Optimizing Quality of Service in Real-Time Systems under Energy Constraints[*]

Ríad Nassiffe
Department of Automation and Systems Engineering
Federal University of Santa Catarina
Florianópolis, SC 88040-900, Brazil
riad@das.ufsc.br

Eduardo Camponogara
Department of Automation and Systems Engineering
Federal University of Santa Catarina
Florianópolis, SC 88040-900, Brazil
camponog@das.ufsc.br

George Lima
Computer Science Department
Federal University of Bahia
Salvador, BA 40170-110, Brazil
gmlima@ufba.br

ABSTRACT

Embedded real-time systems powered by batteries require suitable support for energy-savings at the operating system level. Mechanisms to do so must take into consideration not only energy constraints but also schedulability since, in this kind of system, tasks must execute within predefined time windows. On top of that, it is desired that application quality of service (QoS) is optimized.

In this paper we present a framework capable of maximizing application QoS subject to both schedulability and energy constraints. It is assumed that application tasks may have multiple operating modes, each of which exhibiting a specific QoS level when running at a specific processor operating frequency. Although the formulated problem is NP-Hard, experimental analysis has shown that the derived heuristic to solve it achieves very good approximation results and presents low running time.

Categories and Subject Descriptors

H.4.1 [**Operating Systems**]: Process Management—*Energy Save and Scheduling*

General Terms

Algorithms, Optimization, Performance

Keywords

Real-Time, Energy savings

1. INTRODUCTION

Both the computing power and complexity of real-time embedded systems have grown considerably in the past few years. Nowadays mobile phones have become more like hand held computers, with several multimedia-like tools capable of executing 3D graphical applications, such as the NVIDIA embedded processor Tegra APX Series. This trend can be noticed in several other devices, including sensors, smart cameras, or autonomous robots. Since such devices are often powered by batteries, there must be mechanisms to provide energy-savings, which are usually implemented at the operating system level.

Providing energy-savings for real-time embedded systems requires sophisticated solutions when compared to general purpose operating systems in which it may be enough to manage underlying hardware power-saving functionalities. For example, components such as monitor, hard driver or communication devices can be turned-off or put in sleep states during (or after some) inactivity periods. Upon some predefined wake-up events, such devices are put in the operational mode again. This is done without any explicit time requirements, though. Because of this, such a simple approach does not suffice when it comes to real-time systems whose tasks are specified to execute within predefined time windows in a predictable fashion, that is system schedulability must be guaranteed.

The usual approach to dealing with energy-savings in real-time embedded systems is to adapt real-time scheduling algorithms so that it is possible to take advantage of idle processing time to reduce processor voltage/frequency. In other words, the goal is to minimize energy consumption subject to system schedulability. This approach may not suffice when quality of service (QoS) guarantees must be ensured since energy savings may compromise QoS beyond what is tolerated. Indeed, as reported by Rusu *et al.* [28, 27], there are applications that require maximizing QoS subject to both schedulability and energy constraints. This is the problem we address in this paper.

Another common requirement of new real-time embedded systems is the need of dynamically adjustment due to external sensory data or low-level architecture features, which has been recently pointed out [6]. In particular, reconfigurable systems can be structured as having different modes of operation. For example, the computer vision subsystem of a robot may experience different operating modes due to environment changes. Light conditions, obstacles, vision

[*]This work was supported in part by CNPq.

angle, modifications in the robot goals during its lifetime, and other unpredictable environmental characteristics may be modeled by different operating modes. In this context, support for switching from one operating mode to another is needed at the scheduling level.

In this paper we describe a mechanism capable of dynamically reconfiguring multi-mode applications considering both schedulability and energy constraints. It is assumed that there may be one or more operating modes associated with each task in the system. Each mode in turn is associated with a benefit value, which represents the corresponding QoS level defined by application designers. The goal is to determine at run-time which configuration of the system gives the maximum aggregate benefit value subject to schedulability and energy constraints. As this optimization problem is NP-Hard, an efficient polynomial time heuristic is derived. Simulation results indicate that the derived heuristic achieves very good solutions as compared to the optimum.

The remainder of this paper is structured as follows. Related work is summarized in Section 2. The model for energy consumption considered in this work and the technical notation are presented in Section 3. The proposed approach is described in Section 4 and is assessed by simulation in Section 5. Conclusions are drawn in Section 6.

2. RELATED WORK

Dynamic Voltage Scaling (DVS) is an important and commonly used mechanism to provide energy savings in computer systems. It works by scaling down the processor voltage/frequency. DVS schemes for real-time systems take advantage of information on the system slack/idle times. Off-line schemes use information on the difference between task worst-case computation times and deadlines (*e.g.* [32, 26, 14, 31]) while on-line schemes are based on monitoring the actual task execution times (*e.g.* [29, 15, 34, 36, 22]) and so are usually more effective. In either case, most developed techniques to date focus on the problem of minimizing the processor energy consumption subject to timing constraints. Unlike these approaches, we focus on taking QoS requirements into consideration. Although recent work on DVS has been taken specific QoS aspects into account, such as fault tolerance [35], few of them deal with QoS in general.

The problem addressed here is similar to the one solved by Rusu *et al.* [28, 27]. They also describe an on-line approach to maximizing the system benefit subject to both energy and timing constraints. Nonetheless, they assume an overly restrictive task model, according to which all tasks share the same deadline. Another approach to maximizing QoS under both schedulability and energy constraints has been devised for the Imprecise-Computation model [33]. The authors provide an algorithm to distribute slack between tasks using a best-effort heuristic. Differently from these approaches, our solution is able to give guarantees for energy-savings by solving an optimization problem without imposing extra restrictions on the task model.

The optimization problem addressed in this paper is designed for adaptive systems comprised by a set of tasks, each of which having multiple modes of operation. The goal is to find at run-time a system configuration that maximizes QoS so that the system can adapt itself to possible changes in the application requirements or its environment. There have been several solutions to this problem. For example, Lima *et al.* [18] have considered selecting task modes so that aperiodic requests can be accommodated, leading to graceful transient degradation. Jehuda and Israeli [13] have described a multi-layer reconfiguration framework. The approach by Lee *et al.* [17, 16] has been designed to maximize the system benefit taking into consideration that applications specify a minimum QoS requirement for the needed computation resources. Efficient approximation algorithms to maximizing QoS ensuring deterministic [8] or probabilistic schedulability [7] guarantees have also been given. None of these results, however, take into consideration energy-saving constraints.

3. MODEL AND NOTATION

3.1 Power Model and Energy Management

We assume a single processor system capable of operating at different frequency/voltage levels, which can be selected at run-time. The set of allowed frequencies is denoted by $\mathcal{F}$. Without loss of generality, we assume that the set $\mathcal{F}$ is normalized, namely $\forall f \in \mathcal{F}, 0 < f \leq 1$. For example, if the processor can operate at $0.8GHz$, $1.6GHz$, or $2.8GHz$, $\mathcal{F} = \{0.25, 0.5, 1\}$.

In certain embedded systems, the processor can consume less energy than other devices such as wireless communication cards and hard disks. To this end, this paper uses DVS combined with the Dynamic Power Model (DPM) proposed by Zhu *et al.* [35] to control processor frequency/voltage and manage system power consumption more realistically. Formally, the system power P consumed by some task is expressed as

$$P = P_{\mathrm{S}} + (P^{\mathrm{D}} + P^{\mathrm{I}})h \tag{1}$$

where: P_{S} is the static power necessary to maintain the system devices on sleep mode, which can be disconsidered from the energy model for not being subject to dynamic adjustments; the binary term h models the operating ($h = 1$) and sleep ($h = 0$) system modes; and the terms P^{D} and P^{I} represent the frequency-dependent and frequency-independent active power modes, respectively. The former changes as a function of the processor operating voltage/frequency, while the latter does not since it corresponds to the active power consumed by off-chip components (I/O devices, memory, *etc.*) used by the task.

Further, P^{D} is defined as:

$$P^{\mathrm{D}} = CV^2 f \tag{2}$$

where C is the processor capacitance and V is the processor voltage for a specific operating frequency f. According to [35] the operating frequency should decrease linearly with the reduction of processor voltage.

3.2 Task model

We assume a real-time system comprised of n independent tasks. Tasks may have more than one operating mode. An operating mode of a task is associated with a release frequency and/or a worst-case execution time. Different operating modes of a task provide different QoS levels. For example, a task running at a low/high frequency may give low/high QoS. Further, the application may have distinct versions (implementations) of a task, each of which associated with some QoS level. More costly versions may be designed to provide high QoS and can be activated when system resources are underutilized. We stress that there is no restriction regarding the way task modes are implemented, which is an issue related to the application.

