Document downloaded from:

http://hdl.handle.net/10251/37478

This paper must be cited as:

Simó Ten, JE.; Coronel Parada, JO. (2012). High Performance dynamic voltage/frequency scaling algorithm for real-time dynamic load management and code mobility. Journal of Systems and Software. 85(4):906-919. doi:10.1016/j.jss.2011.11.284.

The final publication is available at

http://dx.doi.org/10.1016/j.jss.2011.11.284

Copyright Elsevier

High performance dynamic voltage/frequency scaling algorithm for real-time dynamic load management

J.O. Coronel[*], J.E. Simó[**]

University Institute of Control Systems and Industrial Computing (AI2), Polytechnic University of Valencia, Camino de Vera s/n, 46022 Valencia, Spain

Keywords:
Dynamic voltage scaling Task migration
Real-time scheduling Power consumption Feasibility analysis

A B S T R A C T

Modern cyber-physical systems assume a complex and dynamic interaction between the real world and the computing system in real-time. In this context, changes in the physical environment trigger changes in the computational load to execute. On the other hand, task migration services offered by networked control systems require also management of dynamic real-time computing load in nodes. In such systems it would be difficult, if not impossible, to analyse off-line all the possible combinations of processor loads. For this reason, it is worthwhile attempting to define new flexible architectures that enable computing systems to adapt to potential changes in the environment.

We assume a system composed by three main components: the first one is responsible of the manage-ment of the requests arisen when new tasks require to be executed. This management component asks to the second component about the resources available to accept the new tasks. The second component performs a feasibility analysis to determine if the new tasks can be accepted coping with its real-time constraints. A new processor speed is also computed. A third component monitors the execution of tasks applying a fixed priority scheduling policy and additionally controlling the frequency of the processor.

This paper focus on the second component providing a "correct" (a task never is accepted if it is not schedulable) and "near-exact" (a task is rarely rejected if it is schedulable) algorithm that can be appli-cable in practice because its low/medium and predictable computational cost. The algorithm analyses task admission in terms of processor frequency scaling. The paper presents the details of a novel algo-rithm to analyse tasks admission and processor frequency assignment. Additionally, we perform several simulations to evaluate the comparative performance of the proposed approach. This evaluation is made in terms of energy consumption, task rejection ratios, and real computing costs. The results of simula-tions show that from the cost, execution predictability, and task acceptance points of view, the proposed algorithm mostly outperforms other constant voltage scaling algorithms.

1. Introduction

Modern cyber-physical systems (CPS) assume a complex and dynamic interaction between the real world and the computing system in real-time (Wolf, 2009). In this scenario, computing systems are commonly formed of many embedded units that are heterogeneously networked. One of the challenges facing future systems is to address the separation of modal control design from the deployment of executable code. In such systems, it would be difficult, if not impossible, to analyze off-line all the possible combinations of processor loads. Because of the large range of control process domains it is difficult to address control modality only through the parameterization of pre-loaded local controllers. For this reason, it is worthwhile attempting to define new flexible architectures that enable computing systems to optimally adapt to potential changes in the environment in a dynamic manner.

From the point of view of embedded control systems, it is well known that the integration of controller design and real-time scheduling is necessary (Cervin, 2003). However, this approach is not enough in dynamic environments with limited computing resources. Therefore, additional mechanisms must be included in the design to optimize the use of resources and provide adaptation to the changing environmental conditions. Likewise, specific quality of services (QoS) levels must be guaranteed to ensure correct system operation. These QoS levels maintain a suitable control performance and a temporal predictability in the control system.

In this context, changes in the system operation mode, or in the environment, may force the disabling of some controllers and the enabling of others. In addition, controller switching mechanisms must be added in both local and distributed forms, resulting

[*] Corresponding author. Tel.: +34 665368512; fax: +34 963877579.
[**] Corresponding author. Tel.: +34 963877000x75706; fax: +34 963877579.
E-mail addresses: jacopa@ai2.upv.es (J.O. Coronel), jsimo@disca.upv.es (J.E. Simó).

in workload changes for the embedded and networked units (Emberson and Bate, 2007; Real and Crespo, 2004). This can have a significant impact on the overall performance and real-time constraints of the system. For this reason, task and component migration approaches (Briao et al., 2007) may contribute to on-line redistribution and reallocation of workload – according to each situation.

An example of this kind of architecture is proposed in Simarro et al. (2008), which is developed in the context of a distributed real-time control system and from the perspective of control kernel middleware (Crespo et al., 2006; Albertos et al., 2006). The proposal includes features such as control basis, controller switching, and code delegation. From the point of view of CPS, these approaches partially bridge the abstraction gap (Lee, 2006).

In summary, cyber-physical systems should be able to deliver new levels of performance and efficiency thanks to sophisticated control computing co-design. Control systems mean that we must change our understanding of computing systems and accept that cyber-physical systems actively engage with the real world and real-time restrictions – and so expend real energy.

This paper explores methods that can be used in task migration when combined with techniques of energy management. These methods can maximize overall energy savings; facilitate thermal chip management by moving tasks away from the hot processing unit; balance the workload or concentration of parallel processing elements; decrease communication among those tasks that are particularly energy-efficient on wireless sensor networks (WSN) systems (Yi et al., 2009; Yuan and Wang, 2008; Kumar and Manimaran, 2007); and help guarantee the fulfillment of real-time system constraints.

Several power-aware techniques have been widely addressed in real-time literature (Piao et al., 2009; Tavares et al., 2008; Aydin et al., 2006; Pillai and Shin, 2001). These papers have discussed dynamic voltage scaling (DVS) of the processor, also known as dynamic voltage and frequency scaling. This approach exploits the convex, and normally quadratic relationship between CPU energy consumption and voltage. Additionally, these techniques have been extended to reduce the energy consumed during memory cycles (Cho and Chang, 2006; Liang et al., 2008) and network energy consumption (Yi et al., 2009; Kumar et al., 2008).

Let's consider a dynamic environment where an embedded and networked system operates with the support of task migration and processor frequency scaling. Assuming that the system knows where and when it must allocate tasks (Briao et al., 2007; Emberson and Bate, 2007), we must perform feasibility analyses when each task arrives and departs. This guarantees that the temporal requirements of the system will be accomplished during the task allocation phase. Additionally, a new processor speed (frequency scaling) should be also computed to enable the system to adapt itself to the new computational workload and so reduce energy consumption. Although some authors have carried out these two phases (feasibility analysis and frequency scaling computation) separately (AlEnawy and Aydin, 2005), these analyses are strongly related and in some cases can be performed together.

The main motivation to use fixed priorities approaches is to accomplish the standard ARINC-653 (AASS Interface, 2003), which is used mainly in Avionics and currently also in Aerospace environment (Windsor and Hjortnaes, 2009). This standard defines a Time and Space partitioning system and each partition should be executed on fixed priority scheduling approaches. An example is the RTEMS O.S. on XtratuM hypervisor, which we are currently working (Galizzi et al., 2011, 2012). However, we use the EDF-based scheduler to compare with the proposed approach by suggestion of a reviewer. As is well known, the EDF algorithm has a higher performance than fixed priority based scheduling, but we have design constraints.

This work mainly focuses on a proposal for a new algorithm to be used on-line during the task allocation and processor speed assignment phases. The algorithm uses fixed priority scheduling schemes with deadlines less than, or equal to, the period of the tasks when the dynamic processor workloads are being handled.

1.1. Related work

Few works cover task migration in the context of embedded and networked systems. Moreover, most algorithms are aimed at multiprocessor systems with soft real-time constraints (Yazdi et al., 2008; Emberson and Bate, 2007; Anderson et al., 2005). Some papers have addressed both task migration and energy consumption minimization (Briao et al., 2007; Chen, 2005). Briao et al. (2007) analyze the impact of task migration and justify its use because it compensates for the performance and energy costs involved in the system.

Several heuristics has been proposed for task allocation problems with deadlines equal to the periods (Mejia-Alvarez et al., 2004; Chen, 2007) when earliest-deadline-first (EDF) scheduling is in use. AlEnawy and Aydin (2005) use algorithms for allocating real-time tasks by applying rate monotonic (RM) scheduling. In their article, a comparison of some admission control algorithms is presented in function of complexity, feasibility, and energy consumption parameters. However, only tasks with deadlines that are equal to the period are analyzed. Moreover, aspects such as predictability of execution, behaviour related to the arrival of tasks, and the real computing cost of the algorithms are not taken into account.

At the CPU-level, DVS techniques can be classified as *static* or *dynamic*. Static algorithms for hard real-time systems use parameters such as period or minimum inter-arrival time, and assume that each task executes its worst-case execution time when selecting the processor frequency, and that this is statically decided before execution. Dynamic algorithms are based on the reclamation of additional slack resulting from the early completions of tasks. These are then used to further reduce the processor frequency and save more energy. These algorithms are applied at run-time.

Using static algorithms we can obtain a single processor frequency that never changes, or obtain variable frequencies that are statically decided before execution. An example of the former is Saewong and Rajkumar (2003), in which an algorithm that chooses a single frequency for a fixed priority scheduling scheme is proposed. In Bini et al. (2005) the authors present a method for approximating any speed level with two given discrete values that are switched as a pulse width modulation signal in order to obtain the average value. For the latter, in Mejia-Alvarez et al. (2004) and Saewong and Rajkumar (2003), the authors propose an approach whereby each task is assigned a different frequency. Several authors have published works that find the optimal voltage schedule for recurrent tasks – examples for periodic tasks include Liu and Mok (2003) and Yun and Kim (2003) and examples for periodic and aperiodic tasks include Zhong et al. (2007) and Scordino and Lipari (2006). Furthermore, speed function and the characterization of the points in time at which speed changes occur has been presented in Liu and Mok (2003) and Gaujal and Navet (2007). However, a drawback in the works with statically established variable frequencies can appear if a task activation is lost or delayed. The drawback is that the entire frequency assignment will be affected, and this leads to missed deadlines.

Some authors, such as Piao et al. (2009), have proposed dynamic algorithms working in combination with static methods. These algorithms can be divided into inter-task and intra-task methods. In the inter-task algorithms, the processor frequency is determined task-by-task, whereas intra-task algorithms may adjust the frequency within the boundaries of a given task. In Kim et al. (2002), a performance comparison of several dynamic techniques is presented. In Pillai and Shin (2001), the authors propose the algorithms named Cycle Conserving DVS for both EDF and RM schedulers and Look-Ahead for EDF. Saewong and Rajkumar (2003), Quan (2004), and Leung (2005) propose a simple dynamic scheme for fixed priorities. Aydin et al. (2004) presents a generic dynamic reclaiming algorithm, as well as an adaptive and speculative speed adjustment mechanism. Dynamic frequency changes for periodic and aperiodic tasks are discussed in Piao et al. (2009).

The use of elastic scheduling to improve DVS management has been proposed in Marinoni and Buttazzo (2007) and in Zhu et al. (2004). Xia et al. (2008) proposes energy management based on a feedback control scheduling methodology. The processor DVS analysis has been extended by other authors to reduce energy consumption in the memory (Cho and Chang, 2006; Liang et al., 2008) and network levels (Yi et al., 2009; Kumar et al., 2008).

However, some of the works mentioned above are based on an analysis of the hyper period. This approach can result in hard computing and is unsuitable for execution in run-time during task migration. Some other works consider only deadlines equal to periods and although this can be considered in specific cases, it can result in hard simplifications being applied in embedded control systems. Additionally, some studies have applied heuristics (Jejurikar and Gupta, 2004; Mejia-Alvarez et al., 2002) to obtain solutions that are sometimes far from optimal.

