On the integration of application level and resource level QoS control for real-time applications

Tommaso Cucinotta*, Luigi Palopoli†, Luca Abeni†, Dario Faggioli*, Giuseppe Lipari*

* *Scuola Superiore Sant'Anna, Pisa, Italy*, {t.cucinotta, d.faggioli, g.lipari}@sssup.it

† *Università di Trento, Trento, Italy*, {luigi.palopoli, luca.abeni}@unitn.it

Index Terms

Real-Time Embedded Systems, Quality of Service Control, Power-Aware Optimisation

Abstract

We consider a dynamic set of soft real-time applications using a set of shared resources. Each application can execute in different modes, each one associated with a level of Quality of Service (QoS). Resources, in their turn, have different modes, each one with a speed and a power consumption, and are managed by a Reservation Based scheduler enabling a dynamic allocation of the fraction of resources (bandwidth) assigned to each application.

To cope with dynamic changes of the application, we advocate an adaptive resource allocation policy organised in two nested feedback loops. The internal loop operates on the scheduling parameter to obtain a resource allocation that meets the temporal constraints of the applications. The external loop operates on the QoS level of the applications and on the power level of the resources to strike a good trade-off between the global QoS and the energy consumption. This loop comes into play whenever the workload of the application exceeds the bounds that permit the internal loop to operate correctly, or whenever it decreases below a level that permit more aggressive choices for the QoS or substantial energy saving.

I. Introduction

In soft real-time applications timing constraints can occasionally be violated without causing a system failure. However, the performance of the system depends on the number of violated

constraints and on the severity of such violations. Therefore, one of the prominent goals for a soft real-time system is to keep under control the violation of timing constraints.

There are many examples of soft real-time applications both in the consumer electronics market and in the industrial application domain. Many of such applications involve video processing: for example, distributed video monitoring for video-surveillance and collection of sensible data (e.g., people counting, monitoring of parking lots, etc.). In industrial control, image recognition is increasingly used to identify objects on conveyor belts, or to find defects in products. Also, even control applications are known to be effectively implementable by soft real-time tasks [1], [2].

For all these applications, a static allocation of the computation resources to the various tasks is rarely appropriate for many reasons: it requires an expensive off-line analysis of the application requirements (e.g., execution times); it reduces the flexibility of the applications design and their robustness to unexpected changes of external conditions.

In this context, we advocate the use of adaptive techniques for the Quality of Service (QoS) management of soft real-time applications. The QoS can be evaluated considering three different dimensions. The first one is the macroscopic QoS perceived by the user. For instance, in a streaming application the QoS can be related to such parameters as the resolution, the frame rate etc. The second dimension is the compliance with the timing constraints. Finally, a third perspective on the system QoS is given by energy consumption. As an example, if the application is run on a portable device, the duration of the battery is directly perceived as a quality indicator. Correspondingly, we can classify adaptive techniques in three groups: *application level* adaptation, in which the application operating modes are adapted to the availability of resources; *resource allocation level* adaptation, in which the resource shares granted to the applications are adapted to the dynamic workload requirements; and *resource power level* adaptation, in which the resource speed (and the corresponding power consumption) is adapted to the requirements of the system.

Adaptation at the resource allocation level can be an effective solution in case of short overload situations. In this case, the detrimental effects on the QoS when the system is overloaded can be controlled by the adaptive scheduler, exploiting the inherent robustness of soft real-time applications to timing faults. However, if the overload persists (due to a structural change), then the problem cannot be solved at the scheduling level and it may be necessary to change the

application mode in order to reduce the workload. On the other hand, if changes in the application level are too frequent, the perceived QoS can be very low. Therefore, application level adaptation cannot be used to control the timing behaviour of an application at each activation of its tasks.

These simple arguments suggest the potential improvement that we can achieve by the combined application of resource level and application level QoS control. However, this combination is far from trivial: the adaptation on the application side has to be appropriately *coordinated* with the adaptation on the resource side, in order to maximise their effectiveness.

Therefore, in this paper we present a coherent software architecture that 1) is soundly rooted in control theory; 2) implements resource-level and application-level adaptation by using a coherent two-level control loop strategy; 3) allows the user to customise the desired QoS metric, and tune the application parameters to maximise the desired metric; 4) is available on a widely used operating system platform.

The advantages of our software architecture are demonstrated on a real application, consisting of an adaptive video encoder, which is capable of dynamically scaling the quality of the video and the corresponding computing requirements.

The paper is organised as follows. In Section II, we quickly report on the state of the art in the field. In Section III, we describe our approach putting the stress on the idea of a double feedback loop. In Section IV, we describe the Linux based software architecture that we have designed to implement the idea and in Section V we report the obtained experimental results. Finally, in Section VI, we offer our conclusions and announce future work directions.

II. Related Work

The application of feedback control techniques to control the evolution of real-time applications has become popular in the research community. The basic idea is to compensate for the fluctuation in the workload generated by the different applications by operating on appropriate *actuators*. The proposed approaches take a different direction according to the type of actuators used and to the way the behaviour of the application is observed. Very roughly, we can classify them in three different areas: 1) application level adaptation, 2) scheduling level adaptation, 3) power level adaptation.

The idea underlying application level adaptation is to operate on a set of parameters exposed by the application in order to increase or decrease its generated workload. This viewpoint is

championed by several researchers. A recent and important example is in the work of Wust et al. [3], a QoS optimisation framework based on a Markov Decision Problem (MDP) approach, with similar goals to the ones considered in this paper. Our work differs in two respects. The first difference is that we aim at supporting QoS in open systems where a MDP is hard to set up because of the limited prior knowledge on the applications. The second one, is that we consider applications for which frequent changes in the operating mode produce annoying effects and should be avoided solving the problem at the scheduling level whenever possible.

The viewpoint taken by scheduling level adaptation is somewhat dual: the QoS is controlled by operating on the scheduling parameters, leaving the application unaware of the adaptation. An approach of this kind is the so called real-rate scheduling developed by Steere and others [4]. The progress of the application is compared with the ideal rate and corrective actions are taken as required. In the real-time community a very significant work has been carried out by Stankovic and others [5], who measure for each task the deadline miss ratio and operate on the deadlines used by an EDF scheduler. In the absence of a precise dynamic model of the scheduler, this approach is to be classified as a heuristic solution. In contrast, Abeni and others [6] propose the use of an adaptive scheme based on the resource reservations schedulers [7], which is one of the cornerstones of the work presented in this paper. The use of a resource reservation scheduler enables a precise description of the evolution of the system and hence a well founded design [8]. A major limitation of adaptive scheduling is its inability to operate when the system is in persistent overload conditions, an issue we address in this paper by mixing application level adaptation with adaptive scheduling.

As far as power adaptation is concerned, most of the work done in the area is based on the well-known dependency of the power consumption of a CPU (using CMOS technology) from the voltage level used for power supply and the operating frequency. When the workload on a CPU is low, we can lower the voltage and hence the frequency. Because there is a quadratic relation between power and voltage, this can lead to remarkable energy savings. Pillai and Shin [9] in a seminal work propose a power aware real-time scheduling technique. A similar research direction is taken by Aydin et al. [10]. However, in our case, we cannot rely on a prior characterisation of the workload and we are interested in adaptively striking a good compromise between QoS and power rather than respecting every single deadline. As a consequence, we perform power adaptation on longer time scales rather than aggressively changing it upon each job execution.

Closer to our approach is the idea of Simunic and others [11]. The authors estimate on-line the arrival and the service rate of frames in a MPEG streaming application (modelled as a M/M/1 queue) and adjust the CPU voltage and frequency accordingly. Adaptive schemes in which power consumption is explicitly considered along with temporal guarantees have been proposed by Qu et al. [12] and in the GRACE-OS [13] architecture, proposed by Yuan and Nahrstedt. Our work differs in many respects. First, power consumption is only one of our optimisation criteria, and it can be traded off with the macroscopic QoS level of the applications. Second, our power adaptation mechanism does not take place at the scheduling level, but it is carried out along with adjustment in the application modes when the system is reconfigured.

A very general framework for QoS optimisation is QRAM [14]. With QRAM the dimensions along which the QoS is evaluated can be multiple and include power consumption. The framework was initially applied offline but recently it has been extended in order to be applicable on-line [15]. Differently from QRAM, we are not proposing a general framework for QoS definition and optimisation. Our main focus is rather on adaptive and combined control of QoS and temporal behaviour. The two aspects are taken care of by two control loops: the external one operating on the QoS level and the internal one operating a fine grained control of the temporal behaviour of the applications. In principle, QRAM could be used to specify the optimisation problem operated by the external loop, along the line suggested by Lee et al. [16].

An approach similar to the one presented in this paper has been proposed by Abeni et al. [17], where an inner controller performs on-line adaptation of the requested bandwidth, and an outer controller performs a slower on-line adaptation of the application QoS level by modulating the task activation rate. However, the approach presented here is more general, in that we can operate on multiple dimensions to change the QoS of the applications and hence the generated workload (and not simply on the rate), and we seek optimum QoS/power-consumption trade-offs in the global system reconfiguration. Another interesting approach recently proposed by Romero et al. [18] uses global ILP optimization for setting application modes and allocating tasks on multiple processors, and adaptive reservations for controlling the deadline-miss ratio of individual tasks. However, power management is not considered in the model.

In order to solve the optimisation problem associated with the external loop with an acceptable cost we use a heuristic. Similar solutions can be found in the work of several researchers. For instance, Rusu et al. [19], propose a heuristic to solve an Integer Linear Program (ILP) that

identifies the optimal configuration of a system with multi-mode periodic real-time applications under deadline and energy constraints. Chen and Kuo also propose [20] various formulations, comprising ILP, and propose heuristic solvers, focusing on the theoretical complexity of the solution strategy. In both cases, the problem addressed by the authors bears some resemblance with the one associated with our external loop. There are, however, significant differences, the most important being that the authors mainly concentrate on a time-slotted setting where tasks share the same period (an extension to multiple periods is hinted to considering the hyper-period). In our case, the use of a Reservation Based (RB) scheduler induces important differences in the way the problem is formulated. Other differences are in the cost function (which in our case has a term penalising power consumption) and in the fact that in the cited paper the authors assume the possibility of choosing a different frequency for each task (whereas we periodically set the frequency of the CPU to be used for all tasks in the next period). These differences in the model prevent the direct application of the heuristics proposed in the cited papers in our framework, although some of the ideas have been adapted and experimented on (as detailed below).

