A Model Considering QoS for Real-Time Systems with Energy and Temperature Constraints

Ríad Nassiffe, Eduardo Camponogara
Dept. of Automation and Syst. Eng.
Federal University of Santa Catarina
Florianópolis, SC, Brazil
Email: riad, camponog@das.ufsc.br

Daniel Mossé
Dept. of Computer Science
University of Pittsburgh
Pittsburgh, PA, USA
Email: mosse@cs.pitt.edu

George Lima
Dept. of Computer Science
Federal University of Bahia
Salvador, BA, Brazil
Email: gmlima@ufba.br

***Abstract*—Saving energy is widespread goal, particularly in battery-supplied real-time embedded systems. Moreover, application lifetime and delivered quality of service are associated with the amount of energy that can be saved. In this context, mechanisms for providing energy savings must be in place and should take into consideration system-wide aspects such as CPU usage and temperature, which influence in the energy consumed by the system. In this paper we describe a model to capture such needs. The model is formalized as a convex optimization problem, with the goal of maximizing system quality of service while being subject to schedulability, energy, and temperature constraints. The CPU frequency and task execution time are decision variables. The effectiveness and behavior of the proposed solution is shown through numerical analysis. We conclude that a good approximate solution is enough for guarantee the constraints.**

I. Introduction

Due to the steady development of embedded systems, the usage of such systems has grown in recent years in host of applications: health monitoring, security systems, *etc*. These applications are often subject to constraints as energy, schedulability, temperature, and system QoS requirements. Since such devices can run different sets of applications with specific goals, several mechanisms have been implemented at the operating system level to keep the systems working properly.

Ensuring energy savings and temperature bound guarantees in a real-time embedded system requires sophisticated solutions, when compared to general purpose operating systems for which it may be enough to manage the underlying hardware power-saving and CPU throttling functionalities.

The constraint of energy comes from the fact that some devices may not be connected to a continuous power source, but rather depend on a battery which is recharged by solar energy, as in [1]. As a result, it is important to know for how long the system can be powered up by the available energy. The lack of energy management may result in failure of the system execution.

The temperature has impact on system power consumption and performance. For example, high temperatures can increase the leakage and cooling power, and even reduce the device lifespan.

In real-time systems the success of task execution depends on temporal and logical correctness. Hence, some real-time systems can accept results with less precision, which implies in less execution time. They are composed of tasks with multiple execution modes (multi-modal) as in multimedia, image and speech processing, and in mathematical applications that can be prematurely halting.

This paper develops a convex model for providing a reconfiguration mechanism, which jointly selects the CPU frequency and task precision or execution mode, while considering the system and application constraints. The solution must be flexible to work on different systems, efficient to provide on-line reconfiguration according the situation, maximize the QoS, and also obey all constraints.

The remainder of this paper is structured as follows. First, in Section II the related work is discussed. Section III presents a general formulation of the problem, with the constraints being detailed after a brief problem introduction. Section IV presents a brief overview of the solution used to solve the proposed model. In Section V, the behavior and efficiency of an algorithm for solving the reconfiguration problem is evaluated numerically. Finally, section VI offers some conclusions.

II. Related Work

To control CPU temperature and save energy the CPU Dynamic Voltage and Frequency Scaling (DVFS) is a technique commonly used in real-time systems, such as the system discussed in [2], [3], [4], [5], [6] that offer guarantees on minimal energy savings and improve the system quality of service. For example, in [6], QoS increases as the probability of deadline task miss decreases.

The effects of temperature in computers is analyzed in [7], [8]. These works show that high temperatures decrease CPU power efficiency, reduces system reliability, and increases the power drawn by the cooling system, with a throttling mechanism that degrades the CPU performance. To address these issues, new models have been proposed in [9], [10] to account for the temperature effects. Nevertheless, these works do not guarantee any energy saving.

In [4], [5] it is proposed an on-line mechanism that guarantees energy savings, ensures schedulability, and avoids

This work was supported by CAPES n°6544/13-4 and CNPq n°478273/2012-6.

QoS degradation. Nevertheless, this mechanism does not take into account the temperature effects.

Our work proposes a new mechanism that guarantees minimal energy savings, considers the temperature in a power-consumption model, supports multi-modal tasks, and further keeps CPU temperature under control to avoid CPU throttling. To achieve these goals, the proposed mechanism selects the CPU frequency for DVFS applications and defines the executing mode of the tasks, while respecting all of the system constraints. The algorithm developed in this work should be run only when the system experiences new conditions, such as when the temperature is close to the maximum or minimum limit, or the energy source has less or more power than predicted.

