QoS Control for Pipelines of Tasks using Multiple Resources

Tommaso Cucinotta, *IEEE Member,* and Luigi Palopoli, *IEEE Member*

***Abstract*—We consider soft real-time applications organised as pipelines of tasks using resources of different type (communication, computation, storage). The applications are assumed to be periodically triggered and the different tasks communicate by unidirectional buffers. The problem we cope with is how to effectively share the resources so that some specified Quality of Service (QoS) requirements are met. The QoS considered here is tightly related to the end-to-end temporal behaviour of the application. To compensate for time-varying resource requirements, we advocate a distributed control approach whereby the scheduling parameters of each task are tuned depending on the temporal behaviour of the application measured by appropriate sensors. The use of real-time scheduling strategies enables a mathematically safe control design, in which the QoS requirements are translated into formal control goals, and formal proofs are provided on the ability of the controller to fulfil these goals. We also offer extensive simulations that validate the approach for multimedia applications.**

I. Introduction

An increasing number of real-time applications are distributed. The data items are processed by a set of interacting components (tasks) that use resources of different type (computation, communication, storage). In order to reduce the cost and the complexity of the system, designers are pushed towards an aggressive sharing of these resources. The price to pay is a reduced predictability for the execution of applications whose Quality of Service (QoS) is strongly related to the temporal behaviour. Indeed, the scheduling delays that an application may suffer on the different resources can significantly change based on the workload fluctuations. In this context, the ability to execute the application respecting its timing constraints heavily depends on the scheduling policy and on the choice of the scheduling parameters. In a distributed context, the problem is even harder since "local" scheduling choices have to be coordinated to achieve a "global" goal.

Traditional approaches to the design of distributed real-time applications use priority based schedulers. For a given choice of scheduling priorities, appropriate analysis techniques enable a designer to compute upper bounds on the end-to-end delays incurred by the application and decide if a given real-time system meets *all* of its deadlines. For many *soft real-time* applications such strong guarantees are neither strictly needed nor useful. It is often the case that we can trade a moderate occurrence of deadline violations for a more efficient utilisation of resources.

In this context, classical approaches have been proven ineffective. In particular, the use of fixed or even dynamic priorities is simply too coarse a tool for controlling the workload fluctuations of very dynamic applications. This limitation has stemmed a strong research activity in the past decade producing scheduling mechanisms that permit a fine grained control on the temporal allocation of resources (e.g., exporting such parameters as the fraction of resource allocated to each task). Solutions of this type are increasingly available even on general-purpose Operating Systems (e.g., Linux). In order to show their full potential, these mechanisms have to be complemented with design procedures guiding the choice of the local scheduling parameters. Very suitable to the soft real-time domain are probabilistic approaches, which apply performance analysis to decide "off-line" the allocation of parameters that allow a system to meet its timing constraints with a given probability.

In this paper, we consider very dynamic environments (open systems), in which applications can be started and terminated at any moment. Moreover resource requirements are difficult to identify in detail and can be strongly time-varying. Therefore, carrying out a complex stochastic design of the system upon every reconfiguration is hardly a viable solution. A better solution to the problem has to be sought in the realm of adaptive algorithms, able to self-tune the scheduling parameters to accommodate the QoS requirements of the applications and this is the approach taken in the paper.

a) Problem presentation and solution strategy: We consider applications composed of pipelines of tasks, operating on data items (or tokens) that traverse sequentially the pipeline. Each task processes tokens by using resources, which are shared with other tasks and managed by the underlying run-time platform through a *reservation-based* scheduling mechanism. The scheduler allocates a fraction of the resource (*bandwidth*) to the competing tasks, and allows for dynamic changes of the allocation at run-time. We will make two important assumptions: each task uses a single resource, and tokens entering the first stage are generated *periodically* with a bounded jitter. Therefore, the pipeline is required to sustain a periodic processing of data, producing the result of the chain of computations with the required level of QoS.

We identify two classes of QoS requirements:

- **strong:** every token must be processed *within* a maximum end-to-end latency; also the output jitter must be controlled;

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".

T. Cucinotta is with the Real-Time Systems Laboratory (ReTiS) of Scuola Superiore Sant'Anna, Pisa, Italy, e-mail: cucinotta@sssup.it.

L. Palopoli is with University of Trento, Trento, Italy, e-mail: palopoli@disi.unitn.it.

- **weak:** there is a *nominal* specification for the end-to-end latency and for the jitter; however, when the system is overloaded the application is allowed to produce the result with an additional delay, which has to be bounded.

In both cases, we require that applications never discard tokens. As an example, a professional interactive video-conferencing system can be considered as an application with strong QoS requirements, since the latency from the acquisition device up to the rendering (remote) device has to be very low and predictable. On the other hand, an IP-TV player is a typical example of application with weak QoS requirements, since moderately larger latencies may be compensated for through an appropriate use of buffering.

The workload of a task can change considerably over its sequence of activations in time. For instance, in a video-monitoring application the encoding/decoding workload can be very low when the scene is still and become much higher when the scene is subject to sharp changes.

The problem addressed in this paper is *how to decide the fractions of resource allocated to an application tasks* so that it receives a specified QoS guarantee. To this end, we advocate an *adaptive* scheme, whereby resource allocation is changed in time to track the time varying resource requirements of the tasks. The core contribution of this paper is the design of a distributed QoS control strategy and resource supervisory policy that, operating *locally* on each task and on each resource, attain a *global* goal: ensuring the specified QoS guarantees to applications. Our strategy consists of two elements. First, we provide a mathematical *model* that describes the temporal evolution of an application. This is done defining a *state variable* that measures the deviation, for each activation of each task, from a reference point. Therefore, the QoS requirements of an application can be formulated as a constraint on the allowed evolution for the trajectories of the state variables. Second, we design a non-linear feedback-based task *controller* and a *resource supervisor* that restrains the evolution of the state-space trajectories in the specified sets. Third, we show theoretical conditions under which the coordinated action of the task controllers and of the resource supervisors provides the application with QoS guarantees. Such conditions allow for the design of a simple admission control policy that admits into the system only those applications for which the QoS requirements can be satisfied.

Although the results and the techniques presented in the paper have been developed for linear topologies (for the sake of simplicity), we believe that extensions to more general cases are possible, like for the DAG topologies discussed in Section VI.

b) Paper outline: After an overview of alternative approaches to the problem in Section II, we offer a system level view for our approach in Section III, and we show how to model the QoS control problem in a control theoretical framework in Section IV. The control design and the formal properties it attains are described in detail in Section V. The effectiveness of the techniques presented in this paper is extensively shown on a set of realistic simulations in Section VII. Finally, some conclusions are drawn in Section VIII, along with possible directions for future work.

II. State of the Art

It is possible to identify different research trends related to our work that we shortly summarise in this section.

a) Application model: An application model tightly related to the one presented in this paper is represented by synchronous data-flow networks [26]. As shown in [7], synchronous data flow networks lend themselves to an effective code generation process, in which an off-line schedule is synthesised that minimises the code length and the buffer size. The model used in this paper is a very restricted case of synchronous data-flow. Indeed, our tasks are organised in linear topologies, the production and the consumption rate are forced to be equal (one token) and we do not aim at an optimised off-line scheduling since our application is inherently distributed and dynamic. In this sense, our framework is closer to the one presented in several papers in the real-time community such as [19]. Interestingly, our scheduling technique allows us to keep in check the length of the buffers by controlling the delays of the different tasks in the pipeline.

b) Soft real-time scheduling: Different scheduling algorithms have been proposed to support the specific needs of soft real-time applications. A first important class approximates the Generalised Processor Sharing concept of a *fluid flow* allocation, in which each application using the resource marks a progress proportional to its weight. Among the algorithms of this class, we can cite Proportional Share [36] and Pfair [6].

Similar are the underlying principles of a family of algorithms as Resource Reservation schedulers [30]. In this case it is possible to associate to each task a pair (Q, P) meaning that the task is reserved a budget equal of at least Q time units every P. In essence, the task is reserved a fraction of the CPU $B = Q/P$. This approach has been successfully applied to the problem of scheduling several types of resources in the resource Kernels project [30]. A very effective implementation of the Resource Reservations is based on the use of aperiodic servers [2]. By using a small enough value for P we can approximate to a reasonable extent and with a controlled overhead [13] the fluid flow allocation of the scheduled resources assumed in our model (See Section III). Similar approximations can also be done using a proportional share scheduler by changing the weights and operating on the scheduling parameters. Therefore, both a resource reservation and a proportional share scheduler could be used as a basic layer for our adaptive scheme.

c) Design of distributed real-time applications: The problem of designing scheduling parameters for distributed real-time applications has received a constant attention in the past few years. Interesting methodologies can be found in [5], [16] for hard real-time transactions and in [22] for soft real-time applications. In all of these cases, the author come up with a static assignment of the scheduling parameters. In this paper, we make the point that a static design can be inefficient (or even impossible) when applications are time-varying. Therefore, we decompose end-to-end constraints into local *control goals* rather than into static parameters. The local controllers are then used to achieve the control goals operating on the scheduling parameters.

d) Adaptive QoS control: Many papers in the past few years propose to operate resource allocation based on a constant monitoring of the QoS experienced by the application. The application, in principle, can run almost unaware of the underlying ongoing adaptation. In [11], the authors propose to adjust the scheduling priorities to maximise the performance of a set of feedback controllers. Likewise, in the networking community adaptivity is expected to play an increasingly important role. An example is [15], where routing decisions are optimised based on a monitoring of the experienced QoS.

More similar to the framework considered in this paper are two recent papers [17], [21] based on the Q-RAM framework [29]. The idea is to associate each task with a custom utility function relating the resource consumption to the QoS. An on-line optimisation problem is then solved to compute the optimal assignment of resources. Our work differs in two main points. First, we restrict to a class of applications for which the QoS is solely related to the latency, whereas in the cost function advocated by the authors of QRAM several dimensions for the QoS can be considered in the cost function. Therefore, we trade generality for accuracy taking into account such aspects as the interaction of the different tasks composing an application. Secondly, algorithms based on Q-RAM typically entail heavy-weight computations whereas our control scheme requires a few algebraic operations and is then applicable on every task activation.

Focusing on soft real-time applications, in [24] the authors use an EDF scheduling algorithm trying to dynamically adapt the deadline miss ratio. In contrast, our QOS metric, the scheduling error, is related to the maximum delay accumulated with respect to the end-to-end deadline, rather than on the probability of missing the deadlines. This allows us to perform a very fine control on the finishing time of all activations of each task, based on a well-founded dynamical system model.

Very much related to our approach is the notion of real-rate scheduling, proposed in [34], [20], where a controller is used to regulate the progress rate of each task. The progress rate is defined as the difference between a time-stamp associated to a computation and the actual time this computation is performed at. An interesting generalisation of the real-rate approach to pipelines of tasks is in [35], where the authors consider a simplified model for the interactions of the tasks (which does not describe the behaviour of the system in overload conditions). In our case, we base our model on a QoS metric (the *virtual scheduling error*), which is different from the real-rate approach because it is referred to the execution of an entire job (rather than to the progress of the task). Moreover, the model is applicable in all workload conditions. Another related paper is [25], where the author considers a feedback control scheme to regulate the utilisation of distributed real-time systems. Our work differs in two respects. First, our purpose is to control the QoS rather than the utilisation (although one effect of the presented controller is to track the resource requirements of the task). Second, the control approach presented in [25] is centralised whereas the one presented in this paper is distributed.

The main ideas underlying our approach were first introduced in [3] and developed in [4], [28]. In these papers, the authors use a scheduling algorithm (the CBS [2]) based on an aperiodic server, which implements a resource reservation scheduling policy. The budget associated to each reservation is used as an actuator in an adaptive scheme. In these papers, the authors restrict to a single resource, whereas in this paper we generalise to pipelines of tasks. Preliminary results on the design of a task controller for strong QoS guarantees have been presented in [27].

Table I
NOTATION SUMMARY

Symb./abbr.		Meaning	Symb./abbr.		Meaning
$\mathcal{A}^{(i)}$	$\mathcal{A}$	i^{th} application	$n^{(i)}$	n	$\mathcal{A}^{(i)}$ length
$\mathcal{R}$		Resources set	$\mathcal{R}_r$		r^{th} resource
$\tau^{(j,i)}$	$\tau^{(j)}$	j^{th} task in $\mathcal{A}^{(i)}$	C_r		$\mathcal{R}_r$ capacity
$J_k^{(j,i)}$	$J_k^{(j)}$	k^{th} job of $\tau^{(j,i)}$	$c_k^{(j,i)}$	$c_k^{(j)}$	$J_k^{(j,i)}$ RUT
$a_k^{(j,i)}$	$a_k^{(j)}$	arrival-time of k^{th} token at $\tau^{(j,i)}$	$r_{-1}^{(r,i)}$	$r_{-1}^{(r)}$	tasks in $\mathcal{A}^{(i)}$ using $\mathcal{R}^{(r)}$
$d_k^{(j,i)}$	$d_k^{(j)}$	$J_k^{(j,i)}$ abs. deadl.	$b_k^{(j,i)}$	$b_k^{(j)}$	$J_k^{(j,i)}$ bw
$T^{(i)}$	T	$\mathcal{A}^{(i)}$ activ. period	$s_k^{(j,i)}$	$s_k^{(j)}$	start-time
$f_k^{(j,i)}$	$f_k^{(j)}$	$J_k^{(j,i)}$ finish time	$\sigma_k^{(j,i)}$	$\sigma_k^{(j)}$	start error
$D^{(i)}$	D	$\mathcal{A}^{(i)}$ end-to-end deadline	$r^{(j,i)}$	$r^{(j)}$	Resource of $\tau^{(j,i)}$
$B_N^{(j)}$	$B_N^{(j)}$	Saturation bw	$P_k^{(j,i)}$	$P_k^{(j)}$	Prediction range for $c_k^{(j,i)}$
$\xi_k^{(i)}$	ξ_k	activation jitter of $\mathcal{A}^{(i)}$	$[h_k^{(j,i)}, H_k^{(j,i)}]$	$[h_k^{(j)}, H_k^{(j)}]$	
$B_Q^{(j,i)}$	$B_Q^{(j)}$	Minimum bandw.	$\epsilon_k^{(j,i)}$	$\epsilon_k^{(j)}$	schedul. error

e) Architectures for QoS control: Many papers have considered the problem of QoS adaptation from a software architecture point of view. For instance, the use of adaptation techniques inside general-purpose Operating Systems is discussed in [31]. Some authors propose instead to perform resource adaptation in a middle-ware layer [38], [14], [18], [33]. The latter work evolves around the QuO [23] middleware framework, which is particularly noteworthy for it utilises the capabilities of CORBA to reduce the impact of QoS management on the application code. In this paper, we do not explicitly deal with software architectural issues, even though the practical feasibility and effectiveness of our techniques have been shown for the sole CPU resource in the context of the AQuoSA project [1], [13].

