Multi-level feedback control for Quality of Service Management *

Tommaso Cucinotta, **Giuseppe Lipari**†
{t.cucinotta, g.lipari}@sssup.it

Luigi Palopoli, **Luca Abeni**‡
{l.palopoli, l.abeni}@unitn.it

Rodrigo Santos§
ierms@criba.edu.ar

Abstract

We consider the problem of power-aware Quality of Service (QoS) control for soft real-time embedded systems. Applications can have time-varying and scarcely known resource requirements, and can be activated and terminated at any time. However, they have the capability to switch among a discrete set of operation modes *with different QoS levels and resource requirements. In addition, the platform provides resources with* power-scaling *capabilities and may be subject to power constraints.*

We present a QoS control architecture achieving optimum trade-offs between overall QoS and power consumption of the system, based on two nested control loops. The external one decides dynamically the optimum configuration for the system, in terms of application QoS modes and resource power modes, while the internal one modulates the resource allocations on a job by job basis, so as to respect timing constraints. We demonstrate the effectiveness of the approach by extensive simulations with trace data of real multimedia applications.

1 Introduction

An emerging class of soft real-time applications is characterised by strongly varying resource requirements and is commonly executed in *open systems*. In our terminology, a computing system is defined "open" if applications can be activated and terminated in any moment generating a time–varying workload. For such applications, classical real-time systems design methodologies are hardly applicable, because they assume *a priori* knowledge both on the application requirements and on the availability of resources. An interesting alternative for ensuring reliable levels of Quality of Service (QoS) is then offered by the application of a *feedback control loop*, in which design parameters can be fine-tuned by measuring the behaviour of the application and/or the level of workload in the system.

This idea has recently become very popular, producing different approaches, that can be roughly classified into two groups: *application-level* adaptation, in which the application operating modes are adapted to the availability of resources; and *resource-level* adaptation, in which the resource shares granted to the applications are adapted to the dynamic workload requirements. 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. Some authors propose to scale down the power of the CPU when the system workload is below a threshold.

In this paper, we make the point that, if we consider each of these different perspectives in isolation, we miss important opportunities for optimising the system behaviour. Indeed, a "low-level" feedback scheme, which simply operates on resource allocation to accommodate the timing constraints of the tasks, is not clearly able to solve permanent overloads of the system, nor is it able to switch to more aggressive setting for the application if the workload in the system becomes low. Conversely, if the feedback loop simply operates with a high QoS application mode, the problem of finding an appropriate allocation of resources remains unaddressed. A static choice of scheduling parameters in this case may lead to an unsatisfactory timing behaviour.

Original contributions. In this paper, a power-aware QoS management architecture for soft real-time systems is presented, that combines application-level and resource-level adaptation. Combining these two approaches is non-trivial: not only a correct "communication" between the two controllers must occur, but their interaction needs also a proper design, so as to avoid undesired instabilities.

We use an adaptive reservation algorithm in the internal loop and a dynamic optimisation algorithm using discrete options for the QoS and power in the external loop. Our most important contribution in this paper is to show an integrated design of these two control loops, based on a well founded model of the system and on a formal statement of the control goals.

Organisation. The paper is organised as follows. In Section 2 we provide an overview of the approach and describe the control goals. In Sections 3 and 4 we describe the inner and outer control schemes, respectively. In Section 5, we re-

*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".

†Tommaso Cucinotta and Giuseppe Lipari are with *Scuola Superiore Sant'Anna*, *Pisa, Italy*

‡Luigi Palopoli and Luca Abeni are with *Università di Trento*, *Trento, Italy*

§Rodrigo Santos is with *Universidad Nacional del Sur*, *Bahía Blanca, Argentina*

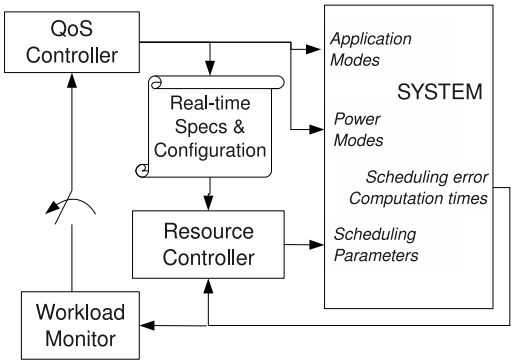

Figure 1. Two-level control loop.

port simulation results that validate the proposed approach. In Section 6, we compare our work with the previous work. Finally, in Section 7 we draw conclusions and present future work directions.