Other authors propose architectural support for multi-level feedback. This is the case of the HOLA-QOS architecture [21] proposed by Valls et al. and of the work done by Kalogeraki and others [22]. Generally speaking, devising middle-ware solutions for QoS management has been an active research area in the last few years. One significant sample in this wide body of literature is the work by Brandt and co-workers [23], in which a middle-ware is used to support application level adaptation. Other relevant middle-ware proposals are the ones by Schmidt et al. [24], and by Zhang et al. [25]. In these cases, there is a real-time scheduling support for the QoS management, which is part of our approach. More importantly, in these proposals, the authors offer a generic support for QoS adaptation but they do not commit to any specific algorithmic solution. In this work, although we propose a middle-ware architecture ourselves and we place a major emphasis on the “internals” of the algorithms for QoS management.

Finally, this work extends and subsumes preliminary results we presented in [26].

III. Proposed Approach

A. Dual-Loop Control Scheme Overview

The two-level feedback-based QoS control scheme proposed in this paper is sketched out in Figure 1. The internal control loop is operated by a *Resource Allocation Controller*. At the end

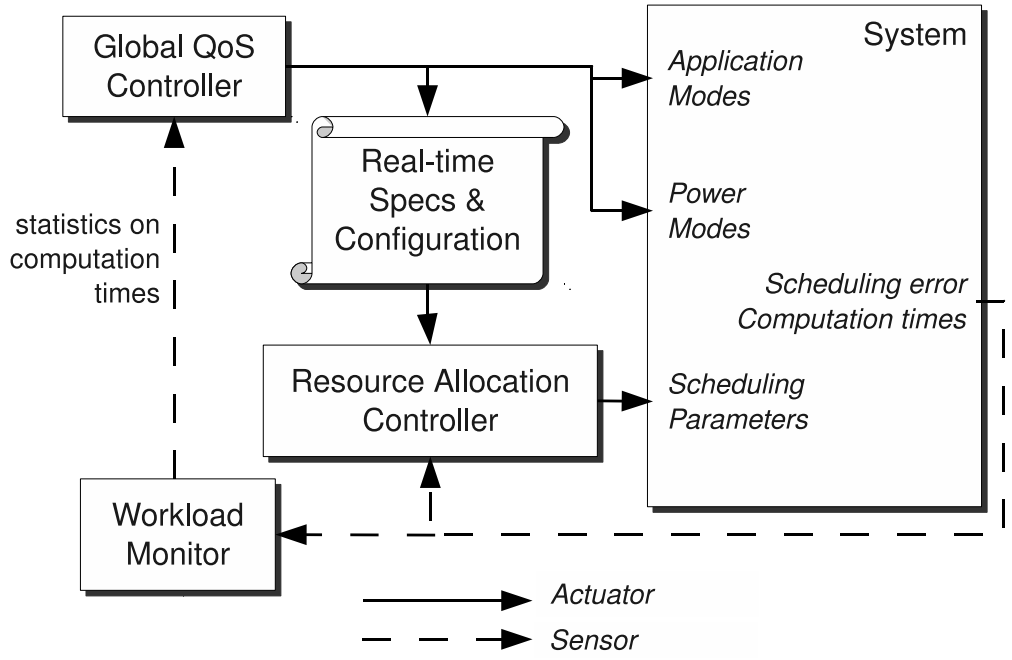


Figure 1. Two-level control loop.

of each job, the controller receives a measurement of the *scheduling error*, a QoS metrics which will be introduced later in III-B2, and of the computation time of the previous job. The controller then decides the fraction of resources (bandwidth) to be used for the next job and actuates this decision by properly changing the *scheduling parameters*. The objective of this control is to have an evolution of the QoS metric complying with the stochastic real-time specification set by the external loop. This goal can be achieved only if some assumptions on the workload generated by the tasks are respected. The *Workload Monitor* records the computation time of the jobs and evaluates the workload generated by the task over a time interval.

An external control loop, operated by a *Global QoS Controller*, provides the application-level and power-level adaptation. It periodically prompts the Workload Monitor to evaluate if the working assumptions for the resource controller are violated. If this is the case, it switches the mode of the applications and the power mode of the resources to change the resource requirements of the tasks. Similarly, if a reduction in the resource requirements is detected, the QoS controller can make more aggressive choices in the application modes increasing the “macroscopic” QoS perceived by the users. The Global QoS Controller chooses at all times a

configuration that maximises a system-wide *QoS index*, achieving the desired trade-offs between QoS of applications and power consumption. When some of the application modes or of the power modes are changed, the QoS controller resets the real-time specifications for the resource controllers adapting them to the new mode and resets the controller.

More details about the two control loops follow in Sections III-C and III-D, after a brief introduction of the necessary concepts and notational elements.

B. Background and Notation

Our system consists of a set of n applications $\mathcal{A}^{(1)}, \ldots, \mathcal{A}^{(n)}$, sharing a pool of m resources $\mathcal{R}^{(1)}, \ldots, \mathcal{R}^{(m)}$. Although in the remainder of the paper we will concentrate on processors, the presented approach may be equally effective in managing other types of resources. In fact, the approach is applicable whenever the resource can be allocated preemptively or with a fine granularity for non-preemptive sections. A network link for which the packet size used for transmission is much smaller than the one used by the application is a good example. Applications can dynamically enter and leave the system.

1) Real-Time Task Model: Each application $\mathcal{A}^{(i)}$ is composed of one or more tasks, each one deployed on a processor. For the sake of simplicity, we assume that in all applications there is at most one task using each CPU: let $\tau^{(i,r)}$ denote the task belonging to $\mathcal{A}^{(i)}$ that uses $\mathcal{R}^{(r)}$, and A_r the set of applications with tasks on $\mathcal{R}^{(r)}$.

A task $\tau^{(i,r)}$ consists of a sequence of jobs, or instances, $J_k^{(i,r)}$. Each job $J_k^{(i,r)}$ arrives (becomes executable) at time $r_k^{(i,r)}$, and finishes at time $f_k^{(i,r)}$ after using $\mathcal{R}^{(r)}$ for a time $c_k^{(i,r)}$. Job $J_k^{(i,r)}$ is associated with a deadline $d_k^{(i,r)}$, which is *met* if $f_k^{(i,r)} \leq d_k^{(i,r)}$ and is *missed* otherwise. In this paper, we focus on *periodically* activated tasks with period $T^{(i)}$ and *relative deadline* equal to the period: $r_{k+1}^{(i,r)} = r_k^{(i,r)} + T^{(i)}$ and $d_k^{(i,r)} = r_k^{(i,r)} + T^{(i)} = r_{k+1}^{(i,r)}$. However, our software architecture supports also aperiodic tasks (see Section IV).

2) Scheduling: Each processor is shared between multiple tasks by using a Reservation Based (RB) scheduling policy. In a multiprocessor setting, we assume tasks are statically partitioned on the processors. In a RB framework, a task $\tau^{(i,r)}$ deployed on a CPU $\mathcal{R}^{(r)}$ is associated a pair $(Q^{(i,r)}, P^{(i,r)})$, said *reservation*, meaning that the scheduling algorithm guarantees to $\tau^{(i,r)}$ a *budget* of $Q^{(i,r)}$ execution time units of the CPU in every *reservation period* $P^{(i,r)}$, whenever in need. The ratio $B^{(i,r)} = Q^{(i,r)}/P^{(i,r)}$ is referred to as *reserved bandwidth* and quantifies

the fraction of the CPU reserved to the task. In our framework, the reserved bandwidth can be dynamically changed by changing the reserved budget $Q^{(i,r)}$ for each job. The reservation period $P^{(i,r)}$ is strictly tied to the task period $T^{(i)}$ (generally equal to the period of the task, or to one of its sub-multiples). This choice maximises the efficiency in resource allocation [27] and allows us to construct a dynamic model for the evolution of the task. As a result, the reservation period only changes when the application period is changed (e.g., for a change in the application mode). The symbol $B_k^{(i,r)}$ will denote the bandwidth allocated to $J_k^{(i,r)}$.

To analyse the timing behaviour of the tasks, it is convenient to introduce, for each job $J_k^{(i,r)}$, the *latest possible finishing time* (LPFT), defined as the end of last reservation period in which the job can finish. As an example, if a task is scheduled through a reservation $(Q^{(i,r)}, P^{(i,r)}) = (1,\, 3)$ and the job starts at 0 with $c_k^{(i,r)} = 3$, the LPFT is 9. Indeed, it takes three reservation periods for the task to complete. This quantity is an upper bound for the finishing time of the job, which is attained if: 1) the job does not receive computation time other than the reserved one, 2) in the last reservation the budget is received right before the deadline. Based on this quantity it is possible to introduce the *scheduling error* $\epsilon_k^{(i,r)}$, defined as the difference between the LPFT and the job deadline (see [6] for details). This quantity can be used to quantify the precision in resource allocation. Indeed, if the scheduling error is negative, then the job received more CPU time than strictly required. Conversely, if the scheduling error is positive, then the job received too little. The scheduling error as just defined has been used in various papers about adaptive real-time scheduling, in which feedback-based control loops control its evolution in a range as close as possible to zero, ranging from linear controllers [6] to non-linear [28] and stochastic based ones [8], [29]. The controller proposed in this paper (see Section III-C) falls in this latter class.

