text{EDF}}$ into the FPS feasibility region, as computed from Equation (17), is $\alpha_{\max} = 0.939839$. Dividing all the periods by $\alpha_{\max}$ we obtain the solution at the boundary

$$\mathbf{T}^{\text{hit}} = (55.0,\ 80.9,\ 113.4,\ 152.5,\ 212.1,\ 428.9) \tag{24}$$

with an increased cost value $v^{\text{hit}} = F(\mathbf{f}^{\text{hit}}) = 4.45946$.

In order to find the vertex solution, from the periods $\mathbf{T}^{\text{hit}}$, we compute the response times R_i, the tuples $\mathbf{n}^{(i)}$ and the minimum period (maximum frequency) corresponding to the outermost vertex (see Eq. (20)), as reported in Table 2. The vertex solution at the current parallelepiped is obtained

Task	R_i	$\mathbf{n}_{i-1}$	$(R_i/n_1^{(i)}, \ldots, R_i/n_{i-1}^{(i)}, R_i)$
τ_2	25	1	$(25, 25)$
τ_3	45	$(1,1)$	$(45, 45, 45)$
τ_4	80	$(2,1,1)$	$(40, 80, 80, 80)$
τ_5	205	$(4,3,2,2)$	$(51.25,\ 68.33,\ 102.5,\ 102.5,\ 205)$
τ_6	420	$(8,6,4,3,2)$	$(52.5, 70, 105, 140, 210, 420)$

Table 2. Response times at $\mathbf{T}^{\text{hit}}$.

by intersecting all the feasibility intervals. Hence:

$$\mathbf{T}^{\text{vert}} = (52.5,\ 70,\ 105,\ 140,\ 210,\ 420) \tag{25}$$

and a performance value $v^{\text{vert}} = 4.36369$ with utilization factor $U^{\text{vert}} = 1$.

IEEE COMPUTER SOCIETY

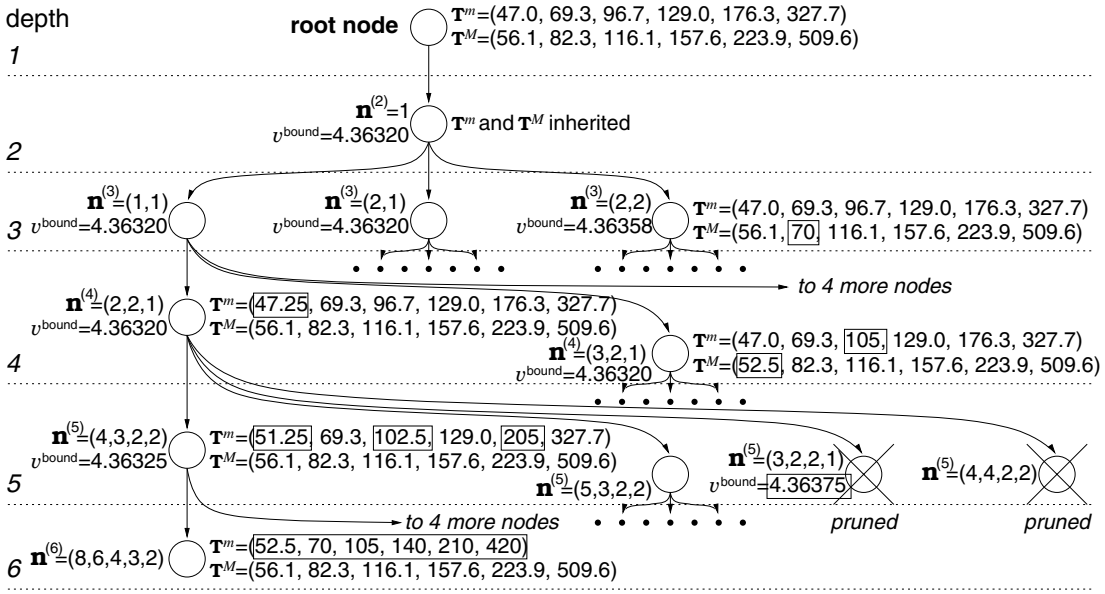

Figure 5. The search tree.