2 System model

We consider a set of soft real-time applications running on a hardware platform consisting of a set of resources (CPU, network, disk, etc). For the purposes of this work we consider CPU-intensive tasks, so the resource needed for executing a task is essentially a processor. More formally, our system consists of a set of n applications $\mathcal{A}^{(1)}, \ldots, \mathcal{A}^{(n)}$, sharing a pool of m processors $\mathcal{R}^{(1)}, \ldots, \mathcal{R}^{(m)}$. Applications can dynamically enter and leave the system, so the number n can change at run-time. Each application $\mathcal{A}^{(i)}$ is comprised of one or more tasks, each one executing on a processor. For the sake of simplicity, and without loss of generality, we assume that in an application there is at most one task per processor: by $\mathcal{T}^{(i,r)}$ we denote the task belonging to application (i) that uses processor (r).

A task $\mathcal{T}^{(i,r)}$ consists of a stream 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 the processor $\mathcal{R}^{(r)}$ for a time $c_k^{(i,r)}$ (in Section 3, we will introduce the concept of stochastic computation times). Job $J_k^{(i,r)}$ is associated 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 do not explicitly consider the interactions between tasks: this is possible assuming that the end-to-end deadlines of the application have been decomposed into partial deadlines for the tasks. We assume that tasks are *periodically* activated ($r_{k+1}^{(i,r)} = r_k^{(i,r)} + T^{(i)}$) and that their *relative deadline* is equal to the period: $d_k^{(i,r)} = r_k^{(i,r)} + T^{(i)} = r_{k+1}^{(i,r)}$.

Each application $\mathcal{A}^{(i)}$ may vary its mode of operation within a finite set $V^{(i)} \triangleq \{1, \ldots, n_{am}^{(i)}\}$ of cardinality $n_{am}^{(i)}$. This means that the algorithm implemented in the application task(s) has a set of discrete options that can be chosen to change the quality. Every mode $j \in V^{(i)}$ is associated with a quality index $q^{(i,j)} \in \mathbb{R}$, which is a measure of the *user satisfaction* when $\mathcal{A}^{(i)}$ executes in the j-th mode.

The hardware platform may consist of one or more processors, exhibiting multiple power operation modes (e.g., by using Dynamic Voltage Scaling), each one associated with a different speed and power consumption. For the sake of simplicity, we assume that the power mode of each processor can be set independently of the others. More formally, each resource $\mathcal{R}^{(r)}$ may vary its operating mode within a finite set $P^{(r)} \triangleq \{1, \ldots, n_{rm}^{(r)}\}$ of cardinality $n_{rm}^{(r)}$. In each mode k, the resource $R^{(r)}$ has a power consumption $p^{(r,k)}$. We make the reasonable assumption that power modes associated with a higher power consumption reduce the execution time of the tasks.

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

2.1 Scheduling

We assume that each processor may be shared between multiple tasks by using a Reservation Based (RB) scheduling policy. We assume a *partitioned* scheduling strategy, where tasks are statically allocated to processors and cannot migrate between them during the application lifetime. How this allocation is performed is out of the scope of this paper.

In a RB framework, a task $\mathcal{T}^{(i,r)}$ using a resource $\mathcal{R}^{(r)}$ is associated a pair $(Q^{(i,r)}, P^{(i,r)})$, said *reservation*, meaning that the scheduling algorithm guarantees to $\mathcal{T}^{(i,r)}$ a *budget* of $Q^{(i,r)}$ execution time units in every *reservation period* $P^{(i,r)}$ for the processor 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 processor reserved to the application. In our framework, the reserved bandwidth for a task can be dynamically changed by changing the reserved budget $Q^{(i,r)}$ for each job (we never change the reservation period). We will use $B_k^{(i,r)}$ to denote the bandwidth required for the k-th job. To analyse the timing behaviour of the tasks, it is convenient to characterise each job $J_k^{(i,r)}$ execution by the *scheduling error* $\epsilon_k^{(i,r)}$. This quantity is a measure of the delay of a job with respect to its deadline (see [5] for a formal definition). If the scheduling error is negative, then the job received an excessive amount of CPU. Conversely, if the scheduling error is positive, then the job received too little and the task is accumulating delays. A positive scheduling error does not always have an immediate impact on the QoS exhibited by the application. For example, in the multimedia domain, buffers are usually used to compensate undesired fluctuations on the finishing times. However, the introduction of buffers increases end-to-end latencies, so for real-time applications with tight responsiveness and interactivity constraints, their use should be limited to the minimum.

