A New EDF Feasibility Test

Bruno Gaujal , Nicolas Navet

No 5125

February 27, 2004

THÈME 1

ISSN 0249-6399 ISRN INRIA/RR--5125--FR+ENG

A New EDF Feasibility Test

Bruno Gaujal [*] , Nicolas Navet [†]

Thème 1 — Réseaux et systèmes
Projet TRIO

Rapport de recherche n˚ 5125 — February 27, 2004 — 21 pages

Abstract: We present a new algorithm for testing the feasibility of a set of non-recurrent tasks (or jobs) with real-time constraints scheduled under the EDF policy (Earliest Deadline First). The proposed feasibility test has a lower complexity than the previously known test. The first step of the algorithm is to construct the Hasse diagram of the set of tasks where the partial order is defined by the inclusion relation on the tasks. Then, the algorithm constructs the shortest path in a geometrical representation at each level of the Hasse diagram. Depending on the maximal slope of each path, the set of tasks is either feasible of not.

The worst-case complexity of this feasibility test depends on structural characteristics of the set of jobs since it is the sum of the levels in the Hasse diagram. Whatever the set of jobs, this is better than the worst-case complexities of existing approaches. Furthermore, we provide a probabilistic analysis of the complexity when the tasks are random. For Poisson arrivals and exponentially distributed latencies, we show that the asymptotic complexity for assessing feasibility is $O(N \log(N))$.

As an interesting by-product, the algorithm provides a new way to derive the best speeds of the processor so as to minimize the total energy consumption while meeting the deadline of each task. The exact complexity of this algorithm (sub-cubic in the number of tasks) is lower than the complexity of all the algorithms solving the same problem, known by the authors.

Key-words: Real-Time Systems, Scheduling, Feasibility, Dynamic Voltage Scaling, Earliest Deadline First.

[*] ENS Lyon - LIP, 46 Allée d'Italie, 69007 Lyon, France. Email: Bruno.Gaujal@ens-lyon.fr
[†] LORIA, Ensem, 2 avenue de la Forêt de la Haye, 54516 Vandoeuvre, France. Email: Nicolas.Navet@loria.fr

(Résumé : tsvp)

Un nouveau test de faisabilité sous EDF

Résumé : Nous présentons un nouvel algorithme pour tester la faisabilité d'un ensemble de tâches non-récurrentes ordonnancées sous EDF. Le test proposé à une complexité plus faible que le test connu jusqu'à présent. La première étape de l'algorithme est de construire le diagramme de Hasse de l'ensemble des tâches où l'ordre partiel est défini par la relation d'inclusion sur les tâches. L'algorithme construit alors le plus court chemin d'une façon géométrique pour chaque niveau du diagramme de Hasse. En fonction de la pente maximale de chacun des plus courts chemins, l'ensemble de tâches est faisable ou non.

La complexité dans le pire cas dépend de caractéristiques structurelles de l'ensemble des tâches car elle est fonction du nombre de tâches à chacun des niveaux du diagramme de Hasse. Quel que soit l'ensemble de tâches, la complexité dans le pire cas est plus faible que celle des approches existantes. De plus, nous présentons une analyse probabiliste quand les caractéristiques des tâches sont aléatoires. Pour des arrivées Poissoniennes et des échéances exponentiellement distribuées, nous montrons que la complexité asymptotique pour décider de la faisabilité est $O(N \log(N))$ où N est le nombre de tâches.

Une implication intéressante de cet algorithme est qu'il fournit une nouvelle façon de calculer les meilleures vitesses du processeur pour minimiser la consommation totale d'énergie tout en respectant les échéances. La complexité exacte de l'algorithme, inférieure à $O(N^3)$, est plus faible que la complexité des algorithmes connus par les auteurs pour résoudre le même problème.

Mots-clé : Systèmes Temps Réel, Ordonnancement, Adaptation Dynamique du Voltage, Earliest Deadline First.

1 Introduction

Definition of the problem. Two important problems related to the scheduling of a set of independent jobs under EDF are addressed in this study : 1) the feasibility analysis and 2) the minimization of the energy consumption.

Existing solutions have shown that these problems are closely correlated; the optimal energy minimization algorithm from Yao et al. [8] embeds the feasibility analysis from Spuri [5] although both results have been developed independently. This characteristic that the solutions of the two problems are tightly linked remains true with the new algorithms developed in this study.

Existing work. In [5], an algorithm for testing the feasibility of a set of jobs scheduled under EDF is presented; it will be called the Spuri's algorithm in the following since it relies on a Theorem stated by Spuri (Theorem 3.5 pp 33 in [5]). The complexity of this algorithm is $O(N^2)$ where N is the cardinal of the set of jobs.

The study published in [8] provides, for independent tasks with deadlines, the EDF schedule that minimizes energy usage while ensuring that deadlines are not missed. This algorithm, termed the Yao's algorithm, runs in $O(N^3)$ in the worst-case (for more details on the complexity analysis, please refer to the end of Section 2). The Yao's algorithm also provides a feasibility test under EDF since, intuitively, if for meeting deadlines the CPU speed needs to be greater than its maximum speed then the set of jobs is not feasible.

Recently, a new approach for solving both problems under the light of a shortest path problem has been proposed in [3] but it is restricted to tasks with FIFO real-time constraints ($a_i \leq a_j \rightarrow d_i \leq d_j$). For such tasks, feasibility testing and energy minimization can be solved in $O(N \log(N))$.