III. SYSTEM-LEVEL DESCRIPTION

a) Application model: We consider a finite set of applications $\left\{\mathcal{A}^{(i)}\right\}$. Each application $\mathcal{A}^{(i)}$ consists of a pipeline of $n^{(i)}$ tasks $\mathcal{A}^{(i)} = \left(\tau^{(1,i)}, \ldots, \tau^{(n^{(i)},i)}\right)$, which are pairwise connected by uni-directional buffers (in Section VI we discuss possible extensions). For the tasks, we adopt a data-flow model of computation as described in [26]. The k^{th} token arrives at the input buffer of task $\tau^{(j,i)}$ at time $a_k^{(j,i)}$, for being processed by job $J_k^{(j,i)}$ (k^{th} instance of $\tau^{(j,i)}$) that starts at time $s_k^{(j,i)}$ and finish at time $f_k^{(j,i)}$, instant in which the token is placed into the input buffer of the next task in the pipeline: $a_k^{(j+1,i)} = f_k^{(j,i)}$. See Table I for a notation summary.

A task can be implemented by an infinite loop that cyclically reads a token from the input buffer, processes the token and outputs the result onto the output buffer. When the input buffer is empty, the task is blocked waiting for a token to process.

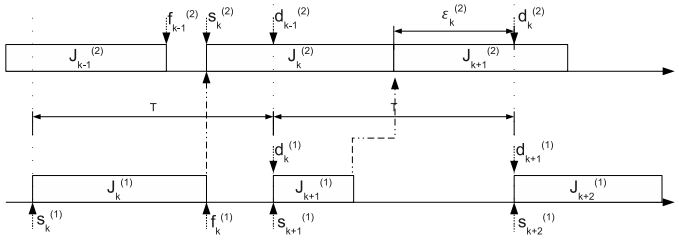


Figure 1. Example showing the activation pattern of the jobs and the notation.

This is simply implemented by a *blocking read* operation. In our model, write operations are non-blocking, and tokens cannot be lost. As shown next (1), the upper bounds on the computation delays introduced by our framework allow us to implement this loss-less and non-blocking semantics by using a limited number of elements in a FIFO queue.

For notational convenience, we will henceforth omit the i superscript when the discussion refers to a single application, as shown in the second column of Table I.

Example Figure 1 shows an example of the activation pattern of the jobs for a single application. The bottom line reports the execution of the first task in the pipeline, whose jobs are periodically activated. For the second task in the pipeline (top line), job $J_k^{(2)}$ cannot start until job $J_k^{(1)}$ finishes and produces the k^{th} token. On the contrary, $J_{k+1}^{(2)}$ starts right after $J_k^{(2)}$ finishes because $J_{k+1}^{(1)}$ has already terminated by that time and the $(k+1)^{th}$ token is already available.

b) Real-time constraints and QoS guarantees: For the purposes of this paper, we will restrict to periodic applications: the source feeds the tokens into the input buffer of the first task $\tau^{(1)}$ with a regular period T, as shown in the example in Figure 1. Actually, our framework tolerates a limited jitter on the time instant at which the k^{th} token enters the pipeline, with respect to the *ideal* time $(k-1)T$. It triggers a chain of computations that eventually produces its result at time $f_k^{(n)}$, n being the length of the pipeline. We require that this chain of computations be terminated within a maximum latency D: $f_k^{(n)} \leq a_k^{(1)} + D$.

For applications receiving *strong QoS guarantees*, this bound is invariably respected. For applications receiving *weak QoS guarantees*, having a latency lower than D is the nominal work condition; in response to a system overload, the delay can become larger, but it remains below an upper bound and when the overload finishes the nominal condition is restored.

c) Resources and Scheduling: For their execution, applications share a pool of resources $\mathcal{R} = \{\mathcal{R}_1, \ldots, \mathcal{R}_R\}$ of possibly different kind. Each task $\tau^{(j,i)}$ is assumed to perform its operations using a single resource[1] $r^{(j,i)}$ ($r_{-1}^{(r,i)}$ denotes the set of tasks in $\mathcal{A}^{(i)}$ that use resource $\mathcal{R}_r$). Every job $J_k^{(j,i)}$ of $\tau^{(j,i)}$ uses the resource for a Resource Usage Time (RUT) denoted by $c_k^{(j,i)}$ that generally changes for each job. This is the time needed by the job when it is allocated the entire resource capacity. For each resource, we assume the availability of a scheduler allowing one to decide the fraction $b_k^{(j,i)}$ devoted to job $J_k^{(j,i)}$.

[1]For resources different from the CPUs, we assume the overall computational overhead due to their management by the competing tasks is bounded and may be accounted for by considering a reduced total CPU capacity.

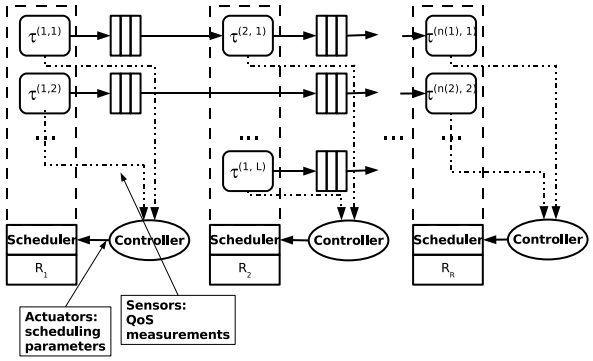


Figure 2. An example scenario of the addressed problem. Several applications are organised as pipelines of tasks sharing a pool of resources. Each resource is managed by a scheduler and by a controller that decides the scheduling parameters based on QoS measurements.

We allow for changes in the bandwidth $b_k^{(j,i)}$ on a job per job basis and even during the job. To preserve the properties of the scheduler, though, we require that the bandwidth assignments respect the following *schedulability condition*: at any point in time t, the total amount of resource allocated to jobs active at time t and using the resource, has to be lower than its capacity. Formally:

$$\forall \mathcal{R}_r \in \mathcal{R}, \forall t \sum_{\substack{i,j,k \,:\, r^{(j,i)} = r \\ \wedge\, s_k^{(j,i)} \leq t < f_k^{(j,i)}}} b_k^{(j,i)} \leq C_r. \quad (1)$$

The capacity $C_r \leq 1$ is related to the efficiency of the scheduling algorithm. For instance, scheduling solutions based on the Constant Bandwidth Server (CBS) [2] or on the Proportionate Fair (Pfair) [6] scheduling attain full resource utilisation (i.e., on a single processor, $C_r = 1$).

d) The control architecture: The control architecture proposed in this paper is described in Figure 2. Each application is horizontally represented as a chain of tasks, where each task is drawn on top of the resource it uses. For each resource, besides the scheduler, we can find an additional software component called `controller`, whose purpose is threefold: admission test, run-time task control, supervision.

When a new application requires admission into the system, it has to specify: the period T, the maximum latency D, and the minimum guaranteed bandwidth that each task composing the application requires for the resource it uses. As shown below, these levels can be chosen according to the required level of QoS (weak or strong). An application is admitted into the system if the guaranteed bandwidth its tasks require on each resource does not invalidate Condition 1. This way, the scheduler is able to operate properly and the negotiated QoS levels can be dynamically guaranteed to the applications. The latter claim results from the application of the control theory we developed, as detailed in Section V.

If the application is admitted into the system, the control algorithms hosted on the different controllers are configured and the different tasks can be started. The run-time task

controller interacts with the scheduler: using the data collected by the operating system upon each start and termination of a job, it formulates a bandwidth request for the next job. Finally, based on the bandwidth requests of the different task controllers, the resource supervisor decides the actual bandwidth allocations that respect Equation (1).

There are several architectural solutions for designing a middle-ware that implements this machinery. For instance, the admission test can be carried out in a centralised or in a distributed way. Moreover, the QoS specification of the application can be exposed to the application or externally attached to it by a wrapping middle-ware component (as shown in [14]). A critical issue is how to notify the start and/or the termination of a job to the framework. We simply propose that the task explicitly performs some calls at the start and end of each job, in the spirit of the architecture proposed in [13]. In this paper, we do not make any specific commitment to any architectural option. Rather, we focus on the description of the QoS control strategy and its formal properties.

IV. Formulating the control problem

In order to safely define a control strategy for this system (as shown in Section V), we need a dynamic model. To this end, we use a state-space approach, where each application is associated with a difference equation:

$$\boldsymbol{\epsilon}_{k+1} = f(\boldsymbol{\epsilon}_k, \mathbf{c}_k, \mathbf{b}_k) \tag{2}$$

where $\boldsymbol{\epsilon}_k = [\epsilon_k^{(1)}, \ldots, \epsilon_k^{(n)}]^T$ is a vector of state variables, $\mathbf{c}_k = [c_k^{(1)}, \ldots, c_k^{(n)}]^T$ is the vector of external (non controllable) inputs, $\mathbf{b}_k = [b_k^{(1)}, \ldots, b_k^{(n)}]^T$ is the vector of command variables. The symbol k refers to the k^{th} token entering the pipeline. The non-controllable inputs are the RUTs required when processing the token through the different stages of the pipeline. The vector of the command variables consists of the bandwidth values used on the different resources $\mathcal{R}^{(1)}$, $\mathcal{R}^{(2)}$, …, $\mathcal{R}^{(n)}$ for the jobs generated by the k^{th} token.

In order to substantiate this model, we need to: 1) identify a suitable vector of the state variables for the system, 2) formulate the control goals by translating the QoS specifications into a desired equilibrium condition for the state-space trajectories, 3) identify the state transition function f, which analytically describes how the command variables affect the evolution of the system. Control design amounts to designing a control law that operates on the input variables $\mathbf{b}_k$ to keep the system in the desired equilibrium and restore it in case of perturbations.

a) The state variables: In order to quantify the temporal evolution of a task, we introduce for each job a *temporal reference* $t_k^{(j)}$. Because of the periodic activation of the pipeline, we consider periodically spaced out temporal references: $t_k^{(j)} = (k+j-1)T$. Now, we define the *scheduling error* as a state variable that quantifies the distance of the job termination from the temporal reference:

$$\epsilon_k^{(j,\,i)} \triangleq f_k^{(j,\,i)} - t_k^{(j,\,i)}. \tag{3}$$

The arrival times of the tokens at the first stage of the pipeline are assumed to be equal to an ideal time $(k-1)T$, plus a jitter ξ_k (which is 0 for perfectly periodic arrivals): $a_k^{(1)} \triangleq (k-1)T + \xi_k$. Contrary to $\epsilon_k^{(j)}$, ξ_k is not actually part of the system state. In fact, it is an exogenous term depending on the evolution of the surrounding environment.

b) Dynamic model: The evolution of a single pipelined application can be described by a discrete event model. Consider the k^{th} token produced by the data source, which determines the subsequent activation of the k^{th} job for all the tasks in the pipeline $J_k^{(1)}, \ldots, J_k^{(n)}$. The pipeline state can be described by a vector of variables $\boldsymbol{\epsilon}_k$ where each component is the scheduling error $\epsilon_k^{(j)}$ that the j^{th} element of the pipeline experiences when it processes the token: $\boldsymbol{\epsilon}_k \triangleq \left[\epsilon_k^{(1)}, \ldots, \epsilon_k^{(n)}\right]^T$. Due to the "fluid flow" allocation model of the scheduler, and assuming for simplicity a constant bandwidth $b_k^{(j)}$ throughout the execution of the job, the time instant $f_k^{(j)}$ in which $J_k^{(j)}$ will write its output into the output buffer is simply given by $f_k^{(j)} = s_k^{(j)} + \frac{c_k^{(j)}}{b_k^{(j)}}$. Since the k^{th} job of $\tau^{(j)}$ can start only after both the $(k-1)^{th}$ job of $\tau^{(j)}$ and the k^{th} job of $\tau^{(j-1)}$ have been completed (performing their write operations into the buffers), we can write $s_k^{(j)} = max\{f_{k-1}^{(j)}, f_k^{(j-1)}\}$ (for the first stage $s_k^{(1)} = max\{f_{k-1}^{(1)}, a_k^{(1)}\}$).

The resulting evolution model for the state of each task $\tau^{(j)}$ is described by the following recursive equations:

$$\epsilon_k^{(j)} = \sigma_k^{(j)} + \frac{c_k^{(j)}}{b_k^{(j)}} - T, \text{ with} \tag{4}$$

$$\sigma_k^{(j)} = \begin{cases} \max\left\{\epsilon_k^{(j-1)},\, \epsilon_{k-1}^{(j)}\right\} & j \geq 2,\, k \geq 2 \\ \epsilon_1^{(j-1)} & j \geq 2,\, k = 1 \\ \max\left\{\epsilon_{k-1}^{(1)},\, \xi_k\right\} & j = 1,\, k \geq 2 \\ \xi_1 & j = 1, k = 1 \end{cases} \tag{5}$$

where, for notational convenience, we introduced the *start error* $\sigma_k^{(j)} \triangleq s_k^{(j)} - t_{k-1}^{(j)}$ as the difference between the start time of $J_k^{(j)}$ and the temporal reference $t_{k-1}^{(j)}$. The function above substantiates with a precise and well-founded expression the general model described in Equation (2), which we will use in the sequel whenever we need a compact expression for the state evolution. As a consequence of this formulation, the set in which each state variable ranges is $\epsilon_k^{(j)} \geq \inf_k \xi_k - jT$.

The model is easily generalisable to the case in which the bandwidth allocated to a task can be changed during a job. For instance, if job $J_k^{(j)}$ executes for $c_{k,\,1}^{(j)}$ units of time with bandwidth $b_{k,\,1}^{(j)}$, for $c_{k,\,2}^{(j)}$ units of time with bandwidth $b_{k,\,2}^{(j)}$, …, for $c_{k,\,s}^{(j)}$ units of time with bandwidth $b_{k,\,s}^{(j)}$, with $\sum_{f=1}^{s} c_{k,\,f}^{(j)} = c_k^{(j)}$, then its scheduling error is computed as:

$$\epsilon_k^{(j)} = \sigma_k^{(j)} + \sum_{f=1}^{s} \frac{c_{k,\,f}^{(j)}}{b_{k,\,f}^{(j)}} - T = \sigma_k^{(j)} + \frac{c_k^{(j)}}{\tilde{b}_k^{(j)}} - T$$

where we introduced the *equivalent bandwidth* $\tilde{b}_k^{(j)}$ as the weighted harmonic average of the used bandwidth values: $\tilde{b}_k^{(j)} \triangleq \sum_{f=1}^{s} c_{k,\,f}^{(j)} / \sum_{f=1}^{s} (c_{k,\,f}^{(j)} / b_{k,\,f}^{(j)})$.

c) Control goals: In typical real-time distributed applications the end-to-end deadline D is decomposed into partial deadlines for the different tasks of the pipeline. The choice of these deadlines is orthogonal to the problem considered in this paper. In our setting, the choice of the deadlines for each task simply corresponds to a set point $\overline{\boldsymbol{\epsilon}} = [\overline{\epsilon}^{(1)}, \overline{\epsilon}^{(2)}, \ldots, \overline{\epsilon}^{(n)}]$, for the state variables. If $D \geq nT$, a convenient choice for the deadlines are the temporal references $d_k^{(j)} = t_k^{(j)}$. In this case the set point for the state variable is simply $\overline{\boldsymbol{\epsilon}} = 0$. Maintaining the state in a neighbourhood of the set point allows us to combine the respect of the end to end latency constraint with an efficient utilisation of resources (finishing the job too early w.r.t. its deadline corresponds to a waste of resources).