III. The Addressed Problem

A model for managing real-time systems with energy consumption, temperature, and schedulability constraints is presented. The CPU frequency and task mode are considered continuous variables, and will be adjusted to maximize system QoS and satisfy the constraints. Although we consider a continuous model, applications are discrete and the algorithm would pick the next discrete mode.

A. System Model

The system is composed of n independent real-time tasks. Each task has more than one operating mode, each being associated with an execution time that should be above $C_i^{\mathrm{D,min}}$ and below $C_i^{\mathrm{D,max}}$. The system uses a single core CPU that operates with a continuous frequency F_i, that will be represented by f_i. Variable f_i will be used to indicate the frequency used by a task, $f_i = \frac{F_i}{F_{\max}}$, where $F_{\max}$ is the maximum frequency supported by the CPU. The frequency f_i varies from the minimum value of $f^{\min}$ to the maximum value of $f^{\max}$.

The tasks share a single processor that runs in a continuous range of frequency, such as ARM1176JZF-S, used by RaspBerry-Pi [11]. This processor simulates a continuous frequency and the voltage operates in the interval [0.8V,1.4V], changing in steps of 0.025V.

The execution time C_i is composed of two terms, C_i^{D} and C_i^{I}, which represent the time an activation of τ_i may spend using the CPU and other devices, respectively. C_i^{D} is considered to be inversely proportional to the value of the selected frequency: C_i^{D} increases by 100% when the processor frequency is reduced by 50% and it will vary in the range from $C_i^{\mathrm{D,min}}$ to $C_i^{\mathrm{D,max}}$. Thus, the total execution time of τ_i is expressed as:

$$C_i = \frac{C_i^{\mathrm{D}}}{f_i} + C_i^{\mathrm{I}}.$$

Processor utilization is determined by the sum of the utilizations of each task τ_i, being defined as:

$$u_i(f_i, C_i^{\mathrm{D}}) = \frac{C_i}{T_i} = \frac{C_i^{\mathrm{D}}}{T_i f_i} + \frac{C_i^{\mathrm{I}}}{T_i}.$$

It is well known that for the tasks to be EDF-schedulable the following condition must hold:

$$\sum_{i=1}^{n} u_i(f_i, C_i^{\mathrm{D}}) \leq 1. \tag{1}$$

Since the configuration of the task frequencies consumes CPU resources, the management algorithm should be invoked only when needed.

B. Problem Formulation

The model is for a system with real-time tasks, energy consumption and temperature limits. System QoS should be kept as high as possible for the prevailing situation, which evolves dynamically depending on the conditions of the environment. A convex optimization model was sought to grant an efficient solution, with the objective function reflecting the quality of service and the constraints representing the system limits.

Under these assumptions, the problem is generically formulated as:

$$\begin{aligned}
P: \quad \max \quad & \sum_{i=1}^{n} G(u_i, \alpha_i, f_i) && \text{(2a)}\\
\text{s.t.}: \quad & \sum_{i=1}^{n} u_i \leq U_l && \text{(2b)}\\
& \sum_{i=1}^{n} P_i \leq \frac{E^\star}{T_{\mathrm{est}}} && \text{(2c)}\\
& f^{\min} \leq f_i \leq f_i^{\max},\ i = 1, \ldots, n && \text{(2d)}\\
& C_i^{\mathrm{D,min}} \leq C_i^{\mathrm{D}} \leq C_i^{\mathrm{D,max}},\ i = 1, \ldots, n && \text{(2e)}\\
& H(f_i) \leq H^{\mathrm{thr}},\ i = 1, \ldots, n && \text{(2f)}
\end{aligned}$$

where:

- Eq. (2a) expresses the task configuration profit;
- Eq. (2b) models CPU utilization according with the selected scheduling policy (see Eq. (1));
- constraint (2c) represents the sum of P_i, that is the power drawn by a task depending on their configuration, where $E^\star$ and T_{est} the available energy and the system lifetime respectively;
- constraint (2d) establishes bounds for f_i;
- Eq. (2e) defines the range for task execution time;
- constraint (2f) ensures that the CPU temperature won't rise above H^{thr} (maximum admissible temperature) with f_i.

The proposed model is designed to accept any scheduling policy, as long as a convex function can determine whether or not a set of tasks is schedulable. Hereafter, we assume a EDF policy for an uniprocessor system, and the schedulability is ensured by Eq. (1). In the next subsections the other constraints will be discussed in detail.