Contribution of the paper. This work is the continuation of [3] where the special case of FIFO constraints has been treated. Here, a solution to the general case is provided, *i.e.* the FIFO assumption is removed. Note that the complexity of the algorithms depends on structural properties of the set of jobs. The exact complexities of the algorithms on a particular set of jobs is provided and, whatever the set of jobs, it is better than the complexities of existing approaches (sub-quadratic in the number of tasks for feasibility and sub-cubic for energy minimization). Furthermore, we provide a probabilistic analysis of the complexities when the tasks are random. We show that the average complexity is always smaller that $O(N\sqrt{N})$. However this bound is not tight because it does not take into account the fact that deadlines have to be larger than arrival times. In the case of Poisson arrivals and exponentially distributed latencies, we show that the asymptotic complexity for assessing feasibility is $O(N \log(N))$.

Model of the system. We consider a single CPU dedicated to the execution of a a finite set of some non-recurrent independent tasks (or "jobs") with real-time constraints. The CPU processing speed u can vary over a continuous range from 0 to 1 (the maximal speed of the processor to fixed to 1 with no loss of generality; this can be achieved by a re-scaling of the size of the jobs). The set of tasks is $\mathcal{T} = \{\tau_1, \ldots, \tau_N\}$ and each task τ_i is characterized by a triplet (a_i, s_i, d_i) where the quantities a_i, s_i, d_i respectively denote the arrival time, the size (time needed to execute the task at the maximum speed) and the deadline of task τ_i.

Organization of the paper. The Yao's algorithm for energy minimization is reminded in Section 2. Section 3 is devoted to our proposal of feasibility test under the EDF policy. The correctness of the algorithm is proved and its complexity is assessed both deterministically and probabilistically. Finally, in section 4, an original algorithm to the problem of minimizing the energy consumption under real-time constraints is proposed.

2 Existing results

In this Section, we explain the Yao's algorithm for energy minimization. We chose to present this construction with full details so that the differences with our approach can be easily highlighted and also since several concepts and notations presented here will be used in subsequent Sections. Note that the Spuri's algorithm for feasibility is the first iteration of the loop of the algorithm in paragraph 2.2.

The speed of the CPU at time t is denoted $v(t)$. We recall the construction of the optimal function $v_{\text{Yao}}(t)$ given by Yao et al. in [8]. This construction minimizes energy consumption while ensuring that no deadline will be missed. Basically, the algorithm works by identifying the time interval, termed the critical interval, over which the highest processing speed is required. The lowest admissible frequency is computed for this interval, the tasks belonging to this interval (i.e. arrival date and deadline inside the interval) are then removed (this step is termed "time compression" in the following) and a sub-problem is constructed with the remaining tasks.

2.1 Time compression

Here we explain how to suppress the interval $[a, d]$ from the time-line and how to modify the set of tasks accordingly. It has been chosen to denote the bounds of the interval by a and d since, in the following, a and d will necessarily be arrival dates and deadline dates respectively. The notation $\tau_i \subseteq [a, d]$ means that task τ_i belongs to $[a, d]$ ($a_i \geq a, d_i \leq d$). When suppressing $[a, d]$, the task set is modified in the following manner :

- if $\tau_i \subseteq [a, d]$ then τ_i is removed,
- for all remaining tasks, s_i is unchanged and
  - if $a_i \in [a, d]$ then $a_i := a$,
  - else if $a_i \geq d$ then $a_i := a_i - (d - a)$,
  - if $d_i \in [a, d]$ then $d_i := a$,
  - else if $d_i \geq d$ then $d_i := d_i - (d - a)$.

The time-compression procedure is illustrated in Figure 1.

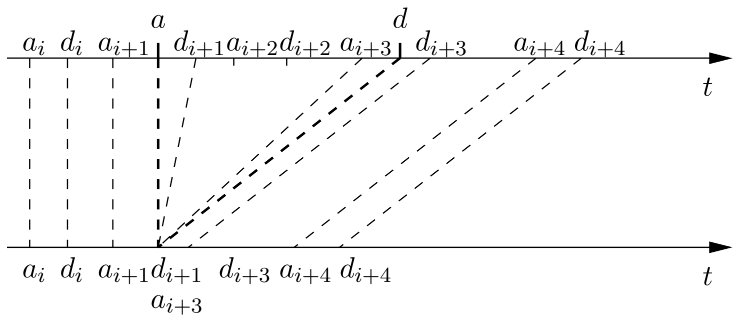

Figure 1: Example of time compression. When suppressing the interval $[a, b]$, the task τ_{i+2} is removed from the set of tasks.

.

2.2 Yao's algorithm

The Yao's algorithm can be described in a convenient way using the following "composition" of functions. Let f and g be real functions with appropriate domains and let $[a, d]$ be a time interval, then function $g \wedge_{[a,d]} f$ is defined as follows :

- if $t < a$, then $g \wedge_{[a,d]} f(t) = g(t)$,
- if $t \in [a,d]$, then $g \wedge_{[a,d]} f(t) = f(t)$,
- if $t > d$, then $g \wedge_{[a,d]} f(t) = g(t - (d - a))$.

Note that this operation is not associative and leaves implicit the domain of all functions involved. If we define $h := g \wedge_{[a,d]} f$ this means that f is defined on $[a,d]$, g is defined over an interval including a, say $[b,c]$, and h is defined over $[b, c+d-a]$.

