Real-Time Systems manuscript No.
(will be inserted by the editor)

Stability of Adaptive Feedback-based Resource Managers for systems with Execution Time Variations

Sergiu Rafiliu · **Petru Eles** · **Zebo Peng**

Received: date / Accepted: date

Abstract Today's embedded systems are exposed to variations in load demand due to complex software applications, dynamic hardware platforms, and the impact of the run-time environment. When these variations are large, and efficiency is required, adaptive on-line resource managers may be deployed on the system to control its resource usage. An often neglected problem is whether these resource managers are stable, meaning that the resource usage is controlled under all possible scenarios. In this paper we develop mathematical models for real-time embedded systems and we derive conditions which, if satisfied, lead to stable systems. For the developed system models, we also determine bounds on the worst case response times of tasks. We also give an intuition of what stability means in a real-time context and we show how it can be applied for several resource managers. We also discuss how our results can be extended in various ways.

Keywords control theory · stability criterion · adaptive real-time systems · utilization control

1 Introduction

Today's embedded systems, together with the real-time applications running on them, have achieved a high level of complexity. Moreover, such systems very often are exposed to varying resource demand (load demand) due to e.g. variable number of tasks in the system or variable execution times of tasks. When these variations are large and system efficiency is required, adaptive on-line resource managers may be deployed to control the system's resource usage. Such managers take the shape of algorithms which run at key moments in time and adjust system parameters (task rates, modes, offsets, priorities, etc.) subject to the measured variations. Among the

Department of Computer and Information Science,
Linköping University, Sweden
E-mail: {sergiu.rafiliu, petru.eles, zebo.peng} @liu.se

goals of such a resource manager is to maximize the resource usage while minimizing the amount of time the system spends in overload situations.

One, often overlooked, question is whether the deployed resource managers are safe, meaning that the resource demand is bounded under all possible runtime scenarios. This notion of safety can be linked with the concept of stability of control systems. In control applications, a controller controls the state of a plant towards a desired stationary point. The combined system is stable if the plant's state remains within a bounded distance from the stationary point, under all possible scenarios. By modeling the real-time system as the plant and the resource manager as the controller, one may be able to reason about the stability of the combined system.

In this work, we consider a real-time system running a set of independent tasks and resource managers that control the processor's utilization. Our aim is to develop general models of the system and determine conditions that a resource manager must meet in order to render it stable (Sections 7 and 8). Our method has the advantage of using a control theoretic setting, and thus the results are given in well defined terms of stability. The conditions that resource managers must follow are simple and can be applied to any controller, based on control theory or not. As an application of our theory we show how to construct simple switching resource managers (Section 11) and we prove the stability of several existing non-control theory based approaches (Section 12). For the developed system models, we also determine bounds on the worst case response times of tasks (Sections 9 and 10). We end with an extensive discussion on how to interpret the presented results and how they can be extended in various ways (Section 13).

This work is an extension of our previous results from [17]. The theory presented there is augmented here by a number of discussions, test cases and examples, to give a better insight into the discussed problems.

The theory presented in this paper has similarities with the theory on queueing networks [21,22]. Apart from the resource manager which has no counter part in queueing networks, our system may be modeled as a queueing network and our result in Theorem 2 has a similar wording with a classical result on queueing networks (see Chapter I from [22]). However, in this work we deal with absolute stability, that is, we bound the behavior of the system in the worst case, while in queueing networks theory stochastic stability is used, where only bounds on the expected behavior are determined. Furthermore, the rest of the results presented in this paper use the concept of resource manager, thus they have no counterpart in queueing networks theory.

2 Related Work

There exists a vast literature regarding resource utilization control, targeting different types of real-time systems. Lee et al. proposed the QRAM algorithm in [1]. The model consists of a number of resources that tasks use and a number of quality dimensions. When the algorithm runs, it optimizes the overall quality subject to keeping the resource constraints.