1.2. Contributions and paper organization

Global energy consumption in the system can contribute to the load balance criteria in the code deployment phase. This criterion can be combined with others such as schedulability, communication delays, control application correctness, and stability to determine the dynamic code movement and on-line load balancing in a system. However, we focus on the problem of task allocation, and more precisely on the feasibility analysis algorithms performed at task arrival or departure, as well as in energy management.

In this paper, we present novel algorithm that perform feasibility analyses and compute new processor frequencies when set tasks arrive and/or depart. Through extensive simulations, we evaluate the performance of this algorithm against other existing feasibility tests that have been adapted to compute the minimum processor frequency. This minimum frequency is computed in terms of energy consumption, acceptability ratio, and real computing costs. In addition, predictability in the execution and behaviour of the algorithms in relation to the continuing arrival of tasks is analyzed. These terms will be explained later, particularly the real computing cost of algorithms. Our algorithms are based on DVS static algorithms and search for a single processor frequency while using a fixed priority scheduling scheme.

This paper is organized as follows. Section 2 describes the task model, processor model, and the evaluation of computational costs. In Section 3 we introduce energy minimization and the task acceptability problem in a dynamic environment with variable loads. In Section 4 an overview of a schedulability method-based feasibility region is presented. In Section 5 the adaptation of several existing feasibility tests for computing the processor scaling factor is shown. Additionally, in this section a proposed $\mathcal{A}$ approach is presented. Section 7 presents the results of the simulations used to evaluate the performance of several approaches when computing the α factor. And finally, Section 8 states our conclusions and future work.

2. System model

2.1. Task model

In this paper we use a periodic task model. Let $\mathcal{T} = \{\tau_1, \tau_2, \ldots, \tau_n\}$ denote a task set of n real-time preemptible tasks running on a uniprocessor system. Each task $\tau_i(T_i, D_i, C_i)$ is characterized by a period T_i, a relative deadline D_i, and a worst-case execution time C_i. We assume critical instants of activations for each task. Tasks are scheduled by a fixed priority scheduler (RM or DM) and ordered from highest to lowest priority, meaning that τ_1 has the highest priority and τ_n has the lowest priority.

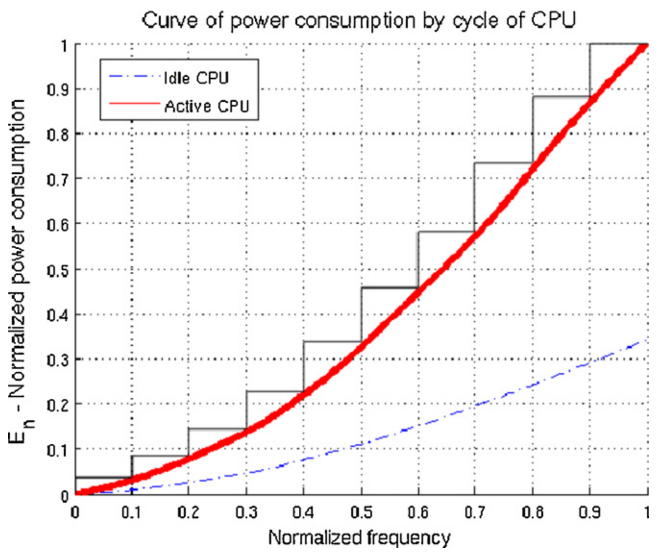

Fig. 1. Normalized reference model of power consumption by CPU cycle as used in this paper.

We consider that only real-time tasks are executed in the CPU, this point being a simplification for the analysis rather than a restriction.

2.2. Processor model

The power consumption ($\mathcal{P}$) per CPU cycle is proportional to $C_L \cdot V_{dd}^2 \cdot f$, where C_L is the average load capacity, V_{dd} is the supply voltage of the processor, and f is the operating frequency of the processor. The maximum operating frequency is a direct consequence of the supply voltage given by $f = K((V_{dd} - V_{th})^{\sigma})/V_{dd}$ where K is a constant specific for a given technology, V_{th} is the threshold voltage, and σ is the velocity saturation index where $1 \le \sigma \le 2$ (Saewong and Rajkumar, 2003; Pouwelse et al., 2001). However, the exact form of the power and frequency relation specifically depends on the processor hardware, and can be expressed in terms of CPU speed as a second or third degree polinomial function (Li and Ding, 2001; Aydin et al., 2004; Zhong et al., 2007; Marinoni and Buttazzo, 2007). In this article, the curve of function $\mathcal{P}$ (see Fig. 1) was obtained for the relationship between power and speed on the Intel XScale processor (Zhong et al., 2007), while taking into account the processor consumption in idle state.

Considering the fact that most commercially available processors only provide a limited number of voltage levels, in Fig. 1, a discrete model has been drawn over the continuous model of the processor. This discrete model consists of 10 levels of power consumption, i.e. we assume that the processor can be scaled to 10 different frequencies. In this way, the continuous model can be mapped to a discrete model.

For the analysis of the behaviour of several DVS (dynamic voltage scaling) algorithms we assume that the processor voltage can be changed continuously, and therefore, also the frequency. If the frequency obtained using these approaches is between two discrete levels, the frequency that should be used is just the higher. In this way we can guarantee that the temporal requirements of the system will be accomplished. Although it is a way simple and fast to extend the analysed algorithms to a discrete model, researchers have proposed algorithms to map continuous voltage levels to discrete levels in a way more optimal. Such as the proposed in Bini et al. (2005).

We therefore do not explore the effect of a limited number of processor frequencies. However, thanks to advances in power-supply electronics and CPU design (Duan and Khatri, 2006) systems are increasingly able to operate using a wider voltage spectrum. An example of this is the evolution in the last years of the Intel processors that uses SpeedStep technology (Intel, 2004). This technology has three versions: SpeedStep, SpeedStep II and SpeedStep III. In the first versions the clock frequency can be stepped in 200 MHz increments and in the last version the CPU varies its frequency in increments of 100 MHz, which increments the number of discrete levels available.

We also assume that the operating frequency will be adjusted by a normalized scaling factor α ($0 < \alpha \leq 1$), which will be the equivalent to C scaled by a factor $1/\alpha$ and not affect the period nor the task deadlines. When α is equal to 0, the processor is in turn-off mode, and when α is 1, the processor runs at maximum clock frequency.

2.3. Evaluation of costs

To compare different proposals, some authors such as Bini and Buttazzo (2004) and Bini et al. (2003) have proposed the use of schedulability complexity tests based on the number of steps and iterations. However, if these comparisons are made from the viewpoint of computational cycles, the results could be strongly altered. For example, the cost of 100 loop sums is not the same as 100 loop divisions. Therefore, the complexity of algorithms to calculate a suitable *alpha* (α) on embedded systems should be evaluated by the number of machine cycles required for each mathematical operation: especially on-line executions. In most embedded systems, it is well known that division operations have a higher computational cost than other operations. The routines of integer division using the Newton–Raphson approach can last between 20 and 100 cycles (Sloss et al., 2004), depending on the implementation and range of input operands. Therefore, these aspects will be considered in the evaluation of algorithms.

We look at the assembly code implementing the algorithms, and counted the execution cycles according to it. Cache is disabled in order to get predictability in the execution of the algorithm. We provide the execution time in machine cycles instead of time units because it will permit to describe the execution time in different ways in function of the processor frequency. For example, the cost at a frequency of 100 MHz is not the same as execute the algorithms at 1 GHz.

3. Scaling frequency and energy consumption with dynamic loads

Energy consumption has been defined as an integral over the time t of $\mathcal{P}$:

$$E(S_y, t) = \int_0^{t_a} \mathcal{P}(F(t, \mathcal{T}(t)))dt$$

where S_y identifies the type of scheduling, and suffix y can be FP or DP for fixed and dynamic priority scheduling, respectively. $\mathcal{P}$ is the power consumed by the CPU cycle and this depends on the CPU frequency function. The clock frequency (F) is composed as a function of time t and $\mathcal{T}$ task sets for each time instant t. Fig. 2 shows an example of the scaling of processor frequency as a function of time and task set. In this figure, two different task sets are represented over time. Obviously, the profile frequency scaling factor

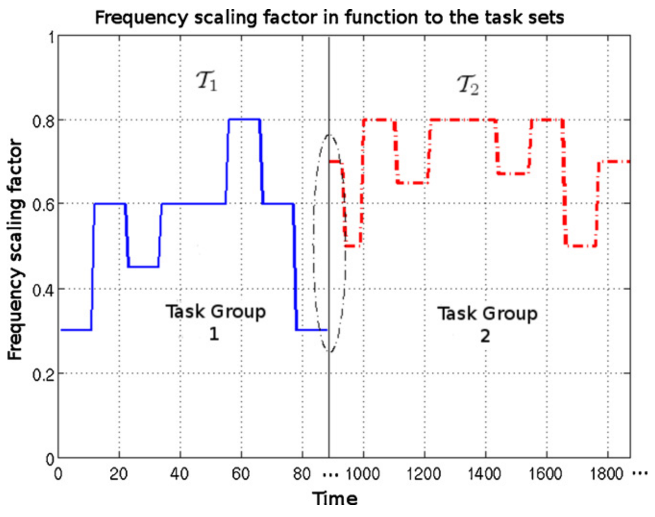

Fig. 2. Example of the scaling of processor frequency α as a function of time and task set.

$\alpha(t)$ should change just when the processor changes the task set. This is because each α is calculated for a given task set.

The switching between task sets is represented in Fig. 2 as a dashed elipse. The resultant task set is not known in advance because these are dynamically defined according to code delegate (Posadas et al., 2008; Emberson and Bate, 2007). Nevertheless, existing mode change protocols for real-time systems (Real and Crespo, 2004) can be adjusted and applied to the movement of real-time components. These change the task set demand by:

- repeating the schedulability analysis for the whole real-time system,
- recalculating the function $\alpha(t)$ for scaling the frequency of the processor.

These analyses can be performed together. One important aspect to consider in task migration is the computational cost involved in recalculating these parameters. This is because this calculation may mean an increase in the energy consumption and also an increase in the time to perform the incorporation or rejection of tasks.

After a migration of components, a *resultant task set* ($\mathcal{T}_{x+1}$) will be understood as a modification of the current task set ($\mathcal{T}_x$) due to inclusions and expulsions of tasks. Therefore, the $\mathcal{T}_{x+1}$ set will be defined as:

$$\mathcal{T}_{x+1}^N = \mathcal{T}_x^n + \mathcal{T}_I^m - \mathcal{T}_E^p = \mathcal{T}_x^n + \Delta\mathcal{T}$$

where the size of $\mathcal{T}_{x+1}$ is $N = n + m - p$. $\mathcal{T}_I$ is the task set that migrates to the processor, and $\mathcal{T}_E$ is the task set that migrates from the processor, where $\mathcal{T}_E \subseteq \mathcal{T}_x$.

The energy consumption function when assuming the mobility of components at instants of time t_x will be given by:

$$E(S_y, t_a) = \int_{t_0}^{t_1} \mathcal{P}(F(t, \mathcal{T}_0))dt + \cdots + \int_{tx}^{t_{x+1}} \mathcal{P}(F(t, \mathcal{T}_x))dt + \cdots + \int_{t_{x+k}}^{t_a} \mathcal{P}(F(t, \mathcal{T}_{x+k}))dt \quad (1)$$

where $F(t, \mathcal{T}_x)$ is the function of processor speed obtained from the task set $\mathcal{T}_x$ during a time period.