For a given set point, an ideal feedback law is a relation $\mathbf{b}_k = u(\boldsymbol{\epsilon}_k)$ such that $\overline{\boldsymbol{\epsilon}} = f(\overline{\boldsymbol{\epsilon}}, \mathbf{c}_k, u(\overline{\boldsymbol{\epsilon}}_k))$. However, this goal cannot realistically be achieved for two reasons: 1) we do not have an *a priori* knowledge of $c_k^{(j)}$, at the time each bandwidth $b_k^{(j)}$ is decided; 2) the limitation of the resources restricts the possible choices of the scheduling parameters.

We cope with the first problem by introducing a variability range $P_k^{(j)}$ for each $c_k^{(j)}$ variable, which is dynamically provided by an external component, the *predictor*. We will denote by $\mathbf{P}_k$ a vector of intervals such that $c_k^{(j)} \in P_k^{(j)}$. Hence, the control value is decided on the basis of the current state *and* of the expected variability range for $\mathbf{c}_k$. In control theoretical terms we have an "unknown-but-bounded" disturbance term.

Concerning the resource limitation, we assume that the controller can rely on a minimum guaranteed bandwidth $B_O^{(j)}$ on each resource $\mathcal{R}_{r^{(j)}}$ (that may at most equal the resource capacity $C_{r^{(j)}}$), for a given application. Also, $\mathbf{B}_O = \left[B_O^{(1)}, \ldots, B_O^{(n)}\right]^T$ will denote the vector of these minimum bandwidths. As detailed in Section V-E, these assumptions can be enforced by an appropriate admission control policy.

Because of the uncertainty in the prediction and of the resource limitation, our desired equilibrium is stretched from the set point $\overline{\boldsymbol{\epsilon}}$ to a (small) set containing it, whose size is related to the tightness of the prediction. This is illustrated through the conceptual trajectories depicted in Figure 3: for applications receiving strong QoS guarantees, we require that the state trajectories (marked as (a)) be always contained in a set $\mathbf{I}$ (the inner rectangle); for applications receiving weak QoS guarantees, the state trajectories (marked as (b)) are allowed to leave the equilibrium set, but they have to be bounded in a larger enclosing set (the outer rectangle). Also, we allow an application originally accepted with a weak QoS guarantee to be switched to a strong guarantee if the workload of the system decreases (e.g., due to the termination of an application). In this case we need the ability to drive the system state from the larger set into the inner one (trajectory marked as (c)).

We can express formally these concepts, adapting from [8];

Definition 1: $\mathbf{L}$ is robustly invariant from $\mathbf{I}$ for the closed loop system if $\forall k_0$, if $\boldsymbol{\epsilon}_{k_0} \in \mathbf{I}$ then $\boldsymbol{\epsilon}_k \in \mathbf{L}\ \forall k \geq k_0$. If $\mathbf{L} = \mathbf{I}$, then we simply say that $\mathbf{I}$ is a robustly invariant set (RIS).

Definition 2: Given a natural number H the set $\mathbf{I}$ is robustly H-steps *attractive* from $\mathbf{L}$ if $\forall k_0$, if $\boldsymbol{\epsilon}_{k_0} \in \mathbf{L}$ then $\boldsymbol{\epsilon}_k \in \mathbf{L}$ $\forall k \in [k_0 + 1,\ k_0 + H - 1]$ and $\boldsymbol{\epsilon}_{k_0+H} \in \mathbf{I}$.

The definitions translate the different type of QoS guarantees introduced at the beginning of the section (see Figure 3) in terms of closed loop stability properties. Namely, the set invariance of $\mathbf{I}$ describes the desired behaviour of an application with strong QoS guarantee, while the invariance of $\mathbf{L}$ from $\mathbf{I}$ corresponds to the weak QoS guarantee. Finally, the attractivity of $\mathbf{I}$ from $\mathbf{L}$ is needed to allow transition from weak to strong guarantee. We qualify these properties by the adjective "robust", because they are must hold despite all possible variations of $\mathbf{c}_k$ inside $\mathbf{P}_k$.

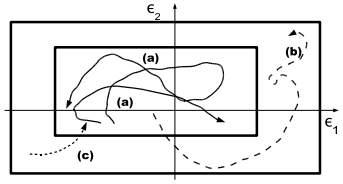


Figure 3. Example trajectories. (a) Application with strong QoS guarantee: the state is constrained in a neighbourhood of the origin. (b) Application with weak QoS guarantee. (c) Transition from weak to strong QoS guarantee.

d) Design parameters: Given our notion of stability, the specification of the control goals corresponds to the specification of the sets $\mathbf{I}$ and $\mathbf{L}$ and of the number of steps H.

As far as the geometry of $\mathbf{I}$ and $\mathbf{L}$ is concerned, we will restrict to hyper-rectangles so that the scheduling error of each stage of the pipeline is constrained in an interval: $\mathbf{I} \triangleq \left[-e^{(i)}, E^{(i)}\right] \times \ldots \times \left[-e^{(n)}, E^{(n)}\right]$, where $\overline{\epsilon}^{(i)} \in \left[-e^{(1)}, E^{(1)}\right]$, and $\mathbf{L} \triangleq \left[-l^{(1)}, L^{(1)}\right] \times \ldots \times \left[-l^{(n)}, L^{(n)}\right]$, with $-l^{(i)} \leq -e^{(i)} \leq E^{(i)} \leq L^{(i)}$.

To understand the convenience of this geometry, consider an application with strong QoS guarantee and set point $\overline{\boldsymbol{\epsilon}} = 0$, and choose the set $\mathbf{I}$ so that, $\forall j$, $e^{(j)} = e > 0$ and $E^{(j)} = E > 0$. By choosing the extremal point E, we can decide the maximum delay that each stage of the pipeline can suffer, which is obviously related to the acceptable QoS level.

The extremal point e is associated to resource consumption: a large value for e means that the task has received too much bandwidth leading to an early completion. Also the measure of the segment $[-e,\ E]$ has an important significance. Indeed, by choosing a small segment we can reduce the output jitter of the application. More importantly, the extremal points of the interval can also be related to the amount of memory required for the buffers, as shown in the following:

Theorem 1: Consider a pipeline consisting of n tasks $\tau^{(1)}, \ldots, \tau^{(n)}$, and assume that the jitter in the production of the first token is always smaller than the period $|\xi_k| \leq T/2, \forall k$. The following statements hold true: I) If $\forall j, \forall k, \epsilon_k^{(j)} \leq E > 0$, then the number of tokens at the j^{th} buffer of the pipeline is upper bounded by $\left\lceil \frac{E}{T} \right\rceil + j + 1$, and the bound is tight (in the sense that there are cases in which it is attained). II) If $\forall j, \forall k, -e \leq \epsilon_k^{(j)} \leq E$, (with $e > 0$ and $E > 0$) then the number of tokens at the j^{th} stage of the pipeline is upper bounded by $\left\lfloor \frac{e+E}{T} \right\rfloor + 2$.

Proof: See Appendix. ■

This result is interesting in many respects. The first information it carries is that if we are able to contain the scheduling error of the different stages below a bound E, then we are able to use a finite number of buffers (for which the theorem

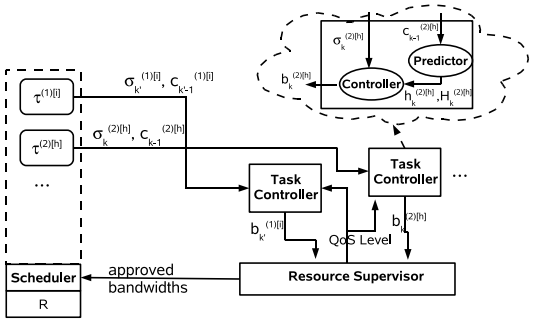


Figure 4. The control scheme adopted in this paper. We zoom in one task controller to show its internal structure.

provides a tight bound) preserving the properties of our model of computation (which is based on non-blocking writes). The second important fact is that, if we are also able to impose a lower bound on the scheduling error then, even with a moderate length of the pipeline, we can have substantial memory savings as compared to the case in which only the upper-bound E is imposed.

V. Control design

A. Generalities

The system described in this paper is comprised of different software applications that evolve independently and asynchronously from each other. Moreover, the different tasks of a single application are loosely synchronised due to the use of intermediate buffers. Therefore, the different components of the state vector are measured at different times (in the state vector $\boldsymbol{\epsilon}_k$, index k refers to a token and not to a time instant). These considerations dictate a decentralised control scheme, as shown in Figure 4: each *resource controller* consists of a *supervisor* and of a collection of *task controllers*.

Control decisions are taken at the *start time of a new job* $J_k^{(j)}$ according to the following scheme: 1) the task controller acquires the start time $\sigma_k^{(j)}$ and the RUT $c_{k-1}^{(j)}$ of the previous job, 2) the predictor uses $c_{k-1}^{(j)}$ to update its state and produce the range $P_k^{(j)} \triangleq \left[h_k^{(j)}, H_k^{(j)}\right]$ in which $c_k^{(j)}$ is expected to fall, 3) the task controller applies the control algorithm computing a range of acceptable bandwidths $\mathcal{U}_k^{(j)} \triangleq \left[\mathcal{B}_L^{(j)}(\sigma_k^{(j)}, P_k^{(j)}), \mathcal{B}_H^{(j)}(\sigma_k^{(j)}, P_k^{(j)})\right]$ for $J_k^{(j)}$; this request remains *active* until the end of the job; 4) the supervisor assigns a bandwidth $b_k^{(j)}$ to $J_k^{(j)}$ and possibly changes the bandwidth allocated to the other tasks respecting Condition (1) and ensuring that a minimum bandwidth $B_O^{(j)}$ is granted *whenever* requested. Our scheme is decentralised since resource supervisors do not communicate with each other, and so task controllers do (the only communication is between the task controllers operating on a resource and the supervisor that manages it). As shown later, despite this loose interaction between the components, their actions fulfils system-wide control goals.

In the sequel we will primarily focus on the design of the task controllers, whilst we will only show a simple supervisor design, reserving an evaluation of alternative possibilities for future work. Likewise, the prediction techniques for the time series of RUTs is mostly out of the scope of this work. For the sake of completeness we will provide a short description of the algorithms we used in Section VII-c.

B. The task controllers

The first point to address for the design of the task controller is how to produce a range of bandwidth values $\mathcal{U}_k^{(j)} \triangleq \left[\mathcal{B}_L^{(j)}(\sigma_k^{(j)}, P_k^{(j)}), \mathcal{B}_H^{(j)}(\sigma_k^{(j)}, P_k^{(j)})\right]$, such that choosing the bandwidth inside this range, robust invariance of $\mathbf{I}$ is achieved. Since it does not make sense for a task controller to choose a bandwidth greater than the capacity $C_{r^{(j)}}$, we introduce a saturation constraint $\mathcal{B}_H^{(j)} \leq B_N^{(j)}$, with $B_N^{(j)} \leq C_{r^{(j)}}$.

To simplify the statement of the theorems and their proofs, we make some assumptions. The first one is that the set $\mathbf{I}$ is an hypercube, i.e., the target variability range for the scheduling error of all tasks in the pipeline is a common interval $[-e,\, E]$. The second assumption is that $e + E \leq T$, which is perfectly reasonable as we want to constrain the scheduling error into a small set. Finally, we assume that the activation jitter is also constrained within the interval $[-e,\, E]$. These assumptions simplify the statements of the theorems below and their proofs. Under these assumptions, a controller achieves robust invariance if and only if it chooses the bandwidth in the range $\left[\mathcal{B}_L^{(j)}, \mathcal{B}_H^{(j)}\right]$:

$$\mathcal{B}_L^{(j)} = \begin{cases} \frac{H_k^{(j)}}{T+E-\sigma_k^{(j)}} & \text{if } -e \leq \sigma_k^{(j)} \leq T + E - \frac{H_k^{(j)}}{B_N^{(j)}} \\ B_N^{(j)} & \text{if } \sigma_k^{(j)} > T + E - \frac{H_k^{(j)}}{B_N^{(j)}} \end{cases}$$
$$\mathcal{B}_H^{(j)} = \begin{cases} \frac{h_k^{(j)}}{T-e-\sigma_k^{(j)}} & \text{if } -e \leq \sigma_k^{(j)} \leq T - e - \frac{h_k^{(j)}}{B_N^{(j)}} \\ B_N^{(j)} & \text{if } \sigma_k^{(j)} > T - e - \frac{h_k^{(j)}}{B_N^{(j)}} \end{cases} \tag{6}$$

This is shown in the following:

Lemma 1:
I) A control algorithm attaining robust invariance of $\mathbf{I} = [-e,\, E]^n$ with $e + E \leq T$ exists if and only if:

$$e + E \geq T\left(\frac{1-\alpha}{\alpha}\right), \tag{7}$$

with $\alpha \triangleq \min_{j=1,\ldots,n}\left\{\inf_k\left\{\alpha_k^{(j)}\right\}\right\}$, $\alpha_k^{(j)} \triangleq \frac{h_k^{(j)}}{H_k^{(j)}}$.
II) Any such algorithm must assign bandwidth values in the range defined by Equation (6) with

$$B_N^{(j)} \geq \frac{\sup_k H_k^{(j)}}{T} \equiv \frac{\tilde{H}^{(j)}}{T}, j = 1, \ldots, n. \tag{8}$$

This range is non-empty under conditions in Equation (7).

Proof: See appendix. ∎

Lemma 1 provides conditions for the correctness of the task controller behaviour when it operates in isolation and all its bandwidth requests can be granted. When task controllers are used in combination with resource supervisors, different QoS guarantees are be provided depending on the guaranteed bandwidths $\mathbf{B}_O^{(i)} = [B_O^{(1,\,i)}, \ldots, B_O^{(n^{(i)},\,i)}]$. In particular, in Theorem 2 and in Theorem 3 we will provide a lower bound for the components of vector $\mathbf{B}_O^{(i)}$ to provide respectively strong and weak QoS guarantee, while in Theorem 4 we will show how to switch among them.