In order for a RB scheduler to work properly, the following relation has to be respected at all times:

$$\forall r \in 1,\ \ldots,\ m \sum_{i \in A_r} B^{(i,r)} \leq U_{lub}^{(r)}, \tag{1}$$

where $U_{lub}^{(r)} \leq 1$ depends on the scheduling algorithm.

3) System Configuration: Each application $\mathcal{A}^{(i)}$ can execute in a mode chosen in a finite set $V^{(i)} \triangleq \{1, \ldots, n_{am}^{(i)}\}$ of cardinality $n_{am}^{(i)}$. Every mode $j \in V^{(i)}$ is associated with a *QoS rate* $q^{(i,j)} \in \mathbb{R}$, which is a measure of the instantaneous user satisfaction when $\mathcal{A}^{(i)}$ executes in the

j-th mode.

Each resource $\mathcal{R}^{(r)}$ may vary its power mode within a finite set $P^{(r)} \triangleq \{1, \ldots, n_{rm}^{(r)}\}$ of cardinality $n_{rm}^{(r)}$ (e.g., by varying the clock frequency and voltage for a CPU). In each mode k, the resource $\mathcal{R}^{(r)}$ has a power consumption $p^{(r,k)}$. Modes associated with a higher power consumption reduce the execution time of the tasks. The optimisation framework presented in this paper is very general and supports arbitrary specification of resource requirements for each available power-mode, as it will be clarified in Section III-D.

In the following, for the purpose of clarity, whenever the discussion refers to a single task and/or resource, the corresponding superscript (i) and/or (r) is omitted.

C. The Resource Allocation Controller

The goal of the resource-level control loop attached to each task is specified in terms of the probability $\pi^{(i,r)}$ that the scheduling error $\epsilon^{(i,r)}$ of the task respects an upper bound $\delta^{(i,r)}$. More formally:

$$\Pr\left\{\epsilon_k^{(i,r)} \leq \delta^{(i,r)}\right\} \geq \pi^{(i,r)}. \tag{2}$$

This goal is achieved by a control scheme (presented in a preliminary version in [29]) consisting of two basic elements: 1) a set of local controllers associated with each task (*task controllers*), and 2) a set of resource supervisors associated with each processor. The purpose of the task controller is to formulate a minimum bandwidth request $\overline{B}_k^{(i,r)}$ that allows the task to respect its timing constraints with probability $\pi^{(i,r)}$. Because the controllers have only a local visibility, they could formulate bandwidth requests exceeding $U_{lub}^{(r)}$ in Condition (1). The resource supervisor in this case can change the value $B_k^{(i,r)}$ of the bandwidth granted to the application so that the condition is respected. The conceptual link between the two components is a minimum bandwidth $B_G^{(i,r)}$ that has to be granted to job $J_k^{(i,r)}$, whenever the task controller formulates a request $\overline{B}_k^{(i,r)} \geq B_G^{(i,r)}$. Clearly, to respect the schedulability constraint in Equation (1), the minimum guaranteed bandwidths have to respect it in turn:

$$\forall r \in \{1, \ \ldots, \ m\}, \sum_{i \in A_r} B_G^{(i,r)} \leq U_{lub}^{(r)}. \tag{3}$$

Indeed, in the worst case each task $\tau^{(i,r)}$ is granted only its minimum guaranteed bandwidth $B_G^{(i,r)}$.

1) Task controllers: Considering a single task, we can approximate the evolution of the scheduling error as follows:

$$\epsilon_{k+1}^{(i,r)} = S(\epsilon_k^{(i,r)}) + \frac{c_{k+1}^{(i,r)}}{B_{k+1}^{(i,r)}} - T^{(i)} \tag{4}$$

where $S(x) = x$ if $x > 0$, and $S(x) = 0$ if $x \leq 0$. Intuitively, if in the previous job the task received an amount of bandwidth greater than or equal to its need (negative scheduling error), there is not any execution backlog on the new job. In this case, the LPFT can be computed starting from the job activation instant. Otherwise, it has to be computed starting from the LPFT of the previous job (hence the function $S(.)$), and the ratio $\frac{c_{k+1}^{(i,r)}}{B_{k+1}^{(i,r)}}$ approximates the number of required reservations.

In order to find a feedback control law that achieves Condition (2), the sequence of computation times $c_k^{(i,r)}$ and of observed scheduling errors $\epsilon_k^{(i,r)}$ are considered as discrete-time, continuous-valued, stochastic processes, related by Equation (4). The problem is to find a function relating $B_{k+1}^{(i,r)}$ to $\epsilon_k^{(i,r)}$ such that Equation 2 is respected.

A reasonable approximation of this ideal design goal is as follows. If the scheduling error is in an *attractivity region* $\mathcal{R}^{(i,r)} = [-T^{(i)}, R^{(i,r)}]$ enclosing the target region $\mathcal{R}_T^{(i,r)} = [-T^{(i)}, \delta^{(i,r)}]$, then it is steered back to $\mathcal{R}_T^{(i,r)}$ with at least the probability $\pi^{(i,r)}$:

$$\Pr\left\{\epsilon_{k+1}^{(i,r)} \in \mathcal{R}_T^{(i,r)} \mid \epsilon_k^{(i,r)} \in \mathcal{R}^{(i,r)}\right\} \geq \pi^{(i,r)}. \tag{5}$$

Informally speaking, the quantity $R^{(i,r)} \geq \delta^{(i,r)}$ represents the maximum value that the scheduling error can take for which the controller is able to control it in the desired set at the next step with a high probability.

As discussed in [29, Proposition 1], assuming that an additional component – *the predictor* – provides the controller with a quantity $H_{k+1}^{(i,r)}$ such that $\Pr\left\{c_{k+1}^{(i,r)} \leq H_{k+1}^{(i,r)}\right\} \geq \pi^{(i,r)}$, and that the minimum bandwidth $B_G^{(i,r)}$ guaranteed to the task satisfies

$$B_G^{(i,r)} \geq \sup_{k \in \mathbb{N}} \frac{H_k^{(i,r)}}{T^{(i)} + \delta^{(i,r)} - R^{(i,r)}}, \tag{6}$$

then Condition (5) is fulfilled by the following control law:

$$\overline{B}_{k+1}^{(i,r)} = \frac{H_{k+1}^{(i,r)}}{T^{(i)} + \delta^{(i,r)} - S(\epsilon_k^{(i,r)})}. \tag{7}$$

The quantity $H_{k+1}^{(i,r)}$ represents a guess of an "upper bound" for the computation time $c_{k+1}^{(i,r)}$. Clearly, choosing a tight bound generally decreases the probability $\pi^{(i,r)}$ of making a correct

guess. The $\sup$ in Equation (6) has to be evaluated over all the (possibly infinite) sequence of jobs. For all practical purposes, this quantity can be retrieved from previous executions of the application or estimated on a trial execution of a sufficient number of jobs.

This result, given a desired value for the $R^{(i,r)} = \inf R_k^{(i,r)}$ and given the maximum resource requirement estimated by the predictor, allows us to identify a minimum bandwidth requirement $B_G^{(i,r)}$ that attains the goal. We can extend the control law proposed above by saturating it to $B_G^{(i,r)}$ for values of the scheduling error greater than $R^{(i,r)}$:

$$\overline{B}^{(i,r)}(\epsilon_k^{(i,r)}) = \begin{cases} \frac{H_{k+1}^{(i,r)}}{T^{(i)}+\delta^{(i,r)}-S(\epsilon_k^{(i,r)})}, & \text{for } \epsilon_k^{(i,r)} \leq R^{(i,r)} \\ B_G^{(i,r)} & \text{otherwise.} \end{cases} \tag{8}$$

As discussed in [29, Proposition 2], if $B_G^{(i,r)}$ dominates the moving average of the computation times over a given time horizon, then this control law also ensures that, if the scheduling error leaves the attractivity region, it returns to it in a finite number of steps and its maximum value can be bounded.

2) Predictor: Many algorithms are available for time series predictions [30]. Generally, they rely on an analysis of the past observed $c_k^{(i,r)}$ samples, where both simple solutions like moving averages, or complex ones based on optimal filtering theory, are possible. A better performance can be obtained by leveraging off the knowledge on the domain of the application (as suggested by Pohlack et al. [31] for MPEG decoding). Any predictor that can be characterised in terms of a probability $\pi^{(i,r)}$ satisfying Equation (2) can fit in our framework.

3) Resource Supervisor: Each resource $\mathcal{R}^{(r)}$ is attached a *supervisor* that comes into play whenever the set of requested reservations violate Equation (1). In such a case, the supervisor must guarantee to each task its minimum bandwidth $B_G^{(i,r)}$. We cope with this problem by using a compression function [32]. Let $\left\{\overline{B}^{(i,r)}\right\}$ denote the bandwidths required by the task controllers at some time t. If $\sum_{i \in T_r} \overline{B}^{(i,r)} \leq U_{lub}^{(r)}$, then the granted bandwidths $B^{(i,r)}$ are simply set equal to the required values: $\forall i \in T_r$, $B^{(i,r)} = \overline{B}^{(i,r)}$. Otherwise, the granted bandwidths $\left\{B^{(i,r)}\right\}$ are set as follows: $\forall i \in T_r$,

$$B^{(i,r)} = B_m^{(i,r)} + \left(U_{lub}^{(r)} - \sum_{j \in T_r} B_m^{(j,r)}\right) \frac{\overline{B}^{(i,r)} - B_m^{(i,r)}}{\sum_{j \in T_r}\left(\overline{B}^{(j,r)} - B_m^{(j,r)}\right)}$$

where $B_m^{(i,r)} \triangleq \min\left\{B_G^{(i,r)}, \overline{B}^{(i,r)}\right\}$.