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

$$\forall r \in 1, \ \ldots, m \sum_i B^{(i,r)} \leq U_{lub}^{(r)}, \qquad (1)$$

where $U_{lub}^{(r)} \leq 1$ depends on the scheduling algorithm used on $\mathcal{R}^{(r)}$. As detailed in Section 4, basing our schedulability condition on a utilisation-based test, leads to the advantage of obtaining a problem formulation falling within the standard class of Integer Linear Programming (ILP) problems.

2.2 Control Specification

The two-level feedback scheme proposed in this paper is sketched in Figure 1. The internal feedback loop performs the resource-level adaptation, and it is depicted in the box labelled *Resource Controller*. There is a separate resource controller for each task in the system. The controller operates at the end of each job; first it measures the computation time and the scheduling error of the task, then it computes the bandwidth for the next job and *actuates* it by changing the scheduling parameters (the reservation budget). The goal of each controller is to keep in check the scheduling error of the corresponding task. As far as some basic assumptions on the workload are respected, the resource controller is able in its turn to respect the specification on the real-time behaviour of the tasks. If some changes occur in the workload generated by the applications that invalidate these assumptions (i.e., a new application enters the system or a running application exhibits a substantial change in the workload characteristics) then the resource controller is no longer able to work properly.

In this case, the external control loop, containing the *QoS controller* box in Figure 1, provides the application-level adaptation. It monitors the system periodically and if detects that some resource controller is not working properly, it switches the modes of the applications and the power modes of the processors to change the minimum bandwidth requirement of the tasks and changes the configuration of the task controllers. Similarly, if a reduction in the average resource requirements is detected, the QoS controller can make more aggressive choices in the application modes increasing the "macroscopic" QoS perceived by the user.

2.3 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 have defined a QoS index $q^{(i,j)}$. If this level is maintained for a time interval Δt (which is equal to the sampling period of the external loop), the total QoS accumulated 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, changing the bit-rate of the play-back of a video stream too frequently 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 $k_s^{(i)} > 0$, which quantifies the quality degradation due to the change. This term does not depend on the duration of the time interval. Therefore, its impact becomes smaller if we increase the sampling period Δt.

Another dimension for evaluating the QoS is energy consumption. For each processor $R^{(r)}$, and each operating mode k, we specify a *power consumption* $p^{(r,k)}$ when the resource is operated in mode k. The energy spent over the sampling interval Δt is simply given by $p^{(r,k)}\Delta t$. Clearly this term has to be accounted for with a negative sign (the greater the energy consumed, the lower the quality). If the information is available, also in this case it is possible to introduce a metric quantifying the energy spent in a transition between two different power modes (j) and (k): $-k_p^{(r)(j,\,k)}$.

These different metrics are clearly in conflict with each other. For instance, if we increase the level of one application, we have to correspondingly increase the quantity of resource dedicated to the application. Likewise, reducing the power mode of a processor produces a longer lifetime for the system but it also increases the computation times of the different tasks. In response to this action, we may be forced to switch to lower QoS application modes so as to prevent overload conditions.

Generally speaking, our problem is one of multi-objective optimisation and is generally addressed by identifying the front of Pareto optimal points[1] and by studying the best trade-offs between the different metrics on this front. In our case, we adopt a simple technique known as "scalarisation" [7]. The idea is to construct a simple linear *utility function* in which each different metric is associated with a weight. In our case, suppose that the optimisation is run at time t and has to decide the parameters for the interval $t + \Delta t$. Let $j^{(i)}$, $k^{(i)}$ denote the application mode of $\mathcal{A}^{(i)}$ before and after the optimisation, and $j^{(r)}$ and $k^{(r)}$ represent the power modes of processor $R^{(r)}$ before and after the optimisation. The utility function is given by:

$$\sum_{i=1}^{n} w_a^{(i)} \left(\Delta t q^{(i,k^{(i)})} - k_a^{(i)} |q^{(i,k^{(i)})} - q^{(i,j^{(i)})}| \right) - \sum_{r=1}^{m} w_p^{(r)} \left(\Delta t p^{(r,k^{(r)})} + k_p^{(r)(k^{(r)},\,j^{(r)})} \right), \quad (2)$$

where the weights $w_a^{(i)}$, $w_p^{(r)}$ are positive real numbers that emphasise/de-emphasise the importance of each application or resource in the search on the Pareto front.