C. Global QoS guarantees

The following result clarifies how the task controller and the supervisor, if configured with appropriate parameters, may interact achieving the robust invariance of a set $\mathbf{I} = [-e, E]^n$.

Theorem 2: Consider an application controlled by the set of task controllers defined by Equation (6), acting under the action of resource supervisors granting a set of minimum guaranteed bandwidths of $\{B_O^{(j)}\}$. Then, robust invariance of $\mathbf{I} \triangleq [-e, E]^n$ is attained iff, in addition to Conditions (7) and (8), the following holds:

$$B_O^{(j)} \geq \frac{\tilde{H}^{(j)}}{T},\ j = 1, \ldots, n. \tag{9}$$

Proof: This result is a rather straightforward implication of Lemma 1. Indeed, any bandwidth assignment in the range identified by Equation (6) guarantees robust invariance of $\mathbf{I}$. In order to guarantee that the supervisor always allows for such an assignment, the minimum bandwidth guaranteed $B_O^{(j)}$ needs to be greater than the saturation value $B_N^{(j)}$ identified in the Lemma, which leads straight to Condition (9). ■

Remark 1: Condition (6) identifies a family of stabilising control laws. The supervisor can choose one element of this family according to different trade-offs. For instance, the value $\mathcal{B}_H^{(j)}$ is the most resource demanding choice but it also tolerates small errors in the estimation of $H_k^{(j)}$ preserving the upper bound of E for $\epsilon_k^{(j)}$. On the other hand, the value $\mathcal{B}_L^{(j)}$ is the choice that most likely tolerates small errors in the estimation of $h_k^{(j)}$ preserving the lower bound of $-e$ for $\epsilon_k^{(j)}$, keeping the allocation of bandwidth at the bare minimum required to achieve the desired QoS goals. The middle point of the range represents a good trade-off between robustness against imprecise estimation of both $c_k^{(j)}$ bounds and $B_O^{(j)}$.

Remark 2: It is easy to show that robust invariance of $\mathbf{I}$ is preserved also if the supervisor changes the value of the bandwidth during the job execution, as long as its value always belongs to the $\left[\mathcal{B}_L^{(j)}, \mathcal{B}_H^{(j)}\right]$ range identified by Condition (6). This is a straightforward consequence of the notion of equivalent bandwidth introduced in Section IV-b. The same remark also applies to the results presented below.

If the minimum bandwidth guaranteed to a task by the supervisor is lower than prescribed by Condition (9), then robust invariance cannot be guaranteed any more, since the condition of the theorem is necessary. However, robust invariance of $\mathbf{L}$ from $\mathbf{I}$ may still be preserved, as shown in the following:

Theorem 3: Consider an application controlled by the set of task controllers defined by Equation (6), acting under the action of resource supervisors granting a set of minimum guaranteed bandwidths of $\{B_O^{(j)}\}$. If, in addition to Conditions (7) and (8), the following ones are met for an integer M:

$$B_O^{(j)} \geq \frac{\sup_k \frac{1}{M} \sum_{h=0}^{M-1} c_{k+h}^{(j)}}{T} \equiv \frac{\tilde{c}_M^{(j)}}{T},\ j = 1, \ldots, n, \tag{10}$$

then Robust Invariance of $\mathbf{L} \triangleq \left[-e,\ L^{(1)}\right] \times \ldots \times \left[-e,\ L^{(n)}\right]$ from $\mathbf{I} \triangleq [-e,\ E]^n$ is ensured, with $L^{(j)} \triangleq E + j(M-1)T$.

Proof: See Appendix. ■

Based on this result, a weak QoS guarantee can be provided if the minimum guaranteed bandwidths exceed the upper bounds of the average resource requirements over a moving time horizon M. The maximum deviation from $\mathbf{I}$ is proportional to M and to the pipeline length.

D. Transitions between guarantee types

The transition from strong to weak guarantees takes place as a result of the supervisor decreasing the minimum guaranteed bandwidth from the values required in Theorem 2 to the one required in Theorem 3. In this case, the continuity between the two levels of guarantees derives from the very definition of $\mathbf{L}$-invariance. The reverse transition, from weak to strong guarantees, is far less obvious. The next states sufficient conditions under which the set $\mathbf{I}$ is robustly attractive from an outer set $\mathbf{L}$, assuming that at some point in time the available bandwidth switches from the value required in Theorem 3 to the one required in Theorem 2.

Theorem 4: Let the assumptions of Theorem 2 hold true for the vector of guaranteed values $B_O^{(j)}$, assume $\theta \triangleq M\left(T - \max_j \frac{\tilde{c}_M^{(j)}}{B_O^{(j)}}\right) > 0$, and let $\tilde{c}_M^{(j)}$ denote the supremal of the moving averages of M consecutive $c_k^{(j)}$ samples: $\tilde{c}_M^{(j)} \triangleq \sup_k \frac{1}{M} \sum_{h=1}^{M} c_{k+h}^{(j)}$. Then, $\mathbf{I} = [-e, E]^n$ is m-steps attractive from $\mathbf{L} = [-e, L]^n$, with $L \geq E$ and $m \geq M \left\lceil \left(L - E - T + \max_j \frac{\tilde{H}^{(j)}}{B_O^{(j)}}\right)/\theta + n \right\rceil$.

Proof: See Appendix. ■

The key point of the theorem above is that, starting from a scheduling error greater than E, if we only assume that the guaranteed bandwidth $B_O^{(j)}$ is set as required in Theorem 2, then the scheduling error can not increase but it is not guaranteed to decrease. This situation occurs in the worst-case scenario in which the RUTs are all set equal to $c_k^{(j)} = H_k^{(j)} = \tilde{H}^{(j)}$. In contrast, the condition identified in the theorem above ($\theta > 0$) ensures that over a time horizon M the average availability of bandwidth strictly exceeds the requirements, thus guaranteeing a reduction of the scheduling error of an amount lower bounded by θ itself. Thanks to this property, this last theorem is able to identify the number of steps m required to move back into $\mathbf{I}$, that clearly increases with the gap $L - E$ (i.e., the distance to cover), with the length of the time horizon M and with the length of the pipeline. On the contrary, increasing θ accelerates the convergence.

E. Resource supervisors

In order to provide QoS guarantees to an application, each resource supervisor has to be able to guarantee a minimum bandwidth whenever a task requires it. The strategy proposed here is based on the combination of a *negotiation phase* and of a *run-time policy*. In the negotiation phase, repeated every time a new application enters or exits the system, the supervisor decides whether an application can be admitted to the system and with what guarantee type. The run-time policy decides how to share the bandwidth among conflicting requests.

Negotiation phase Consider the set of applications

$\mathcal{G} = \left\{\mathcal{A}^{(1)}, \mathcal{A}^{(2)} \ldots \mathcal{A}^{(L)}\right\}$ already running into the system, along with the vectors of bandwidths guaranteed to them: $\left\{\mathbf{B}_O^{(i)}\right\}_{i=1,\ldots,L} = \left\{\left[B_O^{(1,\,i)}, \ldots, B_O^{(n^{(i)},\,i)}\right]\right\}_{i=1,\ldots,L}$. A new application $\mathcal{A}^{(L+1)}$ may be accepted into the system only if the vector of $\mathbf{B}_O^{(L+1)}$, chosen according to Theorem 2 if strong guarantees are required, or to Theorem 3 if only weak guarantees are required, satisfies the following *admission test*:

$$\forall \mathcal{R}_r \in \mathcal{R}, \sum_{i \in \{1,2,\ldots,L+1\}, j \in r_{-1}^{(r,\,i)}} B_O^{(j,\,i)} \leq C_r.$$

We can also admit applications with no QoS guarantee at all (setting $\mathbf{B}_O^{(i)} = 0$), for which the performance depends on the system workload and on the ability of the feedback to reclaim the bandwidth unused by the guaranteed applications.

The information on the minimum guaranteed bandwidth required by each application can derive from prior executions or from trial benchmarking runs. The decision on the type of guarantee that each application can receive can be static, or dynamic. In the latter case, for applications accepted with weak guarantees, it is possible to raise their guarantee as long as the workload of the system permits it.

Run-time supervision policy

Assume, for the sake of simplicity, that at most one task in each application can use a resource $\mathcal{R}_r$ and let $\mathcal{G}_r$ be the set of indices associated to applications that use it. With a slight abuse of notation, we can say that the controller associated to each task $r_{-1}^{(r,\,i)}$ in $\mathcal{A}^{(i)}$ using $\mathcal{R}_r$ requires at time t a bandwidth in the range $\left[\mathcal{B}_L^{(i)}(t), \mathcal{B}_H^{(i)}(t)\right]$ and is guaranteed a minimum bandwidth $B_O^{(i)}$. The supervisor has to guarantee to $\mathcal{A}^{(i)}$ at least $b_{min}^{(i)}(t) \triangleq \min\left\{B_O^{(i)}, \mathcal{B}_L^{(i)}(t)\right\}$. Let the residual bandwidth $b_r(t)$ be defined as $b_r(t) \triangleq C_r - \sum_{i \in \mathcal{G}_r} b_{min}^{(i)}(t)$. The bandwidth assigned to each task is then $b^{(i)}(t) \triangleq b_{min}^{(i)}(t) + \min\left\{\mathcal{B}_H^{(i)}(t) - b_{min}^{(i)}(t), \frac{\mathcal{B}_H^{(i)}(t) - b_{min}^{(i)}(t)}{\sum_{j \in \mathcal{G}_r}(\mathcal{B}_H^{(j)}(t) - b_{min}^{(j)}(t))} b_r(t)\right\}$. Therefore, each task receives a bandwidth of at least $b_{min}^{(i)}(t)$. In absence of overload ($\sum_{j \in \mathcal{G}_r}(\mathcal{B}_H^{(j)}(t) - b_{min}^{(j)}(t)) \leq C_r$), each task is granted the entire extra request $\mathcal{B}_H^{(j)}(t) - b_{min}^{(i)}(t)$, while, in case of overload ($\sum_{j \in \mathcal{G}_r}(\mathcal{B}_H^{(j)}(t) - b_{min}^{(j)}(t)) > C_r$), the extra requests $\{\mathcal{B}_H^{(j)}(t) - b_{min}^{(i)}(t)\}$ of all tasks are simply scaled down according to the available residual bandwidth on the resource $b_r(t)$. In any case, the bandwidth received by the task is in the range $\left[b_{min}^{(i)}(t), \mathcal{B}_H^{(i)}(t)\right]$.

F. Scalability

The approach that we propose is inherently scalable for different reasons. First, in our decentralised approach, each controller focuses on the evolution of a single task in isolation. Therefore, its complexity is not affected by the number of stages in the pipeline. The computational load of a task controller is very low, most of its complexity actually residing in the predictor. As discussed in the simulation section, it is possible to obtain very good results even with predictors requiring no more than a few algebraic operations. Second, the resource supervisors are distributed and, in their run-time evolutions, they only consider the requests of the tasks running on a specific node. As discussed for a single resource in [12], effective supervision algorithms can be implemented with constant complexity with respect to the number of tasks.

VI. Extensions of the approach

The assumptions made in Section V-B on the choice of the intervals $\left[-e^{(j)}, E^{(j)}\right]$ do not invalidate the applicability of the approach to more general cases, but make the statement of the theorems and their proofs much easier to read.

As far as the model of the application is concerned, the results that we found on the pipelines, can be generalised to other types of topologies. First of all, consider a tree topology, directed from the root to the leaves. The root receives the input token with a periodicity T, and each father, upon the termination of a job, forwards the token to all of its children. Each path from the root to a leaf can be thought of as a pipeline associated to a maximum latency constraint. The model in Equation (4) is still applicable. In order for our control strategy to be applicable as well, we have to be able to choose a partial deadline, for each task, which is compatible with the end-to-end latency of all the paths traversing the task. This way, we can consistently choose a target set $[-e, E]$ for the scheduling error of each task and apply the control strategy presented above (along with the analysis in Theorems 2, 3 and 4). In the example in Figure 5.(a), if the latency from R to $N2$ is greater than $2T$, the latency from R to $N4$ is greater than $3T$ and the one from R to $N5$ is greater than $4T$, we can choose a set point for the scheduling error equal to 0 for all tasks and a common target set $[-e, E]$ for all.

Similar generalisations can be done for some Direct Acyclic Graph (DAG) topologies with split and join points (see Figure 5.(b), Figure 5.(c)). In this case the semantic of a split point is the same as for the tree topology. The semantics of a join point is that the task can be started if all incoming jobs have produced their result. Therefore, our model may be extended by adapting the definition of the start-time of a task in Equation (5), to account for the fact that a task joining multiple branches has to wait for the completion of the k^{th} activation of all of them.

Concerning the algorithms for QoS control, if the concurrent branches generated from a split point have all equal length, then our results are easily generalisable. For instance, in the example in Figure 5.(b), we can choose the same partial deadlines for tasks $N1$ and $N2$ and the same target set for the scheduling errors of both. When the length of the concurrent branches is not balanced, the generalisation is not immediate. For instance, in Figure 5.(c), $N2$ and $N3$ need to execute at the same timing as $N1$, thus their partial deadlines need to be shorter. This case would need a generalisation of our methodology for dealing with tasks with different deadlines and target sets. This is outside of the scope of this paper.

VII. Simulation Results

a) Simulation environment and model validation: We simulated a mix of multimedia applications, consisting of

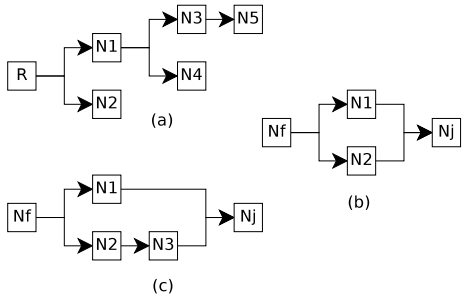


Figure 5. Examples of tree topology (a), DAG topology with balanced (b) and unbalanced (b) concurrent branches.

video decoders and MIDI sound synthesisers. The simulation of the video decoder has been made by modelling the video-path of an MPEG2 decoder as a two-stages pipeline: the first one acquires frames from the disk or from the network and consumes a negligible amount of CPU and the second one decodes them through CPU-intensive computations, drawing the decoded frames on the screen. This model is a simplification of real applications such as the `ffplay` video player. The MIDI sound synthesiser has been modelled as a two-stages pipeline, where the first stage acquires MIDI events and builds the corresponding sound frames, while the second stage records them on the disk, or sends them through the network to the playback device. This second stage was characterised by a fixed frame size, so there was no need to actually attach a controller to it, therefore its performance is not shown in the experiments reported below.