C. Energy Model

We make two assumptions regarding energy: 1) the energy consumption of the tasks accounts for the devices as in [12]; and 2) the temperature increases with power consumption [9], [13]. Based on these assumptions, the DPM (Dynamic Power Model) [14] was chosen to represent the energy consumption of a task τ_i per unit of time:

$$P_i = P_i^{\mathrm{Ind}} + P_i^{\mathrm{Dep}} + P^{\mathrm{Fan}} + P^{\mathrm{Sleep}} \tag{3}$$

where: P_i^{Ind} is the power that is independent of processor frequency, determined by the peripherals used by task τ_i; P_i^{Dep} is the energy per unit of time consumed by the CPU to perform τ_i; P^{Fan} is the power consumed by the fan to cool the system; P^{Sleep} is the power cost to maintain the system devices in the sleep mode, which, being a constant, will be omitted to simplify the presentation.

P_i^{Dep} will be given by the sum of the CPU dynamic and leakage power, being expressed as:

$$P_i^{\mathrm{Dep}} = P_i^{\mathrm{Dyn}} + P_i^{\mathrm{Leak}} \tag{4}$$

with P_i^{Dyn} being defined as $C_{\mathrm{ef}} V_{\mathrm{dd}}^2 F_i$, where F_i is the frequency used by the processor to perform task τ_i, C_{ef} is the capacitance required to execute an instruction, and V_{dd} is the voltage needed to operate the system at a given frequency. According to [15], it is common to affirm that the voltage required by the CPU to run at frequency F_i is given by $V_{\mathrm{dd}} = V_{\max} f_i$.

The power consumption of the ARM architecture [16] can increase 14%, when the temperature reaches 85°C. This behavior occurs due to the leakage power, which has an exponential dependence on the CPU temperature, H, as shown in [8]. This is represented as:

$$P_i^{\mathrm{Leak}} = C_1 + C_2 e^{HC_3} \tag{5}$$

where C_1, C_2, and C_3 are defined by experiments and analysis of the CPU leakage power. Again, according to [8], the sum of the leakage and fan power is a convex-like curve that reaches a minimum at a certain point, which yields the best speed to execute the fan. Hence, P^{Fan} becomes a constant and will be omitted to simplify the presentation.

For the formulation presented here, the power drawn by a task from the system becomes:

$$P_i(f_i, C_i^{\mathrm{D}}) = C_{\mathrm{ef}} V_{\max}^2 f_i^3 F_{\max} + C_1 + C_2 e^{HC_3} + P_i^{\mathrm{Ind}}. \tag{6}$$

By adding up the energy consumed per unit of time by each task, the following expression of the total power required to operate the system is obtained:

$$\sum_{i=1}^{n} P_i(f_i, C_i^{\mathrm{D}}) u_i(f_i, C_i^{\mathrm{D}}) \tag{7}$$

The values of P_i^{Dep} and P_i^{Ind} represent the power consumed by the processor and other devices. Further, u_i shows the power drawn by a task from the system.

The energy consumption of a task is not fixed due to the variability of the computational load, task execution time and mode (which vary according to the environment conditions). Consequently, it is not possible to anticipate how much energy will be needed to accomplish a goal. Therefore, it becomes necessary to estimate execution time and available energy to avoid unnecessary QoS degradation. Hence the following constraint was added to the model, to limit the power consumed by the system:

$$\sum_{i=1}^{n} P_i(f_i, C_i^{\mathrm{D}}) u_i(f_i, C_i^{\mathrm{D}}) \le \frac{E^\star}{T_{\mathrm{est}}} \tag{8}$$

where $\frac{E^\star}{T_{\mathrm{est}}}$ is the available power of the system.

D. Temperature

The CPU temperature is of major importance because cooling systems are not evolving at the same rate of performance and the heat produced by new processors. In [7] it is showed that the CPU power dependents on the temperature, increasing with it. To this end, a Dynamic Thermal Management (DTM) system is used to prevent a CPU from reaching high temperatures and burning.

Despite the extensive research on DTM and its hardware implementation, which with low system overhead, DTM exhibits a zig-zag behavior and can degrade system performance because the processor can run for unknown time at the lowest frequency until cooling the system. This behavior can make a task take more time to finish its execution or even miss the deadline. In order to avoid this undesirable behavior, we add to the model the heat produced by the execution of each task.