$\mathbf{T}^{\text{vert}}$, or equivalently $\mathbf{f}^{\text{vert}}$, is the first solution. It is used to bound the set of candidate solutions that needs to be enumerated in order to find the optimum.

Notice that $F(\mathbf{T}^{\text{vert}})$ is already within 0.01% of the performance at $\mathbf{T}^{\text{EDF}}$. The experiments in Section 6 show that the quality of the solutions $\mathbf{T}^{\text{vert}}$ obtained after this stage is already very close to the optimum.

Once the first admissible solution is found, we compute the upper and lower bounds for task periods delimiting the region to be searched, as explained in Section 4.1. We have

$$\begin{array}{l}\mathbf{T}^M = (56.1,\ 82.3,\ 116.1,\ 157.6,\ 223.9,\ 509.6)\\ \mathbf{T}^m = (47.0,\ 69.3,\ 96.7,\ 129.0,\ 176.3,\ 327.7)\end{array}. \quad (26)$$

Notice that $\mathbf{f}^m = 1/\mathbf{T}^M$ and $\mathbf{f}^M = 1/\mathbf{T}^m$ since the inversion operator is a decreasing function.

From these bounds the tree structure built by the first invocation of the routine VISITTREE is shown in Figure 5.

From the period bounds at the root node the only possible tuple verifying Eq. (14) is $\mathbf{n}^{(2)} = 1$. The period constraints computed from Eq. (10) for $\mathbf{n}^{(2)}$ do not restrict the frequency bounds, which are simply inherited from the parent node. By using the frequency bounds at depth 2, we find only 3 admissible tuples $\mathbf{n}^{(3)}$: $(1,1)$, $(2,1)$ and $(2,2)$. The third node $\mathbf{n}^{(3)} = (2,2)$ at depth 3 lowers T_2^M to 70 from the inherited value of 82.3 (in Figure 5, we highlight the new period bounds by boxing the new value).

We evaluate the cost bounds v^{bound} for all nodes at depth 2 and we proceed visiting the child showing the best value. Two nodes have the same $v^{\text{bound}} = 4.36320$ and we break the tie arbitrarily. First we select $\mathbf{n}^{(3)} = (1,1)$. From this node six tuples $\mathbf{n}^{(4)}$ are possible: $(2,1,1)$, $(2,2,1)$, $(3,2,1)$, $(3,2,2)$, $(3,3,2)$ and $(4,3,2)$. The two best choices are $(2,2,1)$ and $(3,2,1)$ with cost bound 4.36320. By selecting $(2,2,1)$, the bound T_1^m increases to 47.25.

At the next step, the tuples $\mathbf{n}^{(5)}$ that allow the schedulability of τ_5 are only four: $\mathbf{n}^{(5)} = (4,3,2,2)$ with a cost bound of 4.36325, and $(5,3,2,2)$ with a cost bound of 4.36349. The third node $(3,2,2,1)$ has $v^{\text{bound}} = 4.36375$, which is higher than the cost v^{vert} of the current solution. Hence, this subtree can be pruned. The forth tuple $(4,4,2,2)$ can be pruned as well.

Descending from the node $\mathbf{n}^{(5)} = (4,3,2,2)$, five children nodes are possible. Only the lowest cost tuple $\mathbf{n}^{(6)} = (8,6,4,3,2)$ is worth considering. It determines a candidate solution equal to $\mathbf{T}^{\text{vert}}$. Visiting the entire tree we find that $\mathbf{T}^{\text{vert}}$ is also *the global optimum*. This circumstance occurs frequently, especially for few tasks.

Finally, we remark the importance of a good quality first solution $\mathbf{f}^{\text{vert}}$, since the size of the region to be searched depends on $F(\mathbf{f}^{\text{vert}}) = v^{\text{vert}}$. A value close to the optimum is necessary for achieving a manageable size and a poor choice of $\mathbf{f}^{\text{vert}}$ makes the problem intractable.