We assume that the level of QoS associated with a given task mode gives a certain benefit for the application/system. We denote $A_i^{k,j}$ the benefit given by task i when it runs in mode k and with the processor frequency f_j. No restriction on the values of $A_i^{k,j}$ is assumed. The application designer may use any benefit function to set up the values of $A_i^{k,j}$. The definition of benefit function is not the focus of this paper, though. An interested reader can be referred to other sources [5, 25].

Applications for which the mechanism described in this paper is useful may contain either soft or hard tasks. In the former case, it is recommended the use of reservation-based scheduling [20] according to which tasks are given processing bandwidths so that temporal isolation is ensured. Motivated by this need, we assume that there is a server associated with each task. A server is defined by the tuple (Q, T), meaning that the task being served may execute up to Q processing units per T time units. The use of servers here is not part of the mechanism we are going to describe, though. Their use is to make our description independent of the kind of tasks that are being dealt with. Hence, hereafter we assume that the system is composed of a set of n servers $\mathcal{S} = \{S_1, \ldots, S_n\}$.

We do not assume a specific type of bandwidth-reservation mechanism but require that servers are scheduled in the system according to the Earliest Deadline First (EDF) policy. Mechanisms which are in line with this assumption can be found in other works (*e.g.* [20, 1]) and will not be further described. The time needed to switch frequency/voltage is 50 μs in the worst case according to [2]. As we are using EDF and cannot determine the number of preemptions of a task, 50 μs will be added to the application execution time twice, one to represent the time to adjust the frequency/voltage to the values chosen by the algorithm and the other for the application to return to its previous configuration. Thus, 100 μs (0.10 ms) should be added to the execution time of each task.

We denote $S_i = (Q_i^k, T_i^k)$ as the server associated with task i running in mode k. We assume that the execution of a task can be divided in two parts, one related to the term P^{D} and the other to the term P^{I}, which is in line with Eq. (1). Hence, $Q_i^k = Q_i^{k,\mathrm{D}} + Q_i^{k,\mathrm{I}}$, where $Q_i^{k,\mathrm{D}}$ and $Q_i^{k,\mathrm{I}}$ account for the frequency-dependent and frequency-independent parts of the task, respectively.

4. SYSTEM RECONFIGURATION UNDER ENERGY CONSTRAINTS

This section begins by presenting the mathematical formulation of the problem of server reconfiguration under schedulability and energy consumption restrictions. A sample instance is defined to illustrate concepts and algorithms. Lagrangian and surrogate relaxations are developed to compute bounds and parameters that will be handy in the design of a heuristic. Finally, a greedy heuristic based on the surrogate relaxation and Lagrangian multipliers is proposed to find approximate solutions for the server reconfiguration problem.

4.1 Problem Formulation

The model assumes a processor that can run at a frequency f_j chosen from a finite set $\mathcal{F} = \{f_1, \ldots, f_F\}$. The real-time system is composed of a set $\mathcal{S} = \{S_1, \ldots, S_n\}$ of servers. A server S_i operates at a mode k selected from a set $\mathcal{K}_i = \{1, \ldots, K_i\}$ and is characterized by a tuple $(Q_i^{k,\mathrm{I}}, Q_i^{k,\mathrm{D}}, P_i^{k,\mathrm{I}}, T_i^k)$ defining the frequency-independent active budget, the frequency-independent budget, the power consumption, and the period, respectively.

The problem consists in defining the mode of operation and processor frequency for each server so as to maximize the overall system benefit, while ensuring task schedulability and keeping energy consumption below a user-defined limit. In mathematical notation the problem is cast as:

$$P: \quad \max \quad f = \sum_{S_i \in \mathcal{S}} \sum_{(k,j) \in \Omega_i} A_i^{k,j} x_i^{k,j} \tag{3a}$$

$$\text{s.t.}: \sum_{(k,j) \in \Omega_i} x_i^{k,j} = 1, \; S_i \in \mathcal{S} \tag{3b}$$

$$\sum_{S_i \in \mathcal{S}} \sum_{(k,j) \in \Omega_i} u_i^{k,j} x_i^{k,j} \leq 1 \tag{3c}$$

$$\sum_{S_i \in \mathcal{S}} \sum_{(k,j) \in \Omega_i} P_i^{k,j} x_i^{k,j} \leq \beta P^\star \tag{3d}$$

$$x_i^{k,j} \in \{0, 1\}, S_i \in \mathcal{S}, (k,j) \in \Omega_i \tag{3e}$$

where:

- $\Omega_i = \{(k,j) : k \in \mathcal{K}_i, \; f_j \in \mathcal{F}\}$ is the set of all possible mode-frequency configurations of server S_i;
- $u_i^{k,j}$ is the processor utilization of server S_i when it operates at mode k and frequency f_j, being defined as:

$$u_i^{k,j} = \frac{Q_i^{k,\mathrm{D}}}{T_i^k f_j} + \frac{Q_i^{k,\mathrm{I}}}{T_i^k} \tag{4}$$

- $P_i^{k,j}$ is the power consumed by server S_i, operating at mode k and frequency f_j, being defined as:

$$P_i^{k,j} = (P_i^{k,\mathrm{I}} + CV^2 f_j) u_i^{k,j} \tag{5}$$

- $P^\star$ is the maximum power consumption given by $P^\star = \sum_{S_i \in \mathcal{S}} \max\{P_i^{k,j} : (k,j) \in \Omega_i\}$ and $\beta \in (0, 1]$ is a user-defined parameter establishing the limit on energy consumption;
- $A_i^{k,j}$ is the system benefit accrued by running server S_i in mode k and at frequency f_j;
- $x_i^{k,j}$ is a binary variable which takes on value 1 if and only if server S_i runs in mode k and at processor frequency f_j.

The possibility of interruption or admission denial of a server S_i is modeled by setting $A_i^{k,j} = 0$, $u_i^{k,j} = 0$, and $P_i^{k,j} = 0$ for a mode k and processor frequency f_j.

Although this work assumes the EDF schedulability policy, other policies may be used such as Rate Monotonic (RM) by replacing inequality (3c) with

$$\sum_{S_i \in \mathcal{S}} \sum_{(k,j) \in \Omega_i} u_i^{k,j} x_i^{k,j} \leq U_{\text{lub}} \tag{6}$$

where U_{lub} is the CPU utilization limit defined according to the number of tasks. Similar to the schedulability policy, other power management models such as DVS can be used by modifying the energy constraint (3d).

The reconfiguration problem can be extended to control the frequency/voltage of other devices by introducing additional dimensions to the given parameters, currently restricted to the task mode k and processor frequency f_j. For instance, a new dimension could be introduced to model memory frequency.

The server reconfiguration problem under energy constraints is obviously NP-Hard since the standard knapsack problem is reducible to P. Actually, P is a variation of the two-dimensional knapsack problem [30].

A dynamic programming (DP) algorithm can be devised for solving P. Suppose that constraints (3c) and (3d) are multiplied by sufficiently large positive constants Δ and Γ to make the parameters $u_i^{k,j}$ and $P_i^{k,j}$ integers, respectively. The DP algorithm is based on the recursive computation of the function $f_l(\delta, \gamma)$ defined as the optimal objective of the problem $P_l(\delta, \gamma)$, which consists of P restricted to servers $S_1, \ldots, S_l$, the right-hand side of (3c) replaced by $\delta \in \{0, 1, \ldots, \Delta\}$ *(available CPU)*, and the right-hand side of (3d) replaced by $\gamma \in \{0, 1, \ldots, \Gamma\beta P^\star\}$ *(available energy)*.

An nice feature of the DP algorithm is that it can reach an optimal solution. The algorithm runs in $\Theta(\Delta\Gamma\beta P^\star|\Omega|)$ time, where $\Omega = \{(i,k,j) : S_i \in \mathcal{S}, (k,j) \in \Omega_i\}$ is the set of all mode-frequency configurations of all servers. Owing to its running time, the DP algorithm is efficient only for small instances or when the constants Δ, $\Gamma\beta P^\star$, and n are relatively small. Because the DP approach solves a family of reconfiguration problems for varying energy and CPU availability, reconfiguration can be carried out immediately provided that the DP tables are stored in memory.