Yao et al. define the "intensity" of an interval as the workload brought by the tasks belonging to the interval divided by the length of the interval. Intuitively, it is the smallest amount of work that has to be done during the interval for meeting the timing constraints. The intensity over $[a,d]$ is denoted $W_{[a,d]}$ and it is equal to :

$$W_{[a,d]} = \frac{\sum_{\{\tau_k \subseteq [a,d]\}} s_k}{d - a}$$

The algorithm proposed in [8] constructs the function $v_{\text{Yao}}(t)$ which is the optimal speed for minimizing the total energy consumption while respecting deadlines. This algorithm can be decomposed in the following steps :

1. $n := 1$
2. Compute the critical interval $I_n := [a_i, d_j]$ where a_i and d_j are such that
$$W_{[a_i,d_j]} = \max_{0 \leq a \leq d} W_{[a,d]}. \qquad (1)$$
In (1), only values of a and d that are respectively arrival dates and deadline dates have to be considered since $W_{[a,d]}$ possesses local maxima on such points.
3. Over the critical interval at step n, $I_n = [a_i, d_j]$, the function f_n is constant
$$f_n(t) := W_{[a_i,d_j]}.$$
4. Using time-compression, I_n is removed, let $n := n + 1$, return to step 2 if at least one task remains or go to step 4 otherwise.
5. Function v_{Yao} is completely defined : $v_{\text{Yao}} := f_n \wedge_{I_{n-1}} (\cdots \wedge_{I_1} f_1)$.

The above algorithm is cubic in the number of tasks N : there are at most N execution of step 2 which runs in $O(N^2)$ since for each arrival date there are at most N deadlines to consider. In [8], the authors claim, without more details, that using "a suitable data structure such as the segment tree", the running time can be reduced to $O(N \log^2(N))$. However, this was never really achieved as mentioned in [7]. We actually do not know how to obtain an implementation with their algorithm in less than $O(N^3)$ and we are not aware of any paper in the literature that has addressed the problem with a complexity lower than $O(N^3)$.

The rest of the paper is almost entirely devoted to the presentation of a new algorithm with a sub-quadratic complexity to compute a function which coincide with v_{Yao} on the critical interval. This will provide a feasibility test under EDF in sub-quadratic time which is an improvement over existing algorithms running in quadratic time ([8, 5]).

3 New algorithm for feasibility

The algorithm presented here relies on the construction of the Hasse diagram of a partially ordered set (poset) and on the solution of a shortest path problem that has been proposed in [3]. These two points will be first reminded. We then provide the core of the algorithm, prove its correctness and we evaluate its algorithmic complexity both in a deterministic and in a probabilistic way.

3.1 The Hasse diagram

Let us consider the poset P on the set of tasks with the partial order being defined by the strict inclusion of the intervals : $P = \{[a_i, d_i], \subsetneq\}$, where $[a, d] \subsetneq [a', d']$ if $a' < a$ and $d' > d$. A finite poset can be represented by a Hasse diagram (see [6] for more details) and the levels in the diagram are denoted $L_1, \cdots, L_K$ where K is the highest level. One denotes by $L(i)$ the level of the interval $[a_i, d_i]$ in P, it will be called the level of task τ_i. The set of tasks that will be considered for illustration purpose in the following is :

	a	d	s
τ_1	0	22	3
τ_2	2	10	1
τ_3	4	6	1
τ_4	5	13	4
τ_5	8	19	1
τ_6	11	15	2
τ_7	15	24	1

The Hasse diagram associated with this set of tasks is given in Figure 2 b) while Figure 2 a) shows how the intervals $[a_i, d_i]$ overlap.

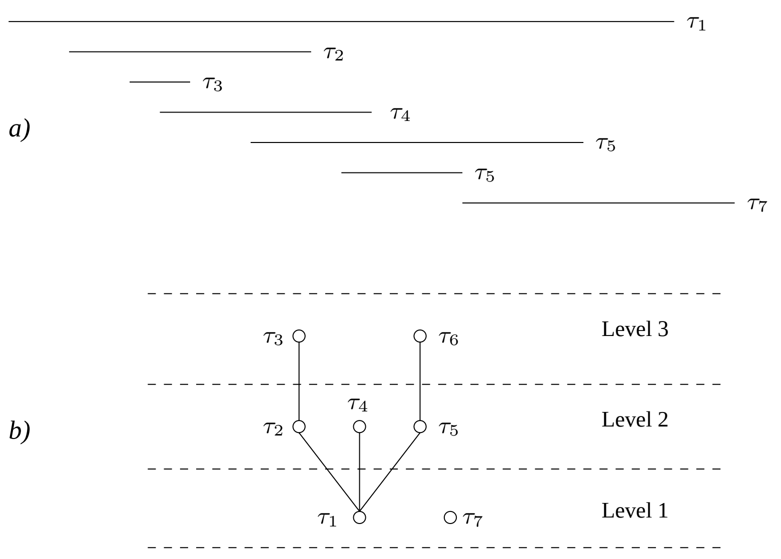

Figure 2: Sub-figure a) represents the overlapping of the intervals $[a_i, d_i]$ while sub-figure b) shows the corresponding Hasse diagram.

We now explain how to compute the levels of all the tasks in the Hasse diagram in $O(N \log(N))$ elementary operations.

An array L will be used to store the levels of the tasks : $L[i]$ is the level of τ_i. An array T stores the value of largest deadline on a given level. The arrays L and T are iteratively constructed during the execution of the algorithm given below :

1. Reorder the set of tasks so that all the indexes of the tasks are given according to the arrival times a_i where ties are broken using the deadlines.