In fact, applying the method by Seto et al. [19] and using the frequency bounds from the Liu and Layland linear constraint, the number of nodes at depth 5 grows to 10^{15}, which makes the search infeasible.

6. Experimental Results

In order to evaluate the performance of the optimization algorithm we ran several experiments onto random task sets. We implemented the search procedure in C language onto a 2.0 GHz Pentium processor. The task computation times C_i are uniformly distributed between 5 and 20. In the experiments, we considered a **minimization** problem with cost

IEEE COMPUTER SOCIETY

function $F(\mathbf{f}) = \sum e^{-\beta_i f_i}$, where the value of the beta coefficients are selected so that the ratio C_i/β_i, is uniformly random between 0.3 and 0.7, which prevents the optimum value from having any frequency component at 0.

Figure 6 shows the average run-time in seconds (with 90% confidence intervals) depending on the number of tasks N, in logarithmic scale. The upper curve is the time needed to complete the search for the optimum. As expected, the time required by the algorithm is exponential. On our simulation platform, the search procedure is feasible when we have less than 12 tasks.

Below, we also depict the time necessary to find the optimal solution. The difference between the two curves represents the time spent visiting the tree without improving the final solution. The lower curve is almost two orders of magnitude lower than the upper curve. This means that *visiting the best cost bound first is an effective strategy* to speed up the search for the optimum. In Figure 6 we also inferred

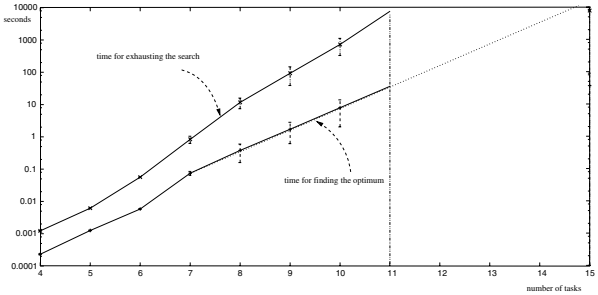

Figure 6. Algorithm run-time.

the run-time needed for finding the optimum when considering larger task sets (lower dotted line). When the number of tasks is less than 16 we expect to find the optimum in a feasible time.

Since the run-time required by the optimal algorithm prevents the applicability for large task sets, it is interesting to evaluate the quality of the first admissible solution $\mathbf{f}^{\text{vert}}$. In fact, the run-time to compute the first candidate solution is just a few milliseconds. Let $\mathbf{f}^{\text{EDF}}$ be the solution onto EDF, $\mathbf{f}^{\text{opt}}$ the optimum for FPS, $\mathbf{f}^{\text{vert}}$ the first vertex and $\mathbf{f}^{\text{LL}}$ the solution on the linear Liu and Layland bound, we computed the ratios $\frac{F(\mathbf{f}^{\text{opt}})}{F(\mathbf{f}^{\text{EDF}})}$, $\frac{F(\mathbf{f}^{\text{vert}})}{F(\mathbf{f}^{\text{EDF}})}$, $\frac{F(\mathbf{f}^{\text{LL}})}{F(\mathbf{f}^{\text{EDF}})}$. Since we are considering a minimization problem all these values are greater than or equal to 1. We plot the density of these quantities, from left to right, in Figure 7, assuming $N = 8$.

As expected, the optimum on the Liu and Layland constant utilization bound is a poor approximation of the actual optimum since the average cost increase is 33% (in the example of Section 5, we had $v^{\text{LL}} = 4.80897$).