A very interesting aspect of the simulated workloads is the substantial fluctuation of the resource usage times, due to the use of compression for the video decoder, and of a variable number of active voices for the MIDI synthesiser.

In order to perform the simulation, we constructed a discrete-event simulator that replicates the evolution of the model described in Section IV, alongside of the control architecture described in Section V.

The adherence of such model with the behaviour of a real Reservation Based scheduler was validated, considering the CPU reservations as implemented within the AQuoSA architecture for the Linux kernel. We considered a periodic task with a period of $40ms$ and with a pseudo-random execution time ranging between 12.5% and 25% of the period. First, we executed 1000 jobs of the task at real-time priority, collecting the sequence of execution times $\{c_k\}$. Then, we executed the same sequence of jobs using the Reservation Based scheduler, with a CPU allocation randomly chosen between 12.5% and 25%, with a fixed time granularity (i.e., server period) of $2ms$. The average difference between the values of the scheduling error as computed through the model in Equation (4) and the ones measured in the experiment was the 3.78% of the average computation time, with a standard deviation of 5.76%. This difference can be reduced by reducing the granularity of the reservation. However, this approach cannot be pushed too far because the introduced overhead is inversely proportional to this quantity (see [13] for more details).

Similar validations exist in literature for other resources. For example, in [37] the authors present a disk scheduler, implemented in the FreeBSD OS, approximating a fluid allocation of the disk. Under appropriate assumptions on the size of the read data chunks, the model of Equation (4) may be applied.

b) Experiments set-up: For the video decoder, we assumed that the first stage reads the frames from a disk with a guaranteed rate of 10Mb/s. This value is remarkably lower than typical rates achieved in modern Operating Systems using best effort schedulers, because the provision of individual temporal guarantees to QoS sensitive applications affects adversely the global throughput [37]. The RUTs of a job have been computed by simply multiplying the frame size by the transfer rate. The second stage is the MPEG decoder that consumes CPU cycles. In this case, the RUTs used in the simulator were chosen equal to the ones measured on a real execution of the `ffplay` player[2] for the video decoder and the `fluidsynth` software[3] for the MIDI synthesis.

For the MPEG player, the task period was dictated by the periodicity of the selected videos: $T = 40ms$, corresponding to 25 frames per second. For both tasks we specified a desired RIS of $[-e,\, E] = \left[-e^{(1)},\, E^{(1)}\right] = \left[-e^{(2)},\, E^{(2)}\right] = [-9ms,\, 9ms]$. Therefore, the RIS size is 45% of the task period, which is perfectly acceptable for this kind of application. For the different experiments, we considered four applications $\mathcal{A}^{(1)}, \ldots, \mathcal{A}^{(4)}$, with the first three being video decoders associated to different video streams, while $\mathcal{A}^{(4)}$ being a sound synthesiser playing a MIDI file. The relevant parameters of the applications are summarised in Table III. In particular, we report the average RUT $E[c_k]$, the maximum moving average of 3 samples $\tilde{c}_3$ and the maximum sample $\max c_k$, all normalised w.r.t. the application periods, set to $40ms$ for the video decoders and $10ms$ for the sound synthesiser.

c) The predictor: The selection of a predictor for the application considered in this paper is conditioned by conflicting requirements. On one hand, our theory is based on the assumption of *exact* prediction (i.e., $c_k^{(j,i)} \in P_k^{(j,i)} = [h_k^{(j,i)},\, H_k^{(j,i)}]$ with probability one). This requirement would induce "large" sizes of the interval $P_k^{(j,i)}$. On the other hand, the minimum attainable size of the RIS is related to the infimum of the ratio $h_k^{(j,i)}/H_k^{(j,i)}$ (see Theorem 2). Therefore, to improve the performance of our controller, we need *aggressive* predictions with small sizes for $P_k^{(j,i)}$. Moreover, the computation overhead due to the QoS control needs to be negligible with respect to the computation requirements of the controlled application.

Under these premises, complex structures like the one presented in [10], are hardly acceptable. In contrast, we used a very simple predictor, composed of two blocks: the first block produces the prediction of a sample (decorrelating the time series), and the second block constructs a range around the predicted point, whose size is related to the observed accuracy of the point prediction over a moving time horizon.

The prediction of the next sample in time series based on past samples is a very broad topic [9]. The simplest predictor we can imagine is a moving average. However, the considered MPEG data set was composed of a sequence of Group of Pictures (GOPs) of fixed length. Therefore, samples in the

[2]More information at the URL: http://www.ffmpeg.org.

[3]More information at the URL: http://www.fluidsynth.org.

same relative position within consecutive GOPs are highly correlated to each other. To account for this effect, we used a set of interleaved and independent moving averages. In the experiments shown below, we will denote this predictor as $MMA-H-L$, where H is the number of considered moving averages and L is the length of each moving average. This can be implemented by using H circular buffers of L elements.

The auto-correlation structure may be fully exploited by using a FIR filter, whose coefficients (taps) are *trained* to minimise the square error. In order to keep the prediction overhead sufficiently low, we did not use on-line adaptation of the FIR coefficients. Instead, we made an off-line optimisation of the taps by using a training set composed of a few movies with characteristics similar to the ones used for the final experiment. We will denote this predictor as $FIR-L$, where L is the number of taps. The on-line execution of the filter in this case requires $2L-1$ algebraic operations (scalar product between the coefficients vector and the sliding samples vector) plus the insertion and extraction operations from the circular buffer. Depending on the value of L, the complexity may become considerably higher than in the case of MMA.

Clearly, the best performance can be obtained when using predictors that exploit a deep knowledge of the application. One of such examples was recently proposed in [32] for MPEG decoding. The prediction is not based on the history, but is produced through a quick parse of the frame. We implemented this application-specific predictor (labelled as AS), and used the produced prediction traces in some of our simulations. We obtained precision and overhead figures similar to the ones of [32]. In particular, we got an excellent accuracy (we measured a maximum relative error below 15%), paying the price of a non-negligible computation time for the predictor (5% of the decoding time).

For the sound synthesis, a very effective predictor is the one that exploits the advance knowledge on the number of active voices that have to be synthesised at each sound frame, so as to adapt accordingly the resource allocation right before starting to synthesise each frame. In fact, in our experiments we measured a cross-correlation between the number of active voices and the frame synthesis times higher than 98%.

Concerning the resource usage times on the disk, in addition to the MMA and FIR predictors described above, we assumed availability of an application-specific predictor that has an exact knowledge of the frame size at the very start of the frame load operation, e.g., reading it immediately from the very first bytes of the frame header. However, also in this case we assumed a relative variability of nearly 15% for the loading time, because, even with algorithms providing guarantees, the disk throughput may exhibit uncontrolled fluctuations due to the pattern of requests coming from other applications.

The second predictor block considers a moving window of N past errors done by the sample predictor. The range is then obtained by computing the upper and lower x percentiles, where N and x are configurable parameters. This technique, which may be implemented very efficiently for fixed values of the parameters, eliminates possible outliers. In the presented experiments, we will append to the predictor name a "$/N-x$" suffix to denote the parameters used for the range computation, e.g., $MMA-12-3/24-87.5$.

d) Single application with strong QoS guarantees: In this paragraph we report the result of a batch of experiments for a single application with strong QoS guarantees. Our purpose is to expose the performance of the task controllers, ruling out possible interferences from the supervisor. The average RUTs for the considered application are reported in the second column of Table III (application $\mathcal{A}^{(2)}$).

In order to have a baseline for assessing the performance of our control schemes, we performed a first set of experiments with fixed bandwidth, with values ranging from slightly above (+30%) up to considerably above (+70%) the mean value of the resource requirements. The results are shown in the top rows of Table II, where the (A) side refers to the first stage and the (B) side to the second one. The first data column (labelled as $p^{(j)}$) reports the experimental probability of the scheduling error being within the target RIS, which can be used as a performance metrics. The probability attained with fixed allocation schemes were consistently below 27%, meaning that for most of the times the application received too little bandwidth. As we can see in the 2^{nd} column of the table, with a fixed allocation equal to 1.3 times the average requirement, we got high values for the average scheduling error. Therefore, the system had transient behaviours with significant deviations from the target RIS. Indeed, the maximum error experienced by the system was more than three times the period (3^{rd} column). To understand this point, we report an excerpt (600 frames) of the temporal evolution in which the resource usage times are significantly above the average value (Figure 6.(a)). In this case a fixed allocation of 1.3 the average requirements was not sufficient to accommodate the temporary requirements and the scheduling error reached very high values (as shown in the first curve of Figures 6.(b) and 6.(c)). The large fluctuation in the scheduling error corresponds to a highly varying latency, which is certainly a very annoying effect for the user. On the other hand, when we increased the allocated bandwidth, we got large (in absolute value) and negative scheduling error. This is shown in the 2^{nd} and 3^{rd} data row of Table II, where e.g., a fixed over-allocation of the 50% achieves an average scheduling error of -18.79% on the first stage and of -12.6% on the second one. The effect is also illustrated by the experimental PDF in Figure 7, which exhibits a compression to the left when we increase the bandwidth reducing the probability of staying in the target RIS. In practical terms, this corresponds to a wasteful use of the resources and to a reduced utilisation of the system.

The evaluation of the adaptive schemes was made considering different predictors. First, we used the AS predictor, where the predicted range has been built around the predicted sample by considering the maximum relative deviation of 15% achieved by the value predictor over a set of sample traces. This resulted in a predicted range that always contained the resource usage time of the next job. In this way, the conditions required by Theorem 2 were perfectly fulfilled. As far as the width of the range is concerned, this predictor attained a value for $\alpha^{(2)} = \inf_k \alpha_k^{(2)} \geq 0.7$. The size of the RIS was consistent with the values of $\alpha^{(1)}$ and $\alpha^{(2)}$, according to the statements in Theorem 2. Therefore, the experimental

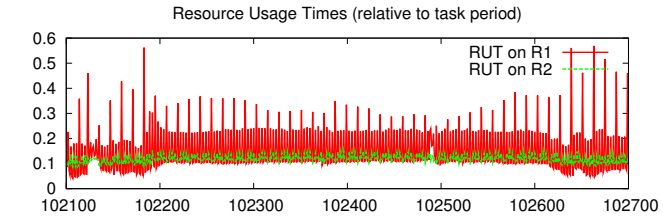
Table II
COMPARISON OF THE PERFORMANCE OF DIFFERENT ALLOCATION SCHEMES ON THE FIRST (A) AND SECOND (B) PIPELINE STAGE.

Experiment	$p^{(1)}$	$E[\epsilon_k^{(1)}]$	$\max \epsilon_k^{(1)}$	$E[b_k^{(1)}]$
Fix. μ+30%	21.66%	135.5%	301.2%	14.58%
Fix. μ+50%	24.02%	-18.79%	58.6%	16.83%
Fix. μ+70%	20.20%	-33.28%	26.9%	19.07%
PI-0.10-0.25	27.9%	-5.75%	206.1%	11.17%
AS	100%	-0.033%	<1.50%	11.56%
FIR-12/24-87.5	94.93%	-3.92%	30.4%	12.10%
MMA-12-3/24-87.5	90.67%	-4.19%	17.1%	12.03%

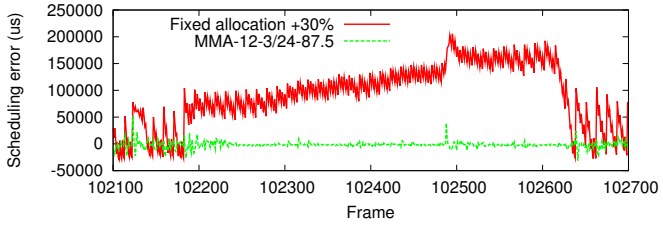
(A)

Experiment	$p^{(2)}$	$E[\epsilon_k^{(2)}]$	$\max \epsilon_k^{(2)}$	$E[b_k^{(2)}]$
Fix. μ+30%	26.52%	153.9%	320.3%	13.40%
Fix. μ+50%	21.62%	-12.60%	47.5%	15.47%
Fix. μ+70%	11.56%	-30.63%	3.10%	17.53%
PI-0.10-0.25	40.0%	9.29%	173.9%	5.28%
AS	100%	-0.037%	<1.50%	10.71%
FIR-12/24-87.5	95.68%	-2.01%	26.6%	11.32%
MMA-12-3/24-87.5	99.51%	-4.77%	<1.8%	11.21%

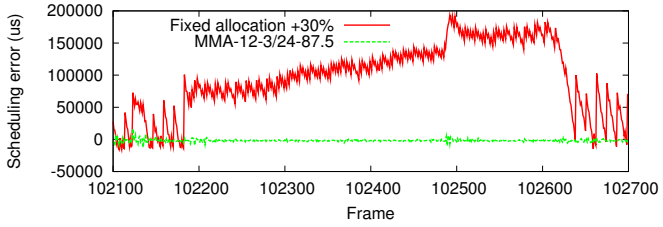
(B)



(a)



(b)



(c)

Figure 6. (a): segments of temporal traces for the RUTs. (b) and (c): scheduling error obtained with a fixed bandwidth 30% greater than the average RUT, and with an invariant-based controller with a MMA-12-3/24-87.5 predictor, for the first (b) and second (c) stages.

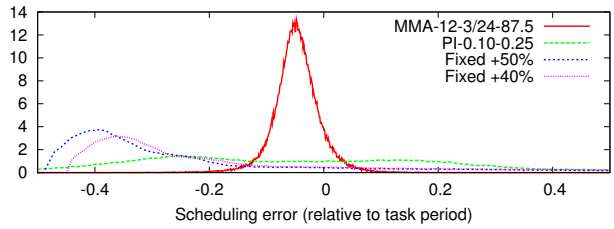


Figure 7. Scheduling error experimental PDF on the second pipeline stage, with different bandwidth allocation strategies.

probability of having the scheduling error inside the RIS was exactly 1. Furthermore, the maximum and the average error is suggestive of an evolution of the scheduling error tightly constrained around the null value.

We evaluated the robustness of the scheme with respect to occasional violations of the assumptions of the theorem. To this end, we applied general purpose predictors: for the video decoders, $MMA-12-3/24-87.5$, consisting of 12 independent averages of 3 samples each (the stream has a fixed GOP of 12 elements), and $FIR-12/24-87.5$. Moreover, the size of the RIS corresponds to a value of $\alpha = 69\%$ that does not account for some isolated spikes of the α_k sequences (less than 5% of the predicted ranges had an $\alpha_k = h_k/H_k$ ratio below such value on the considered traces).