2.4 Constraints

Independently from the configured mode of operation, each application is associated with a soft real-time constraint. This may be specified in terms of the probability $\pi^{(i)}$ that the scheduling error $\epsilon^{(i)}$ respects an upper bound $\delta^{(i)}$. Formally:

$$\Pr\left\{ \epsilon_k^{(i)} \leq \delta^{(i)} \right\} \geq \pi^{(i)}. \quad (3)$$

If the bound $\delta^{(i)}$ is chosen equal to 0, we get the probability of respecting the deadline. In some cases, larger delays can

[1] A point is said Pareto Optimal if no further improvement can be achieved on a metric without lowering the other ones.

be tolerated and a greater value can be chosen for $\delta^{(i)}$.

The control action is subject to two types of constraints. The most important one is the schedulability condition Equation (1). If the inner feedback loop violates this condition, the schedulability algorithm is not in condition to work consistently. Therefore, we can define as "feasible" a vector of application and power modes such that the resource controller is able to enforce the timing requirement in Equation (3) without violating Equation (1). Another constraint that we will consider is on the maximum power consumption for the set of available power-aware processors on the system. This constraint is particularly useful for guaranteeing a minimum lifetime for battery-operated devices, but it may equally be useful in those situations in which environmental conditions constrain the maximum power that the system can dissipate.

3 Resource Level QoS Control

This section presents some background on resource-level adaptation (a more complete description of this work can be found in [14]). The proposed approach is comprised 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 local controller is to formulate a minimum bandwidth request $\overline{B}_k^{(i,r)}$ that allows the task to respect its timing constraints. 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: $\forall r \in \{1, \ldots, m\}$, $\sum_i B_G^{(i,r)} \leq U_{lub}^{(r)}$. Indeed, in the worst case all tasks could require their guaranteed bandwidths.

From now on, to simplify the notation, we will refer to a single processor.

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

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

where $S(x) = x$ if $x > 0$, and $S(x) = 0$ if $x \leq 0$. In order to find a feedback control law to achieve Condition (3), the sequence of computation times $\{c_k\}_{k\in\mathbb{N}}$ and of experimented scheduling errors $\{\epsilon_k\}_{k\in\mathbb{N}}$ are considered as discrete-time, continuous-valued, stochastic processes, related by Equation (4). We denote by $\mathcal{E}_k$ and $\mathcal{C}_k$ the stochastic processes for which ϵ_k and c_k are specific realisations. The problem is therefore, given δ, to find a function relating B_{k+1} to ϵ_k such that $\Pr\{\mathcal{E}_k \leq \delta\} \geq \pi$.

It is very difficult to compute the expression of the scheduling error probability distribution, as resulting from the combination of the computation times distribution, the control law function, and the system evolution equation. Therefore, we consider the following "relaxed" goal. If the scheduling error is in a region $\mathcal{R} = [-T, R]$ enclosing the target region $\mathcal{R}_T = [-T, \delta]$ ($R \geq \delta$), then it is steered to $\mathcal{R}_T$ with at least the probability π:

$$\Pr\{\mathcal{E}_{k+1} \in \mathcal{R}_T \mid \mathcal{E}_k \in \mathcal{R}\} \geq \pi. \tag{5}$$

Therefore, if the scheduling error is sufficiently close to the target set, then it is reduced into the target set with at least the probability π. We will refer to $\mathcal{R}$ as *attractivity region*.