2. $T[0] := \infty$, $T[1] := d_1$, $L[1] := 1$ and for all $k = 1..N$ set $T[k] := 0$.
3. For i from 2 to N do
   (a) Dichotomy-Search of d_i in the current T gives k, the index such that $T[k] > d_i \geq T[k+1]$.
   (b) $T[k+1] := d_i$; $L[i] = k+1$.

Note that the dichotomic search is valid because T is sorted in increasing order. The correctness of this construction comes from the fact that all interval orders (as here) are two dimensional (see for example [2]).

The total complexity of the algorithm is $O(N\log(N))$: the initial sort is in $O(N\log(N))$ and each dichotomy uses $O(\log(N))$ operations and it is repeated N times.

3.2 The shortest path construction

In [3], where the case $K = 1$ is treated (FIFO tasks), the algorithm involves the construction of the shortest path among all the paths between 0 and T with given upper and lower bounds. The construction in the general case also uses this framework, but, this time, we need to construct such a path for each level in the Hasse diagram.

For convenience, one assumes that $a_1 = 0$ (all dates can be shifted to the left) and let $T \overset{\text{def}}{=} \max_i\{d_i\}$. For all tasks with level not smaller than k in the Hasse diagram, we construct two functions $A_k(t)$ and $D_k(t)$ over the interval $[0, T]$ using the following definition:

$$A_k(t) = \sum_{L(i)\geq k} s_i \cdot \mathbf{1}_{[a_i<t]},$$

$$D_k(t) = \sum_{L(i)\geq k} s_i \cdot \mathbf{1}_{[d_i\leq t]}.$$

The functions $A_k(t)$ and $D_k(t)$ are staircase functions (*i.e.* piece-wise constant, with a finite number of pieces). Also note that A is left-continuous while D is right-continuous.

Let g be an arbitrary non-decreasing and strictly convex function. In power aware scheduling, this function is generally considered for modeling the energy consumption of the processor. The consumption at time t depends on the processor speed $u(t)$ and for instance with the CMOS technology, typically, $g(u(t)) \approx \alpha C u(t)^{1+2/(\gamma-1)}$, where $1 \leq \gamma \leq 3, \alpha \geq 0, C \geq 0$, see [4] for more details.

Mathematical Program 1. *Program P_k: Find an integrable function $u_k^* : [0,T] \to \mathbb{R}$ such that*

$$\int_0^T g(u_k^*(s))ds \text{ is minimized,} \tag{2}$$

under the constraints

$$u_k^*(t) \geq 0 \quad \forall t \in [0,T], \tag{3}$$

$$\int_0^t u_k^*(s)ds \leq A_k(t) \quad \forall t \in [0,T], \tag{4}$$

$$\int_0^t u_k^*(s)ds \geq D_k(t) \quad \forall t \in [0,T]. \tag{5}$$

Theorem 1. *The following results are all shown in [3].*

- *i For all k, the optimal solution of program P_k is unique (up to a set of measure 0).*
- *ii If u_k^* is the optimal solution of program P_k where g is strictly convex and non-decreasing, then u_k^* is also an optimal solution of P_k for any other non-decreasing convex function.*
- *iii The optimal solution u_k^* satisfies the following inequality*[1]*:* $\sup_{0\leq t\leq T} u_k^*(t) \leq \sup_{0\leq t\leq T} u_k(t)$, *for all functions u_k satisfying constraints* (3), (4) *and* (5).
- *iv The integral $U_k^* \overset{\text{def}}{=} \int_0^t u_k^*(s)ds$ gives the shortest path from 0 to T while staying between A_k and D_k (see Figure 3 for an illustration).*

Theorem 2. *([3]) If the functions A_k and D_k are given under the form of two ordered lists of points, the construction of $u_k^*(t)$ can be done in linear time [3].*

The algorithm to construct u_k^* presented in [3] is inspired from the linear time algorithm computing the convex hull of n ordered points in the plane.

[1]here, the sup operator stands for the central supremum, since all functions are only defined almost everywhere.

3.3 Description of the feasibility algorithm

One recalls that K is the highest level in the Hasse diagram. We call $\mathcal{T}_k, 1 \leq k \leq K$ the set of all the tasks at levels k and higher. The algorithm for deciding the feasibility is the following :

1. For all $k \in \{1 \cdots, K\}$
   (a) Construct $u_k^*(t)$ the optimal solution of P_k.
   (b) Define $r_k := \sup_{0 \leq t \leq T} u_k^*(t)$.
2. The set of tasks is feasible if and only if $\sup_{k=1}^K r_k \leq 1$.

An example of this construction is given in Figure 3 and Figure 4 using the same set of tasks as in Figure 2. The functions $U_i^* = \int_0^t u_i^*(x)dx$ are displayed instead of $u_i^*(t)$ in order to show why such functions are called shortest paths.

Figure 3 represents $U_3^*(t)$ and $U_2^*(t)$ while Figure 4 shows the construction of $U_1^*(t)$. We get $r_3 = 1/2$, $r_2 = 7/11$ and $r_1 = 12/22$. Note that the maximum $r_2 = 7/11$, is reached by u_2^* over the interval $[4, 15]$ that includes tasks τ_3, τ_4 and τ_6. Actually, this is the critical interval. In our example, the set of tasks is feasible since $r_2 = 7/11 < 1$.

3.4 Correctness of the algorithm

This section is devoted to the proof of the correctness of the algorithm. In the following, I_c denotes the critical interval and W_{I_c}, the intensity over the critical interval is denoted W_c. We also define the function v_{Yao}^k, as the Yao's construction over the set of tasks $\mathcal{T}_k$, $\quad 1 \leq k \leq K$. Therefore, $v_{\text{Yao}}^1 = v_{\text{Yao}}$.