4. An overview of a schedulability—analysis based feasibility region

In this article the schedulability analysis and static minimum solutions to α are addressed from the perspective of an analysis in C-space. In this space, the task computation times C_i are considered as variable parameters, whereas the periods and the deadlines are fixed parameters. Consequently, we obtain a constraint on the C_i variables, which is a function of all T_i and D_i.

The analysis is reduced to verify if a point is inside a region delimited by coordinates C_i. This region is known as $\boldsymbol{R}_n$, and was originally proposed in Lehoczky et al. (1989).

The formal formulation of the feasibility region $\boldsymbol{R}_n$ was derived from Lehoczky et al. (1989) and conveniently demonstrated in Bini and Buttazzo (2004) with the following theorem:

Theorem 1 (Theorem 2 in Bini and Buttazzo (2004)). *When deadlines D_i are equal to periods T_i, the region $\boldsymbol{R}_n$ of a schedulable task set is given by:*

$$\boldsymbol{R}_n(T_1,\dots,T_n,D_1,\dots,D_n)|_{D_i=T_i}$$
$$= \{(C_1,\dots,C_n)\in\mathbb{R}_+^n : \max_{i=1,\dots,n}\min_{t\in\mathcal{S}_i}\sum_{j=1}^{i}\left\lceil\frac{t}{T_j}\right\rceil C_j \le t\} \tag{2}$$

where $\mathcal{S}_i=\{kT_j : j=1,\dots,i,\ k=1,\dots,\lfloor T_i/T_j\rfloor\}$.

The elements of $\mathcal{S}_i$ conceptually represent the arrival times of tasks with a priority higher than τ_i before deadlines D_i and D of task τ_i, and assuming $\phi_i=0$. This subset of values is called *rate monotonic scheduling points* $\mathcal{S}$. In Bini and Buttazzo (2004), this theorem is formulated through logical operators to represent the max ($\wedge$) and min ($\vee$).

For a better understanding and characterization in terms of complexity and implementation costs of this approach, we will apply the following notation for a matrix representation of Eq. (2). The element set $\mathcal{S}_i$ is represented by the matrix:

$$\mathcal{S}_i=\left\{s_{kl}\right\}\ k\in[1,\dots,i]\wedge l\in[1,\dots,\lfloor T_i/T_1\rfloor] \tag{3}$$

where s_{kl} is equal to the product:

$$s_{kl}=\begin{bmatrix}T_1\\T_2\\\cdots\\T_k\end{bmatrix}\otimes\begin{bmatrix}1&2&3&\cdots&\left\lfloor\frac{T_i}{T_k}\right\rfloor\end{bmatrix}=\begin{bmatrix}s_{11}&s_{12}&\cdots&s_{1l}\\s_{21}&s_{22}&\cdots&s_{2l}\\\cdots&\cdots&&\cdots\\s_{k1}&0&\cdots&0\end{bmatrix}$$

The column numbers in each row of the matrix s_{kl} are determined by the difference between period tasks ($\lfloor T_i/T_k\rfloor$), thus several elements from the matrix could be 0, and as a result, the set $\mathcal{S}_i$ can be defined as $\mathcal{S}_i^*=\mathcal{S}_i/s_{kl}\neq 0$.

The total number of elements of $\mathcal{S}_i^*$ is determined by:

$$\text{Size}(\mathcal{S}_i^*)=\sum_{j=1}^{i}\left\lfloor\frac{T_i}{T_j}\right\rfloor \tag{4}$$

Moreover, Eq. (2) can be expressed again as:

$$\max_{i=1,\dots,n}\min M_i(\mathcal{S}_i)\le\mathcal{S}_i \tag{5}$$

where $M_i(\mathcal{S}_i)$ is:

$$M_i(\mathcal{S}_i)=\{m_{kl}\}_i$$
$$\{m_{kl}\}_i=\sum_{j=1}^{i}\left\lceil\frac{s_{kl}}{T_j}\right\rceil C_j\quad s_{kl}\in\mathcal{S}_i^* \tag{6}$$

where m_{kl} is a matrix with the same dimension as the matrix s_{kl}. This will result in a matrix sum:

$$\{m_{kl}\}_i=\begin{bmatrix}\left\lceil\frac{s_{11}}{T_1}\right\rceil C_1\cdots\left\lceil\frac{s_{1l}}{T_1}\right\rceil C_1\\\left\lceil\frac{s_{21}}{T_1}\right\rceil C_1\cdots\left\lceil\frac{s_{2l}}{T_1}\right\rceil C_1\\\cdots\quad\cdots\cdots\\\left\lceil\frac{s_{k1}}{T_1}\right\rceil C_1\cdots\left\lceil\frac{s_{kl}}{T_1}\right\rceil C_1\end{bmatrix}+\cdots+\begin{bmatrix}\left\lceil\frac{s_{11}}{T_i}\right\rceil C_i\cdots\left\lceil\frac{s_{1l}}{T_i}\right\rceil C_i\\\left\lceil\frac{s_{21}}{T_i}\right\rceil C_i\cdots\left\lceil\frac{s_{2l}}{T_i}\right\rceil C_i\\\cdots\quad\cdots\cdots\\\left\lceil\frac{s_{k1}}{T_i}\right\rceil C_i\cdots\left\lceil\frac{s_{kl}}{T_i}\right\rceil C_i\end{bmatrix}$$

From this expression, it is possible to determine for which repeated values s_{kl} gives redundant values of m_{kl} as the result. Therefore, $\mathcal{S}_i^*$ can be redefined as:

$$\mathcal{S}_i^*=\frac{\mathcal{S}_i}{s_{kl}}\neq 0\wedge\exists!s_{kl}$$

Region $\boldsymbol{R}_n$ (Eq. (2)) can be expressed as:

$$\boldsymbol{R}_n(T_1,\dots,T_n,D_1,\dots,D_n)|_{D_i=T_i}$$
$$=\{(C_1,\dots,C_n)\in\mathbb{R}_+^n:\ \max_{i=1,\dots,n}\min M_i(\mathcal{S}_i^*)\le\mathcal{S}_i^*\} \tag{7}$$

where $M_i(\mathcal{S}_i^*)$ is defined in Eq. (6). Region $\boldsymbol{R}_n$ is delimited by planes which define the space where the points C_i must be located. However, for analysis of large task sets, the number of equations to be checked is huge and equals the sum of the number of elements in all S_i (such as shown in Eq. (4)). This prevents a practical on-line application of Theorem 1. Nevertheless, such as is demonstrated in Bini and Buttazzo (2004), solving Eq. (7) results in many equations that are useless, so the idea is to reduce the number of equations by eliminating the redundant elements in $\mathcal{S}_i$. This approach has led to the formulation of the following theorem:

Theorem 2 (Theorem 3 in Bini and Buttazzo (2004)). *The region of the schedulable task sets $\boldsymbol{R}_n$, such as defined by (2), is given by:*

$$\boldsymbol{R}_n(T_1,\dots,T_n,D_1,\dots,D_n)$$
$$=\{(C_1,\dots,C_n)\in\mathbb{R}_+^n:\ \max_{i=1,\dots,n}\ \min_{t\in\mathcal{P}_{i-1}(D_i)} C_i+\sum_{j=1}^{i-1}\left\lceil\frac{t}{T_j}\right\rceil C_j\le t\} \tag{8}$$

where $\mathcal{P}_i(t)$ is defined by the following expression:

$$\begin{cases}\mathcal{P}_0(t)=\{t\}\\\mathcal{P}_i(t)=\mathcal{P}_{i-1}\left(\left\lfloor\frac{t}{T_i}\right\rfloor T_i\right)\cup\mathcal{P}_{i-1}\{t\}\end{cases} \tag{9}$$

The total number of elements of $\mathcal{P}_{i-1}(t)$ is determined by:

$$\text{Size}(\mathcal{P}_{i-1})=\sum_{n=1}^{i}2^{n-1} \tag{10}$$

Note that the difference between this theorem and Theorem 2 is only the presence of the $\mathcal{P}_{i-1}(t)$ set, instead of $\mathcal{S}_i$. $\mathcal{P}_{i-1}(t)$ is a $\mathcal{P}_{i-1}(t)\subseteq\mathcal{S}_i$, and this strongly reduces the number of equations to be evaluated. As a consequence, the time needed to establish whether a set task is inside the region $\boldsymbol{R}_n$ is limited.

Following the same expression as Eq. (5), a periodic task set is feasibly schedulable using Theorem 2 if and only if:

$$\max_{i=1,\dots,n}\min M_i(\mathcal{P}_{i-1})\le\mathcal{P}_{i-1}^* \tag{11}$$

where $\mathcal{P}_{i-1}^*=\mathcal{P}_{i-1}\ /p\ \exists!p$.

5. Computing the processor scaling factor

In this section, we state – as based on the test described above and other schedulability tests – that it is possible to compute the frequency scaling factor α for task sets with $D = T$ and $D \leq T$. Furthermore, we establish that if this same computing approach can be used for analyzing the schedulability of periodic task sets.

Not all execution cycles are scaled for the processor speed because some operations deal with memory, or other I/O devices, whose access time may be fixed (Bini et al., 2005). However, access to memory can also be performed at different speeds (Marvell, 2008), and so we should use two scaling factors: α to scale the processor; and β to scale the system bus. To simplify the analysis, we assume β as a fixed parameter equal to 1, such that the worst-case execution time C can be expressed as a parameter split in two: a parameter C^f which is dependent on the processor frequency; and a fixed parameter C^m which is not.

Therefore, C_i can be expressed as the execution time at the maximum frequency of the processor with respect to the frequency scaling factor α plus a constant execution time:

$$C_i = \frac{C_i^f}{\alpha} + C_i^m \tag{12}$$

5.1. LL approach

Theorem 3. *The computing of the frequency scaling factor α using the well-known approximate schedulability test of Liu Laylan (Sha et al., 2004) (LL) is given by:*

$$\alpha_x^{\mathrm{LL}} = \frac{U_f}{n(2^{1/n} - 1) - U_m} \tag{13}$$

where U_f is the sum of the whole utilization of the part of the task set that depends on the processor frequency, U_m is the utilization of the part of the task set that is independent of the processor.

After a *migration of components*, the new scaling factor LL $\alpha_{x+1}^{\mathrm{LL}}$ is defined as:

$$\alpha_{x+1}^{\mathrm{LL}} = \frac{(U_f^{\alpha_x}(t_x) + \Delta U_f^{\alpha_x}(t_{x+1}))\alpha_x}{n(2^{1/n} - 1) - U_m(t_x) - \Delta U_m(t_{x+1})} \tag{14}$$

where ΔU is the difference of utilization between task inclusion and expulsion. And α_x is the frequency scaling factor before the migration of components. However, it is well-known that the LL test is a sufficient but unnecessary test, and so the scaling factor α^{LL} may not be the minimum α that minimizes energy consumption. This test could be used only for a task set with $D = T$.

5.2. HB approach

Theorem 4. *The frequency scaling factor based on the approximate hyperbolic bound (Bini et al., 2003) test can be computed as:*

$$\alpha_x^{\mathrm{HB}} = \max\{\alpha_n\} \tag{15}$$

where α_n represent the n possible values that can be assigned to alpha, and which are the result of solving the roots of the product of all the n utilizations plus one $\left(\prod_{i=1}^{n}((U_i^f/\alpha_x) + U_i^m + 1) - 2 = 0\right)$ of the task set on the computer system.