Buttazzo et al. [2] introduced the elastic model where each task's rate can change within a certain interval. Rates change when a task is added or removed from the

system. Further work deals with unknown and variable execution times [3], optimal control when the applications are controllers [4], and when dealing with energy minimization [5]. In our previous work [16], we have proposed a number of control algorithms to deal with execution time variations in a uniprocessor system. Lu et al. [6] described a framework for feedback control scheduling, where the source of non-determinism is execution time variation, and the actuation method is admission control and task rate change. Palipoli et al. [7,8] proposed a feedback based technique for adjusting the parameters of a resource reservation scheduling algorithm in reaction to task execution time variations. Cervin et al. [13,14] proposed a method for adjusting control performance of tasks that implement feedback controllers. Combaz et al. [9] proposed a QoS scheme for applications composed of tasks, each described as graphs of subtasks, where each subtask has a number of modes. Liu et al. [10,11] presented a Recursive Least Squares based controller to control the utilization of a distributed system by means of rate adjustment.

We may divide the related work in four classes based on the kind of adaptation they propose (some works may fall in more than one class):

1. *Job Flow adaptation:* These are methods that adapt the incoming flow of task instances (jobs) to the current state of the resource. The adaptation is done by changing task rates or by admission control. Papers that fall in this class are [1,2, 3,4,5,6,12,13].
2. *Resource adaptation:* These methods adapt the resource depending on the current usage pattern required by the application. They do so by changing the capacity of the resource (e.g. through frequency scaling). Methods such as [5] fall in this class.
3. *Task Mode Adaptation:* These are methods that adapt the way the already released jobs get to access the resource. Their adaptation mechanisms are task mode change and job dropping. This class is comprised of methods such as the ones described in [9,12].
4. *Schedule adaptation:* These are methods that try to adapt various parameters of the scheduler to improve the performance of the running applications. Works such as [7,8,9,10,11,14] fall in this class.

The methods in the first three classes achieve similar goals, they try to keep the utilization of the resources at a prescribed level in the face of varying job execution times and/or arrival patterns. While doing so they also try to maximize one or more quality-of-service or performance metrics. Notable exceptions are [6] where the proposed method also takes into account deadline miss ratios and [12] where the method does not control utilization, but instead queues all arriving jobs and concentrates on keeping the queue sizes at certain prescribed levels. With the exception of [12] the works in the first three classes are lacking an explicit modeling of the system in the overloaded state. Since resource utilization is a value that saturates at 100%, by the time a resource manager realizes that the system is overloaded it may not know or it may only have an imprecise estimate of the amount of overload. This makes adaptation to the overload slow and inexact. Simply put, if execution times of jobs increase drastically, the resource manager must wait (possibly for a long time) until these jobs finish and their execution times are known. At this time, the resource manager knows

the correct incoming load into the system and can decide on adjustments, however, the system has already been loaded for a while and jobs have started to queue up. Since this accumulation in queues is unknown to the resource manage, the decided adjustments may not be sufficient to cure the overload and the system may need to go through several iterations before the overload is solved.

In [1] the authors provide a mechanism for adapting task rates to varying resource requirements such that an abstract notion of quality of service is maximized, however, they do not talk about the specific functioning of the system, how and when an overload is detected. In [3,4,5], apart from the adaptation mechanism, the problem of overloads is specifically addressed. The mechanism for solving overloads is based on acting as soon as an overload has been detected and on delaying the next job release of the overloading task until all pending jobs of this task get executed. This method, however, may lead to very often actuation which may be a problem for some applications. In [6] the authors treat overloads by also following the deadline miss ratio of the jobs that execute in a certain interval of time before the resource manager actuation. The deadline miss ratio, however, is also a parameter that saturates at 100%, thus still not being suitable to accurately describe the evolution of the system when overloaded. In [13] the authors propose to overcome the overload issue by using a feedback-feedforward resource manager, where small variations in execution times are handled by a feedback mechanism and large variations are handled, before they happen, by a feedforward mechanism. This means that this method is only applicable to systems where the application can warn the resource manager about large increases of execution times in advance.