Lemma 1. *The following assertions are true:*
i- $\sup_t v_{Yao}(t) = W_c$.
ii- $v_{Yao}(t) \leq 1$ *if and only if the set of tasks is feasible.*
iii- For all k, v_{Yao}^k *is a solution of Program* P_k *(not necessarily optimal).*
iv- For all $1 < k \leq K$, $\sup_t v_{Yao}^k(t) \leq \sup_t v_{Yao}^{k-1}(t)$.

Proof. The proof of points *i* is a direct consequence of the construction of v_{Yao} (see [8]). Point *ii* has been shown in [5]. Point *iii* follows from the fact that all feasible scheduling policies with a processor with speed u should satisfy constraints 3,4 and 5. As for *iv*, this is simply because $\mathcal{T}_k \subseteq \mathcal{T}_{k-1}$ and the maximal intensity grows when new tasks are added. □

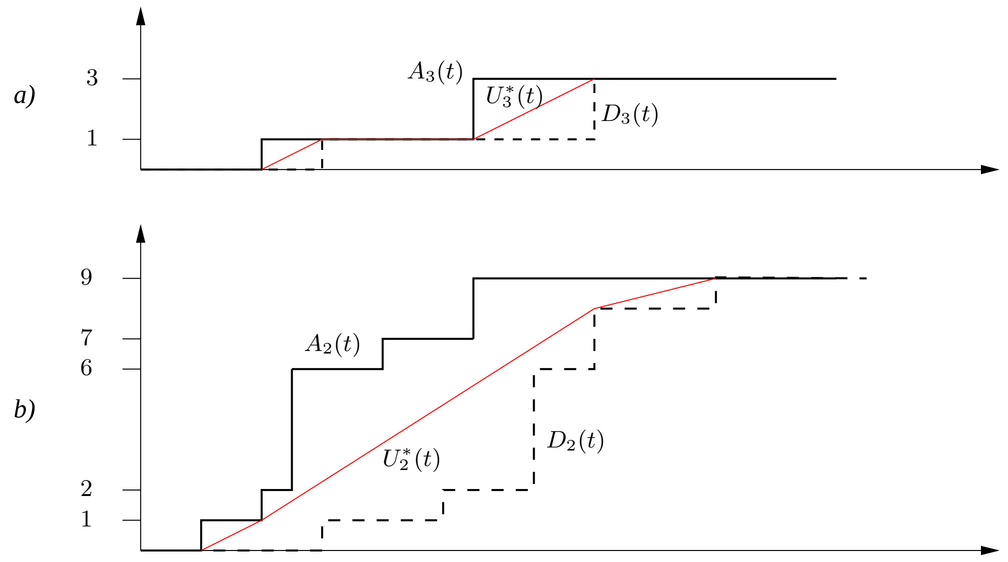

Figure 3: Construction of $u_3^*(t)$ and $u_2^*(t)$. Sub-figure a) shows $U_3^*(t) = \int_0^t u_3^*(x)dx$ and sub-figure b) $U_2^*(t) = \int_0^t v_2(x)dx$.

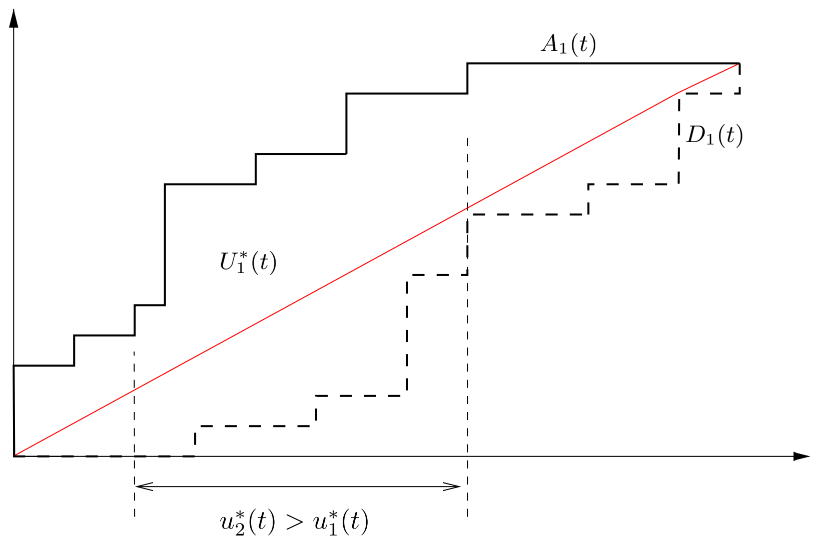

Figure 4: Construction of $U_1^*(t) = \int_0^t u_1^*(x)dx$

Lemma 2. *For all* $t \in [0, T]$, $\quad \sup_t u^*_k(t) \le \sup_t v^k_{Yao}(t)$.

Proof. This is a direct consequence of Lemma 1(*iii*) and Theorem 1(*iii*). □

Lemma 3. $\sup_k r_k \ge W_c$.

Proof. We consider all the tasks included in the critical interval $I_c = [a_c, d_c]$ as one super-task τ_c which size s_c is the sum of the sizes of all tasks included in I_c while its arrival is a_c and its deadline is d_c. Note that the level L_c of this new task is the lowest level of all tasks in I_c.