To obtain the scaling factor after a *migration of components*, it is necessary to solve the product by deducing α_{x+1} from the equation:

$$2 = \prod_{i=1}^{n}\left(\frac{U_i^{f\alpha_x} \cdot \alpha_x}{\alpha_{x+1}} + U_i^m + 1\right) \cdot \frac{\prod_{j=1}^{inc}(((U_j^{f\alpha_x} \cdot \alpha_x)/\alpha_{x+1}) + U_i^m + 1)}{\prod_{k=1}^{exp}(((U_k^{f\alpha_x} \cdot \alpha_x)/\alpha_{x+1}) + U_i^m + 1)}$$

where *inc* and *exp* are, respectively, the number of tasks included and expelled to and from the current ($\mathcal{T}(t_x)$) task set. α_x is the frequency scaling factor before the migration of components. In the same way as the LL approach, the HB approach cannot be used for task sets where $D < T$.

5.3. EDF-U approach

Theorem 5. *The computing of the frequency scaling factor α using the well-known schedulability test EDF utilization based (Sha et al., 2004) when $D = T$ is given by:*

$$\alpha_x^{\mathrm{EDF-U}} = \frac{U_f}{1 - U_m} \tag{16}$$

where U_f is the sum of the whole utilization of the part of the task set that depends on the processor frequency, U_m is the utilization of the part of the task set that is independent of the processor.

When $D < T$, we use a simple and approximate utilization-based admission test. α is calculated in the same way as in Eq. (16), but in the calculation of the utilization instead of using the period is used the deadline: $U = C/D$.

5.4. LLM approach

As an extension of LL test, an utilization bound test for tasks with pre-period deadlines has been used Racu et al. (2005), and some changes were made to find an approximate value for α. We termed this the LLM test.

Theorem 6. *The frequency scaling factor based on an approximate utilization bounds test is given by:*

$$\alpha^{\mathrm{LLM}} = \max_{i=1,\ldots,n} \frac{f_i^f}{U(p, \Delta_i) - f_i^m} \tag{17}$$

where f_i^f and f_i^m are, respectively:

$$f_i^{f/m} = \sum_{j \in H_p} \frac{C_j^{f/m}}{T_j} + \sum_{k \in H_1} \frac{C_k^{f/m}}{T_i} + \frac{C_i^{f/m}}{T_i}$$

where H_1 consists of a set of higher priority tasks that preempt task τ_i only once before its deadline at D_i, and H_p consists of a set of higher priority tasks that may often preempt task τ_i before the deadline at D_i.

$$U(p, \Delta_i) = \begin{cases} p((2\Delta_i)^{1/n} - 1) + 1 - \Delta_i & 0.5 \leq \Delta_i \leq 1 \\ \Delta_i & 0 \leq \Delta_i < 0.5 \end{cases}$$

where Δ_i and p are:

$$\Delta_i = \frac{D_i}{T_i} \quad p = num(H_p) + 1$$

5.5. RTA approach

As is well known, the response time analysis (RTA) (Sha et al., 2004) recurrent approach is an exact test, which we can change to find an exact value for α that minimizes energy consumption. It is based on Saewong and Rajkumar (2003) and we can obtain a static and exact solution to compute the frequency scaling factor for task sets with $D = T$ and $D < T$.

5.6. Approach based on the schedulability region ($\mathcal{P}$ and $\mathcal{S}$ approaches)

Taking into account the above regarding the representation of the feasibility condition as a region in the C-space, it is easy to think that C_i can be tuned so that its values are in the feasibility region. In Bini et al. (2006) this topic is addressed from the perspective of sensitivity analysis.

For our purpose, in Eq. (11) and using Eq. (12), C_i becomes the design variable and a constant parameter. The scaling factor α is the interesting component that results in the following theorem.

Theorem 7. *The static scaling factor of the clock frequency that minimizes CPU energy consumption while preventing missed deadlines when given a task set $\mathcal{T}$ is defined by:*

$$\alpha^{\mathcal{P}}_{\min}(\mathcal{T}) = \max_{i=1,\dots,n} \min M^*_i(\mathcal{P}_{i-1}) \tag{18}$$

where P_{i-1} are the scheduling points defined by Eq. (9). This expression can also be used with the scheduling points S_i:

$$\alpha^{\mathcal{S}}_{\min}(\mathcal{T}) = \max_{i=1,\dots,n} \min M^*_i(\mathcal{S}_i) \tag{19}$$

where $M^*_i(\mathcal{S}_i)$ is given by:

$$M^*_i(\mathcal{S}_i) = \sum_{j=1}^{i} \frac{\lceil s_{kl}/T_j \rceil C^f_j}{t - \lceil s_{kl}/T_j \rceil C^m_j} \quad s_{kl} \in \mathcal{S}^*_i$$

s_{kl} is defined in Eq. (3). C^f and C^m are defined in Eq. (12).

Let the scaling factor be normalized between $0 < \alpha \le 1$, the computer system is feasibly schedulable if and only if α is less than or equal to 1, otherwise, the processor is unable to schedule the task set even though it is running at full speed.

Proof. Based on definitions in Eqs. (5) and (12), and following several changes to simplify the analysis, we have:

$$\max_{i=1,\dots,n} \min M_i(\mathcal{S}_i) \le \mathcal{S}_i$$

$$\max_{i=1,\dots,n} \min \sum_{j=1}^{i} \left\lceil \frac{s_{kl}}{T_j} \right\rceil C_j \le s_{kl} \quad s_{kl} \in \mathcal{S}^*_i$$

$$\max_{i=1,\dots,n} \min \sum_{j=1}^{i} \left\lceil \frac{s_{kl}}{T_j} \right\rceil \left(\frac{C^f_j}{\alpha} + C^m_j \right) \le s_{kl} \quad s_{kl} \in \mathcal{S}^*_i$$

$$\max_{i=1,\dots,n} \min \sum_{j=1}^{i} \left\lceil \frac{s_{kl}}{T_j} \right\rceil \frac{C^f_j}{\alpha} \le s_{kl} - \max_{i=1,\dots,n} \min \sum_{j=1}^{i} \left\lceil \frac{s_{kl}}{T_j} \right\rceil C^m_j$$

deducing α from the expression, we have:

$$\max_{i=1,\dots,n} \min M^*_i(\mathcal{S}_i) \le \alpha \triangleq \alpha_{\min}$$

where the modified matrix M (M^*_i) will be:

$$\begin{aligned} M^*_i(\mathcal{S}_i) &= \{m^*_{kl}\}_i \\ \{m^*_{kl}\}_i &= \sum_{j=1}^{i} \frac{\lceil s_{kl}/T_j \rceil C^f_j}{s_{kl} - \lceil s_{kl}/T_j \rceil C^m_j} \quad s_{kl} \in \mathcal{S}^*_i \end{aligned} \tag{20}$$

These expressions can be expanded to the points $\mathcal{P}_{i-1}$. □

6. Proposed approach $\mathcal{A}$

To bound the schedulability region for periodic fixed priority systems the analysis in the C-space is mainly based on the point set $\mathcal{S}_i$ or $\mathcal{P}_i$ (Theorems 1 and 2). The number of mathematical operations and the accuracy of this analysis are determined by the structure and the number of scheduling points.

In Section 4, two expressions for calculating the schedulability region were described (Eqs. 7 and 11). These regions are delimited by two different point sets ($\mathcal{S}_i$ and $\mathcal{P}_{i-1}$), which differ in size: $\text{Size}(\mathcal{S}_i) \gg \text{Size}(\mathcal{P}_{i-1})$ (Eqs. 4 and 10). The size of point set $\mathcal{S}$ can be much greater than the size of $\mathcal{P}$. However, both approaches are equally precise, and both are sufficient and necessary tests.

6.1. Reduced approach using the schedulability region

In this section we propose a reduction in the number of scheduling points required to compute the schedulability region. The objective of this simplification is to substantially reduce the computational cost of the DVS algorithm, but at the same time, preserve the accuracy of the schedulability analysis.

The reduction in these scheduling points will be based on points $\mathcal{P}_{i-1}$, which in turn are a reduction in the point set $\mathcal{S}_i$ ($\mathcal{P}_{i-1} \subseteq \mathcal{S}_i$). The new point set we call $\mathcal{A}_i$. This $\mathcal{A}_i$ set is a $\mathcal{A}_i \subseteq \mathcal{P}_{i-1}$ and a $\mathcal{A}_i \subseteq \mathcal{S}_i$.

The frequency scaling factor that uses the new set of scheduling points is defined by the following theorem:

Theorem 8. *The minimum scaling factor of the processor frequency that minimizes the energy consumption and guarantees no missed deadline given a task set $\mathcal{T}$, is defined as:*

$$\alpha^{\mathcal{A}}_{\min}(\mathcal{T}) = \max_{i=1,\dots,n} \min M^*_i(\mathcal{A}_i) \tag{21}$$

where M^*_i is defined in Eq. 20, but applied to the points set $\mathcal{A}$. This points set $\mathcal{A}_i$ is equal to:

$$\mathcal{A}_i(D_i) = \{D_i \ \cup \ sched\mathcal{A}_i(D_i)\} \tag{22}$$

where sched$\mathcal{A}(D_i)$ is the characteristic matrix in the computation of the points $\mathcal{A}_i$. sched$\mathcal{A}(D_i)$ is the following matrix expression:

$$\begin{bmatrix} \left\lfloor \frac{D_i}{T_1} \right\rfloor T_1 & \left\lfloor \frac{D_i}{T_2} \right\rfloor T_2 & \cdots & \left\lfloor \frac{D_i}{T_j} \right\rfloor T_j \\ & \left\lfloor \left\lfloor \frac{D_i}{T_2} \right\rfloor \frac{T_2}{T_1} \right\rfloor T_1 & \cdots & \left\lfloor \left\lfloor \frac{D_i}{T_j} \right\rfloor \frac{T_j}{T_{j-1}} \right\rfloor T_{j-1} \\ & & \cdots & \left\lfloor \left\lfloor \left\lfloor \frac{D_i}{T_j} \right\rfloor \frac{T_j}{T_{j-1}} \right\rfloor \frac{T_{j-1}}{T_{j-2}} \right\rfloor T_{j-2} \\ & & \cdots\cdots & \\ & & \left\lfloor \cdots \left\lfloor \left\lfloor \left\lfloor \frac{D_i}{T_j} \right\rfloor \frac{T_j}{T_{j-1}} \right\rfloor \frac{T_{j-1}}{T_{j-2}} \right\rfloor \cdots \frac{T_{j-k-1}}{T_{j-k}} \right\rfloor T_{j-k} & \end{bmatrix}$$

$$\forall j \in [1, \dots, i-1] \wedge \forall k \in [0, \dots, i-2] \tag{23}$$

where the dimension of this matrix is given by $(i-1)x(i-1)$.

Taking into account that the scaling factor has been normalized between $0 < \alpha \le 1$, the whole system will be schedulable if α is less than or equal to 1. If α is greater than 1, there is a low probability that the system can be really schedulable, on the contrary, if α is less than or equal to 1, the system is always schedulable.