In order to better estimate the quality of $\mathbf{f}^{\text{vert}}$ we enlarged the two densities on the left. From this picture we can see that, in the average, the optimization onto FPS incurs in an additional cost of 0.6% w.r.t. EDF. This result confirms that

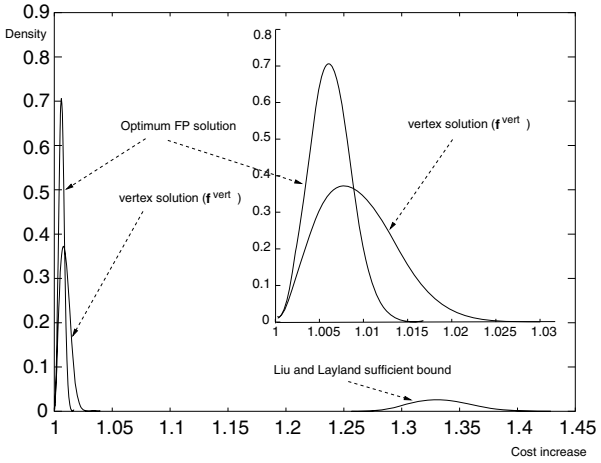

Figure 7. Comparing the quality of solutions.

the benefits of adopting a dynamic priority scheduler (EDF) are negligible, in term of system performance.

If, on the other hand, we simply use the solution $\mathbf{f}^{\text{vert}}$ *without performing the whole tree visit* required by the branch and bound routine, then the average error w.r.t. FPS is 0.3%, which is very low. This intermediate solution is then a very good balance between run-time of the algorithm and quality of the solution.

In conclusion, for the selected cost function (related to applications in the control system domain [20, 17]) we expect to provide the global optimum, with certainty (by exhausting the search), for task sets with no more than 12 tasks. Also, the chance of obtaining the global optimum in a feasible time is considerably high for task sets up to 15 or 16 tasks (as shown by the interpolation of the lower line in Figure 6).

For larger task sets, the first vertex solution, which can be computed with complexity comparable with the standard schedulability test, provides a very good approximation in most, if not all, cases. Furthermore, the search algorithm not only starts with a good quality solution, but it usually produces intermediate solutions that incrementally approach the global optimum. This allows to trade off time for quality and terminate the search when a satisfactory solution is obtained. Early experiments with a different cost metric shows similar results.

7. Conclusions and Future Work

We presented an innovative method for synthesizing the activation rates of a set of periodic real-time tasks that optimizes an application-related performance within the schedulability constraints. Our method exploits the new concept of the exact feasibility region in the space of the tasks' activation rates for a system consisting of periodic tasks scheduled by fixed priorities. With respect to previous work, the search procedure terminates for task sets of

IEEE COMPUTER SOCIETY

practical size. Furthermore, we also provide a method for finding an initial feasible solution of very good quality. The initial solution can be computed with the same complexity of a schedulability analysis test and it provides a good quality alternative for those cases that are computationally intractable.

Further improvements and considerations that have been omitted in the paper for improving the clarity and for lack of space include (1) an improvement that allows reducing the number of nodes in the search tree, and (2) an extension of the algorithm from the FPS case to Rate Monotonic.

The last improvement is particularly important. When task priorities depend on periods, such as in a Rate Monotonic priority assignment, the schedulability region needs to be defined for each possible priority ordering, up to a total of $N!$ subregions. Formally, this consists in a higher level of branching, where all feasible priority orderings are tried. Each one implies a set of additional constraints on the task periods in the optimization problem, which effectively speed up the algorithm. The optimal solution computed on the constant utilization bound for EDF defines the initial priority ordering. The branches for different priority orderings are tried one at a time. When the performance bounding function returns a value worse than the current solution, the priority ordering is pruned.

References

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

[2] H. Aydin, R. Melhem, D. Mossé, and P. Mejía-Alarez. Optimal reward-based scheduling for periodic real-time tasks. *IEEE Transactions on Computers*, 50(2):111–130, Feb. 2001.

[3] F. Balarin, L. Lavagno, C. Passerone, and Y. Watanabe. Processes, interfaces and platforms. Embedded software modeling in Metropolis. In *Proceedings of the* 2^{nd} *International Conference on Embedded Software*, pages 407–416, Grenoble, France, Oct. 2002.

[4] E. Bini and G. C. Buttazzo. Schedulability analysis of periodic fixed priority systems. *IEEE Transactions on Computers*, 53(11):1462–1473, Nov. 2004.