4.2 Sample Instance

A sample instance is defined here with the purpose of illustrating models and algorithms. The tasks were generated with the UUniFast algorithm [3] because it produces scenarios that are neither too pessimistic, nor overly optimistic for the analysis. The sample real-time system consists of $n = 3$ servers with each server S_i having $K_i = 3$ modes of operation. The processor has $F = 3$ frequencies represented by the set $\mathcal{F} = \{1.0, 0.75, 0.5\}$, whereby $f_1 = 1.0$ means that the processor runs at frequency $2.53\,GHz$, $f_2 = 0.75$ for $1.90\,GHz$, and $f_3 = 0.50$ for $1.27\,GHz$. The parameters of Table 1 and Eq. (4) were used to obtain the server utilizations given in Table 2.

Table 1: Server configuration parameters

i	k	$(Q_i^{k,\text{I}}, Q_i^{k,\text{D}}, P_i^{k,\text{I}}, T_i^k)$
	1	$(0.1, 15.70, 0.438, 33)$
1	2	$(0.2, 1.9, 0.438, 66.7)$
	3	$(0.4, 8.2, 0.438, 200)$
	1	$(0.3, 7.6, 0.438, 33)$
2	2	$(1.9, 16.6, 0.438, 66.7)$
	3	$(0.3, 4.8, 0.438, 200)$
	1	$(2.2, 0.03, 0.438, 33)$
3	2	$(1.2, 1.5, 0.438, 66.7)$
	3	$(2.7, 3.7, 0.438, 200)$

Table 2: CPU utilization

	$u_i^{k,j}$		
	$i=1$	$i=2$	$i=3$
$u_i^{1,1}$	47.88%	23.94%	6.76%
$u_i^{1,2}$	63.74%	31.62%	6.79%
$u_i^{1,3}$	95.45%	46.97%	6.85%
$u_i^{2,1}$	3.15%	27.74%	4.05%
$u_i^{2,2}$	4.10%	36.03%	4.80%
$u_i^{2,3}$	6.00%	52.62%	6.30%
$u_i^{3,1}$	4.30%	2.55%	3.20%
$u_i^{3,2}$	5.67%	3.35%	3.82%
$u_i^{3,3}$	8.40%	4.95%	5.05%

The power consumption parameters are $C \approx 6.32411 \times 10^{-9}$, $V_{\max} = 1.25$ [12], and $P_i^{k,\text{I}} = 0.438$ [21] for all $S_i \in \mathcal{S}$, $k \in \mathcal{K}_i$. Using these parameters, the CPU utilizations $u_i^{k,j}$, and the processor frequencies in Eq. (5), the energy consumption of the servers for the various modes and frequencies are obtained as shown in Table 3. The benefits of the system tasks appear in Table 4 depending on operating mode and processor frequency.

The maximum energy consumption $P^\star = 10.5$ of the real-time system is computed from the instance parameters. The sample instance of the problem of reconfiguring real-time serves under schedulability and energy consumption constraints, P, is readily defined with the given parameters.

Table 3: Energy Consumption

	$P_i^{k,j}$		
	$i=1$	$i=2$	$i=3$
$P_i^{1,1}$	12.1794	6.0897	1.7190
$P_i^{1,2}$	7.0015	3.4730	0.7456
$P_i^{1,3}$	3.4010	1.6735	0.2440
$P_i^{2,1}$	0.8009	7.0555	1.0297
$P_i^{2,2}$	0.4502	3.9581	0.5270
$P_i^{2,3}$	0.2137	1.8750	0.2244
$P_i^{3,1}$	1.0938	0.6487	0.8140
$P_i^{3,2}$	0.6225	0.3680	0.4193
$P_i^{3,3}$	0.2993	0.1764	0.1799

The dynamic programming algorithm will find the optimal server reconfiguration if $\Delta = 10^4$ and $\Gamma = 10^4$. Because

Table 4: System benefit

	$A_i^{k,j}$		
	$i=1$	$i=2$	$i=3$
$A_i^{1,1}$	3.0000	1.0000	3.0000
$A_i^{1,2}$	2.5000	0.9658	2.5000
$A_i^{1,3}$	2.0000	0.9316	2.0000
$A_i^{2,1}$	0.0898	3.0000	1.0000
$A_i^{2,2}$	0.0838	2.5000	0.8998
$A_i^{2,3}$	0.0778	2.0000	0.7995
$A_i^{3,1}$	1.0000	0.8631	0.5990
$A_i^{3,2}$	0.7724	0.6703	0.5677
$A_i^{3,3}$	0.5449	0.4775	0.5363

such parameters will make the DP tables large, an approximate version of the DP algorithm was applied to the sample instance with $\Delta = 10^2$ and $\Gamma = 10^2$. Fractional utilizations $\Delta u_i^{k,j}$ and energy consumption $\Gamma P_i^{k,j}$ were rounded to the nearest integer. Despite this approximation, the DP algorithm found the optimal solution for $\beta = 1$, which is $x_1^{3,1} = 1$, $x_2^{2,1} = 1$, and $x_3^{1,1} = 1$ with a total benefit of 7 units. It also found the optimal solution for $\beta = 0.5$, which is $x_1^{3,1} = 1$, $x_2^{2,3} = 1$, and $x_3^{1,1} = 1$ with a total benefit of 6 units. Notice that with the reduction in power supply, the server S_2 was reconfigured to run at a lower processor frequency.

4.3 Relaxations

Let $\mathbf{x} = (x_i^{k,j} : (i,k,j) \in \Omega)$ be a vector collecting all variables of P and let $\mathcal{P} = \{\mathbf{x} : \mathbf{x} \text{ satisfies (3b) through (3e)}\}$ denote its solution space. Then, P is compactly stated as $f^\star = \max\{f(\mathbf{x}) : \mathbf{x} \in \mathcal{P}\}$. A problem R, defined by $r^\star = \max\{r(\mathbf{x}) : \mathbf{x} \in \mathcal{R}\}$, is a relaxation of P if [30]:

i) $\mathcal{P} \subseteq \mathcal{R}$;

ii) $r(\mathbf{x}) \geq f(\mathbf{x})$ for all $\mathbf{x} \in \mathcal{P}$.

Because a relaxation problem must be solved up to optimality for $r^\star$ to induce an upper bound, it is mostly useful when an efficient algorithm exists. Relaxations are key to establishing certificates of quality for feasible solutions and also to the design of algorithms for solving optimization problems. For instance, integer-programming algorithms such as branch-and-bound and cutting-plane rely on the computation of bounds via relaxations.

Lagrangian and surrogate relaxations are developed below for the reconfiguration problem. They will play a part in the design of a heuristic for solving the server reconfiguration problem under schedulability and energy consumption constraints.

4.3.1 Lagrangian Relaxation

Given Lagrange multipliers for the schedulability and energy consumption constraints, $\boldsymbol{\lambda}_\text{u}$ and $\boldsymbol{\lambda}_\text{p}$ respectively, the Lagrangian dual function f_L for the reconfiguration problem is obtained as follows:

$$LP(\boldsymbol{\lambda}):$$

$$f_\text{L}(\boldsymbol{\lambda}) = \max \sum_{(i,k,j)\in\Omega} (A_i^{k,j} - \boldsymbol{\lambda}_\text{u} u_i^{k,j} - \boldsymbol{\lambda}_\text{p} P_i^{k,j}) x_i^{k,j} + \boldsymbol{\lambda}_\text{u} + \boldsymbol{\lambda}_\text{p}\beta P^\star \quad (7a)$$

$$\text{s.t.}: \sum_{(k,j)\in\Omega_i} x_i^{k,j} = 1,\ S_i \in \mathcal{S} \quad (7b)$$

$$x_i^{k,j} \in \{0,1\},\ (i,k,j) \in \Omega \quad (7c)$$

where $\boldsymbol{\lambda} = (\boldsymbol{\lambda}_\text{u}, \boldsymbol{\lambda}_\text{p})$. For any $\boldsymbol{\lambda} \geq \mathbf{0}$, the Lagrangian dual function induces an upper bound $f_\text{L}(\boldsymbol{\lambda}) \geq f^\star$. This dual function has the nice property of being computed analytically as follows:

1. define $(\hat{k},\hat{j})(i) = \arg\max\{(A_i^{k,j} - \boldsymbol{\lambda}_\text{u} u_i^{k,j} - \boldsymbol{\lambda}_\text{p} P_i^{k,j}) : (k,j) \in \Omega_i\}$.
2. the solution $\mathbf{x}(\boldsymbol{\lambda})$ to $LP(\boldsymbol{\lambda})$ is obtained by setting, for all $S_i \in \mathcal{S}$, $x_i^{k,j}(\boldsymbol{\lambda}) = 1$ if $(k,j) = (\hat{k},\hat{j})(i)$ and otherwise $x_i^{k,j}(\boldsymbol{\lambda}) = 0$ for all $(k,j) \in \Omega_i$.