Proof. Schedulability region analysis (Bini and Buttazzo, 2004; Lehoczky et al., 1989) is mainly based on a worst-case workload analysis of the task set. Using the concept of workload, the schedulability condition of a particular task τ_i can be expressed by:

$$C_i + W_{i-1}(D_i) \le D_i \tag{24}$$

The processor demand in the interval [0, t] is defined by $\sum_{j=1}^{i} \lceil t/T_j \rceil C_j$. To be a feasible schedule, the workload in $[t, D]$ should be smaller than the length of the interval, therefore:

$$\forall t \in [0, D_i] \quad W_i(D_i) - W_i(t) \le (D_i - t)$$

The latter equation can also be expressed as follows:

$$\forall t \in [0, D_i] \quad W_i(D_i) \le \sum_{j=1}^{i} \left\lceil \frac{t}{T_j} \right\rceil C_j + (D_i - t) \tag{25}$$

Moreover, we define a term $\varphi_i(D_i)$ as the last instant in [0, D_i] during which the processor is idle:

$$\varphi_i(D_i) = \max\{t \in [0, D_i] \wedge t \notin \text{activetasksininstants}\, t\}$$

From this expression we can induce that the processor is always busy in $[\varphi_i, D_i]$, and thus the workload of the i highest priority tasks during such interval is $(D_i - \varphi_i(D_i))$. The definition of φ is needed

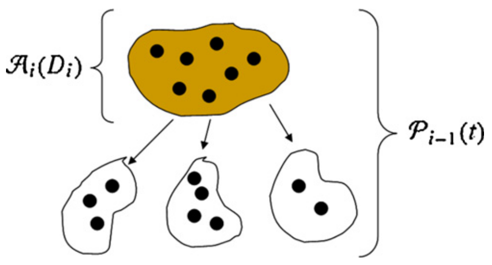

Fig. 3. The set $\mathcal{P}_{i-1}(D_i)$ can be obtained from the set $\mathcal{A}_i$.

because it can be useful for simplifying the computation of $W_i(D_i)$. Hence,

$$W_i(D_i) = \sum_{j=1}^{i} \left\lceil \frac{\varphi_i(D_i)}{T_j} \right\rceil C_j + (D_i - \varphi_i(D_i)) \tag{26}$$

However, using this expression we move from the complexity of the workload estimation itself (using continuous interval $[0, D_i]$) to the complexity of the $\varphi_i(D_i)$ search. Nevertheless, a set of possible values of $\varphi_i(D_i)$ can be restricted. With this aim, an initial set of possible values can be comprised for the deadline of task i and the arrival times of tasks with a priority higher than τ_i before D_i. This is because the last instant of time in which the processor can be idle (φ_i) matches exactly with the arrival times of the tasks. This set is defined by the point set $\mathcal{S}_i$ described in Section 4, such that $\varphi_i(t) \in \mathcal{S}_i(t)$. This set of possible values of S_i can be further reduced by the set of values proposed in Eq. 9, such that $\varphi_i(t) \in \mathcal{P}_i(t)$. From the evaluation of the schedulability points in Expression 25, the minimum value corresponds to the workload i and is the same when $t = \varphi_i(D)$ (Bini and Buttazzo, 2004), that is:

$$W_i(D_i) = \min_{t \in \mathcal{S}_i(D_i) \cup \mathcal{P}_{i-1}(D_i)} \sum_{j=1}^{i} \left\lceil \frac{t}{T_j} \right\rceil C_j + (D_i - t) \tag{27}$$

Using Eq. 24 for a whole task set, we obtain Eqs. 2 and 8.

Having established the principle of the schedulability region and feasibility conditions, we focus on proof of the reduced set $\mathcal{A}$. To obtain the reduced point set $\mathcal{A}$ based on the points $\mathcal{P}$, we propose to extract a characteristic function of the points $\mathcal{P}_{i-1}$, in order to obtain the most important points that enable narrowing and delineating the area where the points $\mathcal{P}_{i-1}$ are located at the time. This characteristic function is defined by means of a matrix expression, which is given in Eq. (23). The set $\mathcal{A}_i$ is given by:

$$A_i(D_i) = \{D_i \ \cup \ sched\mathcal{A}(D_i)\}$$

From the set $\mathcal{A}(D_i)$, we can obtain additional point subsets that in total constitute the set $\mathcal{P}_{i-1}$ (see Fig. 3). The number of elements of $\mathcal{P}_{i-1}(D_i)$ and $\mathcal{A}(D_i)$ are determinated, respectively, by the Eqs. 10 and 29. In Fig. 4, the points of the sets $\mathcal{P}_{i-1}$ and $\mathcal{A}_i$ are compared for a specific set of 15 periodic tasks. Notice that the regions of point concentration $\mathcal{P}_{i-1}$ are delimited by the points obtained from the characteristic matrix $\mathcal{A}_i$.

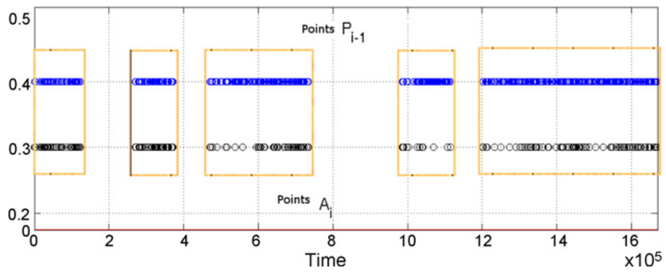

Fig. 4. Comparison of the points $\mathcal{P}_{i-1}$ with the points A_i.

By definition the set $\mathcal{A}$ uses fewer points than $\mathcal{P}$ for the estimation of the values of $\varphi_i(D_i)$, therefore two things can happen:

1. The exact value of $\varphi_i(D_i)$ is within the values obtained in the calculation of $\mathcal{A}_i(D_i)$.
2. There is a difference between the exact value $\varphi_i(D_i)$ and the values obtained by means of $\mathcal{A}_i(D_i)$.

In the first case, we get an exact result for $W_i(D_i)$ and so the schedulability conditions exposed will be accurately tested. Moreover, we could also compute the frequency scaling factor α within a margin of error 0.

In the second case, let the smallest workload W_i correspond with the evaluation at $t = \varphi_i(D_i)$, and assuming that $\varphi_i(D_i)$ is not included in the points $\mathcal{A}_i(D_i)$, then W_i when $t \in \{\mathcal{A}_i(D_i)\}$ is always fulfilled that:

$$\varphi_i(D_i) \notin \mathcal{A}_i(D_i) \Rightarrow W_i(D_i)^{t=\mathcal{A}_i(D_i)} \geq W_i(D_i)^{t=\varphi_i(D_i)} \tag{28}$$

This proves that the result of calculating the frequency scaling factor using set $\mathcal{A}$ is always valid. Using the set $\mathcal{A}$ we will obtain an $\alpha^{\mathcal{A}}$ equal or greater than the exact value. However, we will never obtain a value below the exact scaling factor. This guarantees a result that is *sufficient*, although in some cases not *necessary*.

Based on the analysis of the schedulability region, we have:

$$\max_{i=1,\ldots,n} \min M_i(\mathcal{A}_i) \leq \mathcal{A}_i$$

$$\max_{i=1,\ldots,n} \min \sum_{j=1}^{i} \left\lceil \frac{a}{T_j} \right\rceil C_j \leq a \quad a \in \mathcal{A}_i^*$$

$$\max_{i=1,\ldots,n} \min \sum_{j=1}^{i} \left\lceil \frac{a}{T_j} \right\rceil \left(\frac{C_j^f}{\alpha} + C_j^m \right) \leq a \quad a \in \mathcal{A}_i^*$$

$$\max_{i=1,\ldots,n} \min \sum_{j=1}^{i} \left\lceil \frac{a}{T_j} \right\rceil \frac{C_j^f}{\alpha} \leq a - \max_{i=1,\ldots,n} \min \sum_{j=1}^{i} \left\lceil \frac{a}{T_j} \right\rceil C_j^m$$

where a is the element set that forms the set $\mathcal{A}_i^*$ (see Eq. (22)) and $\mathcal{A}_i^*$:

$$\mathcal{A}_i^* = \frac{\{a\}}{a} \in \mathcal{A}_i \wedge \exists! a \in \mathcal{A}_i$$

The total number of elements of $\mathcal{A}_i$ is determined by:

$$\mathrm{Size}(\mathcal{A}_i) = i + \sum_{n=1}^{i} \frac{n(n-1)}{2} \tag{29}$$

Deducing the factor α from $\{m\}_i$, we have:

$$\max_{i=1,\ldots,n} \min M_i^*(\mathcal{A}_i) \leq \alpha \triangleq \alpha_{min}$$

where

$$M_i^*(\mathcal{A}_i) = \{m^*\}_i$$

$$\{m^*\}_i = \sum_{j=1}^{i} \frac{\lceil a/T_j \rceil C_j^f}{a - \lceil a/T_j \rceil C_j^m} \quad a \in \mathcal{A}_i^*$$

□

We can obtain a maximum error in the computation of α using the set $\mathcal{A}_i$ when $\varphi_i(D_i) \notin \mathcal{A}_i(D_i)$, i.e. when the $\mathcal{A}_i(D_i)$ approach finds an approximate value (φ_i^*) equal to:

$$\mathrm{error}_{\alpha^{\mathcal{A}}} = \sum_{j=1}^{i} \left(\left\lceil \frac{\varphi_i^*(D_i)}{T_j} \right\rceil \frac{1}{\varphi_i^*(D_i)} - \left\lceil \frac{\varphi_i(D_i)}{T_j} \right\rceil \frac{1}{\varphi_i(D_i)} \right) C_j \tag{30}$$

Table 1
Range of periods for task groups A, B and C.

Group	Lower bound (μs)	Upper bound (μs)
A	2000	40,000
B	40,001	600,000
C	600,001	4,000,000

7. Experimental evaluation when $D = T$ and $D \leq T$

We have performed extensive simulations of the algorithm proposed ($\mathcal{A}$ approach) and the other algorithms described ($\mathcal{P}$, RTA, LLM, HB, LL and EDF-U approaches) to experimentally evaluate the performance of the dynamic code delegation for task sets with $D = T$ and $D \leq T$.

We propose three group of random tasks: A, B and C. Group A consists of tasks with short periods, group B consists of tasks with middle periods, and group C consists of tasks with large periods. Each task group consists of 20 tasks. The range of periods for the task groups are shown in Table 1.

Three types of arrival of tasks at the processor (Ll1, Ll2 and Ll3) have also been simulated. For Ll1 the highest priority tasks arrive first, for Ll2 the medium priority tasks arrive first, and for Ll3 the least priority tasks arrive first. The simulation consists of sets of 20 tasks.

These task groups try to represent the tasks set used in several practical applications, such as mobile or humanoid robotics. In these applications usually converge various hierarchical levels of control. These hierarchical levels consist of tasks set with different temporal characteristics, which will depend on the criticality of the activities to be performed. For instance, in a hierarchical robotic system with reactive, tactical and mission levels the distribution of tasks could be like the proposed task groups. The first level (reactive level) consists mainly by high-priority tasks, i.e. tasks with short periods, such as the proposed group A. The second level can be compared with group B, which is formed by tasks with more relaxed periods. It consists of medium priority tasks. In the mission level the temporal constraints are decreased. In this level the tasks used are the least priority, i.e. with large periods such as group C.

Task groups A, B, and C are combined with arrival types Ll1, Ll2 and Ll3, and this results in nine different sets of tasks. These, in turn, will be evaluated on five discrete utilizations U (0.3–0.5–0.7–0.8–0.95).