In [12] the authors propose a model where the state of the system is composed of the sizes of queues where jobs accumulate before being executed and the goal of the adaptation mechanism is to keep this queues at a certain level of occupancy. This model is stemmed from the functioning of web servers. However, queue sizes are values that saturate at 0 (they are positive values) and the proposed model linearizes the behavior of queues to the region where they are not empty. This means that the resource manager must always keep the queues sizes at positive (not necessary small) levels. Since non-empty queue sizes are generally associated with overloaded systems, this means that the system is always kept at a certain level of overload. This behavior may not be acceptable for systems where it is important that end-to-end delays are kept small.

The methods in the fourth class adjust scheduler parameters in an attempt to match the resource demand of each task to a specific share of the capacity of the resource, with the goal of minimizing deadline misses and maximizing various performance or quality-of-service metrics. In [7,8] the authors consider resource-reservation schedulers and propose methods based on adjusting the quota of tasks subject to their demand. In [8] tasks share several resources and the quotas of tasks on all resources are determined together, in order to minimize end-to-end delays. In [10,11] the authors consider distributed systems where tasks are schedulable if the utilization on each resource is kept at or below certain bounds. In [10] the load on one resource is influenced by the load on the other resources via some coefficients which are estimated on-line, while in [11] the model of the system is learned on-line. In [14] the authors develop the control server model for scheduling and propose an approach to

schedule control tasks in order to minimize jitter and latency. These methods tune the scheduler parameters such that jobs are schedulable as long as the incoming load in the system is below a certain bound (less or equal to 1) and they aim at gracefully degrading the quality-of-service experienced by the user when the incoming load is above the bound. However, if the system finds itself in a situation where the incoming load is above 1 for extended periods of time, these methods are powerless at preventing accumulations from happening and possibly leading to system crashes. We view these methods as being complementary with the ones from the former three classes. This view is in agreement with the one presented in [10].

A natural question that arises in all works on adaptive real-time systems, is how much variation can the proposed method handle. The best results that we are aware of have been obtained in [11] where the proposed method for control remains stable for variations where actual execution times are 7 times larger than the expected values. By contrast, the modeling method that we propose in this paper directly embeds the amount of variation existing in a system, and therefore, the stability criterion that we obtain has the power of determining stability for systems with arbitrary large variations.

3 Contributions

In this work we consider a uniprocessor real-time system running a set of independent tasks. We assume that tasks can release jobs at variable rates and we require knowledge of the intervals in which execution times of jobs of tasks vary. We allow these jobs to be scheduled using any kind of non-idling scheduling policy. We consider that the system possesses a resource manager whose job is to adjust task rates subject to the variation in job execution times.

We develop a criterion for resource managers, that if satisfied, renders the adaptive real-time system stable under any load pattern (execution time variation pattern). Stability, in our case, implies bounded load on the resource at all times and can be linked with bounded worst-case response times for jobs of tasks and bounded throughput.

With the criterion developed in this work we guarantee the stability of the system in all possible cases meaning that the proposed framework is suitable even for adaptive hard real-time systems.

The stability criterion that we propose in this paper is simple and intuitive. However, we must go through a somewhat involving modeling and stability proof for the sake of completeness. We consider the modeling of the system very important and we develop a detailed, non-linear model that we use in determining our stability criterion. We develop the model in two steps, first a constrained model (in Section 7) and then the general model (in Section 8). Apart of the fact that the constrained model helps as an intermediate step, we also use it to compare the two models. Since the general model considers the functioning of the system in more detail, it requires more conditions on the resource manager in its stability criterion (Theorem 4) compared with the constrained model (Theorem 3), thus highlighting the peril of not using detailed enough models. Also, considering a resource manager that is stable under the

assumptions of both models we show in Section 10 that the unrestricted assumptions of the general model allow the resource manager to run in such ways that reduce worst-case response times by as much as one order of magnitude. Our models are geared towards describing the behavior of our system in overloaded situations, which means describing the evolution of the job queues (one for each task) where released jobs accumulate before they are executed.