By definition of τ_c, all non critical task in $\mathcal{T}_{L_c}$ are not included in τ_c: non critical tasks starting before τ_c end before τ_c and non critical tasks starting before the end of τ_c have arrived after τ_c. This means that $D_{L_c}(d_c) - A_{L_c}(a_c) \ge s_c$. The construction of $u^*_{L_c}$ implies that $\sup_{t \in I_c} u^*_{L_c}(t) \ge D_{L_c}(d_c) - A_{L_c}(a_c)/d_c - a_c \ge W_c$.

Finally, $\sup_k r_k \ge r_{L_c} \ge \sup_{t \in I_c} u^*_{L_c}(t) \ge W_c$.

□

Theorem 3. *The set of tasks is feasible if and only if* $\sup_{0 \le k \le K} r_k \le 1$.

Proof. This is a direct consequence of Lemmas 1(*i*), 1(*ii*), 2 and 3. □

Note that $u^*_k \neq v^k_{\text{Yao}}$ in general. However, as shown above, the equality holds inside the critical interval for some k.

3.5 Deterministic analysis of the complexity

Assume that tasks have already been sorted by their arrival date as well as by their deadlines. Assume that the levels in the Hasse diagram are also given. The time complexity of constructing u^*_k is linear in the number of tasks at level k and higher as shown in [3]. Computing r_k is also linear in the number of tasks at levels larger than k. Therefore, the complexity of the algorithm to construct v_1 is (up to a multiplicative constant)

$$\sum_{i=1}^{N} L(\tau_i) = \sum_{k=1}^{K} N_i \cdot k$$

where N_i is the number of tasks at level i. The highest complexity corresponds to the case where there is one task at each level lower than K and $N_K = N - (K-1)$ tasks at level K. The complexity is thus $O(KN - K^2/2)$.

Note that this time complexity is linear if all tasks are on level 1 ($K = 1$) which is the best possible case. The worse case arises when $K = N$ with 1 tasks on each level, where the complexity becomes $N^2/2$. However, it is easy to patch this worse case and solve the problem in linear time in that case by merely testing $r_1 \leq 1$. Maybe such a patch exists in all quadratic cases.

Since $K \leq N$, this means that the complexity of the algorithm presented above is at most quadratic. An interesting problem, investigated below, is to estimate the value of K for typical set of tasks.

3.6 Probabilistic analysis of the complexity

In this section, we provide an estimation of K when the characteristics of the tasks are chosen randomly. Recall that K is the number of levels in the Hasse diagram of the N tasks. Another way to define it, is to see it as the longest increasing subsequence in the partial order of the set P.

It is a classical problem to estimate the average length of the longest subsequence in an ordered set. If the arrivals a_i are chosen randomly and the deadlines d_i are also chosen randomly and are independent of the arrivals, then one has the following result.

Lemma 4 ([1]).
$$\mathbb{E}(K) = 2\sqrt{N} + O(\sqrt{N}).$$

However, we can get a much lower estimation on the average K by taking into account the fact that the deadlines are not independent of the arrivals ($d_i \geq a_i$). In that case, the average K is much smaller than $\sqrt{N}$.

For example, here is the proof in the case the arrival process is Poisson and the relative deadlines ($d_i - a_i$) are exponentially distributed.

Lemma 5. *If the arrival process is Poisson and the relative deadlines are exponentially distributed with the same parameter, then* $\mathbb{E}(K) = O(\log(N))$.

Proof. First, note that the problem can be seen as a two dimensional problem by plugging a_i on the x-axis and $-d_i$ on the y axis. In this new framework, K is the longest sequence of component-wise increasing points $(a_{i_1}, -d_{i_1}) < \cdots < (a_{i_K}, -d_{i_K})$.

Now, if we consider any given point $(a_h, -d_h)$, the longest increasing sequence S_h starting in this point is clearly smaller that the total number T_h of points which are larger than $(a_h, -d_h)$. The rest of the proof is a direct derivation giving an upper bound on T_h.

$$\begin{aligned}
&\mathbb{P}(\max_{h=1}^{N} S_h \geq C) \leq N\mathbb{P}(S_H \geq C) \text{ where } H \\
&\text{is a random point} \\
\leq\ & N\int_{x=0}^{\infty} e^{-x} \sum_{k=C-1}^{\infty} \mathbb{P}(k \text{ arrivals in an} \\
&\qquad \text{interval of length } x))e^{-kx}dx \\
\leq\ & N\int_{x=0}^{\infty} e^{-x}\mathbb{P}(\text{ more than } C-1 \text{ arrivals} \\
&\text{in an interval of length } x)e^{-(C-1)x}dx \\
\leq\ & N\int_{x=0}^{\infty} e^{-Cx}\frac{x^C}{C!}dx = \frac{N}{(C)^{C+1}}
\end{aligned}$$

Now, if $C \geq \log(N)$, this probability goes to zero as N goes to infinity. So the tail of K does not go beyond $\log(N)$ for large N. In particular, this implies that $\mathbb{E}(K) = O(\log(N))$.

□

This means that the average complexity is $O(N\log(N))$. Actually, this proof shows a stronger result: when the number of tasks N goes to infinity, the complexity of the algorithm is below $N\log(N)$ with probability one.

Note that the average complexity of the classical algorithm of Spuri is necessarily greater than $C((N^2+N)/2)$ for some constant C whatever the distributions since this is the best possible complexity (corresponding to the case where tasks have FIFO constraints).

4 Energy minimization

In this section we consider the following problem : choose at each time t the speed $u(t)$ in such a way as to execute all tasks within their deadline constraints while minimizing the total energy consumption.