Each test is characterized by an acceptance ratio AR, computational cost CR, and energy consumption ER (see Fig. 5). The results of these simulations will be presented on three components of reference:

- The *cost* of each algorithm will be counted in accordance with the number and type of operations. Each operation is characterized with a predetermined number of machine cycles based on the ARM architecture (Sloss et al., 2004). Therefore, the units in which the computational cost will be measured are machine cycles.
- The component *energy over-consumption* is referred to the energy consumption exceeded by an algorithm when it is compared to energy consumption achieved using an exact algorithm of fixed priority. Using an exact algorithm we get the minimum processor frequency with which it is possible to scheduling a given task set. An example of exact algorithm for fixed priorities is the RTA approach.
- The *rejection ratio* refers to one less than the number of accepted task sets $\mathcal{T}_{pt}$ with respect to those accepted by a necessary and sufficient algorithm $\mathcal{T}_p$: $1 - (\mathcal{T}_{pt}/\mathcal{T}_p)$.

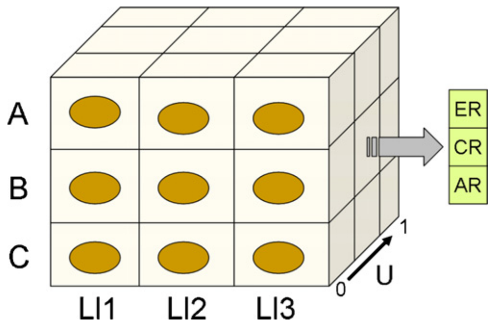

Fig. 5. Test sets for analyzing approaches for computing the scaling factor.

These components were chosen so that a greater magnitude on the axes can be understood as a worse behaviour for an algorithm. In the tests, the worst result will be plotted for each task group and type of arrival. The calculation of α using the $\mathcal{S}$ approach was not used due to the high computational cost involved.

In the energy performance only the consumption caused by execution of whole task set is reflected in the simulations, this consumption depends on the processor frequency obtained by each algorithm. The energy consumption derived from the execution of the algorithm itself was not included in the energy consumption presented in the simulations. It is because this consumption depends on the frequency at which the processor is running when the algorithm requires to be performed. Therefore we consider it more appropriate to compare the cost of the algorithms in independent units to the processor frequency. This considering that the number of machine cycles is proportional to the energy consumption. Mode change protocols should set the processor frequency at which the algorithms must be performed when a new frequency calculation is required. However, these protocols are not within the scope of this paper.

7.1. Rejection ratio versus computational cost

Figs. 6 and 7 show the simulation results of the percentage of rejected tasks in comparison with the computational cost of calculating algorithms.

In the figures, we can see that all approaches for the calculation of α, except for LLM, LL, and HB, have a rejection ratio equal to 0%. The LL approach has a dispersed rejection ratio that increases with utilization and which stretches from 0% to 45%. The HB approach

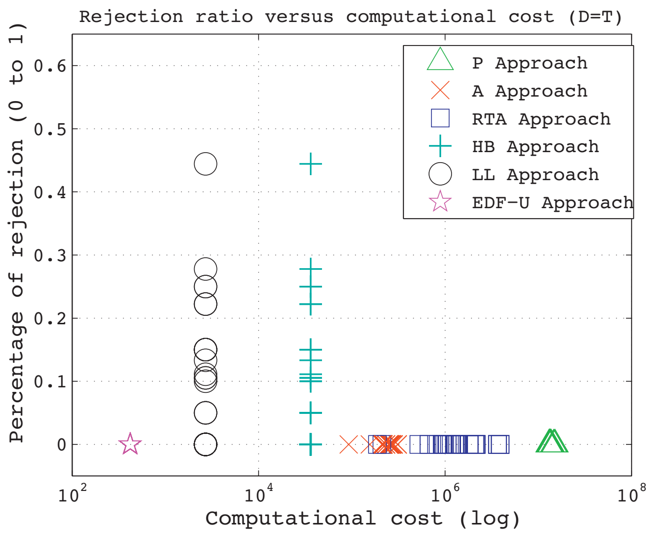

Fig. 6. Rejection ratio versus computational cost when $D = T$. Note that the percentage of over-consumption has been plotted between 0 and 0.6.

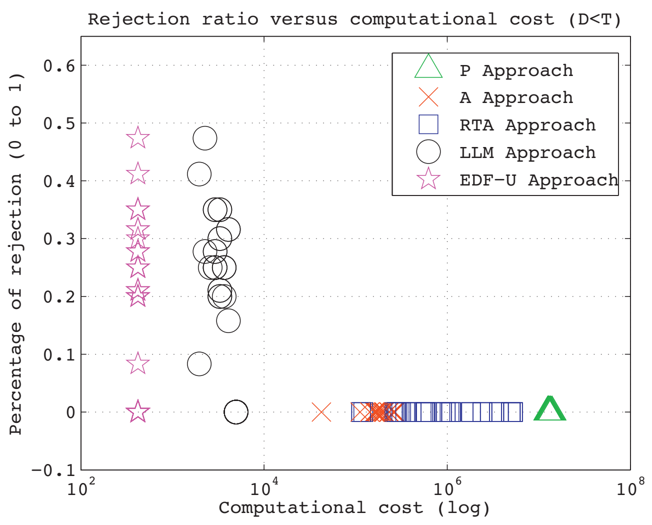

Fig. 7. Rejection ratio versus computational cost when $D < T$.

has a slightly lower rejection percentage than LL. When $D \leq T$, the maximum rejection ratio for the LLM approach is increased to about 50%. The EDF-U based scheduler will has a behaviour similar to LLM approach.

From the perspective of cost, although the methods of EDF-U (when $D \leq T$), LLM, LL, and HB show the largest rejection percentages, they also have lower costs. In both cases, $D = T$ and $D \leq T$, the computational cost of using the RTA approach is dispersed and unpredictable, and their costs are sometimes among the highest of all the methods, exceeded only by the $\mathcal{P}$ approach. The cost is sometimes less than the proposed $\mathcal{A}$ approach.

The $\mathcal{A}$ approach has a middle cost and a rejection ratio of zero for $D = T$ and for $D \leq T$. Furthermore, the proposed method has predictable costs, and their computational cost is almost 20 times less than the cost of the $\mathcal{P}$ approach. Notice that the gap in the magnitude of the cost between the proposed approach and the $\mathcal{P}$ approach increases with the arrival of tasks.

7.2. Over-consumption percentage versus computational cost

Figs. 8 and 9 show the energy over-consumption in function of the computational cost for the worst case. The over-consumptions are due to deviations in the calculation of the scaled factor α compared with the calculation using an exact method. Notice that deviations in alpha lead to quadratic over-consumption – depending

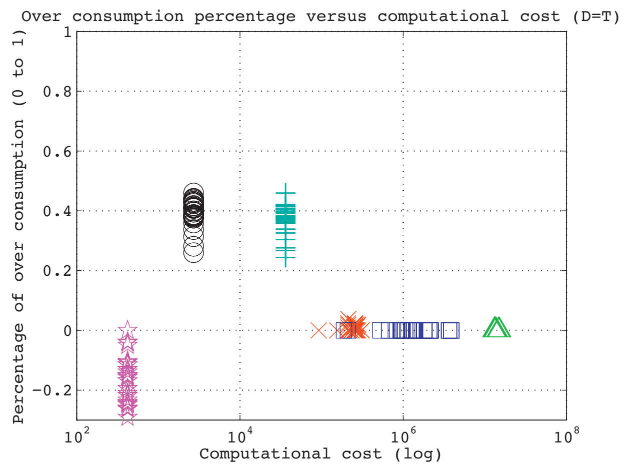

Fig. 8. Over-consumption percentage versus computational cost when $D = T$.

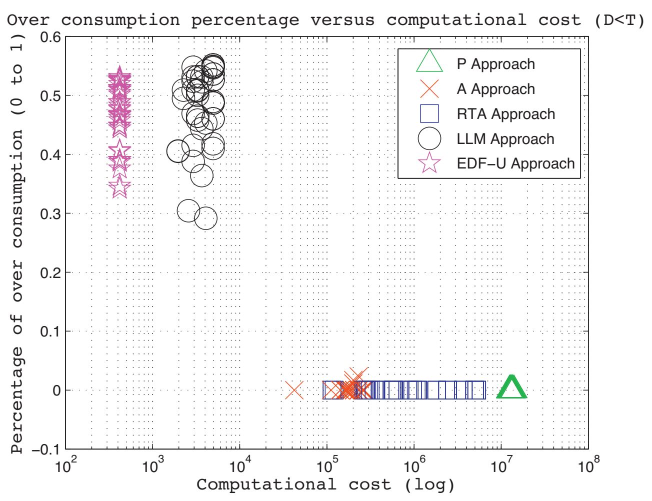

Fig. 9. Over-consumption percentage versus computational cost when $D < T$.

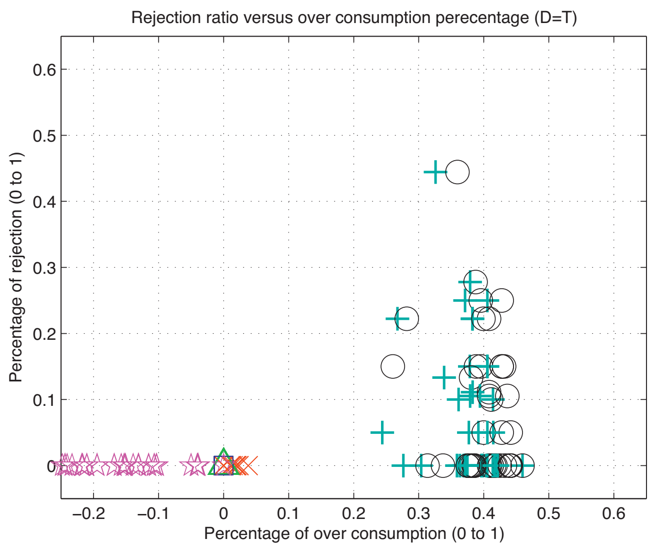

Fig. 10. Rejection ratio versus energy over-consumption percentage when $D = T$.

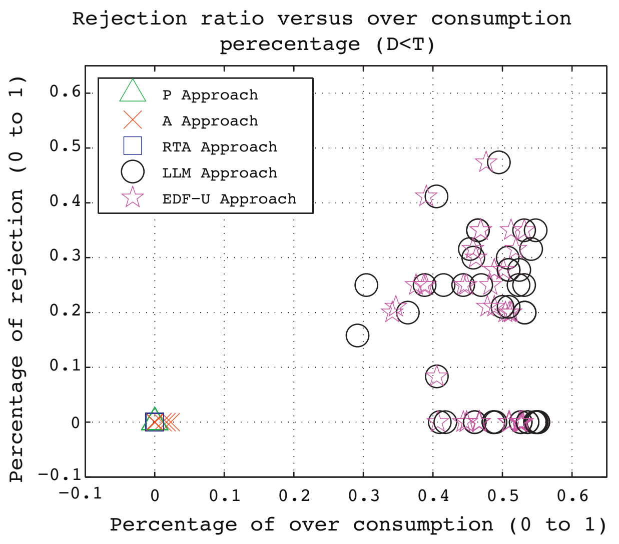

Fig. 11. Rejection ratio versus energy over-consumption percentage when $D < T$.