According to [9], [13], the temperature of a CPU can be modeled as a function that depends on its consumed power and cooling capacity, being expressed as:

$$\frac{P_i^{\mathrm{Dyn}}}{C_{\mathrm{th}}} - \frac{h - h_{\mathrm{amb}}}{R_{\mathrm{th}} C_{\mathrm{th}}} \tag{9}$$

where: $P_i^{\mathrm{Dyn}} = C_{\mathrm{ef}} V_{\max} f_i^3 F_{\max}$, C_{th} is the thermal capacitance of the chip, R_{th} is the thermal resistance of the chip, and h and h_{amb} represent respectively the CPU and ambient temperatures. To simplify the equations the following constants are introduced:

$$a = \frac{C_{\mathrm{ef}} V_{\max} F_{\max}}{C_{\mathrm{th}}}, \quad b = \frac{1}{R_{\mathrm{th}} C_{\mathrm{th}}}.$$

Hence, the dynamic temperature for the CPU to run task τ_i at the selected frequency f_i is given by:

$$H(f_i) = a f_i^3 - b(h - h_{\mathrm{amb}}) + h. \tag{11}$$

The set of constraints of the model does not define an order for task execution, so the only way to ensure that CPU temperature will never go above a limit H^{thr} is:

$$H(f_i) \le H^{\mathrm{thr}}, i = 1, \ldots, n. \tag{12}$$

E. Problem Decision Variables

The proposed model has two decision variables for each task τ_i, namely the frequency f_i and the task execution time C_i^{D}. Basically, the number of decision variables in a system will be twice the number of tasks. f_i limits are defined by $f^{\min} \le f_i \le f^{\max}$, where $f^{\min}$ is the lowest frequency supported by the processor and $f^{\max}$ is the maximum.

The range for C_i^{D} depends on τ_i as $C_i^{\mathrm{D,min}} \leq C_i^{\mathrm{D}} \leq C_i^{\mathrm{D,max}}$, with $C_i^{\mathrm{D,min}}$ being the shortest time required to execute τ_i, whereas $C_i^{\mathrm{D,max}}$ is the maximum. They are defined considering the CPU at $f^{\max}$. These bounds are determined by how much a task can be degraded.

F. Objective Function

According to [4], the task profit must capture aspects as the task weight, the degradation of execution quality, and its importance for the system as the function below:

$$\sum_{i=1}^{n}\Big(\frac{C_i^{\mathrm{I}} f_i}{T_i} + \frac{C_i^{\mathrm{D}}}{T_i}\Big)\alpha_i. \tag{13}$$

This equation has two terms, the first one divides the product of C_i^I and f_i by T_i, which reflects the weight of the independent CPU time of a task and the importance of increase the frequency. The second term represent the weight of the CPU execution time of a task without the frequency effect. And all the terms are multiplied by α_i, this value represents the task importance for the system.

G. Problem Formulation

By using the specific objective function and constraints detailed above, the generic formulation (2) becomes:

$$P_{init}: \quad \max \; f = \sum_{i=1}^{n}\Big(\frac{C_i^{\mathrm{I}} f_i}{T_i} + \frac{C_i^{\mathrm{D}}}{T_i}\Big)\alpha_i \tag{14a}$$

$$\text{s.t.}: \sum_{i=1}^{n} \frac{C_i^{\mathrm{D}}}{T_i f_i} \leq 1 - \sum_{S_i \in \mathcal{S}} \frac{C_i^{\mathrm{I}}}{T_i} \tag{14b}$$

$$\sum_{i=1}^{n}\Big(C_{\mathrm{ef}} V_{\max}^2 f_i^2 F_{\max} \frac{C_i^{\mathrm{D}}}{T_i} + C_{\mathrm{ef}} V_{\max}^2 f_i^3 \tag{14c}$$

$$+ F_{\max}\frac{C_i^{\mathrm{I}}}{T_i} + \frac{C_1 C_i^D}{T_i f_i} + \frac{C_2 e^{HC_3} C_i^D}{T_i f_i} + \tag{14d}$$

$$P_i^{\mathrm{I}} \frac{C_i^{\mathrm{D}}}{T_i f_i}\Big) \leq \frac{E^\star}{T_{\mathrm{est}}} - P^{\mathrm{S}} - \sum_{S_i \in \mathcal{S}}\Big(\frac{P_i^{\mathrm{I}} C_i^{\mathrm{I}}}{T_i} \tag{14e}$$

$$- \frac{C_i^{\mathrm{I}} C_1 - C_i^{\mathrm{I}} C_2 e^{HC_3}}{T_i}\Big) \tag{14f}$$

$$f^{\min} \leq f_i \leq f^{\max}, \; i = 1, \ldots, n \tag{14g}$$

$$C_i^{\mathrm{D,min}} \leq C_i^{\mathrm{D}} \leq C_i^{\mathrm{D,max}}, \; i = 1, \ldots, n \tag{14h}$$

$$H(f_i) \leq H^{\mathrm{thr}}, \; i = 1, \ldots, n \tag{14i}$$

As we have two decision variables, f_i and C_i^{D}, the Problem (14) is not convex. Since an exponential function is convex [17, Cap. 3], Problem (14) is recast in an equivalent form according with the following change of variables:

$$\left\{ \begin{array}{c} f_i = e^{\overline{f_i}} \\ C_i^{\mathrm{D}} = e^{\overline{C_i^{\mathrm{D}}}} \end{array} \right. \iff \left\{ \begin{array}{c} \overline{f_i} = \log(f_i) \\ \overline{C_i^{\mathrm{D}}} = \log(C_i^{\mathrm{D}}) \end{array} \right.$$

which leads to the following equivalent problem:

$$P: \quad \max \; f = \sum_{i=1}^{n}\Big(\log(\frac{C_i^{\mathrm{I}} e^{\overline{f_i}}}{T_i}) + \log(\frac{e^{\overline{C_i^{\mathrm{D}}}}}{T_i})\Big)\alpha_i \tag{15a}$$

$$\text{s.t.}: \sum_{i=1}^{n} \frac{e^{\overline{C_i^{\mathrm{D}}}}}{T_i e^{\overline{f_i}}} \leq 1 - \sum_{S_i \in \mathcal{S}} \frac{C_i^{\mathrm{I}}}{T_i} \tag{15b}$$

$$\sum_{i=1}^{n}\Big(C_{\mathrm{ef}} V_{\max}^2 e^{2\overline{f_i}} F_{\max} \frac{e^{\overline{C_i^{\mathrm{D}}}}}{T_i} + C_{\mathrm{ef}} V_{\max}^2 e^{3\overline{f_i}} \tag{15c}$$

$$F_{\max}\frac{C_i^{\mathrm{I}}}{T_i} + \frac{C_1 e^{\overline{C_i^D}}}{T_i e^{\overline{f_i}}} + \frac{C_2 e^{HC_3} e^{\overline{C_i^D}}}{T_i e^{\overline{f_i}}} + \tag{15d}$$

$$P_i^{\mathrm{I}} \frac{e^{\overline{C_i^{\mathrm{D}}}}}{T_i e^{\overline{f_i}}}\Big) \leq \frac{E^\star}{T_{\mathrm{est}}} - P^{\mathrm{S}} \tag{15e}$$

$$- \sum_{S_i \in \mathcal{S}}\Big(\frac{P_i^{\mathrm{I}} C_i^{\mathrm{I}}}{T_i} - \frac{C_i^{\mathrm{I}} C_1 - C_i^{\mathrm{I}} C_2 e^{HC_3}}{T_i}\Big) \tag{15f}$$

$$H(e^{\overline{f_i}}) \leq H^{\mathrm{thr}}, \; i = 1, \ldots, n \tag{15g}$$

$$\log(f^{\min})s \leq \overline{f_i} \leq \log(f^{\max}), i = 1, \ldots, n \tag{15h}$$

$$\log(C_i^{\mathrm{D,min}}) \leq \overline{C_i^{\mathrm{D}}} \leq \log(C_i^{\mathrm{D,max}}), \; i = 1, \ldots, n \tag{15i}$$

The proposed formulation (15) is a convex-programming problem, and can be solved by efficient algorithms, such as sequential quadratic programming [18].

IV. Sequential Quadratic Programming (SQP)

Sequential quadratic programming (SQP) is a general framework for solving constrained nonlinear problems. Given the current iterate $\mathbf{x}^k = (f_i^k, C_i^{\mathrm{D},k} : i = 1, \ldots, n)$, SQP solves a quadratic program approximating the local optimality conditions, the Karush-Kuhn–Tucker (KKT) conditions [18], and then produces the next iterate. The obtained solution $\widetilde{\mathbf{x}}^k$ defines a search direction $\mathbf{p}^k = (\widetilde{\mathbf{x}}^k - \mathbf{x}^k)$ along which a line-search is applied to define a step $\eta^k > 0$ that yields the next iterate $\mathbf{x}^{k+1} = (\mathbf{x}^k + \eta^k \mathbf{p}^k)$. An efficient implementation of Sequential Quadratic Programming (SQP) is CFSQP [19]. For convex programs, CFSQP will arrive at a global optimum [19].

As the algorithm to solve the problem is iterative, it can be halted at any moment. Hence, it will be divided into two on-line phases. Phase 1 that obtains a feasible solution and Phase 2 executes the algorithm until reach the optimal solution. A solution is feasible if it satisfies all the constraints, not necessarily guaranteeing optimality. Finding the optimum may take considerable time, hence a *tolerance factor* is used to accept a nearly optimal solution. For example, a small tolerance ϵ ensures that the distance of an approximate solution $\mathbf{x}$ to the optimum $\mathbf{x}^\star$ will be bounded by ϵ, namely $\|\mathbf{x} - \mathbf{x}^\star\| \leq \epsilon$.