In Section 10, we determine worst-case response times of tasks assuming an EDF scheduler. We then further tighten these response times assuming further knowledge about the type of resource manager used, by defining two subclasses of stable controllers. We compare these classes in terms of their worst-case response times in Section 10. The bounds developed here are different from the well known worst-case response times derived in literature for EDF, since our system allows overload situations.

Unlike the previous literature (with the possible exception of [8]) in this paper we do not present a particular, customized method for stabilizing a real-time system. We do not present a certain algorithm or develop a particular controller (e.g. PID, LQG, or MPV controller). Instead, we present a criterion which describes a whole class of methods that can be used to stabilize the system. Also, in this work we do not address any performance or quality-of-service metric, since our criterion is independent of the optimality with which a certain resource manager achieves its goals in the setting where it is deployed. The criterion that we propose may be used in the following ways:

1. to determine if an existing resource manager is stable,
2. to help build custom, ad-hoc resource managers which are stable, and
3. to modify existing resource manager to become stable.

We give examples of the usage of our framework in Section 11 where we develop two switching controllers (resource managers) and we prove that three resource manager taken from literature are stable. In Section 12 we show several example runs of adaptive systems with these resource managers.

Finally, we end this paper (Section 13) with a discussion on the stability theory used in this work and final conclusions.

4 Preliminaries

4.1 System and Application

We consider a uniprocessor system running a set of independent tasks (Λ):

$$\Lambda = \{\tau_i, i \in I\}$$

where τ_i is a task and I is a finite index set. A task in the system is defined as a tuple:

$$\tau_i = (\mathbf{P}_i \subseteq [\rho_i^{\min}, \rho_i^{\max}], \mathbf{C}_i \subseteq [c_i^{\min}, c_i^{\max}]), \qquad \rho_i^{\min} \in \mathbf{P}_i,\ c_i^{\max} \in \mathbf{C}_i$$

where $\mathbf{P}_i$ is the set of possible job release rates for the task and $\mathbf{C}_i$ is the set of possible execution times for jobs of this task. The response time of any job of a task represents the interval of time between the release and the finish time of the job. We denote by $\mathbf{P} = \prod_{i \in I} \mathbf{P}_i$ and $\mathbf{C} = \prod_{i \in I} \mathbf{C}_i$ the sets of rates and execution times for the tasks in Λ, and with $\overline{\rho}$ and $\overline{c}$ points in this sets. A job of a task in the system is defined as:

$$\tau_{ij} = \left(c_{ij}, \rho_{ij}, r_{ij}\right)$$

where: c_{ij}, ρ_{ij}, and r_{ij}, are the execution time, rate, and response time of the jth job of task i.

The tasks in the system are scheduled using any scheduler which has the following two properties:

1. it is *non-idling*: it does not leave the processor idle if there are pending jobs;
2. it executes successive jobs of the same task in the order of their release time.

A resource manager is running on the processor, whose goal is to measure execution times, and then adjust job release rates for all tasks, such that the CPU utilization is kept high, and the amount of time spent in overload situations is minimized. We consider the system stable if, under the worst possible run-time scenario, the overload in the system is kept finite, meaning that the system does not keep accumulating jobs without having a way of executing them.

4.2 Processor Behavior in Overload Situations

In overload situations jobs cannot be executed at the rate at which they are released. In this condition, newly released jobs need to wait for the previous ones to finish. We consider that newly released jobs queue up in queues, one for each task. A job τ_{ij} gets executed only after all previous queued up jobs of task τ_i finished executing.