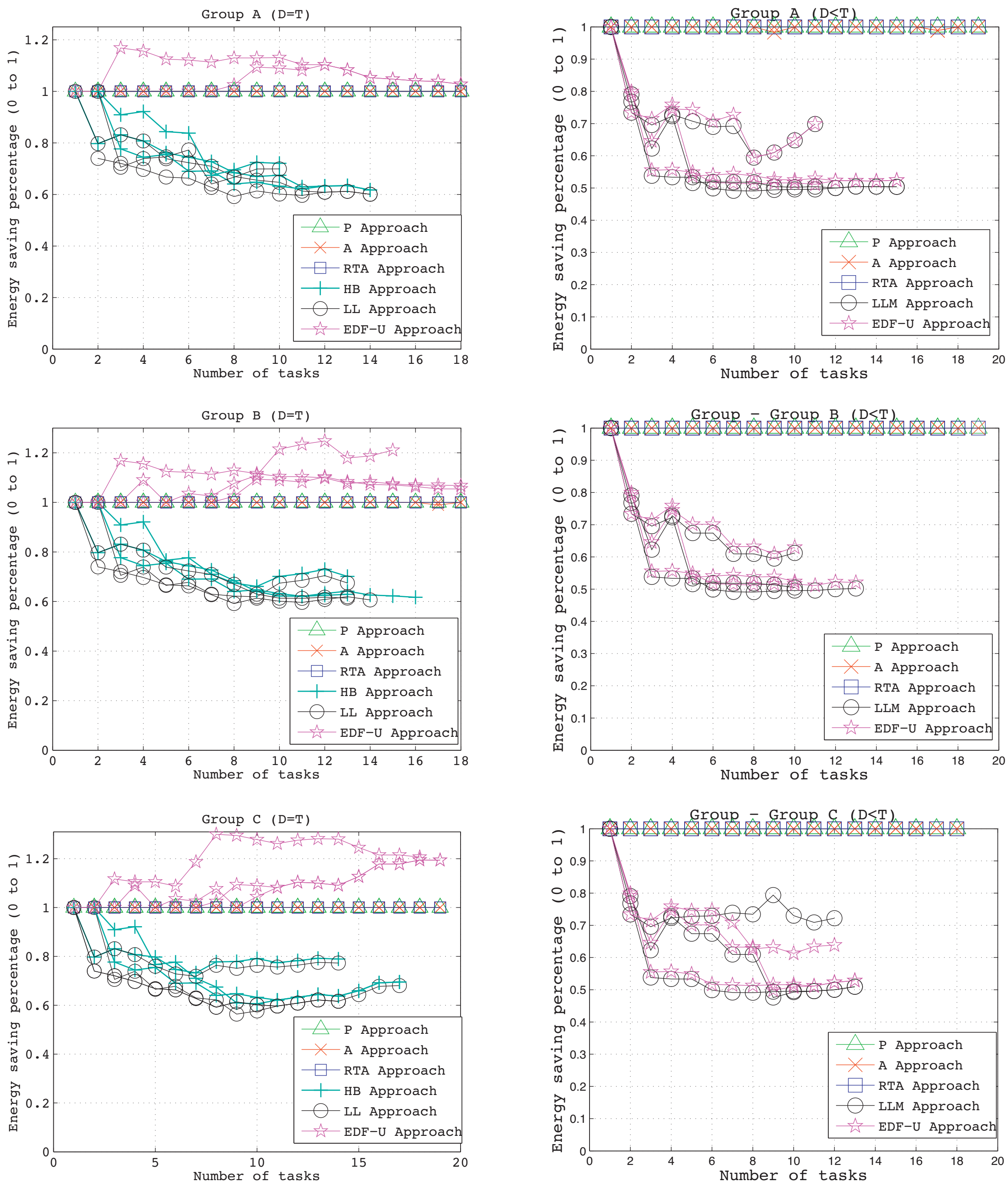

Fig. 12. Evolution of the energy saved with respect to the saving with a exact algorithm in function to the arrival of tasks. Utilization equal to 95%.

on the polynomial expression that describes CPU power consumption.

In Fig. 8 we can observe that LL has a small computational cost, but an energy over-consumption close to 45%. When $D = T$, EDF-U save more energy than the others approaches. For this reason the points of EDF-U are located in negatives values. HB has an over-consumption slightly lower than LL. When $D \leq T$, the LLM approach shows an over-consumption for the worst case of around 30–60%. When $D \leq T$, the EDF-U approach has a similar behaviour than LLM, but it has lower computational cost.

Moreover, we obtain an intermediate computational cost when using the reduced approach $\mathcal{A}$. When $D = T$, the proposed method has a higher concentration of points around the 0% for the worst case, and with some values of over-consumption above 0, and which do not exceed 2.5%. When $D \leq T$, we obtain an over-consumption mostly equal to 0 using the $\mathcal{A}$ approach. A 2% of the plotted points for $\mathcal{A}$ are above 0. For both cases, $D = T$ and $D \leq T$, the approaches $\mathcal{P}$ and RTA show an over-consumption equal to 0, but with higher costs and dispersion, respectively.

7.3. Rejection ratio versus over-consumption percentage

Figs. 10 and 11 show the percentage of rejected tasks in comparison with energy over-consumption percentages.

When $D = T$, the LL and HB approaches are dispersed at the intersection of the areas of between 20% and 48% of over-consumption and the area of between 0% and 45% of rejection ratio. The increase in the percentage in these approaches is determined by the

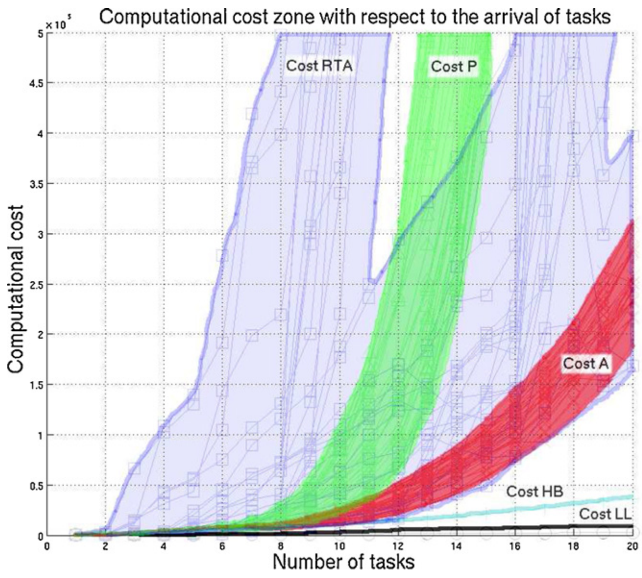

Fig. 13. Evolution of the computational cost of the algorithms with respect to the arrival of new tasks.

utilization and the number of tasks. The $\mathcal{A}$ approach has no points on the axis of the rejection ratio. On the axis of over-consumption, 5% of the plotted points for $\mathcal{A}$ are between 1.2% and 2.5%. The other points are at 0%. EDF-U approach has a rejection ratio equal to 0% and it saves more energy than the others approaches. For this reason the points of EDF-U are located on the axis negative. The other approaches have an over-consumption and a rejection ratio equal to 0.

When $D \leq T$, the LLM approach is dispersed at the intersection of the areas between 30% and 55% of over-consumption and the area of between 0% and 50% of the rejection ratio. EDF-U approach has a behaviour similar to LLM. The RTA, $\mathcal{P}$ and $\mathcal{A}$ approaches have an over-consumption equal to 0 and close to 0, respectively, and a rejection ratio equal to 0.

7.4. Other simulations

Fig. 12 shows separately the evolution of the energy saved with respect to the saving achieved with an exact algorithm in function of the arrival of tasks for each task group (A, B, and C) and their respective arrivals (LL1, LL2 and LL3). It shows a utilization of 95% for the complete tasks set with $D = T$ and $D \leq T$.

Fig. 13 shows the evolution of the computational cost of the proposed algorithm when compared with the other static algorithms and EDF-U algorithm with respect to the arrival of new tasks for the whole test set when $D = T$. The figure shows the variability of the cost for the RTA approach, which varies depending on the task set and increases with the number tasks arriving.

Fig. 13 also shows the evolution of the $\mathcal{P}$ approach, whose cost increases depending on the number of tasks in the order 2^n, where n is the number of tasks. The cost of our static algorithm is smaller when compared to $\mathcal{P}$ and RTA algorithms, and their evolution with respect to the number of arrivals of tasks is far more muffled than $\mathcal{P}$ and RTA.

8. Conclusions

In this paper a new algorithm ($\mathcal{A}$ approach) for computing the processor frequency scaling factor under fixed priority assignment with $D = T$ and $D \leq T$ is proposed. This algorithm is performed with a reduced computational cost and minimizes CPU energy consumption while guaranteeing no missed deadlines. Therefore, it can be used as on-line acceptance test in systems with dynamic workload. However, as is well known, if it is compared with dynamic priority approaches, such as EDF, they have a higher performance than fixed priority based scheduling, but we have design constraints and we have to use fixed priority algorithms in order to complaint the standard ARINC-653 (AASS Interface, 2003).

Table 2
Comparison of approaches to compute a constant frequency scaling factor α.

Alg.	Cost	Rejection ratio	Over-cons. ratio	D/T
$\mathcal{S}$	Very high	None	None	$D \leq T$
$\mathcal{P}$	High	None	None	$D \leq T$
RTA	Unpredictable middle-high	None	None	$D \leq T$
$\mathcal{A}$[a]	Low-middle	None	Very low	$D \leq T$
LLM	Low	Very high	Very high	$D \leq T$
HB	Low	High	High	$D = T$
LL	Very low	High	High	$D = T$
EDF-U aprox.	Very low	Very high	Very high	$D \leq T$
EDF-U	Very low	None	None	$D = T$

[a] Proposed approach $\mathcal{A}$.

Using extensive simulations, we have evaluated the behaviour of several existing feasibility tests that have been adapted to compute a frequency scaling factor α for the proposed approach $\mathcal{A}$. The analysis has been presented from the point of views of energy consumption, task rejection ratio, and real computing costs.

Our simulation results show that the proposed algorithm outperforms other constant voltage scaling algorithms from the cost, predictability, and task acceptance points of view. However, some small deviations were observed in the energy saved. The proposed approach is compared with other approaches. The proposed algorithm enables not only the high precision verification of system feasibility and the computation of α in an environment with a dynamic processor load. The algorithm can also be used in real-time operating systems because the computational cost represents a small system overload.

The main features of the described approach are presented in Table 2.

As a future work, we plan to investigate the case where memory access and the system bus perform at different clock speeds – together with processor frequency scaling (Marvell, 2008). In this case, the combination of these parameters will enable an even greater reduction in energy consumption for the whole computing system. Note that typical values of frequency scaling in XScale processors are 208, 156 and 104 MHz for system bus and 312, 208, 156 and 104 MHz for SRAM memory access. Such as was mentioned in the Section 5, not all execution cycles are scaled for the processor speed because some operations deal with memory or other I/O devices. Therefore, they can be handled according to the activities of the tasks, for instance, while a task does not interact extensively with I/O devices, the access frequency to these can be reduced.

On the other hand, top-level algorithms to reclaim additional idle time resulting from the early completions of tasks will be investigated, such as approaches based on feedback control scheduling methodology. It to be used together with the proposed approach $\mathcal{A}$. Additionally, testing of the proposed method on real hardware will be addressed.

Acknowledgements

This work has been supported by the Spanish Government as part of the SIDIRELI project (DPI2008-06737-C02-02), COBAMI project (DPI2011-28507-C02-02) and by the Generalitat Valenciana (Project ACOMP-2010-038).

References

AASS Interface, 2003. ARINC653, Arinc Specification 653-1. RTCA.

Albertos, P., Crespo, A., Simó, J., 2006. Control kernel: a key concept in embedded control systems. In: 4th IFAC Symposium on Mechatronic Systems.

AlEnawy, T., Aydin, H., 2005. Energy-aware task allocation for rate monotonic scheduling. In: Proceedings of the 11th IEEE Real Time and Embedded Technology and Applications Symposium (RTAS), pp. 213–223.