V. Numerical Analyses

This section presents the results of three numerical analyses based on the CFSQP solver; namely, the impact of tolerance, the impact of phases, and the time to solve the mathematical problem (15). With these analyses the solution overhead, and effectiveness are evaluated.

A. Experimental Setup

To evaluate the proposed solution, 200 synthetic sets with 5, 10, and 20 tasks were generated with the UUniFast algorithm [20]. Each task set has a system utilization of 80%, this value was set based on values chosen by [1], [10], in which 60% is the CPU usage and 20% represents C_i^{I}. $C_i^{\mathrm{D,min}}$ is 80% of $C_i^{\mathrm{D,max}}$, the deadline is equal to the task period, which is 1 s and P_i^{I} varies from 2 to 93 mW

The experiments were performed using a Raspberry-pi model B, operating at $700Mhz$ with $512Mb$ of RAM. This hardware platform runs an embedded version of the Linux system. The energy parameters used in the experiments were: $V = 1.2$; $f^{\max} = 700MHz$; $f^{\min} = 100MHz$; $P^{\mathrm{S}} = 0$; $C_{\mathrm{ef}} = 1.0035 \times 10^{-10}$. In order to find C_{ef} the CPU maximum power was considered to be $0.85W$. Moreover, $E^{\star}/T_{\mathrm{est}}$ was set as 5% more than the power needed for providing the highest QoS. This is to avoid border conditions, forcing a reconfiguration in the system. Thermal parameters were defined according to [8]: the CPU maximum temperature was set to 48°C; the environment temperature was 26°C; and the CPU temperature was 42°C. Two values for the tolerance were assumed: $\epsilon = 10^{-1}$ (high tolerance); and $\epsilon = 10^{-4}$ (low tolerance).

In the figures below, we present most of the results in the form of box and whiskers plots, where the line in the box represents the median value, the bottom and top of the box are the 25th and 75th percentile, lines below and above the box represent the minimal and maximum values, except for the outliers which are shown as crosses.

B. Impact of Tolerance (ϵ)

Figure 1 illustrates the utilization of the CPU and the difference in power consumption of Phase 2, according to ϵ variation. The first graph of the figure shows how much CPU is consumed when we go from a low tolerance to a high tolerance. We can see a small variation (about 1%–3%) in the average usage of CPU resources. And the second graph shows that a low tolerance (that is, being more strict in how close the solution has to be from optimal) will cause savings of less than 3mW on average, which represents less than 1% of the power consumed by the system.

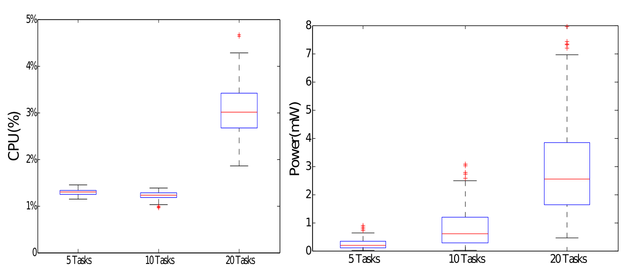

Fig. 1. Difference between the power for low tolerance and high tolerance, for Phase 2, for different numbers of tasks.

The difference in percentage of profit between the low and high tolerance for Phase 2 with 20 tasks is under 0.02%, implying that higher tolerance will influence the profitability of the system in less than 1%. From this results, we can show that a high tolerance (10^{-1}) is sufficient to obtain good solutions.

C. Impact of Phases (feasible vs optimal)

Figure 2 has two graphs, one shows the difference of CPU usage and the other the difference of power consumed from Phase 2 to 1 in the low tolerance scenario; the high tolerance scenario (not shown) has the same behavior. In the CPU usage graph we noted that as the number of tasks increases, the CPU utilization decreases slightly and in Phase 2 the utilization improves almost 12%. From the

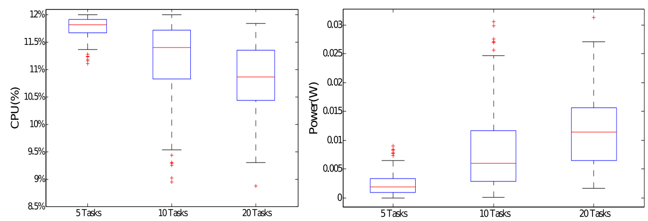

Fig. 2. The graphs represent the difference of CPU usage and power of Phases 2 and 1 in the low tolerance scenario, respectively.

second graph of Figure 2 is possible to note that the difference increases with the number of tasks, implying that Phase 1 is more effective with more tasks (i.e., consumes less power). However, the amount of saved power by not running Phase 2 is insignificant.