The performance of the control algorithm with the two different predictors is shown in the last two rows of Table II. The probability of staying in the RIS remains well above 90% notwithstanding the occasional failures in the prediction, and the maximum error is always below 30%. It is interesting to look at how the scheduling error is distributed inside the RIS (see Figure 7). The PDF is mostly concentrated around the null value and the tails are very short. It is remarkable that this significant improvement with respect to static allocation is obtained with an average bandwidth request very close to the average requirements of the application.

The improvement achieved by the adaptive scheme on the temporal evolution is shown, in the same segments of 600 samples considered above, in the second curve of Figures 6.(b) and 6.(c), where it is possible to see how the adaptation quickly reacts to the temporary increment of the resource usage times. In a small number of steps the scheduling error is reduced to a value close to 0, which is the RIS centre.

Finally, we compared the performance achieved by our technique with the one achieved through a switching proportional-integral controller, where each control value is computed as a linear combination of the value output at the previous step and the scheduling error of the last two jobs (a detailed discussion may be found in [4]). We tuned the poles of the controller as $z1 = 0.10$ and $z2 = 0.25$ (thus we label it as $PI - 0.10 - 0.25$). As highlighted in the corresponding row of Table II, this controller keeps the average scheduling error very close to 0 (below 10% in absolute value), with a somewhat limited deviation, which would be impossible with a static allocation. As shown in Figure 7, the performance of this controller results quite poor when compared to the one achieved by the control strategy proposed in this paper, which has a similar computational complexity.

e) QoS Supervisor and multiple applications: We also simulated the concurrent run of all the three applications $\mathcal{A}^{(1)}$, $\mathcal{A}^{(2)}$ and $\mathcal{A}^{(3)}$, in order to gauge the effects of the interaction between task controllers and resource supervisors. In this case, it was impossible to provide strong QoS guarantees to all applications, since the sum of the worst-case bandwidth requirements on the first resource was of 142.78%, exceeding

Table III
PARAMETERS FOR THE MULTI-APPLICATION SCENARIO. FOR EACH TASK, ALL PARAMETER VALUES ARE EXPRESSED RELATIVELY TO THE PERIOD.

	$\mathcal{A}^{(1)}$	$\mathcal{A}^{(2)}$	$\mathcal{A}^{(3)}$	$\mathcal{A}^{(4)}$	Tot.
T	$40ms$	$40ms$	$40ms$	$10ms$	-
$E[c_k^{(1)}]$	8.18%	11.14%	4.46%	-	23.78%
$\hat{c}_3^{(1)}$	22.91%	**26.55%**	**16.33%**	-	65.79%
$\max\ c_k$	**44.30%**	58.33%	40.15%	-	142.78%
$E[c_k^{(2)}]$	10.82%	10.30%	12.16%	3.70%	36.98%
$\hat{c}_3^{(2)}$	17.20%	**17.46%**	**8.49%**	14.42%	57.57%
$\max\ c_k$	**27.88%**	30.74%	22.60%	**25.04**%	106.26%

Table IV
RESULTS OF THE SIMULATION FOR THE MULTI-APPLICATION SCENARIO, IN THE CASES OF AS (TOP HALF) AND FIR (BOTTOM HALF) PREDICTORS.

	$\mathcal{A}^{(1)}$	$\mathcal{A}^{(2)}$	$\mathcal{A}^{(3)}$	$\mathcal{A}^{(4)}$
QoS guarantee	strong	weak	weak	strong
$Prob\{\epsilon_k^{(1)} \in [-e, E]\}$	100%	99.92%	99.77%	-
$\max\ \epsilon_k^{(1)}$	<1.50%	<2.50%	<1.50%	-
$Prob\{\epsilon_k^{(2)} \in [-e, E]\}$	100%	100%	100%	100%
$\max\ \epsilon_k^{(2)}$	<1.50%	<1.50%	<1.50%	<1.50%
$Prob\{\epsilon_k^{(1)} \in [-e, E]\}$	91.07%	89.61%	90.91%	-
$\max\ \epsilon_k^{(1)}$	50.8%	303.1%	578.0%	-
$E[\alpha_k^{(1)}]$	83.40%	83.60%	85.40%	-
$Prob\{\epsilon_k^{(2)} \in [-e, E]\}$	95.83%	98.20%	93.00%	98.03%
$\max\ \epsilon_k^{(2)}$	29.5%	213.8%	344.0%	31.19%
$E[\alpha_k^{(2)}]$	79.77%	80.56%	84.61%	83.36%

the maximum resource capacity (100%). Therefore, we chose to provide strong QoS guarantees only to the first application, and weak QoS guarantees to the second and third ones. As a result, based on Theorem 3, the scheduling errors $\epsilon_k^{(2,2)}$ and $\epsilon_k^{(2,3)}$ are allowed to roam within an interval set upper bounded by $L = E + 4T = 169ms$. With this choice of QoS guarantees, the sum of guaranteed bandwidths are 87.18% for $\mathcal{R}_1$ and 78.87% for $\mathcal{R}_2$, both below the total capacities (see Table III, in which the guaranteed bandwidths assigned to the applications are shown in boldface).

In the simulation, we used our control scheme with both the AS predictors and the $FIR - 0.7/24 - 87.5$ predictors introduced in the previous subsection. The results of the simulation are reported in Table IV. When controlling with the AS predictors (top half of the table), the performance of the applications with strong guarantees, $\mathcal{A}^{(1)}$ and $\mathcal{A}^{(1)}$, is not affected by the presence of the other applications in the system. Therefore, the experimental probability of keeping the scheduling error within the RIS is 1, and the maximum scheduling error is below 1.5% of the task period. On the other hand, the two applications that were given only weak guarantees exhibit a lower performance. With respect to the results reported in the Table II of subsection VII-d, in which the application was run in isolation, we have a slightly lower probability of staying in the RIS and a degradation also in the maximum error. Both effects are quite moderate (e.g., the maximum scheduling error remains below 3% of the period), because heavy overload conditions are not frequent.

Switching to a different scenario in which a FIR predictor (bottom half of the table) is used, the average performance measurements are not heavily degraded. Indeed the probability of remaining inside the RIS is decreased by an amount close to 10% (for the first stage) and to 5% for the second one. More interesting are the effects on the maximum error. In this case, we can see that the deviation on the applications with strong QoS guarantees are below 51% for the first stage and below 30% for the second stage. Such deviations are due to the wrong predictions that invalidate the assumptions of Theorem 2 but are sensibly smaller than the ones undergone by applications with weak QoS guarantees. The maximum error remained below the bound established in Theorem 3 even in presence of wrong predictions.

As a concluding remark, not only do the experiments presented in this section validate the theory of this paper, but they also display a very good degree of robustness of the proposed scheme with respect to unmodeled effects or of partial knowledge of the parameters of the application.

VIII. CONCLUSIONS AND FUTURE WORK

In this paper, we have shown the application of a feedback controller to the problem of dynamic allocation of resources to time-sensitive applications consisting of pipelines of tasks.

The use of a scheduling mechanism that approximates a fluid partitioning of the resources enables the definition of a precise dynamic model for the system. This is used to design a feedback controller that provides QoS guarantees on the closed loop system performance.

There are many important open issues, which will motivate future research efforts. A first obvious direction is the implementation of the framework in a real architecture. In the context of the AQuoSA project, we already implemented adaptive resource management techniques for the CPU. The extension to other types of resources is currently under way. Another line of research regards the resource supervisors. In this paper we have considered a very simple scheme, but the task level adaptation techniques shown in the paper can very well be adapted to different and more general techniques. Another direction we plan to investigate is the possible integration between a resource level and an application level feedback. Finally, the extension to more general topologies than pipelines, briefly introduced in Section VI, is also to be seen as a natural extension of the framework. We believe that the modelling effort and the results provided in this paper can offer a sound underpinning for these future developments.

REFERENCES

[1] L. Abeni, T. Cucinotta, G. Lipari, L. Marzario, and L. Palopoli. QoS management through adaptive reservations. *Real-Time Systems Journal*, 29(2-3):131–155, March 2005.

[2] Luca Abeni and Giorgio Buttazzo. Integrating multimedia applications in hard real-time systems. In *Proceedings of the IEEE Real-Time Systems Symposium*, Madrid, Spain, December 1998.

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

[4] Luca Abeni, Luigi Palopoli, Giuseppe Lipari, and Jonathan Walpole. Analysis of a reservation-based feedback scheduler. In *Proc. of the Real-Time Systems Symposium*, Austin, Texas, November 2002.

[5] Neil C. Audsley, Alan Burns, Mike F. Richardson, and Andy J. Wellings. Data consistency in hard real-time systems. *Informatica (Slovenia)*, 19(2), 1995.

[6] S.K. Baruah, N.K. Cohen, C.G. Plaxton, and D.A. Varvel. Proportionate progress: A notion of fairness in resource allocation. *Algorithmica*, 6, 1996.

[7] Shuvra S. Battacharyya, Edward A. Lee, and Praveen K. Murthy. *Software Synthesis from Dataflow Graphs*. Kluwer Academic Publishers, Norwell, MA, USA, 1996.

[8] F. Blanchini. Set invariance in control. *Automatica*, 1999.

[9] George Edward Pelham Box and Gwilym Jenkins. *Time Series Analysis, Forecasting and Control*. Holden-Day, Incorporated, 1990.

[10] G. Calafiore and M.C. Camp. Interval predictors for unknown dynamical systems: an assessment of reliability. In 41^{st} *IEEE Conference on Decision and Control (CDC02)*, 2002, December 2002.

[11] Anton Cervin, Johan Eker, Bo Bernhardsson, and Karl-Erik Årzén. Feedback-feedforward scheduling of control tasks. *Real-Time Systems*, 2002.

[12] Tommaso Cucinotta. Access control for adaptive reservations on multi-user systems. In *Proceedings of the* 14^{th} *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS 2008)*, St. Louis, MO, United States, April 2008. IEEE.

[13] Tommaso Cucinotta, Luigi Palopoli, Luca Marzario, and Giuseppe Lipari. AQuoSA – Adaptive Quality of Service Architecture. *Software – Practice and Experience*, 39(1):1–31, January 2009.

[14] Eric Eide, Tim Stack, John Regehr, and Jay Lepreau. Dynamic cpu management for real-time, middleware-based systems. In *Proc. of 10th IEEE Real-Time and Embedded Technology and Applications Symposium*, Toronto, Canada, May 2004.

[15] E. Gelenbe, R. Lent, and A. Nunez. Self-aware networks and QoS. *Proceedings of the IEEE*, 92(9):1478–1489, 2004.

[16] R. Gerber, Seongsoo Hong, and M. Saksena. Guaranteeing real-time requirements with resource-based calibration of periodic processes. *Software Engineering, IEEE Transactions on*, 21(7):579–592, Jul 1995.

[17] Sourav Ghosh, Jeffery Hansen, Ragunathan (Raj) Rajkumar, and John 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.

[18] Christopher D. Gill, Jeanna M. Gossett, David Corman, Joseph P. Loyall, Richard E. Schantz, Michael Atighetchi, and Douglas C. Schmidt. Integrated adaptive QoS management in middleware: A case study. *Real-Time Systems*, 29(2-3):101–130, march 2005.

[19] Steve Goddard and Kevin Jeffay. Managing latency and buffer requirements in processing graph chains. *Comput. J.*, 44(6):486–503, 2001.

[20] Ashvin Goel, Jonathan Walpole, and Molly Shor. Real-rate scheduling. In *Proceedings of Real-time and Embedded Technology and Applications Symposium*, page 434, 2004.

[21] Fumiko Harada, Toshimitsu Ushio, and Yukikazu Nakamoto. Adaptive resource allocation control for fair QoS management. *IEEE Transactions on Computers*, 56(3):344–357, 2007.

[22] Dong-In Kang, Richard Gerber, and Manas Saksena. Parametric design synthesis of distributed embedded systems. *IEEE Trans. Computers*, 49(11):1155–1169, 2000.

[23] Yamuna Krishnamurthy, Vishal Kachroo, David A. Karr, Craig Rodrigues, Joseph P. Loyall, Richard E. Schantz, and Douglas C. Schmidt. Integration of QoS-enabled distributed object computing middleware for developing next-generation distributed application. In *LCTES/OM*, pages 230–237, 2001.

[24] C. Lu, J. Stankovic, G. Tao, 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.

[25] Chenyang Lu, Xiaorui Wang, and X. Koutsoukos. Feedback utilization control in distributed real-time systems with end-to-end tasks. *Parallel and Distributed Systems, IEEE Transactions on*, 16(6):550–561, June 2005.

[26] Walid A. Najjar, Edward A. Lee, and Guang R. Gao. Advances in the dataflow computational model. *Parallel Computing*, 25(13-14):1907–1929, 1999.

[27] L. Palopoli and T. Cucinotta. Feedback scheduling for pipelines of tasks. In *Proc. of 10th conference on Hybrid Systems Computation and Control 2007 (HSCC07)*, Pisa, Italy, April 2007. Springer Verlag, Lecture notes in computer science.

[28] Luigi Palopoli, Tommaso Cucinotta, and Antonio Bicchi. Quality of service control in soft real-time applications. In *Proc. of the IEEE 2003 conference on decision and control (CDC03)*, Maui, Hawai, USA, December 2003.

[29] Ragunathan Rajkumar, Chen Lee, John P. Lehoczky, and Daniel P. Siewiorek. Practical solutions for QoS-based resource allocation. In *RTSS*, pages 296–306, 1998.

[30] Raj Rajkumar, Kanaka Juvva, Anastasio Molano, and Shuichi 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.

[31] John Regehr and John A. Stankovic. Augmented CPU Reservations: Towards predictable execution on general-purpose operating systems. In *Proceedings of the IEEE Real-Time Technology and Applications Symposium (RTAS 2001)*, Taipei, Taiwan, May 2001.

[32] Michael Roitzsch and Martin 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.

[33] Nishanth Shankaran, Xenofon D. Koutsoukos, Douglas C. Schmidt, Yuan Xue, and Chenyang Lu. Hierarchical control of multiple resources in distributed real-time and embedded systems. In *ECRTS'06: Proceedings of the 18th Euromicro Conference on Real-Time Systems*, pages 151–160, Washington, DC, USA, 2006. IEEE Computer Society.

[34] David Steere, Ashvin Goel, Joshua Gruenberg, Dylan McNamee, Calton Pu, and Jonathan Walpole. A feedback-driven proportion allocator for real-rate scheduling. In *Proceedings of the Third usenix-osdi*. pub-usenix, feb 1999.

[35] David Steere, Molly H. Shor, Ashvin Goel, Jonathan Walpole, and Calton P. Control and modeling issues in computer operating systems: Resource management for real-rate computer applications. In *Proceedings of 39th IEEE Conference on Decision and Control (CDC00)*, December 2000.