Anderson, J.H., Bud, V., Devi, U.C.,2005. An EDF-based scheduling algorithm for multiprocessor soft real-time systems. In: ECRTS '05: Proceedings of the 17th Euromicro Conference on Real-Time Systems. IEEE Computer Society, Washington, DC, USA, pp. 199–208.

Aydin, H., Devadas, V., Zhu, D.,2006. System-level energy management for periodic real-time tasks. In: RTSS '06: Proceedings of the 27th IEEE International Real-Time Systems Symposium. IEEE Computer Society, Washington, DC, USA, pp. 313–322.

Aydin, H., Melhem, R., Mossé, D., Mejía-Alvarez, P., 2004. Power-aware scheduling for periodic real-time tasks. IEEE Transactions on Computers 53, 584–600.

Bini, E., Buttazzo, G., 2004. Schedulability analysis of periodic fixed priority systems. IEEE Transactions on Computers 53, 1462–1473.

Bini, E., Buttazzo, G., Lipari, G.,2005. Speed modulation in energy-aware real-time systems. In: ECRTS '05: Proceedings of the 17th Euromicro Conference on Real-Time Systems. IEEE Computer Society, Washington, DC, USA, pp. 3–10.

Bini, E., Buttazzo, G.C., Buttazzo, G.M., 2003. Rate monotonic analysis: the hyperbolic bound. IEEE Transactions on Computers 52, 933–942.

Bini, E., Natale, M.D., Buttazzo, G.C., 2006. Sensitivity analysis for fixed-priority real-time systems. In: Proceedings of the 18th Euromicro Conference on Real-Time Systems, Dresden, Germany.

Briao, E.W., Barcelos, D., Wronski, F., Wagner, F.R., 2007. Impact of task migration in NOC-based MPSOCS for soft real-time applications. In: IFIP International Conference on Very Large Scale Integration (VLSI), pp. 296–299.

Cervin, A., 2003. Integrated control and real-time scheduling. Ph.D. Thesis. Department of Automatic Control, Lund University ISRN LUTFD2/TFRT-1065-SE.

Chen, 2007. Energy-efficient real-time Task Scheduling with Task Rejection.

Chen, J.-J.,2005. Multiprocessor energy-efficient scheduling for real-time tasks with different power characteristics. In: ICPP '05: Proceedings of the 2005 International Conference on Parallel Processing. IEEE Computer Society, Washington, DC, USA, pp. 13–20.

Cho, Y., Chang, N., 2006. Energy-aware clock-frequency assignment in microprocessors and memory devices for dynamic voltage scaling. IEEE Transactions on Computer-aided Design of Integrated Circuits and Systems 26, 1030–1040.

Crespo, A., Albertos, P., Balbestre, P., Vall'es, M., Lluesma, M., Sim'o, J., 2006. Schedulability issues in complex embedded control systems. IEEE International Conference on Control Applications.

Duan, C., Khatri, S.P., 2006. Computing during supply voltage switching in DVS enabled real-time processors. In: IEEE ISCAS Conference, p. 2254.

Emberson, P., Bate, I.,2007. Minimising task migration and priority changes in mode transitions. In: RTAS '07: Proceedings of the 13th IEEE Real Time and Embedded Technology and Applications Symposium. IEEE Computer Society, Washington, DC, USA, pp. 158–167.

Galizzi, J., Metge, J.-J., Arberet, P., Morand, E., F.V., Crespo, A., Masmano, M., Coronel, J., Ripoll, I, V.B.U.P., Roubert, F., Scuri, C., V.T., T.N., 2011. Porting of LVCUGEN (TSP-based solution) on CPU-ITAR-FREE board. In: DASIA.

Galizzi, J., Metge, J.-J., Arberet, P., Morand, E., F.V., Crespo, A., Masmano, M., Coronel, J., Ripoll, I., V.B.U.P., Roubert, F., Scuri, C., V.T., T.N., 2012. LVCUGEN (TSP-based solution) and first porting feedback. In: Embedded Real Time Software and Systems.

Gaujal, B., Navet, N., 2007. Dynamic Voltage Scaling Under EDF Revisited.

Intel, 2004. Enhanced Intel speedstep technology for the Intel Pentium M processor. In: White Paper.

Jejurikar, R., Gupta, R.,2004. Dynamic voltage scaling for systemwide energy minimization in real-time embedded systems. In: ISLPED '04: Proceedings of the 2004 international Symposium on Low Power Electronics and Design. ACM, New York, NY, USA, pp. 78–81.

Kim, W., Shin, D., Yun, H.-S., Kim, J., Min, S.L.,2002. Performance comparison of dynamic voltage scaling algorithms for hard real-time systems. In: RTAS '02: Proceedings of the Eighth IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'02). IEEE Computer Society, Washington, DC, USA, p. 219.

Kumar, G.S.A., Manimaran, G.,2007. Energy-aware scheduling of real-time tasks in wireless networked embedded systems. In: RTSS '07: Proceedings of the 28th IEEE International Real-Time Systems Symposium. IEEE Computer Society, Washington, DC, USA, pp. 15–24.

Kumar, G.S.A., Manimaran, G., Wang, Z., 2008. End-to-end energy management in networked real-time embedded systems. IEEE Transactions on Parallel and Distributed Systems 19, 1498–1510.

Lee, E.A., 2006. Cyber-physical systems – are computing foundations adequate? In: NFS Workshop on Cyber-Physical Systems.

Lehoczky, J., Sha, L., Ding, Y., 1989. The rate-monotonic scheduling algorithm: exact characterization and average case behavior. In: 10th IEEE Real-Time Systems Symposium, pp. 166–172.

Leung, L.-F., 2005. Exploiting Dynamic Workload Variation in Low Energy.

Li, T., Ding, C., 2001. Instruction balance and its relation to program energy consumption. In: International Workshop Languages and Compilers for Parallel Computing.

Liang, W.-Y., Chen, S.-C., Chang, Y.-L., Fang, J.-P., 2008. Memory-aware dynamic voltage and frequency prediction for portable devices. In: 14th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA), pp. 229–236.

Liu, Y., Mok, A., 2003. An integrated approach for applying dynamic voltage scaling to hard real-time systems. In: 9th IEEE Real-Time and Embedded Technology and Applications Symposium, Toronto, Canada.

Marinoni, M., Buttazzo, G., 2007. Elastic DVS management in processors with discrete voltage/frequency modes. IEEE Transactions on Industrial Informatics, 3.

Marvell, 2008. System and timer configuration developers manual. In: Marvell PXA3xx Processor Family, vol. I.

Mejia-Alvarez, P., Levner, E., Mosse, D., 2002. Power-optimized scheduling server for real-time tasks. In: Proceedings of the Eighth IEEE Real-Time and Embedded Technology and Applications Symposium, pp. 239–250.

Mejia-Alvarez, P., Levner, E., Mossé, D., 2004. Adaptive scheduling server for power-aware real-time tasks. ACM Transactions on Embedded Computing Systems 3, 284–306.

Piao, X., Kim, H., Cho, Y., Park, M., Han, S., Park, M., Cho, S.,2009. Energy consumption optimization of real-time embedded systems. In: ICESS '09: Proceedings of the 2009 International Conference on Embedded Software and Systems. IEEE Computer Society, Washington, DC, USA, pp. 281–287.

Pillai, P., Shin, K., 2001. Real-time dynamic voltage scaling for low-power embedded operating systems. In: ACM symposium on operating systems principles, Banff, Canada.

Posadas, J.L., Poza, J.L., Sim, J.E., Benet, G., Blanes, F., 2008. Agent based distributed architecture for mobile robot control. Engineering Applications of Artificial Intelligence 21, 805–823.

Pouwelse, J., Langendoen, K., Sips, H., 2001. Dynamic voltage scaling on a low-power microprocessor. In: Seventh Internacional Conference Mobile Computing and Networking (MOBICOM).

Quan, 2004. Fixed Priority Scheduling for Reducing Overall Energy on Variable Voltage Processors.

Racu, R., Jersak, M., Ernst, R., 2005. Applying sensitivity analysis in real-time distributed systems. In: 11th IEEE Real-Time Technology and Applications Symposium (RTAS), IEEE Computer Society, pp. 160–169.

Real, J., Crespo, A., 2004. Mode change protocols for real-time systems: a survey and a new proposal. Real-time Systems 26, 161–197.

Saewong, S., Rajkumar, R.R.,2003. Practical voltage-scaling for fixed-priority rt-systems. In: RTAS '03: Proceedings of the 9th IEEE Real-Time and Embedded Technology and Applications Symposium. IEEE Computer Society, Washington, DC, USA, p. 106.

Scordino, C., Lipari, G., 2006. A resource reservation algorithm for power-aware scheduling of periodic and aperiodic real-time tasks. IEEE Transactions on Computers 55, 1509–1522.

Sha, L., Abdelzaher, T., Arzn, K.-E., Cervin, A., Baker, T., Burns, A., Buttazzo, G., Caccamo, M., Lehoczky, J., Mok, A.K., 2004. Real-time scheduling theory: a historical perspective. Real-time Systems 28:23, 101155.

Simarro, R., Coronel, J., Simó, J., Blanes, J., 2008. Hierarchical and distributed embedded control kernel. In: 17th IFAC World Congress, Seoul, Korea.

Sloss, A.N., Symes, D., Wright, C., 2004. Arm System Developer's Guide (Designing and Optimizing System Software). Primera ed. Elsevier.

Tavares, E., Maciel, P., Silva, B., Oliveira Jr., M.N., 2008. Hard real-time tasks' scheduling considering voltage scaling, precedence and exclusion relations. Information Processing Letters 108, 50–59.

Windsor, J., Hjortnaes, K., 2009. Time and space partitioning in spacecraft avionics. In: IEEE International Conference on Space Mission Challenges for Information Technology, pp. 13–20.

Wolf, W., 2009. Cyber-physical system. IEEE Computer: Innovative Technology for Computer Professionals 42, 88–89.

Xia, F., Tian, Y.-C., Sun, Y., Dong, J., 2008. Control-theoretic dynamic voltage scaling for embedded controllers. IET Computers & Digital Techniques 2, 377–385.

Yazdi, H., Salmani, H., Khatib-Astaneh, N., Salmani, M., Fard, A., 2008. Exploiting laxity for heterogeneous multiprocessor real-time scheduling. In: 3rd International Conference on Information and Communication Technologies: From Theory to Applications (ICTTA), pp. 1–6.

Yi, J., Poellabauer, C., Hu, X.S., Simmer, J., Zhang, L.,2009. Energy-conscious co-scheduling of tasks and packets in wireless real-time environments. In: RTAS '09: Proceedings of the 2009 15th IEEE Real-Time and Embedded Technology and Applications Symposium. IEEE Computer Society, Washington, DC, USA, pp. 265–274.

Yuan, Z., Wang, G., 2008. Power management for real-time tasks in wireless networked embedded systems. In: International Conference on Computer Science and Software Engineering, vol. 4, pp. 118–121.

Yun, H.-S., Kim, J., 2003. On energy-optimal voltage scheduling for fixed-priority hard real-time systems. ACM Transactions on Embedded Computing Systems 2, 393–430.

Zhong, Xiliang, Xu, Cheng-Zhong, 2007. Frequency-aware energy optimization for real-time periodic and aperiodic tasks. SIGPLAN Notices 42, 21–30.

Zhu, D., Melhem, R., Mossé, D., 2004. The effects of energy management on reliability in real-time embedded systems. In: International Conference on Computer-aided Design.