A control law achieving this goal can be constructed as shown in the following result (whose proof is in [14]):

Theorem 1 *Assuming that the controller has knowledge of a boundary H_{k+1} for the computation times such that:*

$$\Pr\{\mathcal{C}_{k+1} \leq H_{k+1}\} \geq \pi, \tag{6}$$

and that the minimum bandwidth B_G guaranteed to the task satisfies

$$B_G \geq \sup_k \frac{H_k}{T + \delta - R}, \tag{7}$$

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

$$\overline{B}_{k+1} = \frac{H_{k+1}}{T + \delta - S(\epsilon_k)}. \tag{8}$$

Following the same line of reasoning as in the theorem above, for a *given* B_G, the controller fulfils the requirement if $\epsilon_k \leq R_k \triangleq T + \delta - \frac{H_{k+1}}{B_G}$. We can extend the control law proposed above by saturating it to B_G for values of the scheduling error greater than R_k:

$$\overline{B}(\epsilon_k) = \begin{cases} \frac{H_{k+1}}{T+\delta-S(\epsilon_k)}, & \text{for } \epsilon_k \leq R_k \\ B_G & \text{otherwise.} \end{cases} \tag{9}$$

As discussed in [14], this saturated law, together with a further constraint on the minimum guaranteed bandwidth, guarantees that, if at some point in time the scheduling error leaves the attractivity region, it returns to it in a finite number of steps, and the maximum value it can take is bounded.

Predictor. Many algorithms are available for time series predictions [6]. Generally, these rely on an analysis of the past observed c_k samples, where both simple solutions like moving averages, or complex ones based on optimal filtering theory, are possible.

A better performance can be obtained customising the predictor by using knowledge on the domain of the application. Thus we suggest that, whenever possible, the predictor should be provided by the application itself. For example, a very effective approach for prediction of MPEG decoding times can be found in [18], where a quick parsing of

a frame before starting its decoding process is used to perform an accurate estimation of the decoding time for that frame, with a sustainable overhead. For the MPEG streams structured with a Group of Pictures of fixed size, an interesting trade-off between prediction accuracy and overhead is achieved by the algorithm we proposed in [4], where each sample is estimated using multiple interleaved moving averages. This way, we can take advantage of the knowledge on the structure of the MPEG stream.

Resource Supervisor. The supervisor 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 $B_G^{(i)}$. This problem can be approached in several ways. A simple, yet effective solution, is to use a compression function. Let $\left\{\overline{B}^{(i)}\right\}$ denote the bandwidths required by the task controllers at some time t. If $\sum_i \overline{B}^{(i)} \leq U_{lub}$, then the granted bandwidths $B^{(i)}$ are simply set equal to the required values: $\forall i,\ B^{(i)} = \overline{B}^{(i)}$. Otherwise, set the granted bandwidths $\left\{B^{(i)}\right\}$ as follows:

$$\forall i,\ B^{(i)} = B_m^{(i)} + \left(U_{lub} - \sum_j B_m^{(j)}\right) \frac{\overline{B}^{(i)} - B_m^{(i)}}{\sum_j \left(\overline{B}^{(j)} - B_m^{(j)}\right)}$$

with $B_m^{(i)} \triangleq \min\left\{B_G^{(i)},\ \overline{B}^{(i)}\right\}$. Due to the supervisor action, the bandwidth allocated to a task can change during a job execution. This does not affect the properties of the task controllers, as long as the change respects the minimum guaranteed bandwidth. The technological solutions to carry out this change are discussed in detail in [11].

4 Application-level QoS control

The outer control loop, i.e., the QoS controller, performs an optimisation whose decision variables are the modes of the application 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 (as computed by the controller at the previous sampling instant). Similarly, we introduce for each resource a 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 resource $\mathcal{R}^{(r)}$.

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 $\mathbf{p}^{(r)} = [p^{(r,1)}, \ldots, p^{(r,n_{rm}^{(r)})}]$. Finally, for each application $\mathcal{A}^{(i)}$ and resource $\mathcal{R}^{(r)}$, we introduce the matrix $\mathbf{B}_G^{(i,r)} \triangleq [B_G^{(i,j)(r,k)}]_{j,k}$ 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 B_G represent the minimum guaranteed bandwidth required to each resource controller to sustain the performance specification in Equation (5). This quantity can be computed, given an an estimate of $\sup_k \{H_k\}$, by using Equation (7). Such estimates may be based on information acquired upon each sampling 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.

Now, we may formalise the problem as:

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

subject to the constraints:

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

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

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

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

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. For notational convenience we omitted the term related to the change in the power mode. The problem has bi-linear constraints and absolute values in the objective function. However, it may be transformed in a standard Integer Linear Programming (ILP) problem through simple transformations. If the problem does not have solutions, the outer loop may remove one application (or reject it if it is being admitted).