4.3 Resource Utilization and Schedulability

The resource considered in this paper is the CPU time. The *resource utilization*, in any interval of time h, is the fraction of h when the CPU is non-idle. This is a positive number less/equal 1:

$$u = \frac{\text{non-idle time}}{h}, u \in [0,1]$$

The *resource demand*, in an interval of time h, is:

$$u^D = \frac{C_{\text{previous}} + C_{\text{current}}}{h}, u^D \geq 0$$

where C_{current} is the sum of execution times of all jobs released during h and C_{previous} is the sum of execution times of all queued up jobs, released before the beginning of the interval. The resource demand may be larger then 1. Figure 1 shows an example with three tasks. At time instance t_1, task τ_1 has 2 jobs in its queue, τ_2 has 4, and τ_3 has 1. C_{previous} will be the sum of the execution times of all these jobs. Between t_1 and

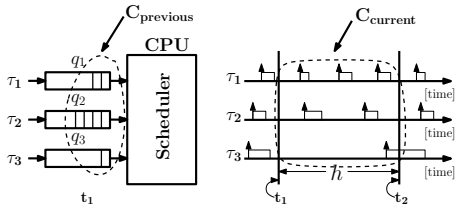


Fig. 1: Resource demand in the system, during time interval h.

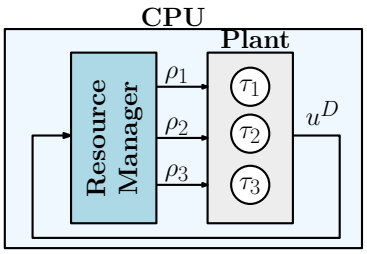


Fig. 2: A system is seen as a control loop, where the resource manager controls the resource demand by adjusting task rates. The tasks constitute the controlled plant.

t_2, τ_1 releases 3 new jobs, τ_2 releases 2, and τ_3 1. In this case C_{current} will be equal to the sum of execution times of all these six jobs.

When the resource demand is less than or equal to 1, then it will be equal with the resource utilization and the system will be schedulable; all the demand can be executed. When the resource demand is above 1 the system is overloaded, only a portion of the demand can be executed (namely 1), and jobs start accumulating in queues. Also, we must note that execution times change with every job, and they are unknown before the job's completion. Therefore, at any moment, the resource demand can only be predicted.

We consider that the employed resource manager controls the system by adjusting job release rates and, thus, controlling the resource demand.

4.4 Control Theoretic View of a Real-Time System and its Parameters

The model of our system can be depicted as in Figure 2. While the tasks and the resource manager run on the same CPU, from the modeling perspective the tasks form the plant, and the resource manager is the controller controlling the resource demand of the plant by means of adjusting task rates.

We are interested in describing the behavior of the our system at discrete moments in time $t_{[k]}$, $k \in \mathbb{Z}_+$, when the resource manager (controller) actuates. Figure 3 presents a example system with n tasks, and shows how it evolves during the time interval $[t_{[k]}, t_{[k+1]}]$. At $t_{[k]}$, the controller will choose new task rates $\rho_{i[k+1]}$ (possibly based on knowledge of queue sizes $q_{i[k]}$ and estimates of future average job execution times $c_{i[k+1]}$). For task τ_1 it happens that new jobs get released immediately after the new rate is available, however this is not the general behavior of the system. Jobs of tasks are released only after the remaining parts of the period of the last released jobs finish. We than say that the new jobs get released with an offset $\phi_{i[k]}$ after $t_{[k]}$. This offset may also be larger than the period of the controller $h = t_{[k+1]} - t_{[k]}$, as it happens for τ_n.

The presentation here is only a qualitative description of the behavior of the system, that we give in order to introduce the parameters of interest regarding its evolution. In Sections 7 and 8 we will develop concrete models of our system.