[5] R. Gerber, S. Hong, and M. Saksena. Guaranteeing real-time requirements with resource-based calibration of periodic processes. *IEEE Transaction on Software Engineering*, 21(7):579–592, July 1995.

[6] J. Goossens and P. Richard. Performance optimization for hard real-time fixed priority tasks. In *Proceedings of the* 12^{th} *International Conference on Real-Time Systems*, Paris, France, Mar. 2004.

[7] M. Joseph and P. K. Pandya. Finding response times in a real-time system. *The Computer Journal*, 29(5):390–395, Oct. 1986.

[8] W. Kim, D. Shin, H. S. Yun, J. Kim, and S. L. Min. Performance comparison of dynamic voltage scaling algorithms for hard real-time systems. In *Proceedings of the Real-Time Application Symposium*, pages 219–228, San Diego, CA, June 2002.

[9] T.-W. Kuo and A. K. Mok. Load adjustment in adaptive real-time systems. In *Proceedings of the* 12^{th} *IEEE Real-Time Systems Symposium*, pages 160–170, San Antonio, TX USA, Dec. 1991.

[10] C. Lee, J. P. Lehoczky, D. Sieworek, R. Rajkumar, and J. Hansen. A scalable solution to the multi-resource QoS problem. In *Proceedings of the* 20^{th} *IEEE Real-Time Systems Symposium*, pages 315–326, Phoenix, AZ, Dec. 1999.

[11] J. P. Lehoczky, L. Sha, and Y. Ding. The rate-monotonic scheduling algorithm: Exact characterization and average case behavior. In *Proceedings of the* 10^{th} *IEEE Real-Time Systems Symposium*, pages 166–171, Santa Monica, CA USA, Dec. 1989.

[12] G. Lipari and E. Bini. Resource partitioning among real-time applications. In *Proceedings of the* 15^{th} *Euromicro Conference on Real-Time Systems*, pages 151–158, Porto, Portugal, July 2003.

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

[14] P. Martí, C. Lin, S. A. Brandt, M. Velasco, and J. M. Fuertes. Optimal state feedback based resource allocation for resource-constrained control tasks. In *Proceedings of the* 25^{th} *IEEE Real-Time Systems Symposium*, pages 161–172, Lisbon, Portugal, Dec. 2004.

[15] A. Moore. Extending the RT profile to support the OSEK infrastructure. In *Proceedings of the* 5^{th} *IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*, pages 341–347, Washington, DC, Apr. 2002.

[16] OMG standard. UML profile for schedulability and time. Available at `http://www.omg.org`.

[17] L. Palopoli, C. Pinello, A. Sangiovanni-Vincentelli, L. El Ghaoui, and A. Bicchi. Synthesis of robust control systems under resource constraints. In *Proceedings of the Hybrid Systems: Computation and Control Conference*, Stanford, CA, Mar. 2002.

[18] M. Saksena and P. Karvelas. Designing for schedulability: Integrating schedulability analysis with object-oriented design. In *Proceedings of the* 12^{th} *Euromicro Conference on Real-Time Systems*, pages 101–108, Stockholm, Sweden, June 2000.

[19] D. Seto, J. P. Lehoczky, and L. Sha. Task period selection and schedulability in real-time systems. In *Proceedings of the* 19^{th} *IEEE Real-Time Systems Symposium*, pages 188–198, Madrid, Spain, Dec. 1998.

[20] D. Seto, J. P. Lehoczky, L. Sha, and K. G. Shin. On task schedulability in real-time control systems. In *Proceedings of the* 17^{th} *IEEE Real-Time Systems Symposium*, pages 13–21, Washington, DC, Dec. 1996.

[21] M. Sjödin and H. Hansson. Improved response-time analysis calculations. In *Proceedings of the* 19^{th} *IEEE Real-Time Systems Symposium*, pages 399–408, Madrid, Spain, Dec. 1998.

IEEE COMPUTER SOCIETY