The Lagrangian dual problem consists in finding the vector $\boldsymbol{\lambda}^\star$ that minimizes the upper bound f_L [9]. Formally, the Lagrangian dual is cast as:

$$LD:\ f_\text{L}(\boldsymbol{\lambda}^\star) = \min_{\boldsymbol{\lambda}\geq\mathbf{0}} f_\text{L}(\boldsymbol{\lambda}) \quad (8)$$

According to duality theory, the Lagrangian dual problem is convex because $f_\text{L}(\boldsymbol{\lambda})$ is convex and nondifferentiable. Although the nondifferentiability of f_L prevents the use of efficient gradient-based algorithms such as the damped Newton's method [4], a subgradient algorithm (SGA) can minimize f_L since subgradients are easily computed.

A vector $\boldsymbol{\xi}(\boldsymbol{\lambda}) \in \mathbb{R}^2$ is a subgradient for f_L at $\boldsymbol{\lambda}$ if $f_\text{L}(\bar{\boldsymbol{\lambda}}) \geq f_\text{L}(\boldsymbol{\lambda}) + \boldsymbol{\xi}(\boldsymbol{\lambda})'(\bar{\boldsymbol{\lambda}} - \boldsymbol{\lambda})$ for all $\bar{\boldsymbol{\lambda}} \geq \mathbf{0}$. Given a solution $\mathbf{x}(\boldsymbol{\lambda}) = (x_i^{k,j}(\boldsymbol{\lambda}) : (i,k,j) \in \Omega)$ to $LP(\boldsymbol{\lambda})$, a subgradient for f_L at $\boldsymbol{\lambda}$ is obtained as follows:

$$\boldsymbol{\xi}(\boldsymbol{\lambda}) = \begin{bmatrix} \boldsymbol{\xi}_\text{u}(\boldsymbol{\lambda}) \\ \boldsymbol{\xi}_\text{p}(\boldsymbol{\lambda}) \end{bmatrix} = \begin{bmatrix} 1 - \sum_{(i,k,j)\in\Omega} u_i^{k,j} x_i^{k,j}(\boldsymbol{\lambda}) \\ \beta P^\star - \sum_{(i,k,j)\in\Omega} P_i^{k,j} x_i^{k,j}(\boldsymbol{\lambda}) \end{bmatrix} \quad (9)$$

The availability of subgradients allows us to develop a subgradient algorithm to solve, at least approximately, the Lagrangian dual. The subgradient algorithm is detailed in Algorithm 1. Under certain conditions, this algorithm produces a series of multipliers that converges to the optimal Lagrangian vector $\boldsymbol{\lambda}^\star$. In particular, Theorem 10.4 of [30] ensures that the series $\{f_\text{L}(\boldsymbol{\lambda}^{(k)})\}_{k=0}^{\infty}$ converges to $f_\text{L}(\boldsymbol{\lambda}^\star)$ if $\mu^{(0)}$ and $\rho < 1$ are sufficiently large, and $k^\text{max} = \infty$.

The performance of the subgradient method depends greatly on the sequence $\{\mu^{(k)}\}$ of the step lengths for the subgradients. This means that the parameters $\mu^{(0)}$ and ρ will have to be tuned for the particular problem at hand. The running time of SGA is controlled by the iteration limit k^max. Its step 6 requires scanning all the parameters defining the reconfiguration problem P, so the computational cost for this

Algorithm 1 Subgradient Algorithm (SGA)

```
1: Input: servers $\mathcal{S}$, $\{(A_i^{k,j}, u_i^{k,j}, P_i^{k,j}) : (i,k,j) \in \Omega\}$, $\beta$,
   initial Lagrangian multiplier $\boldsymbol{\lambda}^{(0)}$, initial step $\mu^{(0)}$, decreasing rate $\rho < 1$, tolerance $\eta$, and iteration limit $k^{\max}$
2: $k := 0$ {iteration counter}
3: $f_{\mathrm{L}}^{\mathrm{best}} := \infty$ {best upper bound}
4: $f^{\mathrm{best}} := -\infty$ {best feasible objective}
5: repeat
6:    solve $LP(\boldsymbol{\lambda}^{(k)})$, obtaining $\mathbf{x}(\boldsymbol{\lambda}^{(k)})$ and $f_{\mathrm{L}}(\boldsymbol{\lambda}^{(k)})$
7:    if $f_{\mathrm{L}}(\boldsymbol{\lambda}^{(k)}) < f_{\mathrm{L}}^{\mathrm{best}}$ then
8:       $\boldsymbol{\lambda}^{\mathrm{best}} := \boldsymbol{\lambda}^{(k)}$
9:       $f_{\mathrm{L}}^{\mathrm{best}} := f_{\mathrm{L}}(\boldsymbol{\lambda}^{(k)})$
10:   end if
11:   if $\mathbf{x}(\boldsymbol{\lambda}^{(k)})$ is feasible for $P$ and $f(\mathbf{x}(\boldsymbol{\lambda}^{(k)})) > f^{\mathrm{best}}$
      then
12:      $\mathbf{x}^{\mathrm{best}} := \mathbf{x}(\boldsymbol{\lambda}^{(k)})$
13:      $f^{\mathrm{best}} := f(\mathbf{x}^{\mathrm{best}})$
14:   end if
15:   compute subgradient $\boldsymbol{\xi}(\boldsymbol{\lambda}^{(k)})$ according to Eq. (9)
      using $\mathbf{x}(\boldsymbol{\lambda}^{(k)})$
16:   obtain the next multipliers:
         $\boldsymbol{\lambda}_{\mathrm{u}}^{(k+1)} := \max\{0, \boldsymbol{\lambda}_{\mathrm{u}}^{(k)} - \mu^{(k)}\boldsymbol{\xi}_{\mathrm{u}}(\boldsymbol{\lambda}^{(k)})\}$
         $\boldsymbol{\lambda}_{\mathrm{p}}^{(k+1)} := \max\{0, \boldsymbol{\lambda}_{\mathrm{p}}^{(k)} - \mu^{(k)}\boldsymbol{\xi}_{\mathrm{p}}(\boldsymbol{\lambda}^{(k)})\}$
17:   $\mu^{(k+1)} := \rho\mu^{(k)}$
18:   $k := k + 1$
19: until ($k > k^{\max}$ or $\frac{\|\boldsymbol{\lambda}^{(k+1)} - \boldsymbol{\lambda}^{(k)}\|}{\|\boldsymbol{\lambda}^{(k)}\|} \le \eta$)
20: return $\boldsymbol{\lambda}^{\mathrm{best}}$, $f_{\mathrm{L}}^{\mathrm{best}}$, $\mathbf{x}^{\mathrm{best}}$, and $f^{\mathrm{best}}$
```

step is $\Theta(|\Omega|)$. Since all the other steps take time proportional to n or less, SGA runs in $\mathcal{O}(|\Omega|k^{\max})$ time.

SGA was applied to the sample instance with $\boldsymbol{\lambda}^{(0)} = \mathbf{1}$, $\mu^{(0)} = 1$, $\rho = 0.95$, $\eta = 10^{-3}$, and $k^{\max} = 200$:

- for $\beta = 1$, the algorithm yielded $f_{\mathrm{L}}^{\mathrm{best}} = 7.6131$ with $\boldsymbol{\lambda}^{\mathrm{best}} = (0.3484, 0.1707)$, and $f^{\mathrm{best}} = 5.7724$ induced by the solution $x_1^{3,2} = 1$, $x_2^{2,3} = 1$, and $x_3^{1,1} = 1$.
- for $\beta = 0.5$, the algorithm produced $f_{\mathrm{L}}^{\mathrm{best}} = 6.2437$ with $\boldsymbol{\lambda}^{\mathrm{best}} = (0, 0.4335)$, and $f^{\mathrm{best}} = 6$ induced by the solution $x_1^{3,1} = 1$, $x_2^{2,3} = 1$, and $x_3^{1,1} = 1$. Notice that this solution coincides with the optimal solution attained by the DP algorithm.

Two other Lagrangian relaxations are obtained by dualizing only the schedulability constraint (3c) or the energy consumption constraint (3d), denoted LD_{u} and LD_{p} respectively. The computation of the dual functions of these relaxations is more costly than the computation of $f_L(\boldsymbol{\lambda})$ since they entail solving a generalization of the knapsack problem. However, $LP(\boldsymbol{\lambda})$ has the integral property, meaning that the solution of its continuous relaxation is integer [10]. This implies that the lowest upper bound $f_{\mathrm{L}}(\boldsymbol{\lambda}^\star)$ has the same value of the bound obtained by solving the linear-programming relaxation of P. On the other, the relaxations LD_{u} or LD_{p} do not have the integral property, implying that the upper bounds obtained by these dual problems may be lower than the linear-programming bound.