In our experiments, the profit difference between Phases 1 and 2 with low tolerance is relatively low, approximately 2.5%. This observation is in accordance with Figure 2, which shows a small variation in resource consumption (CPU and power) in configurations of Phase 1 and 2. However, as it will be shown in the next section, Phase 2 takes significantly longer to find a quasi-optimal solution.

From this analysis, it is possible to verify the effectiveness of Phase 1, and the gain in QoS induced by a low tolerance is very small. The configurations of Phase 2 used all available resource to ensure the minimum QoS degradation, while obeying the constraints.

D. Execution Time Analysis

Figures 3 and 4 show that Phase 1 can be efficiently solved by CFSQP in all scenarios. Considering a low

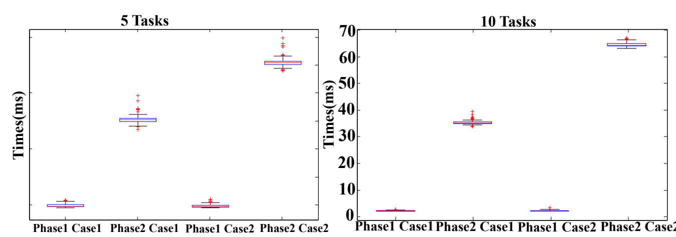

Fig. 3. Execution times in ms of CFSQP for 5 and 10 tasks.

tolerance, for the respective scenarios of 5, 10 and 20 tasks, the lowest, highest and mean execution time in milliseconds to solve Phase 1 were $\{1.6, 2.2, 1.7\}$, $\{2, 3.5, 2.3\}$ and $\{3.5, 5.2, 4\}$. The cumulative execution time for Phase 1 and Phase 2 were: $\{11.5, 13.9, 12.2\}$, $\{45.8, 48.7, 46.9\}$ and $\{63.0, 67.1, 64.6\}$. The graphs in Figures 3 and 4, show that the execution time required for executing Phase 2 is more sensitive to size of the task set and tolerance values.

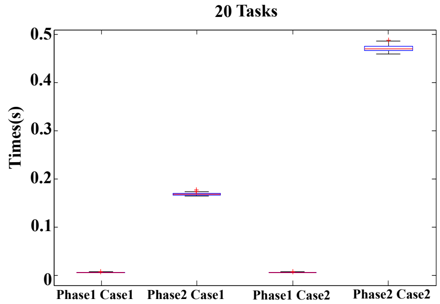

Fig. 4. Execution times in ms of CFSQP for 5 and 10 tasks.

In conclusion, the presented results indicate that the distribution of the execution time is reasonably concentrated around the mean value, indicating small variations on the execution time of the CFSQP. A high tolerance implies in low QoS degradation and less execution time. Furthermore, if the worst scenario is known it is possible to solve the problem off-line and use the answer to solve the problem on-line, this method is known as Warm Start.

E. Impact of Energy Budget

We analyze the behavior of CFSQP with varying energy availability, which we decrease by 5%, up to the value of 80% of the energy needed for the best configuration. Also, the mean execution time taken by CFSQP for 20 tasks and $\epsilon = 10^{-4}$ is shown in Table I. Some tests were done with less than 80%, but it was not enough to schedule most of the sets, so we do not include it in the table.

TABLE I. CFSQP Mean Execution Time And Profit Varying Energy Availability

Energy Available:	95%	90%	85%	80%
Phase 1 exec. time	$0.01S$	$0.01S$	$0.011S$	$0.011S$
Phase 2 exec. time	$0.85S$	$1.07S$	$1.17S$	$1.28S$
Profit difference(%)	3.9	3.8	3.5	2.7

Table I shows that Phase 1 can be efficiently solved with profit very close to the one of Phase 2, implying in a low resources consumption. Note that as the available energy decreases, the profit difference between phases decreases.

VI. Conclusion

A new reconfiguration mechanism that provides energy-savings with guarantees and temperature management in real-time system was presented, along the algorithm for its solution. A numerical analysis evaluated the efficiency and behavior of the used algorithm. The proposed model tends to be more representative of real world systems than current models, due to the temperature constraint. Another feature of the model is the convexity, which enables efficient softwares or methods discussed in [17].

Our model does not depend on a specific scheduling policy; it supports any policy that can ensure the schedulability of a task set in a single core by a convex function. Moreover, our scheme can be complemented by algorithms that increase the energy-saving by using slack time (see [6]). Future work will consider the validation of the model in a real system and comparison with other solutions.