4.1 Solving the optimisation problem

The exact solution of a BLP has generally an exponential complexity in the number of decision variables. Taking inspiration from [1], we developed a greedy heuristic which is quadratic in the number of tasks and resources and linear in the number of levels, and works as follows:

1. set application modes to min QoS, and resource modes to max power;

na	nam	nr	nrm	v_{opt}	t_{opt}	v_{heu}	t_{heu}
8	6	1	5	704	75984	700	25
12	6	1	5	904	49703	900	58
16	6	1	5	904	104789	883	87
20	6	1	5	900	187304	883	105
24	6	1	5	900	282101	879	128
28	6	1	5	908	403547	875	147

Table 1. Number of variables (v_*) and computation times (t_*) obtained when using the exact ($*_{opt}$) and the heuristic ($*_{heu}$) solvers, in various cases.

2. 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;
3. select the change in either the QoS mode of an application, or the power mode of a resource that maximises the objective function increment;
4. repeat from step 2.

Since at each step the algorithm always identifies a feasible solution, the execution can be interrupted at any iteration. Therefore, it is possible to consider different trade-offs between the computation time and the result accuracy.

We ran a set of experiments on problems of different sizes, in order to evaluate the effectiveness of the heuristic approach, as compared to an exact solution, which has been obtained by using the GLPK library[2]. For each problem, we measured the time needed for the execution of both the exact and the heuristic solvers, and the respective values of the objective function that were achieved. The experiments have been conducted on an AMD Turion64 at 1800 MHz with 512MB of RAM. Results are reported in Table 1, where the first four columns represent the problem configuration, in terms of number of applications (na column), number of QoS modes (nam column), number of resources (nr column) and number of power modes (nrm column). Then, the achieved objective value function and the average execution times (in μs) achieved for the exact and the heuristic solvers are shown. While the generic GLPK-based solver requires execution times that may be hardly deemed acceptable on an embedded system, the heuristic solver instead has execution times that are orders of magnitude below, with a value of the achieved objective function that is only slightly below the absolute optimum.

5 Simulation results

The proposed QoS control framework has been simulated on data gathered from real applications[3]. Both QoS control levels have been implemented in ARSim[4], an open-source simulation tool we developed in order to evaluate the behaviour of different adaptive reservation policies on the evolution of the scheduling error.

Table 2. QoS control loop parameters

Parameter	Value	Parameter	Value
$w_p^{(1)} \cdot p^{(1,lofreq)}$	0	$w_p^{(1)} \cdot p^{(1,hifreq)}$	60
$\delta^{(i)}$	5ms	$\pi^{(i)}$	83.3%
$w_a^{(i)}$	1	$w_s^{(i)}$	0
$q^{(i,DVD)}$	100	$q^{(i,SVCD)}$	67
$q^{(i,VCD)}$	24		

We set up a simple experiment emulating three MPEG streaming tasks running on the same CPU. Each task has three operation modes, corresponding to decoding three distinct stream types with different resolutions: DVD, SVCD and VCD. The shared CPU has 5 power modes, corresponding to operating frequencies in the set 3.05 GHz, 2.68 GHz, 1.92 GHz, 1.15 GHz and 767 MHz.

In order to keep the simulation realistic, we used execution times measured on real traces of the frame decoding times as measured during the execution of a modified version of *ffprobe*, an MPEG decoder based on the *ffmpeg* library, on an Intel Celeron D at 3.05 GHz with 1 GB of RAM, while decoding MPEG movie segments. Measurements have been averaged over multiple executions of the program run at SCHED_FIFO policy, and without using the disk (with all the necessary data in a RAM disk). The multiple application modes and CPU modes have been accounted for by acquiring decoding time traces for each combination of the 3 application modes, and 2 out of the 5 available CPU modes, with a total of 6 traces per movie.

Multi-Mode predictor. The multi-mode predictor is used to construct the matrix B_G introduced in the previous section starting from the knowledge of some of its elements (the one related to the current mode).