In this work we choose the Lagrangian dual function f_{L} dualizing the schedulability and energy consumption constraints for its simplicity, low computational requirements, and the need of combining these two resources in the design of a density greedy heuristic. A linear-programming algorithm specially tailored for P could be used instead of the subgradient algorithm to obtain the optimal dual vector $\boldsymbol{\lambda}^\star$ and upper bound $f_{\mathrm{L}}^\star$. However, a linear-programming algorithm is much more complex than SGA, invariably consuming more memory and potentially more time to reach an optimal solution.

4.3.2 Surrogate Relaxation

The surrogate relaxation is obtained by combining restrictions (3c) and (3d) with surrogate multipliers $\boldsymbol{\tau} = (\boldsymbol{\tau}_{\mathrm{u}}, \boldsymbol{\tau}_{\mathrm{p}}) \ge \mathbf{0}$ into a single constraint [11]. More precisely, the inequalities (3c) and (3d) are replaced with the surrogate inequality:

$$\sum_{(i,k,j)\in\Omega} (\boldsymbol{\tau}_{\mathrm{u}} u_i^{k,j} + \boldsymbol{\tau}_{\mathrm{p}} P_i^{k,j}) x_i^{k,j} \le \boldsymbol{\tau}_{\mathrm{u}} + \boldsymbol{\tau}_{\mathrm{p}}\beta P^\star \tag{10}$$

Given a vector $\boldsymbol{\tau} \ge \mathbf{0}$, the surrogate dual function f_{S} is computed by solving the problem:

$$SP(\boldsymbol{\tau}):$$

$$f_{\mathrm{S}}(\boldsymbol{\tau}) = \max \sum_{(i,k,j)\in\Omega} A_i^{k,j} x_i^{k,j} \tag{11a}$$

$$\text{s.t.}: \sum_{(k,j)\in\Omega_i} x_i^{k,j} = 1,\ S_i \in \mathcal{S} \tag{11b}$$

$$\sum_{(i,k,j)\in\Omega} (\boldsymbol{\tau}_{\mathrm{u}} u_i^{k,j} + \boldsymbol{\tau}_{\mathrm{p}} P_i^{k,j}) x_i^{k,j} \le \boldsymbol{\tau}_{\mathrm{u}} + \boldsymbol{\tau}_{\mathrm{p}}\beta P^\star \tag{11c}$$

$$x_i^{k,j} \in \{0,1\}, (i,k,j) \in \Omega \tag{11d}$$

Because $f_{\mathrm{S}}(\boldsymbol{\tau}) \ge f$ for any $\boldsymbol{\tau} \ge \mathbf{0}$, the surrogate dual function yields an upper bound for the reconfiguration problem. Unlike the Lagrangian dual function $f_{\mathrm{L}}(\boldsymbol{\lambda})$ which is computed analytically, the surrogate dual function $f_{\mathrm{S}}(\boldsymbol{\lambda})$ is computationally hard since it generalizes the classic knapsack problem. The surrogate dual expresses the desire of minimizing the surrogate upper bound, being defined as the problem:

$$SD:\ f_{\mathrm{S}}(\boldsymbol{\tau}^\star) = \min_{\boldsymbol{\tau}\ge\mathbf{0}} f_{\mathrm{S}}(\boldsymbol{\tau}) \tag{12}$$

In this paper, we do not attempt to compute the surrogate dual function, let alone solve the surrogate dual. Instead, the surrogate dual function and problem will be useful in the design of a heuristic for solving the reconfiguration problem approximately.

4.4 Density Greedy Heuristic

Based on the procedure developed in [19] for the two-constraint knapsack problem, we propose a heuristic for the reconfiguration problem. The heuristic consists in using the subgradient algorithm to obtain an approximation $\widetilde{\boldsymbol{\lambda}}$ to the optimal Lagrangian multipliers $\boldsymbol{\lambda}^\star$, and then solving $SP(\boldsymbol{\tau})$ for $\boldsymbol{\tau} = \widetilde{\boldsymbol{\lambda}}$ with a density-greedy strategy.

DEFINITION 1. *$X \subseteq \Omega$ is a feasible set of the server configurations if:*

1. *for each $S_i \in \mathcal{S}$ there exists exactly one configuration $(i,k,j) \in X$, that is, $|\Omega_i \cap X| = 1$ for all S_i;*
2. *$\sum_{(i,k,j)\in X} u_i^{k,j} \le 1$ and $\sum_{(i,k,j)\in X} P_i^{k,j} \le \beta P^\star$.*

Assumption 1. *There exists a feasible set $X(\beta)$ of the server configurations for the given energy consumption limit.*

Assumption 2. *For all $S_i \in \mathcal{S}$ and $(k,j),(k',j') \in \Omega_i$, if $u_i^{k,j} < u_i^{k',j'}$ and $P_i^{k,j} < P_i^{k',j'}$ then $A_i^{k,j} < A_i^{k',j'}$. Otherwise, the reconfiguration (k',j') could be discarded.*

In the sample instance, the set $X(\beta) = \{(1,3,1), (2,3,1), (3,3,1)\}$ is feasible for $\beta = 0.5$ because $\sum_{(i,k,j)\in X(\beta)} P_i^{k,j} = 2.5565 < 5.25 = \beta P^\star$ and $\sum_{(i,k,j)\in X(\beta)} u_i^{k,j} = 0.1005 < 1$. Notice that $X(\beta)$ was obtained by selecting modes of operation with low quality for the tasks being managed by the servers.

Let $\Omega(X) = \Omega - X$ and $\Omega_i(X) = \{(k,j) \in \Omega_i : (i,k,j) \notin X\}$ be the configuration sets without the configurations appearing in X. Further, for any S_i let $(k,j)_{X,i}$ be the configuration of S_i appearing in X, *i.e.*, $(i,(k,j)_{X,i}) \in X$. With this notation, X is used to set up a problem $P(X)$ equivalent to the reconfiguration problem P. By substituting $(1 - \sum_{(k,j)\in\Omega_i(X)} x_i^{k,j})$ for $x_i^{(k,j)_{X,i}}$, we arrive at the following form of the reconfiguration problem:

$P(X)$:

$$f = \max \sum_{(i,k,j)\in\Omega(X)} \Delta A_i^{k,j} x_i^{k,j} + \sum_{S_i\in\mathcal{S}} A_i^{(k,j)_{X,i}} \quad (13a)$$

Subject to:

$$\sum_{(k,j)\in\Omega_i(X)} x_i^{k,j} \le 1,\ S_i \in \mathcal{S} \quad (13b)$$

$$\sum_{(i,k,j)\in\Omega(X)} \Delta u_i^{k,j} x_i^{k,j} \le 1 - \sum_{S_i\in\mathcal{S}} u_i^{(k,j)_{X,i}} \quad (13c)$$

$$\sum_{(i,k,j)\in\Omega(X)} \Delta P_i^{k,j} x_i^{k,j} \le \beta P^\star - \sum_{S_i\in\mathcal{S}} P_i^{(k,j)_{X,i}} \quad (13d)$$

$$x_i^{k,j} \in \{0,1\}, (i,k,j) \in \Omega(X) \quad (13e)$$

where $\Delta u_i^{k,j} = u_i^{k,j} - u_i^{(k',j')_{X,i}}$, $\Delta P_i^{k,j} = P_i^{k,j} - P_i^{(k',j')_{X,i}}$, and $\Delta A_i^{k,j} = A_i^{k,j} - A_i^{(k',j')_{X,i}}$ for all $(i,k,j) \in \Omega(X)$.