D. The Global QoS Controller

1) Macroscopic QoS: The QoS, in our setting, can be evaluated using different metrics. Namely, for every application $\mathcal{A}^{(i)}$ and for every possible mode j, we define an instantaneous *QoS rate* $q^{(i,j)}$. If this rate is maintained for a time interval Δt (the "sampling" period of the external loop, from here on referred to as *optimisation period*), then the accumulated QoS in the interval is given by $q^{(i,j)}\Delta t$. For certain types of applications, changing the application mode too often may diminish the QoS experienced by the user. For example, while playing a video stream, changing too frequently the bit-rate can be very annoying. To model this detrimental effect, when going from mode (j) to mode (k), we introduce a negative QoS metric given by $-k_a^{(i)}|q^{(i,k)} - q^{(i,j)}|$, with $k_a^{(i)} > 0$, which penalises switches in the application mode. This term is not multiplied by the time interval duration, thus its impact is lower at higher values of the optimisation period Δt.

Another dimension for evaluating the QoS is *energy consumption*. The energy spent over the sampling interval Δt by resource $\mathcal{R}^{(r)}$ operating in mode k is given by $p^{(r,k)}\Delta t$. This metric has to be accounted for with a negative sign. If appropriate for the specific architecture, it is possible to introduce a metric quantifying the energy spent in a transition between two different power modes (j) and (k): $-k_p^{(r)(j,k)}$. In addition to associating energy consumption with a QoS metric, we support also constraints on the maximum consumed power, e.g., dictated by a desired lifetime for battery operated devices.

These different metrics are clearly in trade-off. For instance, reducing the power mode of a resource is an advantage for the energy consumption but it also increases the computation times of the different tasks generating potential overload conditions that have to be solved reducing quality of the applications. The problem is a multi-objective optimisation, and we solve it by a simple technique known as "scalarisation" [33]. The utility function is built as a weighted linear combination of scalar terms, each one representing a different possible optimisation dimension (see Equation (9) below).

2) Optimisation Problem Set-up: The global QoS controller performs an optimisation whose decision variables are the modes of the applications and of the processors. Because these levels are discrete, we can conveniently set up the problem as a Boolean linear program (BLP). To this end, we introduce a vector of Boolean variables $\mathbf{x}^{(i)} = [x^{(i,1)}, \ldots, x^{(i,n_{am}^{(i)})}]$, where $x^{(i,j)}$ is 1

if the mode j is selected for application $\mathcal{A}^{(i)}$, 0 otherwise. The vector $\tilde{\mathbf{x}}^{(i)}$ denotes the current configuration of $\mathcal{A}^{(i)}$ (as computed by the controller at the previous optimisation). Similarly, we introduce for each resource $\mathcal{R}^{(r)}$ the vector of Boolean variables $\mathbf{y}^{(r)} = [y^{(r,1)}, \ldots, y^{(r,n_{rm}^{(r)})}]$, where $y^{(r,j)}$ is 1 if the mode j is selected for $\mathcal{R}^{(r)}$, and the vector $\tilde{\mathbf{y}}^{(r)}$ to denote the current configuration of $\mathcal{R}^{(r)}$.

We introduce a vector notation also for the QoS levels $\mathbf{q}^{(i)} = [q^{(i,1)}, \ldots, q^{(i,n_{am}^{(i)})}]$ and the power consumptions $\mathbf{p}^{(r)} = [p^{(r,1)}, \ldots, p^{(r,n_{rm}^{(r)})}]$. Also, we introduce the matrix $\mathbf{K}_p^{(r)} = [k_p^{(r)(j,k)}]$ of the power-switching penalties. Finally, for each application $\mathcal{A}^{(i)}$ and resource $\mathcal{R}^{(r)}$, we introduce the matrix $\mathbf{B}_G^{(i,r)} \triangleq [B_G^{(i,r)(j,k)}]_{j,k}$ containing on each row j the requirements of the application mode j for the various resource modes, and on each column k the requirements of the resource mode k for the various application modes. More precisely, the components of $\mathbf{B}_G^{(i,r)}$ represent the minimum guaranteed bandwidths required by task controllers to sustain the performance specification in Equation (5). This quantity can be computed, given an an estimate of $\sup_k \left\{ H_k^{(i,r)} \right\}$, by using Equation (6). Such estimates may be based on information acquired upon each optimisation period from the Resource Controller (see Figure 1), thus they may be time-varying. Concerning the resource requirements related to configurations of the system that are not currently in use, we assume that they can be estimated, based on the values measured on the current configuration, by application-dependent interpolators (the so called *multi-mode predictors*). In the next section, we will provide details for a specific application domain.

3) Formal Problem Statement: Now, we are in condition to formalise the problem as:

$$\max_{\mathbf{x}^{(i)}, \mathbf{y}^{(r)}} \sum_{i=1}^{n} w_a^{(i)} \left(\Delta T \mathbf{q}^{(i)} \cdot \mathbf{x}^{(i)} - k_a^{(i)} \left| \mathbf{q}^{(i)} \cdot \mathbf{x}^{(i)} - \mathbf{q}^{(i)} \cdot \tilde{\mathbf{x}}^{(i)} \right| \right) - \sum_{r=1}^{m} w_p^{(r)} \left(\Delta T \mathbf{p}^{(r)} \cdot \mathbf{y}^{(r)} + \tilde{\mathbf{y}}^{(r)T} \mathbf{K}_p^{(r)(j,k)} \mathbf{y}^{(r)} \right) \tag{9}$$

where "$\cdot$" denotes the scalar product, "$.^T$" denotes the matrix/vector transposition, and the weights $\{w_a^{(i)}\}$ and $\{w_p^{(r)}\}$ are positive real numbers that configure the relative emphasis of applications QoS and power consumption in the search on the Pareto front. The maximisation is subject to the constraints:

$$\sum_{r=1}^{m} \mathbf{p}^{(r)} \cdot \mathbf{y}^{(r)} \leq P$$

$$\sum_{i=1}^{n} \mathbf{x}^{(r)T} \mathbf{B}_G^{(i,r)} \mathbf{y}^{(i)} \leq U_{lub}^{(r)}, \;\; r = 1, \ldots, m$$

$$\sum_{j} x^{(i,j)} = 1, \; i = 1, \ldots, n \quad x^{(i,j)} \in \{0, 1\}, \; \forall i, j$$

$$\sum_{j} y^{(r,j)} = 1, \; r = 1, \ldots, m \quad y^{(r,j)} \in \{0, 1\}, \; \forall r, j$$

where the first constraint limits the maximum instantaneous power consumption P sustainable by the system, while the next constraints represent the consistency conditions in Equation (1), for the various possible configurations. The quantity P could in principle be time-varying to accommodate the possible changes in the current level of the battery or be chosen to obtain a desired lifetime for the system. The problem has bilinear constraints and absolute values in the objective function. However, it can be transformed in a standard BLP problem through simple transformations.

In our framework, each application is associated an additional fictitious switching mode corresponding to a null resource requirement (for each resource power mode) and to a null QoS rate. This way, the presented BLP program has always a feasible solution. If the optimiser selects the fictitious mode for an application $\mathcal{A}^{(i)}$, then: if $\mathcal{A}^{(i)}$ is requiring admission into the system, then the request is turned down; if $\mathcal{A}^{(i)}$ was admitted previously, then it is dynamically dismissed.

4) Solving the optimisation problem: The exact solution of a BLP has generally an exponential complexity in the number of decision variables. Because our approach requires that the optimisation problem be solved online (to close the external loop), it is mandatory to use some kind of heuristic simplifying its solution. The proposed framework does not specify any particular heuristic, and is open to the adoption of any heuristic, as long as it produces feasible solutions. As an example we implemented some simple heuristics, featuring different performance/overhead tradeoffs depending on the class of problem they are applied to.

The simplest heuristic that we have developed was inspired by the work of Abdelzaher et. al. [34]. It is a greedy heuristic, quadratic in the number of tasks and resources and linear in the number of levels. The algorithm is readily described:

1) set the current configuration with application modes to minimum QoS and resource modes to maximum power;
2) if the current configuration is feasible, then proceed to step 3, otherwise if the power constraint is violated, then:
    a) compute the changes of the objective function due to changing each resource mode to the next available one; only changes keeping feasibility of the non-power constraints are considered; if no such changes exist, then exit with error;
    b) update the current configuration with the change that minimises the overall power consumption;
    c) repeat from step 2;
3) compute the changes of the objective function due to changing each application or resource mode to the next one; only changes that do not violate constraints are considered; if no such changes exist, then exit and return the current configuration;
4) update the current configuration with the change in either the QoS mode of an application, or the power mode of a resource that maximises the objective function increment;
5) repeat from step 3.

From the 3-rd step, the execution can be interrupted at any iteration with a feasible solution. Therefore, it is possible to consider different trade-offs between solution time and accuracy. This heuristic is defined Greedy Cost (GC). Being a greedy heuristic, the algorithm is clearly subject to the issue of *local optima*. To overcome the problem we have investigated two different approaches. The first one is inspired from the work of Rusu et al. [19]. The idea is that if at each step we simply make the choice that maximises the cost function, we could saturate the resource constraints preventing subsequent improvements. Therefore, we can amend step 2a by maximising the ratio between the QoS improvement achieved by a choice and the norm of the vector of increased resource utilisation incurred by the choice. This algorithm is referred to as Greedy Cost/Utilisation (GCU) heuristic.

An alternative idea entails a more substantial change. In steps 2 and 3 of the above algorithm, instead of considering merely the solution change leading to the maximum power consumption decrement or objective function increment, we considered the K changes leading to the K best moves, producing a *set of current configurations*. Then, we proceeded by carrying on the heuristic

"in parallel" from step 2 on all the current configurations. At each repetition, the size of the set of the currently considered solutions increases by K (however a lower number is generally kept, due to unfeasible and duplicate solutions), then at most K out of them are selected for proceeding to the next step. This heuristic is called multi-path search (MPS) heuristic.