The function v_{Yao} actually provides a solution to the problem of minimizing the total energy used by the processor when the immediate energy consumption of a processor going at speed u, is of the form $g(u)$, where g is convex and increasing. This is the result claimed in [8]. In the following, we present two different algorithms that enable to compute the sequence of optimal frequencies with a lower complexity than Yao et al.'s algorithm.

4.1 Algorithm 1

A straightforward way to determine the optimal processor speeds is to iterate the algorithm for assessing feasibility : once the critical interval is identified it is removed and the feasibility algorithm is applied on the resulting sub-problem. This leads to the following algorithm :

1. For $k = 1..K$ construct $u_k^*(t)$. Let $r_i = \sup_{0 \le t \le T} u_k^*(t)$; the critical interval $[a_i, b_j]$ is such that $W_{[a_i,b_j]} = \sup_k r_k$.

2. Set the speed on $[a_i, b_j]$ equal to $W_{[a_i,b_j]}$. Using time compression, remove interval $[a_i, b_j]$. Return to step 1 if at least one task remains.

It should be noted that K, the number of levels in the Hasse diagram, cannot increase after the compression of the critical interval. Indeed if task τ_1 is not strictly included in task τ_2 before time compression, then τ_1 is not strictly included in task τ_2 after compression. On the other hand, note that K may decrease by time compression. Indeed, if τ_1 is strictly included in τ_2 before time compression and if the deadlines (or the arrival times) of both tasks fall in the compressed interval, then they are not strictly included anymore after time compression.

Since there is at most N critical intervals, the feasibility algorithm (step 1) is invoked at most N times and the worst-case complexity of the algorithm is $O(KN^2)$. In the average case, where tasks are chosen randomly with Poisson distributions for the arrival times and exponential latencies, the average complexity is $O(N^2 \log N)$. The correctness of this algorithm is implied by the correctness of the feasibility algorithm (see Theorem 3).

4.2 Algorithm 2

Another way to define v_{Yao} is by using a procedure close to the one used for feasibility.

Mathematical Program 2. *For all $1 \le k \le K$, consider the integrable function $v_k : [0, T] \to \mathbb{R}$ such that*

$$\int_0^T g(v_k(s))ds \text{ is minimized,} \tag{6}$$

under the constraints

$$\begin{aligned} v_k(t) &\geq v_{k+1}(t) \quad \forall t \in [0,T], & (7)\\ \int_0^t v_k(s)ds &\leq A_k(t) \quad \forall t \in [0,T], & (8)\\ \int_0^t v_k(s)ds &\geq D_k(t) \quad \forall t \in [0,T]. & (9) \end{aligned}$$

As for Mathematical Program 1, one can check that the function v_k is unique up to a set of measure 0.

Theorem 4. *The functions v_1 and v_{Yao} coincide: $v_1(t) = v_{Yao}(t), \quad \forall 0 \leq t \leq T$.*

Proof. (sketch) Consider the critical interval I_c in the domain of definition of all tasks, $D = [0,T]$. A direct induction on K shows that $v_1(t) = \sup_k u_k^*(t)$ over the critical interval I_c. By using Theorem 3, this shows that $v_1(t) = v_{\text{Yao}}(t)$ if $t \in I_c$.

Now, we compress time by I_c. we consider the new set of tasks $\mathcal{T}\backslash I_c$ compressed by the interval I_c. The new domain of definition is denoted $D\backslash I_c$. If f is any function defined on D for the set of tasks $\mathcal{T}$, we denote by $f\backslash I_c$ the same function constructed on $D\backslash I_c$ for the compressed set of tasks $\mathcal{T}\backslash I_c$.

We first check by induction on K that the function $v_1 = (v_1\backslash I_c) \wedge_{I_c} v_1$. If no critical tasks are at level 1, then $v_2 = (v_2\backslash I_c) \wedge_{I_c} v_2$ by induction. Furthermore, $v_1 = v_2$ over I_c because no tasks on level one are critical. This means that $v_1 = (v_1\backslash I_c) \wedge_{I_c} v_1$. If some critical tasks are on level one, we play the same trick as in the proof of lemma 3 and replace all critical tasks by one super-task of level 1. Then, $v_1 = (v_1\backslash I_c)$ over $D\backslash I_c$ and $v_1 = W_{i_c}$, the rate of the critical interval over I_c. This is direct by using the constraints used to define v_1.

Also by construction, it is clear that $v_{\text{Yao}} = (v_{\text{Yao}}\backslash I_c) \wedge_{I_c} v_{\text{Yao}}$.

Finally, using the fact that $v_{\text{Yao}}\backslash I_c$ and $v_1\backslash I_c$ also coincide over the critical interval in $D\backslash I_c$ shows that v_{Yao} and v_1 coincide over both critical intervals. By iterating this construction, one sees that v_{Yao} and v_1 are equal over all $[0,T]$. □

This gives an analytical way to define the constraints for feasibility. Now, here is a possible construction of v_1.

1. For k from K downto 1 do

   (a) Construct $u_k^*(t)$. Set $v_k := u_k^*$.

(b) As long as there exists an interval I over which $v_k(t) < v_{k+1}(t)$, remove all tasks included in I of level not smaller than $k+1$ and use time-compression for I. Do this for all such intervals. and construct $u_k^* \backslash I$ again over the compressed set of tasks and set $v_k := (u_k^* \backslash I) \wedge_I v_{k+1}$.