[36] Ian Stoica, Hussein Abdel-Wahab, Kevin Jeffay, Sanjoy K. Baruah, Johannes E. Gehrke, and C. Greg Plaxton. A proportional share resource allocation algorithm for real-time, time-shared systems. In *Proceedings of the IEEE Real-Time Systems Symposium*, December 1996.

[37] Paolo Valente and Luigi Rizzo. Hybrid: achieving deterministic fairness and high throughput in disk scheduling. In *Proceeding of CCCT 2004*, August 2004.

[38] Ronghua Zhang, Chenyang Lu, Tarek F. Abdelzaher, and John A. Stankovic. Controlware: A middleware architecture for feedback control of software performance. In *Proc. of International Conference on Distributed Computing Systems*, Vienna, Austria, July 2002.

Tommaso Cucinotta graduated in Computer Engineering at the University of Pisa in 2000, and received the PhD degree in Computer Engineering from the Scuola Superiore Sant'Anna of Pisa in 2004. He is Assistant Professor of Computer Engineering at the Real-Time Systems Laboratory (ReTiS) of Scuola Superiore Sant'Anna. His main research activities are in the areas of real-time and embedded systems, with a particular focus on real-time support for general-purpose Operating Systems, and security, with a particular focus on smart-card based authentication.

Luigi Palopoli graduated in Computer Engineering at the University of Pisa in 1992, and received the PhD degree in Computer Engineering from Scuola Superiore Sant'Anna in 2002. He is Assistant Professor of Computer Engineering in the University of Trento. His main research activities are in embedded system design with a particular focus on resource–aware control design and adaptive mechanisms for QoS management. He has served in the program committee of different conferences in the area of real-time and control systems.

APPENDIX

SUPPLEMENTAL MATERIAL

A. Proof of Theorem 1

Proof [Theorem 1]:

[CLAIM 1] Consider a pipeline composed of j tasks. The tokens that entered the pipeline in a generic time interval $[0, t]$ are those ones associated to jobs $J_k^{(j)}$ with arrival time less than or equal to $t : a_k^{(1)} \leq t$, i.e., $(k-1)T + \xi_k \leq t \iff k \leq \left\lfloor \frac{t}{T} \right\rfloor + 2$. In the same interval, the tokens that already exited the pipeline are at least those ones associated to jobs k of the last task j for which the temporal reference $t_k^{(j)}$, plus the maximum delay E, is less than or equal to t (assume $t \geq jT + E$ for the sake of simplicity): $t_k^{(j)} + E \leq t$, i.e., $(j+k-1)T + E \leq t \iff k \leq \left\lfloor \frac{t-E}{T} \right\rfloor - j + 1$. Introducing the symbols $x \triangleq \frac{t-E}{T} - \left\lfloor \frac{t-E}{T} \right\rfloor \in [0, 1[$, and $y \triangleq \frac{t}{T} - \left\lfloor \frac{t}{T} \right\rfloor \in [0, 1[$, for the maximum number $N^{(j)}(t)$ of tokens still in the pipeline, comprising the one possibly being computed at the j^{th} stage, we have: $N^{(j)}(t) \leq \left\lfloor \frac{t}{T} \right\rfloor + 2 - \left\lfloor \frac{t-E}{T} \right\rfloor + j - 1 = \frac{E}{T} + j + y - x < \frac{E}{T} + j + 1$. The claim follows considering that $N^{(j)}(t) \in \mathbb{N}$.

In order to show that the bound is tight consider a possible execution where the first incoming j tokens are executed very quickly (in a time lower than an arbitrarily small real number δ) by all the tasks in the pipeline, except the j^{th} one, in which they finish with the maximum allowed delay: $f_k^{(j)} = t_k^{(j)} + E = (j+k-1)T + E$. In this situation, at time $f_1^{(j)} = jT + E$, the number of tokens in the pipeline from the first to the j^{th} buffer is given exactly by $j + \left\lceil \frac{E}{T} \right\rceil$.

[CLAIM 2] Consider a generic task $\tau^{(j)}$ in the pipeline with $j \geq 2$. At a generic time instant t under the hypotheses of the theorem, the tokens that have been inserted into the j^{th} input buffer are the ones processed by one of the completed jobs of task $\tau^{(j-1)}$. These jobs are contained in the set of the jobs k such that $t_k^{(j-1)} - e \leq t$. Recalling that $t_k^{(j-1)} = (j+k-2)T$, this set is given by $\{k \in \mathbb{N}\, s.t.\, k \leq \frac{t}{T} + \frac{e}{T} + 2 - j\}$, i.e. $\{k \in \mathbb{N}\, s.t.\, k \leq \left\lfloor \frac{t+e}{T} \right\rfloor + 2 - j\}$. Similarly, the set of tokens that, at time t, have not yet been output, are at most those for which $t_k^{(j)} + E > t$. This set is given by $\{k \in \mathbb{N}\, s.t.\, k > \frac{t-E}{T} + 1 - j\}$, which is equal to the set $\{k \in \mathbb{N}\, s.t.\, k > \left\lfloor \frac{t-E}{T} \right\rfloor + 2 - j\}$. Overall, the set of tokens that can be in the j^{th} buffer task at time t (including the one which is possibly being processed), is given by $\{k \in \mathbb{N}\, s.t.\, \left\lfloor \frac{t-E}{T} \right\rfloor + 2 - j < k \leq \left\lfloor \frac{t+e}{T} \right\rfloor + 2 - j\}$. Focusing for the sake of simplicity to the case $t > E + jT$, the cardinality of the set is upper bounded by $\left\lfloor \frac{t+e}{T} \right\rfloor - \left\lfloor \frac{t-E}{T} \right\rfloor$. If we introduce the quantities $w \triangleq \frac{t+e}{T} - \left\lfloor \frac{t+e}{T} \right\rfloor \in [0, 1[$ and $z \triangleq \frac{t-E}{T} - \left\lfloor \frac{t-E}{T} \right\rfloor \in [0, 1[$, the cardinality is upper-bounded by $\frac{t+e}{T} - w - \frac{t-E}{T} + z \leq \frac{e+E}{T} + 1$. Considering that it is restricted to be a natural number, we come up with the upper-bound $\left\lfloor \frac{e+E}{T} \right\rfloor + 1$.

Focusing on the first stage $j = 1$, the tokens already arrived at time t are, as said above, those ones k for which $k \leq \left\lfloor \frac{t}{T} \right\rfloor + 1$, which are included in the set $\{k\, s.t.\, k \leq \left\lfloor \frac{t+e}{T} \right\rfloor + 2 - j\}$. Hence, we can apply the same reasoning shown above and obtain a conservative estimation of the number of tokens in the buffer.

The case $t < E + jT$ may be handled in a similar way. ■

B. Proof of Lemma 1

To prove the Lemma, we need some preliminary results.

As a first step, we offer necessary and sufficient conditions for robust invariance over *a single step* and for *a single stage.*

Lemma 2: Let $\mathcal{I}$ denote the interval $[-e,\, E]$ (with $e + E \leq T$) and consider the job $J_k^{(j)}$ of task $\tau^{(j)}$.

1) There exists a bandwidth assignment guaranteeing that $\epsilon_k^{(j)} \in \mathcal{I}\ \forall \sigma_k^{(j)} \in \mathcal{I}, \forall c_k^{(j)} \in P_k^{(j)} = \left[h_k^{(j)},\, H_k^{(j)}\right]$, if and only if $e + E \geq T\left(\frac{1-\alpha_k^{(j)}}{\alpha_k^{(j)}}\right)$.
2) The bandwidth assignments guaranteeing these properties are uniquely identified by the range defined in Equation (6) if $B_N^{(j)} \geq \frac{H_k^{(j)}}{T}$.

Proof: Consider job $J_k^{(j)}$ of task $\tau^{(j)}$. In view of Equation (4), we can have $\epsilon_k^{(j)} \in [-e, E]$ iff $T - e - \sigma_k^{(j)} \leq \frac{c_k^{(j)}}{b_k^{(j)}} \leq T + E - \sigma_k^{(j)}$. Since $e + E < T$ and we are interested in values of $\sigma_k^{(j)} \leq E$, it is possible to re-write this condition as: $\frac{c_k^{(j)}}{T+E-\sigma_k^{(j)}} \leq b_k^{(j)} \leq \frac{c_k^{(j)}}{T-e-\sigma_k^{(j)}}$. As $c_k^{(j)}$ is not known, and it may take any value in the range $[h_k^{(j)}, H_k^{(j)}]$, the only possibility for the controller not to violate last equation is the choice of a bandwidth value belonging to the intersection of all the ranges corresponding to any possible value of $c_k^{(j)}$, i.e. $\frac{H_k^{(j)}}{T+E-\sigma_k^{(j)}} \leq b_k^{(j)} \leq \frac{h_k^{(j)}}{T-e-\sigma_k^{(j)}}$. Such a choice exists iff $e + \alpha_k^{(j)} E \geq (1 - \alpha_k^{(j)})(T - \sigma_k^{(j)})$. The latter condition must hold for any possible value of the start time $\sigma_k^{(j)}$ we are interested in. So, the first claim is proved by observing that $\sigma_k^{(j)}$ is allowed to take values in the range $[-e,\, E]$. The proof of the second claim is obtained for all j by observing that, in the interval $\sigma_k^{(j)} \in [-e,\, E]$, the variability range for $b_k^{(j)}$ identified above is identical to the one in Equation (6) iff $B_N^{(j)}$ does not saturate the lower bound of the range, i.e. if $B_N^{(j)} \geq \frac{H_k^{(j)}}{T+E-\sigma_k^{(j)}}$, where the most restrictive condition happens when $\sigma_k^{(j)} = E$. ■

As a second step, the following provides sufficient conditions for ensuring robust invariance *over a single-step* for the *entire pipeline*.

Lemma 3: Suppose the system state for the $(k-1)^{th}$ token is in the invariant set $\boldsymbol{\epsilon}_{k-1} \in \mathbf{I}$. Then, any bandwidth assignment in the control range defined in Equation (6) guarantees $\boldsymbol{\epsilon}_k \in \mathbf{I}\ \forall \mathbf{c}_k \in \mathbf{P}_k$ if the following conditions hold:

$$\begin{cases} e + E & \geq \frac{T\left(1-\underline{\alpha}_k^{(n)}\right)}{\underline{\alpha}_k^{(n)}} \\ \max_{j \in \{1..n\}} \left\{ \frac{H_k^{(j)}}{B_N^{(j)}} \right\} & \leq T \end{cases}$$

where $\underline{\alpha}_k^{(j)} \triangleq \min_{l \in \{1,\, ...,\, j\}} \left\{ \alpha_k^{(l)} \right\}$.

Proof: Proof is obtained by proving the following statement $S_k^{(j)}$ by induction on the stages j of the pipeline (*spatial*

induction):

$$S_k^{(j)}: \begin{cases} \epsilon_{k-1}^{(l)} \in \mathcal{I} \forall l \in \{1,\ldots,j\} \\ e + E \geq T\left(\frac{1-\underline{\alpha}_k^{(j)}}{\underline{\alpha}_k^{(j)}}\right) \text{ for } j = 1,\, 2,\, n \\ \max_{l\in\{1..j\}}\left\{\frac{H_k^{(l)}}{B_N^{(l)}}\right\} \leq T \\ b_k^{(l)} \in \mathcal{U}_k^{(l)} \forall l \in \{1,\ldots,j\} \end{cases} \Rightarrow$$
$$\forall l \in \{1,\ldots,j\},\, \forall c_k^{(l)} \in P_k^{(l)},\, \epsilon_k^{(l)} \in \mathcal{I}.$$

For the base inductive case $j = 1$, note that hypothesis in the left-hand side of $S_k^{(1)}$ and the system model in Equation (4) imply that $\sigma_k^{(1)} \in \mathcal{I} = [-e,\, E]$, hence the statement for $j = 1$ of Lemma 2 may be applied, proving the truth of statement $S_k^{(1)}$.

For a generic $j \geq 2$, suppose $S_k^{(l)}$ holds for any $l \in \{1..j-1\}$, and suppose the assumptions of $S_k^{(j)}$ hold. The inductive hypothesis ensures that, applying the proposed control law to stages $\{1,\ldots,j-1\}$, ensures $\epsilon_k^{(j-1)} \in \mathcal{I}$. This fact, along with the system model in Equation (4), guarantees that $\sigma_k^{(j)} \in \mathcal{I}$. Therefore, the statement for $j \geq 2$ of Lemma 2 proves the truth of $S_k^{(j)}$.

Finally, the Lemma proof follows by observing that its statement corresponds to $S_k^{(n)}$, with n equal to the number of stages of the pipeline. ■

We are now in condition to offer the proof of Lemma 1.

Proof [Lemma 1]:

[CLAIM 1] First, we focus on sufficiency, which may be proved by choosing an arbitrary value for k_0 and proving the following statement $S(k)$ inductively for any $k > k_0$ (*temporal* induction):

$$S(k): \begin{cases} \max_j\left\{\max_{h\in]k_0,k]}\frac{H_h^{(j)}}{B_N^{(j)}}\right\} \leq T \\ e + E \geq T\left(\frac{1-\alpha_{k_0,k}^{(j)}}{\alpha_{k_0,k}^{(j)}}\right) j \geq 2 \\ \forall h \in]k_0,k] \mathbf{b}_h \in \mathcal{U}_h \end{cases} \Rightarrow$$
$$, \quad \forall h \in]k_0,k]\, \forall \mathbf{c}_h \in \mathbf{P}_h,\, \boldsymbol{\epsilon}_h \in \mathbf{I}$$

where $\alpha_{k_0,k}^{(j)} \triangleq \min_{h\in]k_0,k]} \alpha_h^{(j)}$, and $\mathcal{U}_k$ is the vector of control ranges defined in Equation (6).

Consider the base inductive case $k = k_0 + 1$. The left hand side of statement $S(k)$ corresponds to the hypotheses of Lemma 3. Therefore, we can conclude that $\boldsymbol{\epsilon}_k \in \mathbf{I}$.

For a generic k, assume $S(k-1)$ holds. Then, under the conditions of the left-hand side of $S(k-1)$, the proposed control law leads to $\boldsymbol{\epsilon}_{k-1} \in \mathbf{I}$. This condition, along with the left-hand side of statement $S(k)$, allows us to apply Lemma 3, proving the truth of $S(k)$.

For the induction principle, $S(k)$ is true for *any* $k > k_0$, hence the sufficiency claim is proved.

Concerning necessity, in Definition 1 robust invariance is required to hold for each k_0. Therefore, if we consider only a single evolution step of the state vector from k_0 to $k_0 + 1$, Lemma 2 requires $\frac{H_{k_0}^{(1)}}{T} \leq B_N^{(1)}$, $\frac{H_{k_0}^{(j)}}{T} \leq B_N^{(j)}$ and $e + E \geq T\left(\frac{1-\alpha_{k_0}^{(j)}}{\alpha_{k_0}^{(j)}}\right)$ as necessary conditions to guarantee that $\boldsymbol{\epsilon_{k_0+1}} \in \mathbf{I}$. Therefore, by considering any possible value for k_0, we obtain that all of these conditions (at varying k_0) must hold true, leading to the claim proof.