Our heuristic is based on the density greedy approach of Martello and Toth [19], which was developed for the two-constraint 0-1 knapsack problem. Given a feasible set X of server configurations, the proposed heuristic solves $P(X)$ approximately as follows:

1. solve the Lagrangian dual LD with the subgradient algorithm to obtain Lagrangian multipliers $\widetilde{\boldsymbol{\lambda}} = \boldsymbol{\lambda}^{\text{best}}$ approximating $\boldsymbol{\lambda}^\star$ and a feasible solution $\mathbf{x}^{\text{best}}$; define a feasible configuration set X using $\mathbf{x}^{\text{best}}$ if this candidate solution is feasible, otherwise use a predefined feasible solution $X(\beta)$;
2. define $P(X)$ using X;
3. obtain the surrogate problem $SP(X,\boldsymbol{\tau})$ of $P(X)$ using $\boldsymbol{\tau} = \widetilde{\boldsymbol{\lambda}}$, in a manner similar to the surrogate problem $SP(\boldsymbol{\tau})$;
4. follow the density greedy strategy sorting the configurations in nonincreasing order of the ratio of benefit to the combination of CPU utilization and energy consumption weighted by the Lagrangian multipliers; in mathematical notation, this ratio is given by:
$$\Delta\widetilde{A}_i^{k,j} = \frac{\Delta A_i^{k,j}}{\widetilde{\boldsymbol{\lambda}}_{\text{u}} \Delta u_i^{k,j} + \widetilde{\boldsymbol{\lambda}}_{\text{p}} \Delta P_i^{k,j}}$$
5. examine the configurations in the sorted order, changing server configuration depending on whether or not the new configuration improves overall benefit, while ensuring feasibility.

The pseudo-code of the density greedy heuristic (DGH) appears in Algorithm 2. The complexity of DGH is determined as follows. Step 2 consists in applying SGA which takes $\mathcal{O}(|\Omega|k^{\max})$ time. Steps 3-8 can be performed in $\Theta(n)$ time. Step 9 takes $\Theta(|\Omega|\log|\Omega|)$ to sort the mode-frequency configurations for all servers. Step 11 can be performed in $\Theta(1)$ time using an array. Since steps 12-16 run in constant time, the for-loop 10-18 runs in $\Theta(|\Omega|)$ time. Thus, the running time of DGH is $\mathcal{O}(|\Omega|k^{\max} + |\Omega|\log|\Omega|)$ being dominated by SGA and the sorting step.

Algorithm 2 Density Greedy Heuristic (DGH)

```
1: Input: servers $\mathcal{S}$, $\{(A_i^{k,j}, u_i^{k,j}, P_i^{k,j}) : (i,k,j) \in \Omega\}$, $\beta$,
   feasible set $X(\beta)$, initial Lagrangian multipliers $\boldsymbol{\lambda}^{(0)}$, initial step $\mu^{(0)}$,
   decreasing rate $\rho < 1$, tolerance $\eta$, and iteration limit $k^{\max}$
2: run SGA with the given parameters to obtain $\widetilde{\boldsymbol{\lambda}} = \boldsymbol{\lambda}^{\text{best}}$
   approximating $\boldsymbol{\lambda}^\star$, $f_{\text{L}}^{\text{best}}$, and $\mathbf{x}^{\text{best}}$
3: if $\mathbf{x}^{\text{best}}$ is feasible for $P$ then
4:     $X := \{(i,k,j) : x_i^{k,j} = 1 \text{ in } \mathbf{x}^{\text{best}}\}$
5: else
6:     $X := X(\beta)$ is the initial feasible solution
7: end if
8: $u(X) := \sum_{(i,k,j)\in X} u_i^{k,j}$, $P(X) := \sum_{(i,k,j)\in X} P_i^{k,j}$,
   $f(X) := \sum_{(i,k,j)\in X} A_i^{k,j}$
9: obtain $\widehat{\Omega}(X) := \langle (i,k,j)(1), (i,j,k)(2), \ldots, (i,k,j)(T) \rangle$
   by sorting $\Omega(X)$ in nonincreasing order of $\Delta\widetilde{A}_i^{k,j}$, where
   $T := |\Omega(X)|$
10: for $t = 1$ to $T$ do
11:     find $(i,k,j) \in X$ such that $i = i(t)$
12:     if $(A_i^{k(t),j(t)} \ge A_i^{k,j}) \wedge (u(X) - u_i^{k,j} + u_i^{k(t),j(t)} \le 1)$
        $\wedge\ (P(X) - P_i^{k,j} + P_i^{k(t),j(t)} \le \beta P^\star)$ then
13:         $X := (X - \{(i,k,j)\}) \cup \{(i,k(t),j(t))\}$
14:         $u(X) := u(X) - u_i^{k,j} + u_i^{k(t),j(t)}$
15:         $P(X) := P(X) - P_i^{k,j} + P_i^{k(t),j(t)}$
16:         $f(X) := f(X) - A_i^{k,j} + A_i^{k(t),j(t)}$
17:     end if
18: end for
19: return $X$, $f(X)$, $u(X)$, $P(X)$, and $f_{\text{L}}^{\text{best}}$
```

The heuristic yields a feasible solution X which is not worse than the initial, feasible solution. The quality of X is estimated by the upper bound $f_{\rm L}^{\rm best}$, that is, $f^\star - f(X) \leq f_{\rm L}^{\rm best} - f(X)$.

The density greedy heuristic was applied to the sample instance. DGH used in step 2 the same parameters used by SGA when it was applied to the sample instance.

For $\beta = 0.5$, SGA yielded $\boldsymbol{\lambda}^{\rm best} = (0, 0.4336)$ and found an optimal solution $X^\star = \{(1,3,1),(2,2,3),(3,1,1)\}$ with $f(X^\star) = 6$, $U(X^\star) = 0.6352$, and $P(X^\star) = 4.6878$. By using the feasible solution $X(\beta) = \{(1,3,1),\,(2,3,1),(3,3,1)\}$ with $f(X(\beta)) = 2.4621$, DGH managed to find the optimal solution $X^\star$.

For $\beta = 1.0$, SGA yielded $\boldsymbol{\lambda}^{\rm best} = (0.3485, 0.1709)$ and found a feasible solution $X = \{(1,3,2),(2,2,3),(3,1,1)\}$ with $f(X) = 5.7724$, $U(X) = 0.6489$, and $P(X) = 4.2165$. Using the initial solution X and $\boldsymbol{\lambda}^{\rm best}$ obtained with SGA, DGH found the optimal solution $X^\star$ with $f(X^\star) = 7$, $U(X^\star) = 0.3880$, and $P(X^\star) = 9.8683$.

5. EVALUATION

This section presents numerical results from the solution of the reconfiguration problem using an off-the-shelf optimization solver and the density greedy heuristic. These solution approaches are compared with respect to solution time and quality.

5.1 Simulation Scenarios

To evaluate the performance of the heuristic and solver, sets of tasks were generated using the UUniFast algorithm proposed in [3], which produces representative scenarios devoid of task sets that are too pessimistic or overly optimistic for the analysis. Five scenarios were put together with 10, 20, 30, 40, and 50 tasks. Ten different instances for each scenario were obtained from independent runs of UUniFast.

In all scenarios, the real-time system is composed by tasks that are handled by dedicated servers. A server $S_i \in \mathcal{S}$ manages the execution of task i, which can run in one of $K_i = 3$ different modes of operation and $F = 4$ possible processor frequencies. The processor supports frequencies that are restricted to the set $\mathcal{F} = \{1.0, 0.75, 0.45, 0.25\}$ where the maximum frequency is $2.56\,GHz$. The parameters $Q_i^{k,\rm D}$ and $Q_i^{k,\rm I}$ were produced by the UUniFast algorithm, with $\sum_{i=1}^{n} u_i^{k,1} = 1$ for $k = 1, 2, 3$. The power consumption parameters are the same of the sample instance, with the exception of $P^\star$ which models the maximum energy consumption of the system.

The evaluation was performed in MacBookPro5,5 with a $2.53\,GHz$ Intel Core 2 Duo, 4GB on a Debian (Linux Kernel 2.6.32-5-amd64) operating system and Gcc 4.4.5. The mixed-integer linear programming (MILP) solver Cplex version 12.2.0 was used to find the optimal reconfiguration for all the instances by solving problem P. Cplex is one of the most efficient MILP solvers available in the market.

The benefit value $A_i^{k,j}$ of each possible task configuration was set according to the CPU and frequency utilization. Processor usage was taken into consideration to avoid CPU idle time. Frequency was also considered to avoid selection of tasks with low power consumption and poor quality when energy is still available for the system.

In order to estimate the running time of the Cplex solver in an embedded system platform, such as ARM9 [2], the experiments measured the number of instructions performed. According to [2], an ARM968E-S with $470\,MHz$ can execute 517 DMIP (Dhrystone MIPS), which is equivalent to 517 millions of instructions per second. DMIP is the ratio between the score of the Dhrystone Benchmark program and 1757, which is the score obtained per second on a VAX 11/780, nominally a 1 MIPS machine [23].