The two ideas can be combined, producing the Multi-path Search with Cost/Utilisation (MPSCU), in which the search is carried out on multiple paths and the increments of the cost function are weighted against the introduced utilisation.

a) Examples: To the simple purpose of illustrating the practical applicability of the heuristics implemented in the framework, we propose here some examples of the overhead/performance tradeoffs that we obtained on several test cases.

A test case corresponds to a different choice of values for the following parameters: number of applications, number of resources, number of resource modes and number of application modes. For each test case, we selected 100 different optimisation problems. This selection was made randomly, ensuring the correctness of the problem (a higher QoS has to be associated with a higher execution time, and a smaller power consumption has to be associated with a higher computation time). For each problem instance, we applied the different heuristics and found the exact solution by using the GNU Linear Programming Toolkit (GLPK) API[1]. For each test case and for each solution algorithm, we recorded the average of optimal value and execution time for the different problems, obtained on an Intel Core 2 Duo P9600 CPU at 2.66 GHz (only 1 CPU was actually used by the solver).

The results are depicted in Figure 2. Each curve represents the performance of the different heuristics for one of the test cases on a QoS value/computation overhead plane. For all the test cases we considered 2 resources, 3 application modes and 3 power modes. The difference between the three test cases lies in the number of applications: 4 for the test case associated with the curve at the bottom, 8 for the one associated with the intermediate curve and 12 for the curve on the top of the plot. The vertical offset between the curves is clearly due to the increased number of applications, which corresponds to a larger accumulated value for the QoS.

A first comment on the results is that an exact solution with GLPK is not affordable in our setting (it requires a computation time in the order of a fraction of seconds even for small sized

[1] More information is available at the URL: `http://www.gnu.org/software/glpk`.

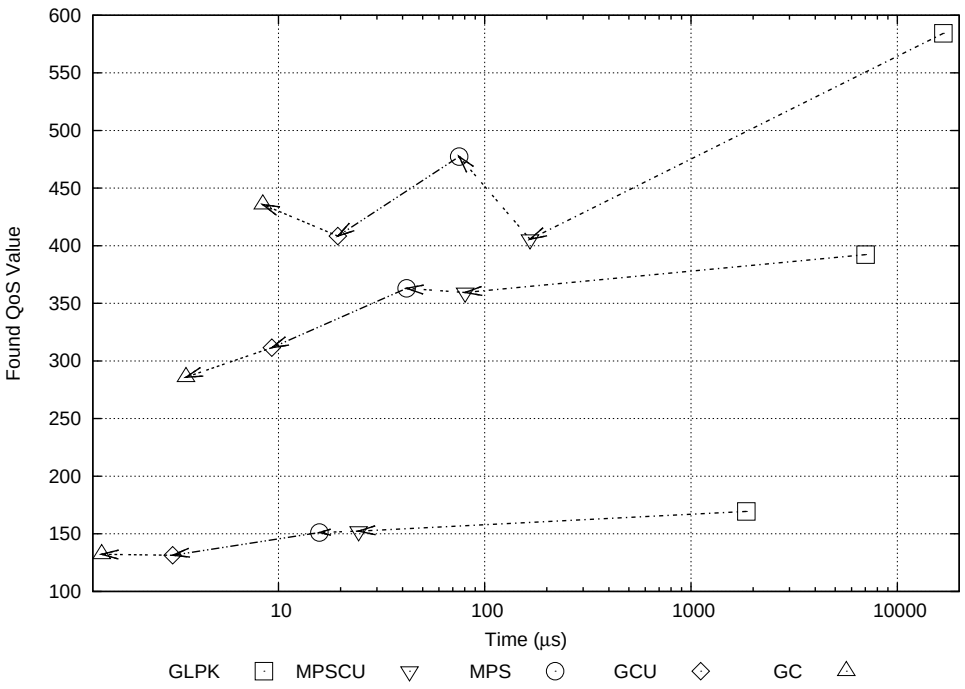


Figure 2. QoS index versus computation time for different test cases and different heuristics.

problems). A second comment is that the use of Greedy Heuristics requires a computation time an order of magnitude below the multi-path heuristics (which in our case scanned four paths in parallel). We conjecture that this gap can be reduced by appropriate optimisations in the code, but is very likely to remain significant. An execution overhead of 10μs is probably affordable in many applications, but the payoff in terms of cost function is often moderate. For the test case on the top of the plot, the use of the utilisation to weigh the increment of the cost function is not apparently convenient.

The simple conclusion that we draw from our experiments is that none of our proposals displayed a striking convenience over the other ones for all possible scenarios, and the selection of the heuristic should be made based on the characteristics of the considered system and applications, the performance cost/tradeoffs specifically identified for the different classes of applications, and similar considerations. A more complete evaluation of these heuristics is beyond the scope of this paper and is reserved for future work.

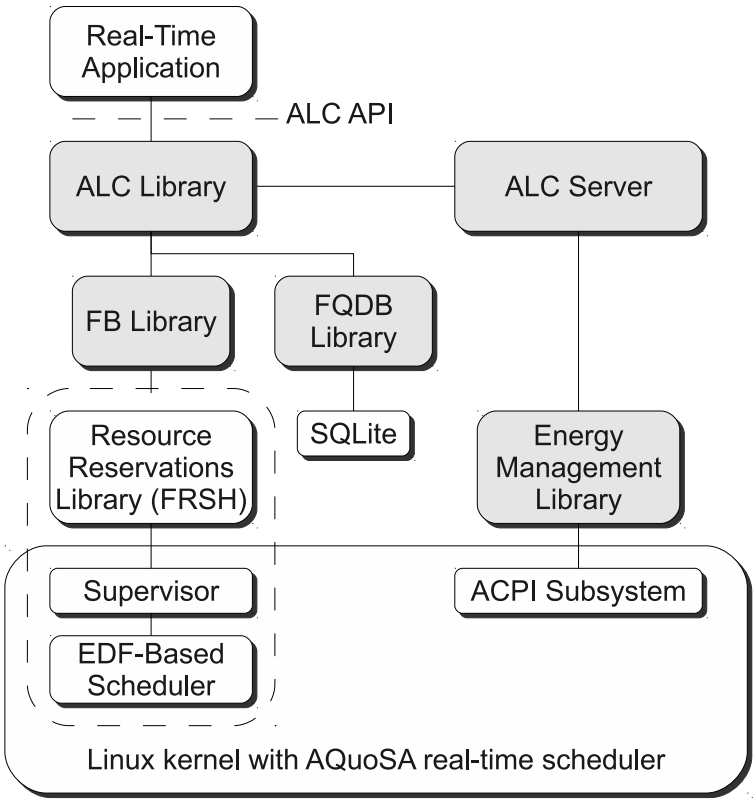


Figure 3. Main components of the software architecture implementing the proposed power-aware QoS management infrastructure. Greyed blocks correspond to the implementation of the dual-loop architecture proposed in this paper.

IV. Software Architecture

The control architecture described in Section III has been implemented in the software architecture shown in Figure 3. The main components of the architecture are the following.

The `ALC Library` is the interface between the application and the `ALC Server` implementing the Global QoS controller. The application has access to this library through the `ALC API` allowing it to specify its resource requirements and its operation modes, in terms of associated QoS values and timing requirements. The library negotiates with the `ALC Server` an appropriate operating mode for the application, depending on the system workload.

The `FQDB Library` allows applications to specify the workload requirements associated with each operation mode and power mode. This information can be stored from previous

runs in a SQLite[2] database. When queried about a configuration that has not been stored yet, the library performs an on-line interpolation of the information collected on-line by using application-specific and resource-specific plugins. This query is only made when the application is activated, therefore, assuming that activations are spaced out by a reasonable amount (say in the order of 100ms or more) the introduced overhead is negligible. An *application plugin* implements a particular mathematical model that describes how the workload reshapes when changing application-level parameters, such as video resolution or colour depth. A *resource plugin* implements a model that describes how computation times scale with respect to resource mode parameters, such as the CPU frequency. These plugins implement the *multi-mode predictor* functionality introduced in Section III-D.

The ALC Server implements the global power/QoS aware optimisation strategy, and performs admission control and periodic re-configuration of applications and resources. After the optimisation problem is solved, each application is notified (via the ALC Library) the mode it has to use for the subsequent activations. Likewise, the ALC Server reconfigures the power-mode of the system resources via a specific Energy Management Library. This provides a unified view on the available power-modes of the underlying physical resources, along with the associated power consumption figures, and allows the ALC Server to reconfigure the power-mode of each resource. Currently, power management for processors is supported via the cpufreq[3] infrastructure.

The FB Library implements the inner control-loop at the resource allocation level. It performs the actual allocation of the CPU via the Resource Reservations Library. It gets from the real-time application notification of the begin and end of each job, and performs on-line adaptation of the reserved budget, so as to meet the declared timing constraints of the application (in terms of periodicity and deadline-miss ratio). The library does not necessarily require a periodic task model, but it supports the optional possibility for the task to specify the deadline at every job start. This way, the bandwidth may be easily adapted also for aperiodic jobs, where it needs to be computed depending on the actual start-time of the job and the available time till the deadline.

[2]More information is available at http://www.sqlite.org.

[3]More information is available at: http://www.kernel.org/pub/linux/utils/kernel/cpufreq/cpufreq.html

The Resource Reservations component, enclosed in a dashed box, is responsible for performing the real-time scheduling of tasks, according to the parameters supplied by the FB Library. This is currently accomplished by using the FRSH API [35] developed in the context of the FRESCOR project[4]. This is a cross-platform API designed for meeting the needs of both hard and soft real-time applications, and it has been implemented on the Linux OS (when enriched with the AQuoSA[5] scheduler [32]), on MaRTE[6], PaRTiKle[7] and Enea OSE[8] operating systems. Furthermore, the use of the FRSH API allows real-time applications to take advantage of real-time scheduling services for the disk and network resources.

The Supervisor is a kernel-level component of the AQuoSA real-time scheduler which receives the independent budget adaptation requests of various clients (coming from the FB Library), and scales them down in order to not violate the scheduler consistency relationship, if needed.