2. Return v_1.

An example of construction is given in Figure 5 using the same set of tasks as in Figure 2. The construction goes as follows: $v_3 = u_3^*$ and therefore it is not displayed here (see Figure 3.a). It also happens that $v_2 = u_2^*$. The integral V_2 is displayed in Figure 5 as well as U_1^*. Since $v_2 \geq u_1^*$ over the interval $I_c = [4, 15]$, we have to compute $u_1^* \backslash [4, 15]$ over the set of tasks compressed by $[4, 15]$. The function v_1 such that $v_1 = v_2$ over $[4, 15]$ and $v_1 = u_1^* \backslash [4, 15]$ on the rest, is, by definition, $v_1 = (u_1^* \backslash [4, 15]) \wedge_{[4,15]} v_2$. Now, we check that $v_1 \geq v_2$ and the algorithm stops. The integral V_1 of v_1 is displayed in Figure 5.

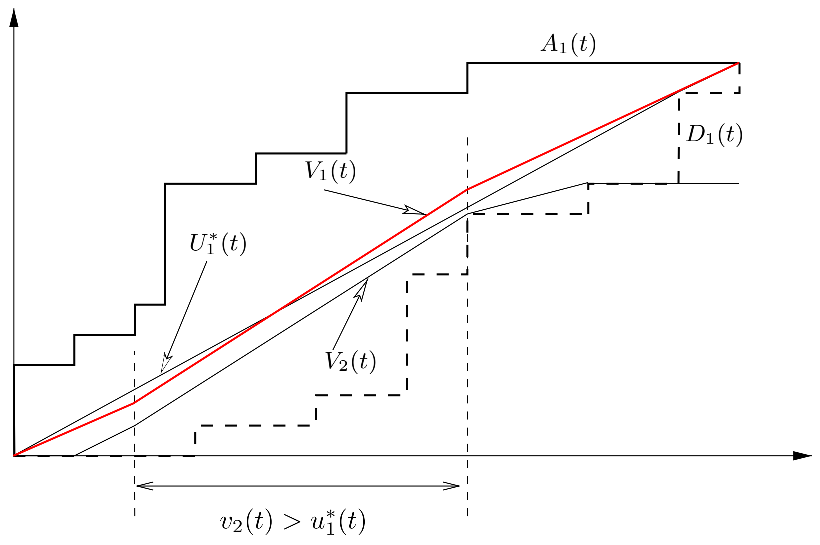

Figure 5: Construction of $v_1(t)$. The function $V_1(t) = \int_0^t v_1(x)dx$ is represented.

The algorithmic complexity of this construction is in the worse case

$$C(\sum_{k=1}^{K-1} n_k n_{k+1})$$

where n_k is the number of tasks on level k and higher and C is a constant. In the worse case, this complexity, up to the constant C, is again $O(KN^2)$. However, it is important to notice that since the time complexity has a different structure than the construction presented in Section 4.1. Indeed, it only depends on the initial Hasse diagram of all that tasks while the previous construction depends on the successive Hasse diagrams after the times compressions. Therefore, this construction is faster than the construction presented in Section 4.1 in some cases (for example if the successive critical intervals involve a single task at each step).

In the average case, where tasks are chosen randomly with Poisson distributions for the arrival times and exponential latencies, the average complexity is also $O(N^2 \log N)$.

4.3 Extensions

If g is not convex or if only a finite number of speeds are available, then, it is still possible to patch both solutions by replacing g by its convex hull, as done in [3] or computing the best solution with a finite number of speeds as done in [3].

5 Conclusion

In this study we proposed a new algorithm for assessing the feasibility of a set of independent jobs scheduled under EDF. This algorithm outperforms the existing approach of Spuri in terms of complexity. In future work, it may be possible that further complexity improvements can be achieved with local optimization procedures when tasks exhibit specific characteristics (for instance when some tasks are imbricated).

The other contribution of this study is a new algorithm for finding the optimal voltage schedule. This proposal possesses a lower complexity than the classical algorithm of Yao et al. and it enables several extensions such as finite number of speeds (see [3]).

Acknowledgments

The authors would like to thank James Martin who pointed out reference [1] and helped with the computations for the average complexity.

References

[1] D. Aldous and P. Diaconis. Hammersley's interacting particle process and longest increasing subsequences. *Probability Theory and Related Fields*, (103):199–213, 1995.

[2] P. Fishburn. *Interval Orders and Interval Graphs: A Study of Partially Ordered Sets*. Interscience Series in Discrete Mathematics. Wiley, 1985.

[3] B. Gaujal, N. Navet, and C. Walsh. Shortest path algorithms for real-time scheduling with minimal energy use. *submitted to ACM TECS*, 2003. Available as INRIA Research Report RR-4886.

[4] F. Gruian. *Energy-Centric Scheduling for Real-Time Systems*. PhD thesis, Lund Institute of Technology, 2002.

[5] J.A. Stankovic, M. Spuri, K. Ramamritham, and G.C. Buttazo. *Deadline Scheduling for Real-Time Systems: EDF and Related Algorithms*. Kluwer Academic Publisher, 1998.

[6] E. Weisstein. *The CRC Concise Encyclopedia of Mathematics*. CRC Press, 1999. ISBN 0-8493-9640-9.

[7] F. Yao. Complexity of the Yao Demers Shenker algorithm. Private communication, 2003.

[8] F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced CPU energy. In *Proceedings of lEEE Annual Foundations of Computer Science*, pages 374–382, 1995.

Éditeur
INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
http://www.inria.fr
ISSN 0249-6399