The number of instructions executed by the solver to find an optimal solution to the reconfiguration problem was measured with Pintool [24], a computational tool for dynamic instrumentation of programs. By dividing the total number of instructions by the number of instructions processed per second in an ARM9 processor, we obtain an estimate for the running time of the solver in an embedded system.

5.2 Cplex Computational Results

Figure 1 shows the overall system benefit induced by Cplex for all the scenarios, as a function of the energy availability parameter β. Cplex solved the reconfiguration problem P given by expressions (3a)-(3e). The system benefit is the average over the 10 instances of each scenario.

Figure 1 indicates that the overall system benefit declines with a decreasing β, which is in line with the intuition that system performance degrades when energy availability is reduced. When less energy is available, servers are forced to run at low frequencies and with modes that consume less CPU, thereby accruing less benefit to the system.

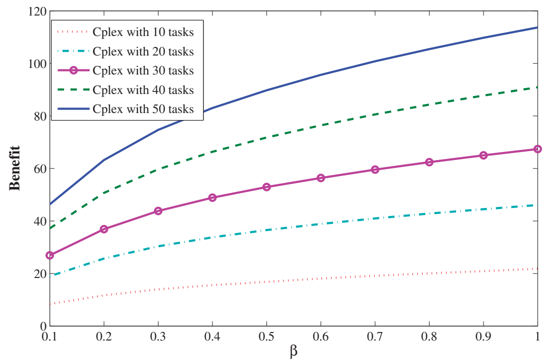

Figure 1: Overall system benefit achieved by Cplex as a function of energy availability.

To establish a fair estimate of the running time of Cplex in an embedded system, we consider an ARM9 processor running at $470\,MHz$, ARM968E-S. The estimated running time of Cplex on ARM968E-S appears in Figure 2. It is important to mention that the reconfiguration problem is solved only when the system undergoes environment changes. If the average running time is not acceptable, and a worst-case running time must be ensured, the solver may be forced to run up to a time limit or reduce the quality of the solution. Setting a tolerance of 4% to the best solution, the worst-case running time of Cplex dropped from 0.79s to 0.15s to solve the instance with 50 tasks. By bounding the solution time or quality, the reconfiguration obtained by solving P approximately will not be optimal, but a feasible one that respects the constraints.

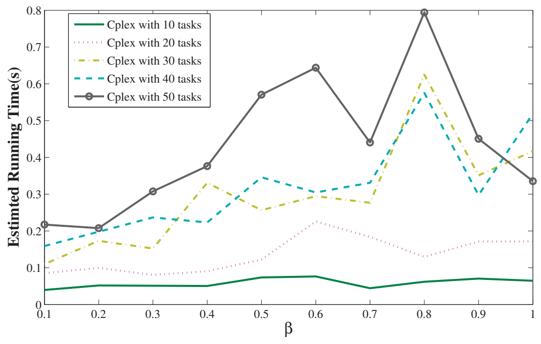

Figure 2: Running time of Cplex to reach the optimal solution of all instances for varying β on a processor ARM968E-S running at $470\,MHz$.

The CPU time necessary to solve a problem instance with Cplex does not necessarily increase with problem size, which is not an atypical behavior in the domain of integer programming as observed in Figure 2. Incidentally, for $\beta = 1.0$, Cplex spent more time to solve the instances of the scenarios with 30 and 40 tasks than those of the scenario with 50 tasks. For all the other values tested for β, the set of 50 tasks executed more instructions than the other sets.

5.3 Heuristic Computational Results

Figure 3 shows the overall system benefit induced by the heuristic for the same scenarios of the Cplex (Figure 1). The results indicate that the system benefit achieved by the heuristic is similar to what was obtained with Cplex. Figure 4 shows the running time of the heuristic to solve the reconfiguration problems, for all instances and varying energy availability. Notice that the average running time of the heuristic is below $16\,ms$ ($0.016\,s$), much faster than Cplex that has an average running time below $800\,ms$ ($0.8\,s$).

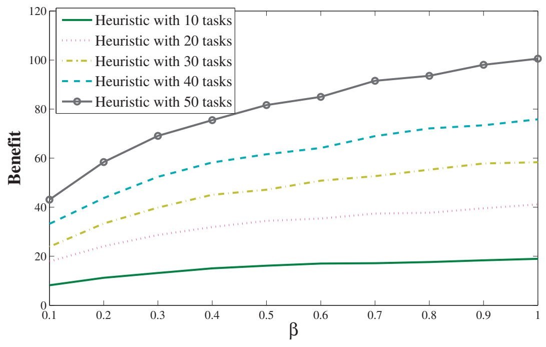

Figure 3: Overall system benefit achieved by the heuristic as a function of energy availability.

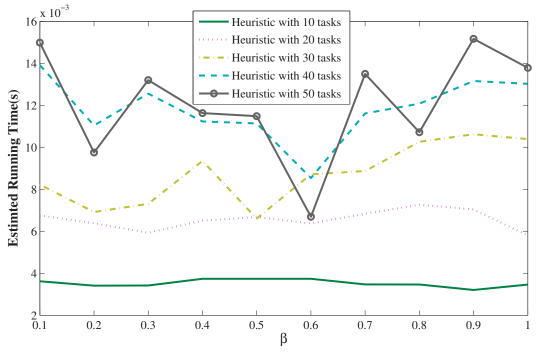

Figure 4: Running time of the heuristic applied to all instances for varying β on a processor ARM968E-S running at $470\,MHz$.

5.4 Remarks

Figure 5 compares the solution yielded by the density greedy heuristic with the optimal reconfiguration found by Cplex, depicting the distance of the heuristic solution from the optimum (error). The average error is approximately 9.9% and the maximum is about 16.5%. To some extent, the quality of the heuristic solution depends on the quality of the Lagrangian multipliers used as surrogate multipliers. For this reason, the subgradient algorithm was tuned so as to induce a good performance of the density greedy heuristic.

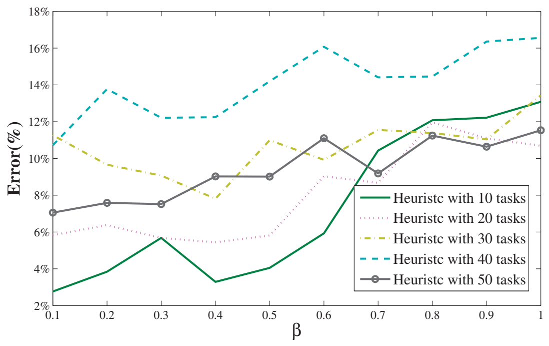

Figure 5: Comparison between the solution produced by the heuristic and the optimal reconfiguration obtained with Cplex.

Figure 6 illustrates the variation of selected modes according to the value of β. The mode selection behavior induced by the optimal solution (Cplex) is not shown because it was similar to the one of the heuristic. It is possible to conclude that the system tends to use the third mode, which generates more benefit by consuming more energy and CPU, as energy availability increases. In the instances chosen for analysis, the tasks consume less energy when they run at low frequency as confirmed by Figure 7.

As an MILP solver, Cplex uses a branch-and-bound search combined with cutting-plane generation and sophisticated procedures for bounding, variable branching, and guiding the search, all of which play a part on efficiency. Thus, the difference in solution time between Cplex and DGH is

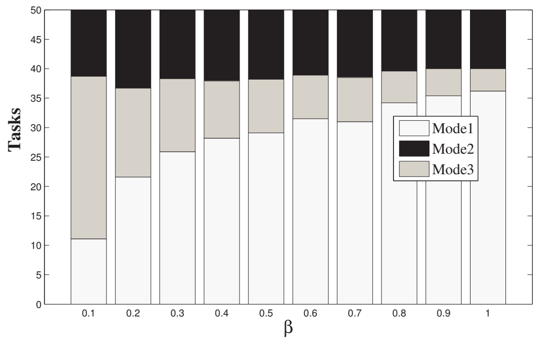

Figure 6: Behavior of mode selection induced by the heuristic solution in the 50 tasks scenario as a function of energy availability.

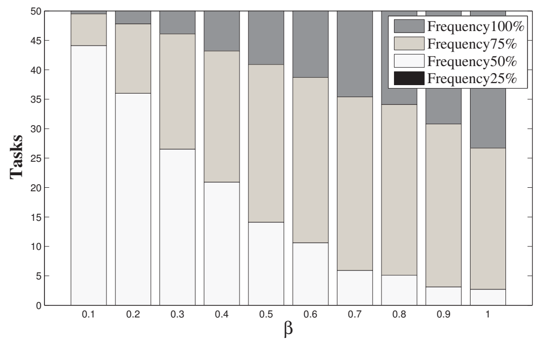

Figure 7: Behavior of frequency selection according to the heuristic solution in the 50 tasks scenario as a function of energy availability.

not surprising, since the former enumerates all solutions either explicitly or implicitly to find the global optimum. On the other hand, the heuristic produces an approximate solution with a certificate of quality induced by the Lagrangian bound. An experiment that could be carried out consists in halting Cplex after running for the same amount of time that DGH runs, in which case Cplex might produce a good solution. Nevertheless, Cplex is a highly sophisticated software that consumes far more memory than DGH.