The ALC Server is realised as a stand-alone server process, which communicates with instances of the ALC Library residing within the application process by using a TCP/IP connection at the initial registration of the application within the framework, and by means of a shared memory segment, which allows for a very fast communication between the server and the clients. However, the TCP/IP channel is planned to be leveraged in future extensions of the framework for the management of distributed real-time applications.

V. Experimental Results

In order to produce an experimental validation of the approach, we installed the architecture described in Section IV on an ASUS EEE PC endowed with an Intel(R) Atom(TM) N270 CPU at 1.60GHz, with Dynamic Voltage Scaling capabilities, operated by a Linux Kernel (2.6.29 series) extended with our AQuoSA real-time scheduler [32]. Among the solvers for the global optimization problem introduced in Section III-D4, given the limited computing capabilities of the platform, we decided to used Greedy Cost, the least computing intensive solver, in all the

[4]More information is available at http://www.frescor.org.

[5]More information is available at http://aquosa.sf.net.

[6]More information is available at http://marte.unican.es.

[7]More information is available at http://www.e-rtl.org/partikle.

[8]More information is available at http://www.enea.com.

experiments that follow. Also, the maximum power constraint, corresponding to a minimum life-time of the system, was not used, as our platform was not a battery-operated one.

The type of applications that we have considered represent a large class of multimedia applications that one can encounter in modern industrial contexts such as video-surveillance and visual based control. An important feature of the considered scenario is the extremely dynamic behaviour of the system: new streams can be activated and deactivated based on environmental conditions and on the requests of human operators.

To simulate this challenging situation, we considered two types of real-time applications, developed on purpose. The first one is a *real-time streaming* application (henceforth `streamer`), which periodically grabs video frames from a `v4l2` device[9] at $25fps$, encodes them in MPEG-4 video, and sends the resulting video stream over the network via the Real-Time Transport Protocol (RTP, see RFC 3550 and RFC 3016 for more details). To reduce the latency, each video frame is encoded before the next one is grabbed (so we have an implicit relative deadline equal to the period $T = 40ms$), and only frames of type `I` and `P` are used. The application uses the MPEG-4 video encoder provided by the `libavcodec` library[10], and it uses a dynamic reconfiguration of the video resolution at the encoder input in order to achieve variable QoS levels.

The second application that we considered, called `sample-app`, is essentially a synthetic periodic task generating a random-walk around a controlled utilisation value, specified via command-line parameters. The application can switch between 2 different operating modes, corresponding to decreasing workload requirements and declared QoS rates.

We show the results from three separate experiments. The first one displays the advantages of the QoS optimisation, the second one focuses on power-management and the third one presents overhead measurements.

A. QoS Optimisation

In this section, we show how our dual-feedback loop reacts to changes in workload deciding a new configuration for the system. We considered 6 instances of the `sample-app` application,

[9]More information about the Video4Linux 2 (v4l2) API is available at: `http://linuxtv.org/downloads/v4l-dvb-apis/`.

[10]More information is available at: `http://www.ffmpeg.org`

started at equally spaced out times. The relevant parameters of the applications are summarised in the first group of rows of Table I. Each column refers to one instance of the application. The first row reports the time when the application is started. The second row reports the QoS rate for each mode, introduced in Section III-B3 and used in the problem formalisation in Equation (9). The third row reports the "target" probability of deadline miss (DM) required to the inner control loop (as specified in Equation (2)). For the predictor, we adopted a simple scheme based on a percentile estimator. In essence the algorithm keeps track of the past 12 samples of the computation time and selects a value greater than a percentile of the collected sample given by the desired probability of deadline miss. Clearly, in this way, the condition in Equation (2) is *approximatively* satisfied. The multi-mode prediction was easy in this case, because the application was designed in such a way as to increase the required workload by a factor of 4 when the required QoS is stepped up. The workload associated with each mode has been stored in the database from previous executions and this information can be retrieved when the application requires admission.

In this experiment, we disabled the power-aware logic by setting the w_P weight equal to 0 in the cost function. The weights $\{w_a^{(i)}\}$ for the different applications were all set to 1, meaning that all applications are equally important. The global optimisation was carried out with a sampling period of 1s.

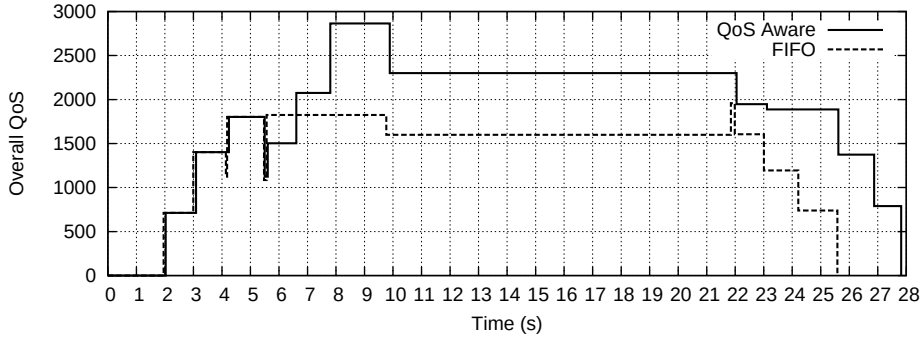
First, we disabled the dynamic rejection capability of the framework, so applications were accepted on the basis of a `FIFO` policy. In this case, if a new application saturates the system, it is simply rejected without any consideration on its QoS. However, we retained both the inner resource allocation control loop and the periodic optimisation loop. The result of the experiments in this case is shown in the dashed curve of Figure 4, where we plot the value of the achieved overall QoS index over time. After an initial time needed to refine the workload estimate from the inner feedback loops (the values used at admission-control time, measured during previous runs of the applications, were clearly optimistic, in this case), the system finds its optimal configuration.

In the second run, we turned on the dynamic rejection capability, whereby the system evaluates the QoS before deciding which application should be dismissed. After the same time for fine-tuning the workload estimates, the system stabilises on a configuration with a far higher overall QoS index. This is an obvious consequence of the fact that the adoption of a FIFO policy in

Instance	1^{st}	2^{nd}	3^{rd}	4^{th}	5^{th}	6^{th}
Approx. start-time	2.0s	3.1s	4.3s	5.5s	6.7s	7.9s
$q^{(i,1)}$ (QoS in mode 1)	353	411	321	514	445	577
$q^{(i,2)}$ (QoS in mode 2)	712	691	680	739	797	789
$\pi^{(i,1)}$ (DM in mode 1)	8.3%	8.3%	8.3%	8.3%	8.3%	8.3%
$\pi^{(i,1)}$ (DM in mode 2)	25%	25%	25%	25%	25%	25%
Total Jobs (QoS)	500	500	32	500	500	500
DM Ratio (QoS)	12.4%	6.6%	75%	16.6%	17%	16%
Avg QoS (QoS)	418	428	680	526	548	605
Total Jobs (FIFO)	500	500	500	500	Rej	Rej
DM Ratio (FIFO)	9.6%	6.8%	10.8%	6.8%	-	-
Avg QoS (FIFO)	418	428	380	542	-	-

Table I

TOP HALF: APPLICATION PARAMETERS FOR THE EXPERIMENTS ON THE QOS. BOTTOM HALF: DEADLINE MISS RATIO AND AVERAGE QOS LEVEL EXPERIENCED BY INDIVIDUAL APPLICATIONS.



(a)

Figure 4. Time evolution of the overall QoS index while starting and dropping the 6 `sample-app` applications.

deciding the applications to discard was not optimal. It is very interesting to take a closer look at how the two feedback loops interact. To this end, Figure 5 shows the computation times experienced by the different jobs of the first application. In the interval between the 50^{th} and the 80^{th} job, the application workload increases, and the inner feedback control loop increases the allocated bandwidth grow in response, up to a point where the budget allocation (not shown) is

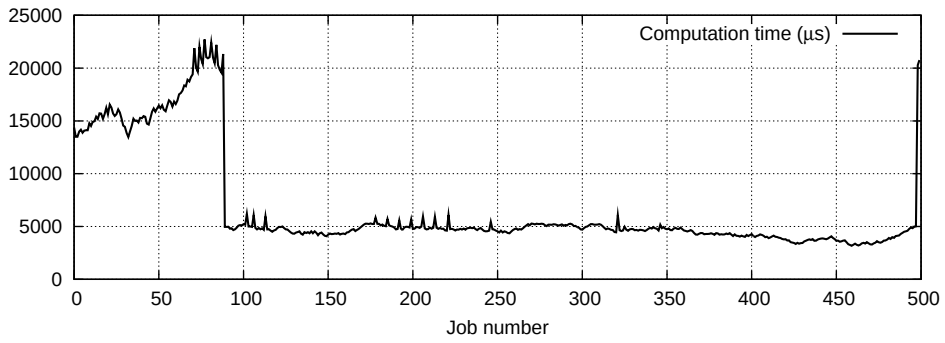


Figure 5. Computation times of the `sample-app` application jobs.

saturated by the Supervisor. At a certain time (close to job 80), the global QoS controller detects the problem and automatically scales down the mode of the application, reducing its workload requirements in order to gain back the schedulability of the system.

The deadline-miss values and average QoS values experienced by the individual applications are shown in Table I. It can be seen that, while the FIFO strategy (middle rows) decided to reject the last two applications, the optimum QoS-aware strategy decided instead to admit them, at the cost of dynamically dismissing the 3^{rd} application after 32 jobs. This way it achieves a QoS index (averaged over time) 32% higher than the FIFO policy (approximately 1850 vs 1400).