By analysing the traces obtained in the previous step, we could observe that computation times vary linearly with the frequency ratios between the two modes. Also, when switching resolution, the computation times vary linearly with the ratios among the number of pixels of the modes. Thereby, a simple linear transformation mapping the resource requirements of one mode onto the other can be obtained by interpolation. Indeed, we verified that, transforming one trace using the linear transformations, led to an average relative error with respect to the actually measured data, of less than 10%, for all the other possible configurations. More details can be found in [10].

QoS Control Set-up. In the resource-level controller we used a very simple history-based predictor, built of two parts: a sample estimator, plus a percentile estimator. The

[2]GNU Linear Programming kit (GLPK), more information at the URL: http://www.gnu.org/software/glpk.

[3]We would like to thank Alberto Donadoni for adapting the *ffmprobe* software to gather the necessary application data.

[4]More information at the URL: https://gna.org/svn/?group=arsim.

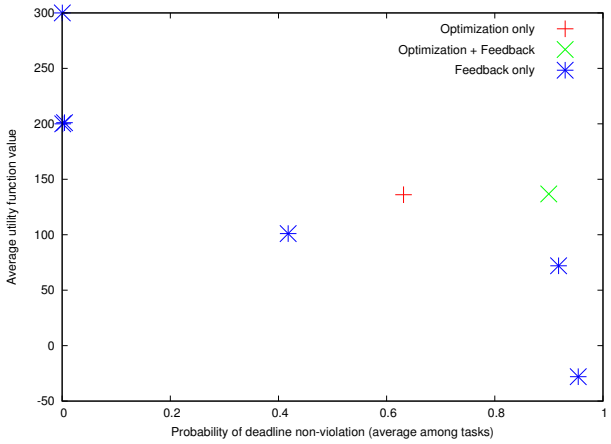

Figure 2. Comparison of different adaptation strategies.

former uses 3 interleaved moving averages, as described in Section 3. The second part consists in an algorithm that observes the prediction errors done by the first part over a sliding time horizon of 12 frames, and estimates the p^{th} percentile of such errors. The obtained error percentile is finally summed to the estimated sample as output by the first part, to obtain H_k. Table 2 summarises the configuration set-up for the QoS control loops.

Comparison with a single feedback loop. In Figure 2 we summarise the simulation results for the set up described above, showing a comparison between the performance achieved with the external loop only (Optimisation only), with the internal loop only (Feedback only) and with the combination of the two.

For each application we measured the *experimental probability* of meeting the deadline and the cost function in Equation (3). Both metrics were computed for the three applications and then averaged over a time horizon and over the three applications. Each experiment is then reported as a point in the plot. In particular, the first metric is reported on the X axis and the second one on the Y axis.

First, we consider the application of resource feedback only, with a static configuration of the application and power modes. We repeated this experiment with different configurations, so we have multiple points in the plot. The obtained results highlight that, with some configurations, we can get very good probability of respecting the deadlines. In fact, if the selection of the modes leads to a sustainable workload, the probabilities of meeting the deadline are close to 98%. The cost function value, though, is very low. On the other hand, if we increase the modes, we get values of the cost function around 300 (left-top corner), but a very low probability of meeting the deadline. It is difficult to get a high value for both the QoS and the probability of meeting the deadlines because the controller is not able to switch the mode even when the current workload permits it.

On the other hand, the external feedback alone selects modes that generate a sustainable workload over time. The probability of meeting the deadline, though, is disappointing since the system does not efficiently distribute the bandwidth to applications. Only the combination of the two feedback loops achieves at once a good probability of meeting the deadline and an acceptable average QoS value.

6 Related Work

While various works have been done on both QoS optimisation and adaptive scheduling, the problem of integrating these two levels of adaptation has been considered only in a few of them. In [3], a dual-loop QoS control strategy was proposed. An inner controller performed on-line adaptation of the requested bandwidth, which was re-modulated by adopting a system-wide compression function; an outer controller performed a slower on-line adaptation of the application QoS level by modulating the task activation rate. In [8], a middleware was presented to dynamically adapt the QoS levels of applications, however the idea of tuning the resource allocations for multiple applications by maximising a cost function quantifying the overall system QoS was proposed in QRAM [17], where the optimisation was done off-line due to the heavyweight computations. The QRAM approach has also been used on-line in an adaptive scheme [13]. In [12], a hierarchical multi-level QoS control architecture was presented, however the paper focused on the software components and interactions, rather than on the description of the system-wide optimisation goals achieved by the infrastructure. In [22], a QoS optimisation framework was presented whose goals are similar to the one presented in this paper, in that an objective function to maximise was defined, that depends on deadline misses, application QoS levels, and QoS level changes, and whose evolution depends on a well-defined evolution model not far from the one used in this paper. However, each application was optimised in isolation, and a global optimisation strategy was left as future work.