An ARM968E-S processor has the power efficiency of 0.00011 W/MHz [2]. Based on this fact, Cplex needs 0.794 seconds whereas DGH needs only 0.0152 seconds in the worst case to solve the reconfiguration problem for a system with 50 tasks on a processor running at $470MHz$, which consumes $0.0410498W$ for Cplex and $0.00078584W$ for DGH.

Because Cplex and DGH consume energy and CPU cycles, the design of the real-time system should consider reconfiguration as a system task and define the events that trigger reconfiguration. For example, if a robot receives a mission that takes an hour to be accomplished but with a battery that can last for only 40 min, the reconfiguration should be applied to degrade the quality of the tasks. Other events that can trigger system reconfiguration are the interruption of energy supply and the admission of new tasks.

6. CONCLUSION

In this paper, we have described a framework for dynamic reconfiguration of multi-mode real-time applications. The framework has been formalized as an optimization problem whose goal is to maximize application QoS subject to both schedulability and energy constraints. The energy consumption model was based on DPM to account for both on-chip and off-chip energy usage by the system.

Because the reconfiguration task generalizes the 2-constraint knapsack problem, which is NP-Hard, a heuristic was developed to find approximate solutions with low computational cost. The heuristic follows a greedy strategy to solve the associated surrogate problem defined by Lagrangian multipliers obtained with a subgradient algorithm.

The proposed framework and heuristic were implemented in C++ and applied to five real-time systems of varying size. The computational results showed that the heuristic is able to find high quality solutions approximating the optimum, which was computed with a top-notch mixed-integer linear-programming solver (Cplex). As observed in the experiments, the reconfiguration mechanism tends to choose operating modes that improve system benefit with the increase of energy availability. Indeed, the obtained results encourage the implementation of the proposed framework in a real-time embedded operating system, a research step to be considered in future work.

7. REFERENCES

[1] L. Abeni and G. Buttazzo. Resource reservation in dynamic real-time systems. *Real-Time Systems*, 27(2):123–167, 2004.

[2] ARM. Application notes and tutorials, Jun 2011.

[3] E. Bini and G. Buttazzo. Measuring the performance of schedulability tests. *Real-Time Systems*, 30(1):129–154, 2005.

[4] S. Boyd and L. Vandenberghe. *Convex Optimization.* Cambridge University Press, 2004.

[5] A. Burns, D. Prasad, A. Bondavalli, F. D. Giandomenico, K. Ramamritham, J. Stankovic, and L. Strigini. The meaning and role of value in scheduling flexible real-time systems. *Journal of Systems Architecture*, 46(4):305–325, 2000.

[6] G. Buttazzo. Research trends in real-time computing for embedded systems. *ACM SIGBED Review*, 3(3), 2006.

[7] E. Camponogara, A. B. de Oliveira, and G. Lima. Optimization-based dynamic reconfiguration of real-time schedulers with support for stochastic processor consumption. *IEEE Transactions on Industrial Informatics*, 6(4):594–609, Nov 2010.

[8] A. B. de Oliveira, E. Camponogara, and G. Lima. Dynamic reconfiguration in reservation-based scheduling: An optimization approach. In *Proceedings of the 15th IEEE Symposium on Real-Time and Embedded Technology and Applications*, pages 173–182, Apr 2009.

[9] M. L. Fisher. The Lagrangian relaxation method for solving integer programming problems. *Management Science*, 50(12):1861–1871, 2004.

[10] A. M. Geoffrion. Lagrangian relaxation an its uses in

integer programming. *Mathematical Programming Study*, 2:82–114, 1974.

[11] F. Glover. Surrogate constraints. *Operations Research*, 16(4):741–749, July-August 1968.

[12] Intel. Intel Technical Documents - Intel core 2 duo mobile processor, oct 2010.

[13] J. Jehuda and A. Israeli. Automated meta-control for adaptable real-time software. *Real-Time Systems*, 14(2):107–134, 1998.

[14] R. Jejurikar, C. Pereira, and R. Gupta. Leakage aware dynamic voltage scaling for real-time embedded systems. In *Proceedings of the 41st Annual Design Automation Conference*, pages 275–280, 2004.

[15] W. Kim, J. Kim, and S. Min. A dynamic voltage scaling algorithm for dynamic-priority hard real-time systems using slack time analysis. In *Proceedings of the Conference on Design, Automation and Test in Europe*, 2002.

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

[17] C. Lee, J. P. Lehoczky, R. Rajkumar, and D. Siewiorek. On quality of service optimization with discrete QoS options. In *Proceedings 5th IEEE Real-time Technology and Applications Symposium*, pages 276–286, 1999.

[18] G. Lima, E. Camponogara, and A. C. Sokolonski. Dynamic reconfiguration for adaptive multiversion real-time systems. In *Proceedings of the 20th IEEE Euromicro Conference on Real-Time Systems*, pages 115–124, 2008.

[19] S. Martello and P. Toth. An exact algorithm for the two-constraint 0–1 knapsack problem. *Operations Research*, 51(5):826–835, 2003.

[20] C. W. Mercer, S. Savage, and H. Tokuda. Processor capacity reserves for multimedia operating systems. Technical Report CMU-CS-93-157, Carnegie Mellon University, 1993.

[21] Micron. Technical Notes, oct 2010.

[22] P. Pillai and K. G. Shin. Real-time dynamic voltage scaling for low-power embedded operating systems. *SIGOPS Operating Systems Review*, 35:89–102, Oct 2001.

[23] N. Pinckney, T. Barr, M. Dayringer, M. McKnett, N. Jiang, C. Nygaard, D. M. Harris, J. Stanley, and B. Phillips. A MIPS R2000 implementation. In *Proceedings of the 45th Annual Design Automation Conference*, pages 102–107, 2008.

[24] Pintool. A dynamic binary instrumentation tool. http://www.pintool.org/, Jul 2011. Online; accessed July-2011.

[25] D. Prasad and A. Burns. A value-based scheduling approach for real-time autonomous vehicle control. *Robotica*, 18(3):273–279, 2000.

[26] G. Quan and X. Hu. Energy efficient fixed-priority scheduling for real-time systems on variable voltage processors. In *Proceedings of the Design Automation Conference*, pages 828–833, 2001.

[27] C. Rusu, R. Melhem, and D. Mossé. Multi-version scheduling in rechargeable energy-aware real-time systems. *Journal of Embedded Computing*, 1(2):271–283, 2005.

[28] C. A. Rusu, R. Melhem, and D. Mossé. Maximizing the system value while satisfying time and energy constraints. *IBM Journal of Research and Development*, 47(5-6):689–702, Sept 2003.

[29] Y. Shin, K. Choi, and T. Sakurai. Power optimization of real-time embedded systems on variable speed processors. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 365–368, 2000.

[30] L. A. Wolsey. *Integer Programming*. John Wiley & Sons, New York, NY, 1998.

[31] C. Xian, Y.-H. Lu, and Z. Li. Dynamic voltage scaling for multitasking real-time systems with uncertain execution time. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 27(8):1467–1478, Aug 2008.

[32] F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced CPU energy. In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science*, pages 374–382, Oct 1995.

[33] H. Yu, B. Veeravalli, and Y. Ha. Dynamic scheduling of imprecise-computation tasks in maximizing QoS under energy constraints for embedded systems. In *Proceedings of the Asia and South Pacific Design Automation Conference*, pages 452–455, March 2008.

[34] X. Zhong and C.-Z. Xu. Energy-aware modeling and scheduling for dynamic voltage scaling with statistical real-time guarantee. *IEEE Transactions on Computers*, 56(3):358–372, March 2007.

[35] D. Zhu, R. Melhem, and D. Mossé. The effects of energy management on reliability in real-time embedded systems. In *Proceedings of the International Conference on Computer-Aided Design*, pages 35–40, 2004.

[36] Y. Zhu and F. Mueller. Feedback EDF scheduling of real-time tasks exploiting dynamic voltage scaling. *Real-time Systems*, 31(1-3):33–63, 2005.