As far as the temporal properties of the tasks are concerned, we can see that the deadline miss ratio that we achieved is *generally* intermediate between the value required in mode 1 and the value required in mode 2. This is suggestive of a behaviour of the resource controller within the specification (since the applications dynamically switch between the two modes). This result is due to the fact that: 1) the percentile predictor predicts a value $H_{k+1}^{(i,r)}$ such that $\Pr\left\{c_{k+1}^{(i,r)} \leq H_{k+1}^{(i,r)}\right\} \geq \pi^{(i,r)}$, with $\pi^{(i,r)}$ greater than or equal to the probability specified in the first part of Table I, 2) the assumption in Equation (6) is respected. An exception to the latter condition is offered by the 3^{rd} application for the QoS-aware strategy case. At some point the workload generated by the application prevents the resource controller to accommodate its requests and the probability of deadline miss grows to 75%. The application is later dismissed by the outer loop.

As a variant of this experiment, we started our applications with only the high QoS mode

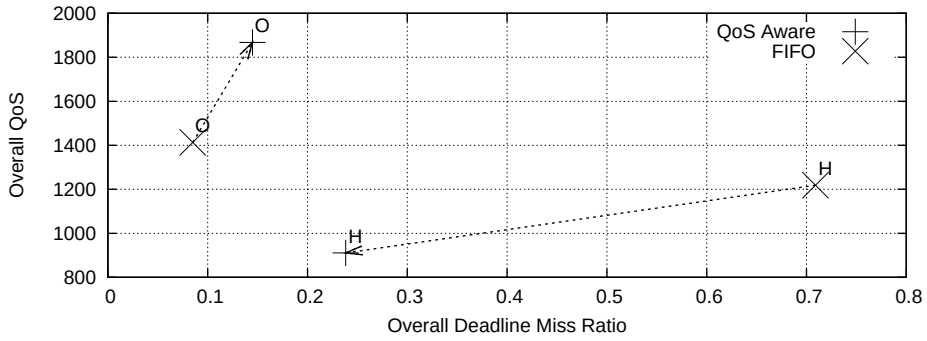


Figure 6. Achieved average QoS index versus average deadline-miss ratio under various configurations.

available. This way the only action available for the external loop is to reject the applications. Figure 6 plots on the x axis the average deadline-miss ratio among the applications, and on the y axis the QoS index, averaged over the entire experiment run. As some instances were rejected dynamically during their execution, the average values refer to the respective time-life of the various instances. Each point on the plot corresponds to an experiment run, and arrows connect points corresponding to runs with the same configuration. We can see that with only high-QoS applications (points with the H labels), the QoS aware admission-control achieves a worse QoS index than the FIFO policy, but this is merely the price it has to pay for keeping the DM ratio within the target specification values. In fact, in such a case, after admission of the applications on the basis of historical workload data, the actual workloads grow unexpectedly and persistently, and the inner QoS control-loop cannot operate at its set-point anymore, due to the impossibility to grant the necessary $\{B_G^{(i,r)}\}$ minimum bandwidths. Clearly, this is a case where a dynamic reconfiguration is needed, achieved by the QoS aware policy by dismissing one of the applications (hence, the loss in the QoS index).

By enabling mode-switching for the applications, we obtained the two points labelled as O, corresponding to the same runs shown in Figure 4. The self-tuning capability of the framework produces a good performance even in the FIFO case. However, the possibility to admit new applications based on a QoS optimisation logic improves the QoS index by $+32\%$ (as discussed above), at the cost of a slight worse DM ratio (which remains within the required bounds).

Mode	Frequency	Power (W)	QoS Penalty
Mode 1	1.60 GHz	2.5	750
Mode 3	1.07 GHz	1.0	500
Mode 4	0.80 GHz	0.7	300

Table II

POWER MODES OF THE INTEL ATOM WE USED, AND ASSOCIATED PENALTIES ON THE QOS INDEX.

B. Power/QoS Optimisation

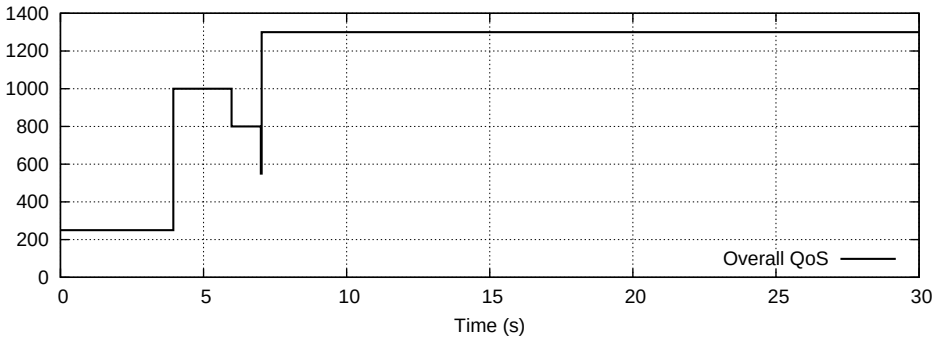
For the experiment shown in this section, we enabled the power management capability. In Table II, we describe the technical data of the power modes, along with the QoS penalty term we introduced for each mode in the QoS Index. The power consumption figures are the maximum values as from the CPU specification [36].

In this experiment, we used multiple instances of the `streamer` application, one acquiring from a real video source, the others acquiring from fictitious video sources provided by means of the `vivi.ko` kernel module, started at intervals of $1s$. Figure 7 (a) shows the overall QoS of the system over time, while in (b) we report the power-mode of the CPU. Between time 4 and 7 we can see that, while the system admits new applications, the CPU power mode is driven from the lowest frequency to the highest frequency mode. The plot in Figure (a) also shows a transient during which the system lowers the QoS mode of the admitted applications in order to account for the actually sensed application workload, clearly higher than the value estimated at admission-control time.

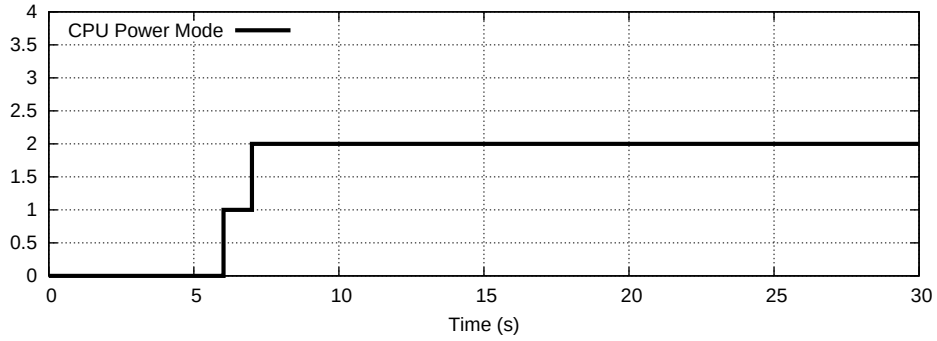
The measured deadline-miss ratios for the two admitted `streamer` instances have been of 0.073 and 0.003, against configured values of 0.

C. Overhead Measurements

We performed an evaluation of the overheads associated to our framework by focusing on the most critical elements, which are triggered at run-time and may possibly impact on the applications performance. Basically, the overhead of our dual-loop control architecture is evaluated separately for the inner loop, where we measured the time needed to run the feedback-based QoS control logic, and the outer loop, where we measured the time needed to solve on-line the



(a)



(b)

Figure 7. Overall QoS index (a), comprising both QoS of applications and penalties due to power consumption of the CPU, and corresponding CPU power mode (b), while starting two instances of the `streamer` application.

optimisation problem. All measurements have been done at the maximum CPU frequency of $1.60\ GHz$.

D. Inner Loop Overheads

In order to measure the overheads associated to the bandwidth control logic of the inner loop, we ran 10 instances of the `streamer` application with 500 jobs per-run, and we measured the time needed to recompute the bandwidth values at each job end. Results are summarised in Table III. As we can see, the average overhead (first row) is of about $46\mu s$, which is a perfectly sustainable figure as compared to the average job execution time of nearly $17ms$ in the lowest QoS mode. Also, we measured the time needed to update the load figures for all application and power modes into the server optimisation problem, obtaining an average value (second row) of $35.62\mu s$. This value is particularly low thanks to the use of the shared memory communications

Description	Avg (μs)	Dev (μs)	Min (μs)	Max (μs)
Bandwidth Adaptation	45.96	8.14	40.00	204.0
Update of Loads	35.62	31.21	25.00	168.0
Admission Control	19863	6716	14978	44485

Table III

OVERHEAD FOR DIFFERENT OPERATIONS.

between the server and the clients. Also, note that, while the bandwidth adaptation is performed every job period, the loads update is only performed every optimisation period.

E. Admission Control Overhead

In the same experiment just described, we also measured the time needed to perform the admission control of the new application, involving both the time for setting-up the client-server communications and allocating the necessary resources, the time for uploading the load figures into the optimisation problem, and the time for finding the new solution. This time has been measured as (third row of Table III) close to $20ms$ (with the GC solver). Such value is surely of importance, however it does not impact the application performance, because this operation is needed before the application enters its main functional loop.

VI. CONCLUSIONS

In this paper, we presented a novel QoS management framework for soft real-time applications based on the application of two QoS control loops. A proper design of these two controllers allows for the optimisation of the performance of multi-mode real-time applications running on hardware with power-switching capabilities. We also presented a software architecture that implements the idea. We provided extensive experimental evidence of the effectiveness of the framework in optimising the overall QoS index keeping in check the real-time behaviour of the applications. The validation was carried out both on synthetic applications and on a real multimedia encoder (featuring dynamic QoS modes). The measurement collected on our implementation in the Linux Kernel revealed that the overhead is acceptable for the considered class of applications.

In the future, we plan to investigate on the integration of a more general power consumption model, which also accounts for the dependency of the power on the expected load (which we assumed to be kept as practically constant and close to saturation, in the current framework), in addition to the frequency. For example, most modern CPUs designed for laptops may switch very quickly to and from *idle* states, even without reconfiguring the CPU frequency and voltage. Also, we plan to extend our framework to the networking resource, so as to properly model networking elements in distributed real-time applications. Correspondingly, we plan to study distributed solutions for power-aware QoS management. Another generalisation we plan is in the direction of allowing each application to have more than one task per CPU. This way we could consider multi-threaded applications, which are certainly very important in modern applications. Finally, we plan to investigate on possible application-specific improvements on the heuristics proposed in this paper.