References

[1] S. Liu, Q. Wu, and Q. Qiu, “An Adaptive Scheduling and Voltage/Frequency Selection Algorithm for Real-Time Energy Harvesting Systems,” in *Proc. of the ACM/IEEE 46th Conf. on Des. Automation*, 2009, pp. 782–787.

[2] C. Rusu, R. Melhem, and D. Mossé, “Multi-Version Scheduling in Rechargeable Energy-Aware Real-Time Systems,” *Journal of Embedded Computing*, vol. 1, no. 2, pp. 271–283, 2005.

[3] L. Niu, “Energy Efficient Scheduling for Real-Time Embedded Systems with QoS Guarantee,” *Real-Time Syst.*, vol. 47, no. 2, pp. 75–108, 2011.

[4] R. Nassiffe, E. Camponogara, and G. Lima, “Optimizing QoS in Energy-Aware Real-Time Systems,” *ACM SIGBED Review*, vol. 10, no. 2, pp. 25–25, 2013.

[5] R. Nassiffe, E. Camponogara, G. Lima, and D. Mossé, “Optimizing QoS in Adaptive Real-Time Systems With Energy Constraint Varying CPU Frequency,” in *Proc. of the Brazilian Symposium on Computing Syst. Engineering*, 2013.

[6] R. Xu, D. Mossé, and R. Melhem, “Minimizing Expected Energy Consumption in Real-Time Systems Through Dynamic Voltage Scaling,” *ACM Transactions on Computer Syst.*, vol. 25, no. 4, Dec. 2007.

[7] W. Liao, L. He, and K. Lepak, “Temperature and Supply Voltage Aware Performance and Power Modeling at Microarchitecture Level,” *IEEE Transactions on Computer-Aided Des. of Integrated Circuits and Syst.*, vol. 24, pp. 1042–1053, 2005.

[8] M. Zapater, J. Ayala, J. Moya, K. Vaidyanathan, K. Gross, and A. Coskun, “Leakage and Temperature Aware Server Control for Improving Energy Efficiency in Data Centers,” in *Proc. of the Conf. on Des., Automation and Test in Europe*, 2013, pp. 266–269.

[9] J. Gu and G. Qu, “Incorporating Temperature-Leakage Interdependency Into Dynamic Voltage Scaling for Real-Time Systems,” in *Proc. of the 24th IEEE Int. Conf. on Application-Specific Syst., Architectures and Processors*, 2013, pp. 289–296.

[10] R. Rao and S. Vrudhula, “Performance Optimal Processor Throttling Under Thermal Constraints,” in *Proc. of Int. Conf. on Compilers, Architecture, and Synthesis for Embedded Syst.*, 2007, pp. 257–266.

[11] *Intelligent Energy Controller*, ARM.

[12] H. Aydin, R. Melhem, D. Mossé, and P. Mejía-Alvarez, “Power-Aware Scheduling for Periodic Real-Time Tasks,” *IEEE Transactions on Computers*, vol. 53, no. 5, pp. 584–600, 2004.

[13] Y. Fu, N. Kottenstette, Y. Chen, C. Lu, X. Koutsoukos, and H. Wang, “Feedback Thermal Control for Real-Time Systems,” in *Proc. of the 16th IEEE Real-Time and Embedded Technology and Applications Symposium*, 2010, pp. 111–120.

[14] B. Zhao and H. Aydin, “Minimizing Expected Energy Consumption Through Optimal Integration of DVS and DPM,” in *Proc. of the Int. Conf. on Computer-Aided Des.*, 2009, pp. 449 –456.

[15] C. Kyung and S. Yoo, *Energy-Aware System Design: Algorithms and Architectures.* Springer, 2011.

[16] K. Flautner, D. Flynn, and M. Rives, “A Combined Hardware-Software Approach for Low-Power SoCs: Applying Adaptive Voltage Scaling and Intelligent Energy Management Software,” in *Proc of the 29th European Solid-State Circuits Conf.*, 2003.

[17] S. Boyd and L. Vandenberghe, *Convex Optimization.* Cambridge, 2004.

[18] P. T. Boggs and J. W. Tolle, “Sequential quadratic programming,” *Acta Numerica*, vol. 4, pp. 1–51, 1995.

[19] C. T. Lawrence and A. Zhou, J.L. end Tits, “User’s Guide for CFSQP Version 2.0: A C Code for Solving (Large Scale) Constrained Nonlinear (Minimax) Optimization Problems, Generating Iterates Satisfying All Inequality Constraints,” Electrical Engineering Dept., University of Maryland, Tech. Rep., 1994.

[20] E. Bini and G. Buttazzo, “Measuring the Performance of Schedulability Tests,” *Real-Time Syst.*, vol. 30, no. 1, pp. 129–154, 2005.