[CLAIM 2] The claim proof may be obtained by observing that the proposed control law produces a non-empty bandwidth value for a given j and k pair iff conditions of Lemma 2 hold for that particular pair of values. Now, the claim requires that the proposed control law produce a non-empty value for each possible j and k values, which happens iff conditions of Lemma 2 for each possible j and k values hold true. The intersection of these conditions may be written as:

$$\begin{cases} e + E \geq T\,\sup_k\left(\frac{1-\alpha_k^{(j)}}{\alpha_k^{(j)}}\right) & j \geq 2 \\ \frac{\sup_k H_k^{(j)}}{T} \leq B_N^{(j)} & j \geq 1 \end{cases}$$

Now, the theorem claim is obtained by observing that the function $\frac{1-\alpha}{\alpha}$ is strictly decreasing in the range $\alpha \in]0,1[$, thus $\sup_k \frac{1-\alpha_k^{(j)}}{\alpha_k^{(j)}} = \frac{1-\inf_k \alpha_k^{(j)}}{\inf_k \alpha_k^{(j)}}$. ■

C. Proof of Theorem 3

First, we need the following preliminary result:

Lemma 4: Consider a generic stage j of an application pipeline controlled by the set of task controllers defined by Equation (6), acting under the action of resource supervisors granting a set of minimum guaranteed bandwidths of $\{B_O^{(j)}\}$. Let $E^{(n)} \geq E^{(n-1)} \geq \ldots \geq E^{(1)} \geq 0$ be a set of reals such that $\epsilon_{k_0}^{(j)} \leq E^{(j)}$. Assume that $\epsilon_k^{(j-1)} \leq L^{(j-1)} \forall k \geq k_0$ (if $j > 1$) with $L^{(j-1)} \leq E^{(j)}$ and that, for some integer M, $B_O^{(j)} \geq \tilde{c}_M^{(j)} = \sup_{\overline{k}} \frac{1}{M}\sum_{k=\overline{k}+1}^{\overline{k}+M} \frac{c_k^{(j)}}{T}$, then $\epsilon_k^{(j)} \leq L^{(j)} \forall k > k_0$, with $L^{(j)} \triangleq E^{(j)} + (M-1)T$.

Proof: The first step is to show that if at a given step k_0 the scheduling error starts from a value below the $E^{(j)}$ bound ($\epsilon_{k_0}^{(j)} \leq E^{(j)}$), then 1) it will return below the same bound within a time horizon of M steps (i.e. there exists a number $h \in [1,\, M]$ such that $\epsilon_{k_0+h}^{(j)} \leq E^{(j)}$) 2) during the time horizon of M steps, the scheduling error cannot grow beyond $L^{(j)}$.

To prove the first fact, consider that, if for any $h \in [1, M]$ $B_O^{(j)} \geq \mathcal{B}_L^{(j)}$, then, by construction, the controller will choose a bandwidth s.t. $\epsilon_{k_0+h}^{(j)} \leq E^{(j)}$ and the fact is verified. Therefore we can focus on the case in which 1) the controller receives a bandwidth greater than or equal to $B_O^{(j)}$ but strictly lower than $\mathcal{B}_L^{(j)}$, 2) $\epsilon_{k_0+h}^{(j)} > E^{(j)}$ for all $h \in [1, M-1]$.

The latter condition also implies, for $j = 1$, that $\epsilon_{k_0+h-1}^{(j)} \geq 0$, while for $j > 1$, that $\epsilon_{k_0+h-1}^{(j)} > L^{(j-1)} \geq \epsilon_{k_0+h}^{(j-1)}$ by hypothesis. Therefore, from the evolution model, we have that the j^{th} stage evolves without "interferences" from the stage $j-1$ due to the max in Equation (4). Hence, its state after M

steps is upper-bounded by:

$$\begin{aligned}\epsilon_{k_0+M}^{(j)} &= \epsilon_{k_0}^{(j)} + \sum_{h=1}^{M} \frac{c_{k_0+h}^{(j)}}{b_h^{(j)}} - MT \\ &\leq E^{(j)} + \sum_{h=1}^{M} \frac{c_{k_0+h}^{(j)}}{B_O^{(j)}} - MT \\ &\leq E^{(j)}\end{aligned}$$

(where the last passage is due to the lemma premises on $B_O^{(j)}$, that may be written as $\sum_{h=1}^{M} \frac{c_{k_0+h}^{(j)}}{B_O^{(j)}} \leq MT$).

To prove the second fact, consider that the maximum possible scheduling error is upper-bounded by:

$$\epsilon_{k_0} + \max_{h\in 1,\dots M}\{\sum_{k=k_0+1}^{k_0+h} \frac{c_k^{(j)}}{B_O^{(j)}} - hT\},$$

whose maximum value, under the lemma premises, is attained in the worst-case scenario in which $c_{k_0+1}^{(j)} = B_O^{(j)}MT$ and $c_k^{(j)} = 0$ for $k = k_0+2, \dots, k_0+M$. Therefore, this maximum value is upper-bounded by $E^{(j)}+MT-T$. The proof is ended considering all possible initial steps k_0. ∎

Now, we are in condition to offer a sketch of the proof of Theorem 3.

Proof Sketch[Theorem 3]: As long as we choose $b_k^{(j)} \leq \mathcal{B}_H$, we have $\epsilon_{k-1}^{(j)} \geq -e \implies \epsilon_k^{(j)} \geq -e$. Under the premises of the theorem this is always a feasible choice, therefore we can focus on the upper bound of $\epsilon_k^{(j)}$ only.

The theorem proof is obtained by (spatial) induction on the following statement:

$S(j)$: if $\forall l \leq j\ B_O^{(l)} \geq \tilde{c}_k^{(l)}$ then $\epsilon_k^{(j)} \leq E+j(M-1)T \forall k > k_0$.

Concerning the base induction case $j = 1$, Lemma 4 applies directly with $E^{(1)} = E \geq 0$, leading to $S(1)$.

For $j > 1$, the inductive assumption $S(j-1)$ provides an upper-bound for $\epsilon_k^{(j-1)}$: $\epsilon_k^{(j-1)} \leq E+(j-1)(M-1)T$. Such value may be used as $E^{(j)} = L^{(j-1)}$ in the application of Lemma 4, leading to the truth of $S(j)$.

By induction, we may conclude that $S(n)$ is true, what is equivalent to the theorem claim. ∎

D. Proof of Theorem 4.

To show the Theorem, we will use two preliminary Lemmas regarding the properties of the control range in Equation (6) when it is used with the minimum bandwidth guaranteed in Theorem 2. The first Lemma describes some elementary properties of the controller when the scheduling error is *outside* of the robust invariant set $\mathbf{I}$.

Lemma 5: Let conditions of Theorem 2 hold, and let $F_k^{(j)}$ be the value of $\sigma_k^{(i)}$ where $B_L^{(j)}(\sigma_k^{(j)}, P_k^{(j)})$ intersects $B_O^{(j)}$: $F_k^{(j)} \triangleq T + E - \frac{H_k^{(j)}}{B_O^{(j)}}$. The family of bandwidth assignments identified in Equation (6) ensures that

1) if $\sigma_k^{(j)} > F_k^{(j)}$ then $-e \leq \epsilon_k^{(j)} \leq \sigma_k^{(j)}$;
2) if $-e \leq \sigma_k^{(j)} \leq F_k^{(j)}$, then $\epsilon_k^{(j)} \in [-e, E]$.

Proof: Descends from the assumptions of Theorem 2 and from the construction of the control range in Equation (6). ∎

The second Lemma shows how the control law reduces the bounds in which the scheduling error lies at each step.

Lemma 6: Consider a generic k_0 and assume $\forall j\, \epsilon_{k_0}^{(j)} \in [-e, L]$ with $L \geq E$, and that the assumptions of Theorem 4 hold. Then:

1) the scheduling error is contained in the following set:
$$\begin{aligned}&\forall j, \forall N \in [(j-1)M, m] \\ &\epsilon_{k_0+N}^{(j)} \in \left[-e, \max\left\{E, L - (\left\lfloor \tfrac{N}{M} \right\rfloor - j + 1)\theta\right\}\right].\end{aligned} \tag{11}$$
2) $\forall j, \forall N \in [1, (j-1)M-1]\, \epsilon_{k_0+N}^{(j)} \in [-e, L]$.

Proof: Let $F_k^{(j)}$ be defined as in Lemma 5. We prove the claim by induction on the stages of the pipeline. In the first stage, $\sigma_{k_0+N}^{(1)} = \max\{0, \epsilon_{k_0+N-1}^{(1)}\}$. In view of the first property in Lemma 5 and of the evolution of the system in Equation (4), $\epsilon_{k_0+N}^{(1)} \geq -e$ is easily verified by induction on N. As far as the upper bound is concerned, we observe that, if for any $k \in [k_0, k_{0+N}]\ \epsilon_k^{(1)} \leq F_k^{(1)}$, then the claim is obtained as an immediate result of the second claim in Lemma 5. On the contrary, assuming that for all $k \in [k_0, k_0+N]\ \epsilon_k^{(1)} > F_k^{(1)}$, then we can use the minimum guaranteed value $B_O^{(1)}$ as a lower bound of the bandwidth assigned by the supervisor. Therefore:

$$\begin{aligned}\epsilon_{k_0+N}^{(1)} \leq& \epsilon_{k_0}^{(1)} + \frac{\sum_{h=1}^{N} c_{k_0+h}^{(1)}}{B_O^{(1)}} - NT \\ =& \epsilon_{k_0}^{(1)} + \frac{\sum_{h=1}^{\lfloor \frac{N}{M} \rfloor M} c_{k_0+h}^{(1)}}{B_O^{(1)}} - \left\lfloor \frac{N}{M} \right\rfloor MT + \\ &+ \frac{\sum_{h=\lfloor \frac{N}{M} \rfloor M+1}^{N} c_{k_0+h}^{(1)}}{B_O^{(1)}} - (N - \left\lfloor \frac{N}{M} \right\rfloor M)T \\ \leq& \epsilon_{k_0}^{(1)} + \frac{\sum_{h=1}^{\lfloor \frac{N}{M} \rfloor M} c_{k_0+h}^{(1)}}{B_O^{(1)}} - \left\lfloor \frac{N}{M} \right\rfloor MT \\ \leq& L - \left\lfloor \frac{N}{M} \right\rfloor \theta,\end{aligned}$$

where the step before the last one is obtained observing that, due to the assumption on $B_O^{(j)}$, any term $\frac{c_k^{(j)}}{B_O^{(j)}}$ is negative. This ends the proof of the property for the first stage.

For $j \geq 2$, let us assume the lemma holds true for stage $j-1$. As $\sigma_k^{(j)} = \max\{\epsilon_k^{(j-1)}, \epsilon_{k-1}^{(j)}\}$ and by inductive hypothesis $\epsilon_k^{(j-1)} \geq -e$, we have $\sigma_k^{(j)} \geq -e$. Hence, in view of Lemma 5, we have $\epsilon_k^{(j)} \geq -e$. As far as the upper bound is concerned, for $N < (j-2)M$, as well as for $N \leq (j-1)M$, the inductive hypothesis ensures $\epsilon_{k_0+N}^{(j-1)} \leq L$. This leads to the second lemma claim through an easy proof by induction applying the first claim of Lemma 5. Also, for $N \geq (j-1)M$, the inductive hypothesis ensures $\epsilon_{k_0+N}^{(j-1)} \leq \max\left\{E, L - (\left\lfloor \tfrac{N}{M} \right\rfloor - j + 2)\theta\right\}$. Now, if for any $k \in [k_0, k_0+N]\ \epsilon_k^{(j)}$ becomes lower than such value, then the property would be easily verified for j. So we focus on the case in which this does not happen, thus the previous stage does not "interfere" with the evolution of the stage j, because the upper bound for $\epsilon_k^{(j-1)}$ is always lower than $\epsilon_{k-1}^{(j)}$. Therefore, for $N \geq (j-1)M$, an upper bound for $\epsilon_{k_0+N}^{(j)}$ is built as follows:

$$\begin{aligned}
\epsilon_{k_0+N}^{(j)} &\leq \epsilon_{k_0+(j-1)M}^{(j)} + \sum_{h=(j-1)M+1}^{N} \frac{c_{k_0+h}^{(j)}}{B_O^{(j)}} - NT \\
&\leq \epsilon_{k_0+(j-1)M}^{(j)} + \sum_{h=(j-1)M+1}^{\left\lfloor \frac{N}{M} \right\rfloor M} \frac{c_{k_0+h}^{(j)}}{B_O^{(j)}} + \\
&\quad - \left(\left\lfloor \frac{N}{M} \right\rfloor - j + 1 \right) MT \\
&\leq \epsilon_{k_0+(j-1)M}^{(j)} - \left(\left\lfloor \frac{N}{M} \right\rfloor - j + 1 \right) M \left(T - \frac{\tilde{c}_M^{(j)}}{B_O^{(j)}} \right) \\
&\leq L - \left(\left\lfloor \frac{N}{M} \right\rfloor - j + 1 \right) \theta.
\end{aligned}$$

where the upper bound for $\epsilon_{k_0+(j-1)M}^{(j)}$ is a direct consequence of the second lemma claim we already proved. ■

Now we can give the proof of the Theorem.

Proof [Theorem 4]: As an immediate consequence of Lemma 6, we have that for any stage j if $\epsilon_{k_0}^{(j)} \in [-e,\, L]$ then $\epsilon_k^{(j)} \in [-e,\, L]\, \forall k \in [k_0+1,\, k_0+m-1]$. Therefore, we have to prove that: $\epsilon_{k_0+m}^{(j)} \in [-e,\, E]$, which in view of Lemma 5, is true if $\sigma_{k_0+m}^{(j)} = \max\left\{ \epsilon_{k_0+m}^{(j-1)},\, \epsilon_{k_0+m-1}^{(j)} \right\} \leq F_{k_0+m}^{(j)} = T + E - \frac{H_{k_0+m}^{(j)}}{B_O^{(j)}}$. This condition is verified if: $L - \left(\left\lfloor \frac{m}{M} \right\rfloor - j + 1 \right) \theta \leq T + E - \frac{\tilde{H}^{(j)}}{B_O^{(j)}}$ which is verified for all $j \in [1, n]$ if $m \geq M \left(\frac{L - E - T + \frac{\tilde{H}}{B_O^{(j)}}}{\theta} + n \right)$. ■