ACKNOWLEDGEMENT

The research leading to these results has received funding from the European Community's Seventh Framework Programme FP7 under grant agreement n.214777 "IRMOS – Interactive Realtime Multimedia Applications on Service Oriented Infrastructures", and n.IST-2008-224428 "CHAT - Control of Heterogeneous Automation Systems". We would like to thank prof. Luigi Rizzo for having kindly provided the hardware. Finally, our special thanks to the reviewers who helped us improve the paper.

REFERENCES

[1] L. Palopoli, L. Abeni, F. Conticelli, M. Di Natale, and G. Buttazzo, "Real-time control system analysis: an integrated approach," *Real-Time Systems Symposium, IEEE International*, vol. 0, p. 131, 2000.

[2] A. Cervin and J. Eker, "Control-scheduling codesign of real-time systems: The control server approach," *J. Embedded Comput.*, vol. 1, no. 2, pp. 209–224, 2005.

[3] C. C. Wüst, L. Steffens, W. F. Verhaegh, R. J. Bril, and C. Hentschel, "Qos control strategies for high-quality video processing," *Real-Time Syst.*, vol. 30, no. 1-2, pp. 7–29, 2005.

[4] D. Steere, A. Goel, J. Gruenberg, D. McNamee, C. Pu, and J. Walpole, "A feedback-driven proportion allocator for real-rate scheduling," in *Proceedings of the Third usenix-osdi*. pub-usenix, feb 1999.

[5] G. T. C. Lu, J. Stankovic and S. Son, "Feedback control real-time scheduling: Framework, modeling and algorithms," *Special issue of RT Systems Journal on Control-Theoretic Approaches to Real-Time Computing*, vol. 23, no. 1/2, September 2002.

[6] L. Abeni, L. Palopoli, G. Lipari, and J. Walpole, "Analysis of a reservation-based feedback scheduler," in *Proc. of the Real-Time Systems Symposium*, Austin, Texas, November 2002.

[7] R. Rajkumar, K. Juvva, A. Molano, and S. Oikawa, "Resource kernels: A resource-centric approach to real-time and multimedia systems," in *Proceedings of the SPIE/ACM Conference on Multimedia Computing and Networking*, January 1998.

[8] L. Abeni, T. Cucinotta, G. Lipari, L. Marzario, and L. Palopoli, "QoS management through adaptive reservations," *Real-Time Systems Journal*, vol. 29, no. 2-3, pp. 131–155, March 2005.

[9] P. Pillai and K. Shin, "Real-time dynamic voltage scaling for low-power embedded operating systems," in *Proceedings of the eighteenth ACM symposium on Operating systems principles*. ACM, 2001, p. 102.

[10] H. Aydin, V. Devadas, and D. Zhu, "System-level energy management for periodic real-time tasks," in *27th IEEE International Real-Time Systems Symposium, 2006. RTSS'06*, 2006, pp. 313–322.

[11] T. Simunic, L. Benini, A. Acquaviva, P. Glynn, and G. De Micheli, "Dynamic voltage scaling and power management for portable systems," in *Proceedings of the 38th annual Design Automation Conference*. ACM, 2001, p. 529.

[12] G. Qu and M. Potkonjak, "Energy minimization with guaranteed quality of service," in *Proc. 2000 International Symposium on Low Power Electronics and Design*, 2000, pp. 43–48.

[13] W. Yuan and K. Nahrstedt, "Energy-efficient soft real-time cpu scheduling for mobile multimedia systems," in *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*. New York, NY, USA: ACM, 2003, pp. 149–163.

[14] R. Rajkumar, C. Lee, J. Lehoczky, and D. Siewiorek, "A resource allocation model for qos management," in *Proc. 18th IEEE Real-Time Systems Symposium*, San Francisco, 1997, pp. 298–307.

[15] S. Ghosh, J. Hansen, R. R. Rajkumar, and J. Lehoczky, "Integrated resource management and scheduling with multi-resource constraints," in *Proceedings of the 25th IEEE International Real-Time Systems Symposium (RTSS04)*. Washington, DC, USA: IEEE Computer Society, 2004, pp. 12–22.

[16] C. Lee, J. Lehoczky, R. Rajkumar, and D. Siewiorek, "On quality of service optimization with discrete qos options," in *Proc. 5th IEEE Real-Time Technology and Applications Symposium*, Vancouver, 1999, pp. 276–286.

[17] L. Abeni and G. Buttazzo, "Hierarchical qos management for time sensitive applications," in *Proceedings of the IEEE Real-Time Technology and Applications Symposium (RTAS 2001)*, Taipei, Taiwan, May 2001.

[18] V. R. Segovia, K.-E. Årzén, S. Schorr, R. Guerra, G. Fohler, J. Eker, and H. Gustafsson, "Adaptive resource management framework for mobile terminals - the ACTORS approach," in *Proc. of the Workshop on Adaptive Resource Management (WARM 2010)*, Stocholm, Sweden, April 2010.

[19] C. A. Rusu, R. Melhem, and D. Mossé, "Maximizing the system value while satisfying time and energy constraints," *IBM J. Res. Dev.*, vol. 47, no. 5-6, pp. 689–702, 2003.

[20] J.-J. Chen and T.-W. Kuo, "Voltage scaling scheduling for periodic real-time tasks in reward maximization," *Real-Time Systems Symposium, IEEE International*, vol. 0, pp. 345–355, 2005.

[21] M. García-Valls, A. Alonso, J. Ruiz, and A. M. Groba, "An architecture of a quality of service resource manager middleware for flexible embedded multimedia systems." in *SEM*, ser. Lecture Notes in Computer Science, A. Coen-Porisini and A. van der Hoek, Eds., vol. 2596. Springer, 2002, pp. 36–55. [Online]. Available: http://dblp.uni-trier.de/db/conf/edo/sem2002.html#VallsARG02

[22] V. Kalogeraki, P. Melliar-Smith, and L. Moser, "Using multiple feedback loops for object profiling, scheduling and migration in soft real-time distributed object systems," in *Proceedings of the IEEE 2nd International Symposium on Object-Oriented Real-Time Distributed Computing*, pp. 291–300.

[23] S. A. Brandt, G. J. Nutt, T. Berk, and J. E. Mankovich, "A dynamic quality of service middleware agent for mediating

application resource usage," in *IEEE Real-Time Systems Symposium*, 1998, pp. 307–.

[24] R. E. Schantz, J. P. Loyall, C. Rodrigues, D. C. Schmidt, Y. Krishnamurthy, and I. Pyarali, "Flexible and adaptive qos control for distributed real-time and embedded middleware," in *Middleware '03: Proceedings of the ACM/IFIP/USENIX 2003 International Conference on Middleware*. New York, NY, USA: Springer-Verlag New York, Inc., 2003, pp. 374–393.

[25] R. Zhang, C. Lu, T. Abdelzaher, and J. Stankovic, "Controlware: A middleware architecture for feedback control of software performance," in *International Conference on Distributed Computing Systems*, vol. 22. IEEE Computer Society; 1999, 2002, pp. 301–310.

[26] T. Cucinotta, G. Lipari, L. Palopoli, L. Abeni, and R. Santos, "Multi-level feedback control for quality of service management," in *Proceedings of the* 14^{th} *IEEE International Conference on Emerging Technologies and Factory Automation*, Mallorca, Spain, September 2009.

[27] T. Cucinotta, L. Abeni, L. Palopoli, and F. Checconi, "The wizard of os: a heartbeat for legacy multimedia applications," in *Embedded Systems for Real-Time Multimedia, 2009. ESTIMedia 2009. IEEE/ACM/IFIP 7th Workshop on*, oct. 2009, pp. 70 –79.

[28] T. Cucinotta and L. Palopoli, "Qos control for pipelines of tasks using multiple resources," *IEEE Transactions on Computers*, 2009.

[29] L. Palopoli, L. Abeni, T. Cucinotta, G. Lipari, and S. K. Baruah, "Weighted feedback reclaiming for multimedia applications," in *Proceedings of the* 6^{th} *IEEE Workshop on Embedded Systems for Real-Time Multimedia (ESTIMedia 2008)*, Atlanta, Georgia, United States, October 2008, pp. 121–126.

[30] G. E. P. Box and G. Jenkins, *Time Series Analysis, Forecasting and Control*. Holden-Day, Incorporated, 1990.

[31] M. Roitzsch and M. Pohlack, "Principles for the Prediction of Video Decoding Times applied to MPEG-1/2 and MPEG-4 Part 2 Video," in *Proceedings of the 27th IEEE Real-Time Systems Symposium (RTSS 06)*. Rio de Janeiro, Brazil: IEEE, December 2006. [Online]. Available: http://os.inf.tu-dresden.de/papers_ps/roitzsch06predict.pdf

[32] L. Palopoli, T. Cucinotta, L. Marzario, and G. Lipari, "AQuoSA — adaptive quality of service architecture," *Software – Practice and Experience*, vol. 39, no. 1, pp. 1–31, 2009.

[33] S. Boyd and L. Vandenberghe, *Convex Optimization*. New York, NY, USA: Cambridge University Press, 2004.

[34] T. F. Abdelzaher, E. M. Atkins, and K. G. Shin, "Qos negotiation in real-time systems and its application to automated flight control," in *IEEE Real Time Technology and Applications Symposium*, 1997, pp. 228–238.

[35] M. G. Harbour and M. T. de Esteban, "Architecture and contract model for integrated resources II," Universidad de Cantabria, Tech. Rep., 2008.

[36] *Mobile Intel AtomTM Processor N270 Single Core*, Intel, May 2008.