None of the above works addressed the system-wide QoS optimisation by means of a QoS optimisation problem, formulated so as to fit within a class of well-known problems (ILP), by accounting also for the power-related issues, like done in this paper. However, the outer control loop we propose is inspired by QRAM.

Adaptive schemes in which power consumption is explicitly considered along with temporal guarantees have been proposed in [19] and [15], but in those papers scalability of the resource requirements depending on a time-varying QoS level is absent.

A fundamental ingredient of most of the adaptive schemes is a scheduling algorithm that allows for fine grained control of the resource allocation, like *resource reservation schedulers* [16], and the feedback-based scheduling algorithms built on top of them known as *adaptive reservations* [2]. This idea enables a well founded de-

sign for a QoS control algorithm [4], and constitutes a fundamental brick of the present paper. In [20], a middleware is proposed for adaptive QoS control using real-time scheduling facilities at the computation and network levels, however power-related issues are not considered.

Similar ideas, although based on different scheduling algorithms, are the ones behind the notion of real-rate scheduling [21]. In other cases [9], the use of different classical schedulers does not permit to write a precise dynamic model for the plant, and the adaptive algorithm can be mainly regarded as a heuristic solution.

7 Conclusions

In this paper, we presented a multilevel feedback approach to the problem of QoS management. The outer feedback loop manages the operation modes of the applications and the power modes of the resources, while the inner loop is focused on timing constraints. We have shown that in our framework the two levels coexist nicely and their combination produces better results than the application of one of them in isolation. Our future work will be aimed at the development of a middleware architecture, built on top of the AQuoSA framework [11], that will implement the ideas shown in the paper. Much work remains to be done also on the theoretical side, in which we will use control theoretical tools to study and formalise the stability properties of the combination of the two feedback loops.

References

[1] 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*, pages 228–238, 1997.

[2] L. Abeni and G. Buttazzo. Adaptive bandwidth reservation for multimedia computing. In *Proceedings of the IEEE Real Time Computing Systems and Applications*, Hong Kong, December 1999.

[3] 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.

[4] L. Abeni, T. Cucinotta, G. Lipari, L. Marzario, and L. Palopoli. Adaptive reservations in a linux based environment. In *Proceeding of the Real-Time Application Symposium (RTAS 04)*, Toronto (Canada), May 2004. IEEE.

[5] 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.

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

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

[8] 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*, pages 307–, 1998.

[9] 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*, 23(1/2), September 2002.

[10] T. Cucinotta, G. Lipari, and L. Palopoli. Control algorithms for coordinated resource-level and application-level adaptation ii. Deliverable D-AQ2v2, FRESCOR EU project (FP6/2005/IST/5-034026), May 2008.

[11] T. Cucinotta, L. Palopoli, L. Marzario, and G. Lipari. AQuoSA – adaptive quality of service architecture. *Software – Practice and Experience*, 2008.

[12] 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 A. Coen-Porisini and A. van der Hoek, editors, *SEM*, volume 2596 of *Lecture Notes in Computer Science*, pages 36–55. Springer, 2002.

[13] 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)*, pages 12–22, Washington, DC, USA, 2004. IEEE Computer Society.

[14] L. Palopoli, L. Abeni, T. Cucinotta, G. Lipari, and S. K. Baruah. Weighted feedback reclaiming for multimedia applications. In *ESTImedia*, pages 121–126, 2008.

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

[16] 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.

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

[18] 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, December 2006. IEEE.

[19] R. Santos, M. B. D'Amico, J. Orozco, P. Doñate, L. Ordinez, and D. Donari. Power and real time requirements integrated with an adaptive resource reservation soft real-time scheduler. In *Proc. 31st Latin American Conference of Informatics*, Cali, 2005.

[20] 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*, pages 374–393, New York, NY, USA, 2003. Springer-Verlag New York, Inc.

[21] 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.

[22] 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.*, 30(1-2):7–29, 2005.