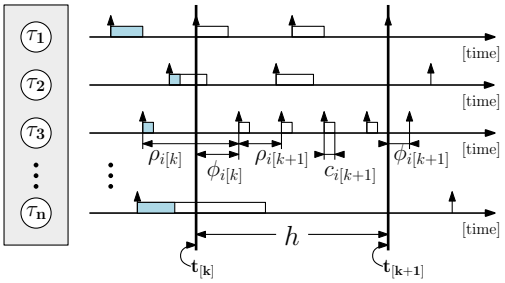


Fig. 3: Parameters related to the resource demand, and the state of the system.

5 Problem Formulation

We consider any task set Λ and a resource manager whose job is to keep the resource demand $u^D = 1$ by adjusting task rates. Our goal is to model the resource demand, the behavior of the real-time system and resource manager in order to determine conditions under which the whole system is stable. By stability we mean that the resource demand in the system is bounded and job response times do not grow infinitely.

6 Control Theoretic Notions of System Modeling and Stability

This section covers some background material that we use later in the modeling and stability analysis sections of this paper.

6.1 System Modeling

We will model our real-time system as a discrete-time dynamical system which evolves at discrete moments in time $t_{[k]}$. The system is described by a system of difference equations.

$$F(\bar{x}_{[k+1]}, \bar{x}_{[k]}) = 0 \tag{1}$$

where $\bar{x}_{[k+1]}$ and $\bar{x}_{[k]}$ are the state vectors of the system at the future ($t_{[k+1]}$) and the current ($t_{[k]}$) time moments, and F is some function of the state vectors. In our real-time system, the state is represented by the resource demand u^D and, therefore, the state vector will be comprised of the elements that determine it: queue sizes, task rates, and execution times. In this work we model the ensemble of controller and plant together, thus parameters that may be regarded as inputs (i.e. task rates) and disturbances (execution times) are embedded in the state.

6.2 Stability of Discrete-Time Dynamical Systems

A discrete-time dynamical system is a tuple $\{\mathscr{T}, \mathscr{X}, \mathscr{A}, \mathscr{S}\}$ where $\mathscr{T} = \{t_{[k]} | k \in \mathbb{N}, t_{[k]} > t_{[k-1]} > \cdots > t_{[0]} = 0\}$ is the set of actuation times, $\mathscr{X}$ is the state space,

$\mathscr{A} \subset \mathscr{X}$ is the set of all initial states of the system ($\bar{x}_{[0]}$), and $\mathscr{S}$ is the set of all trajectories (all solutions of (1) : $\bar{x}_{[k]} = p(k, \bar{x}_{[0]})$ where $t_{[k]} \in \mathscr{T}$ and $\bar{x}_{[0]} \in \mathscr{A}$). Also the state space must be a metric space $(\mathscr{X}, d)$, where $d : \mathscr{X} \times \mathscr{X} \to \mathbb{R}^+$ is a distance function between two state vectors in $\mathscr{X}$. We use the notation $d(\bar{x}, \mathscr{M}) = \inf_{\bar{y} \in \mathscr{M}} \{d(\bar{x}, \bar{y})\}$ to describe the distance from a state to a set $\mathscr{M}$.

A dynamical system's state is desired to belong to a preselected bounded set of points $\mathscr{M} \subset \mathscr{A}$. Because of the noises in the system, this condition does not hold. Under this premise a dynamical system is said to be stable if its state remains "close" to $\mathscr{M}$ under all noise patterns and initial conditions.

For our real-time system, we will consider the notion of *stability in the sense of Lagrange* [18], where a system is stable if all trajectories starting from states within a ball of size δ around $\mathscr{M}$ are bounded in a larger ball of size $\gamma(\delta)$ around $\mathscr{M}$: $\bar{x}_{[0]} \in B(\delta) \Rightarrow p(k, \bar{x}_{[0]}) \leq B(\gamma(\delta))$. To test for stability, we shall use the following theorem described in [18]:

Theorem 1 (uniform boundedness) *Let us consider the above described dynamical system. Assume that there exists a function $V : \mathscr{X} \to \mathbb{R}_+$ and two strictly increasing functions $\varphi_1, \varphi_2 : \mathbb{R}_+ \to \mathbb{R}_+$ with $\lim_{r \to \infty} \varphi_i(r) = \infty, i = \overline{1,2}$, such that*

$$\varphi_1(d(\bar{x}, \mathscr{M})) \leq V(\bar{x}) \leq \varphi_2(d(\bar{x}, \mathscr{M})) \tag{2}$$

for all $\bar{x} \in \mathscr{X}$ whenever $d(\bar{x}, \mathscr{M}) \geq \Omega$, where Ω is a positive constant.

Also, assume that $V(p(k, \bar{x}_{[0]}))$ is non-increasing for all trajectories $p(k, \bar{x}_{[0]})$ from $\mathscr{S}$ whenever $d(p(k, \bar{x}_{[0]}), \mathscr{M}) \geq \Omega$. Assume that there exists a constant $\Psi \geq \Omega$ such that $d(p(k+1, \bar{x}_{[0]}), \mathscr{M}) \leq \Psi$ whenever $d(p(k, \bar{x}_{[0]}), \mathscr{M}) \leq \Omega$.

If the above assumptions hold, then the system is uniformly bounded.

Any system that satisfies the above theorem is stable, and this means that its state becomes trapped in the ball of size Ψ around the set $\mathscr{M}$, after a certain amount of time (possibly infinite), regardless of the initial state $\bar{x}_{[0]}$. In practice however, only initial states within Ω will be of relevance to us, making Ψ a measure of the worst-case performance of the system. We shall give further insight into this stability condition in Section 13.1.

In Sections 7 and 8 we demonstrate stability by showing uniform boundedness according to Theorem 1. Given the definition of the state for our models (Sections 7.2 and 8.1) such a stability implicitly means that queue sizes, response times, and resource demand are bounded.

7 Constrained Model of the Real-Time System

In this section we develop a simple but constrained model for our system and determine conditions under which it is stable. In Section 8 we will derive a generalized model of the system.

7.1 Assumptions

The constrained model is based on the assumption that the time interval between two successive actuations of the controller (the controller's period) $h = t_{[k+1]} - t_{[k]}$ is much larger then the largest possible offset in the system:

$$\phi_{\max} < \max_{i \in I} \left\{ \frac{1}{\rho_i^{\min}} \right\} \ll h$$

Since h is large, we can neglect offsets in this model. Systems described by this model will be called *constrained systems*.

7.2 Model

We will start to model our systems from the definition of resource demand (Section 4.3). By disregarding task offsets, the formula is:

$$u^D_{[k]} = \frac{1}{h} \Big(\underbrace{\sum_{i \in I} c_{i[k]} \cdot q_{i[k-1]}}_{C_{\text{previous}}} + \underbrace{\sum_{i \in I} c_{i[k]} \cdot \rho_{i[k]} \cdot h}_{C_{\text{current}}} \Big) \tag{3}$$

The first sum represents the accumulation of execution times from previously released, but not executed (queued up) jobs. The second sum represents the accumulation of execution times from jobs that are released during $[t_{[k-1]}, t_{[k]}]$. We note that $c_{i[k]}$ represents the average execution time of the jobs of τ_i that were executed during $[t_{[k-1]}, t_{[k]}]$.

As we can observe from Equation (3), the model of our system must contain states for the queue sizes, execution times, and rates. The task queues evolve in time depending on the actual arrival time of jobs, on the actual execution times of each job, and on the schedulers used in the system. In between any two moments of time $t_{[k]}$ and $t_{[k+1]}$, the value $h \cdot u^D_{[k+1]}$ represents the number of jobs, given as an amount of execution time, that need to be executed. The resource can execute only an amount equaling to at most $t_{[k+1]} - t_{[k]} = h$. The rest will then remain as jobs in the task queues. We thus have:

$$\sum_{i \in I} c_{i[k+1]} \cdot q_{i[k+1]} = h \cdot \max \left\{ 0, u^D_{[k+1]} - 1 \right\} \tag{4a}$$

$$\overline{q^{\text{p}}}_{[k+1]} = \overline{q}_{[k]} \tag{4b}$$

We use the notation $\overline{q}_{[k]}$ for the vector of queue sizes, and $\overline{q^{\text{p}}}_{[k]}$ for the vector of queue sizes at the previous time instance. Equation (4b) is only needed because we wish to write the resource demand as a function of the state of the system $u^D_{[k]} \overset{not}{=} u^D(\bar{x}_{[k]})$.

Next, we model the average execution time of the jobs that will be executed in the current interval of time $[t_{[k]}, t_{[k+1]}]$, as being some function $f_p : \mathscr{X} \to \mathbf{C}$ of the previous

state. This function is unknown to us because execution time are disturbances that vary in unknown ways.

$$\overline{c}_{[k+1]} = f_p(\overline{x}_{[k]}) \tag{4c}$$

By the assumptions made in Section 4.1, we know that $\overline{c}_{[k+1]} \in \mathbf{C}$ and this is sufficient knowledge for us.

Equations (4a) to (4c) represent the model of the plant. We complete the model of the system with the model of the controller (resource manager). The controller's job is to compute new rates:

$$\overline{\rho}_{[k+1]} = f_c(\overline{x}_{[k]}) \tag{5}$$

Here we simply model the control law as being any function of the state of the system. The state vector of the system is:

$$\overline{x}_{[k]} = \left(\overline{q}_{[k]}, \overline{q^{\mathrm{p}}}_{[k]}, \overline{c}_{[k]}, \overline{\rho}_{[k]}\right)^T.$$

Equations (4a) to (4c), and (5) represent the model of our real-time system. It is a model of a discrete-time dynamical system $\{\mathscr{T}, \mathscr{X}, \mathscr{A}, \mathscr{S}\}$ where $\mathscr{T}$ is defined as in Section 6.2, $\mathscr{X} = \mathbb{R}_+^n \times \mathbb{R}_+^n \times \mathbf{C} \times \mathbf{P}$, $\mathscr{A} = \mathscr{X}$ and S is the set of all solutions of equations (4a) to (4c), and (5). We make the observation that $\mathscr{X} \subset \mathbb{R}^{4n}$ where n is the number of tasks in the system and, therefore, is a metric space for the usual distances. Throughout the rest of this paper we will use the Chebyshev distance [19]:

$$d(\overline{x}, \overline{y}) = \sup_{i=\overline{1,4n}} \{|x_i - y_i|\}$$

The above model describes a class of systems, namely the systems generated by all combinations of existing f_p and f_c functions, and all allowable schedulers (see Section 4), for all instances of applications Λ. *Our goal is to determine the class of controllers that leads to stable systems.*

7.3 Stability

In this section we want to determine conditions that our controller must satisfy, in order for the system to be stable (satisfy Theorem 1). We first define the notions involved in Theorem 1 and then we determine the stability conditions.

We define the set of states $\mathscr{M}$ as being all the states where $u^D = 1$. The following lemma determines that $\mathscr{M}$ is bounded.

Lemma 1 *The set $\mathfrak{M}^\alpha = \{\overline{x} \in \mathscr{X} | u^D(\overline{x}) = \alpha\}$ is bounded, where $\alpha > 0$ is an arbitrary constant.*

Proof From equation (4a), (4b) and (3) we have

$$\sum_{i \in I} c_i \cdot q_i^{\mathrm{p}} = h\big(\alpha - \sum_{i \in I} c_i \cdot \rho_i\big)$$

and it quickly follows that the largest distance between two states in $\mathfrak{M}^{\alpha}$ is bounded by

$$d^{\alpha} = \max_{j \in I} \{|q_j^{\alpha} - 0|, |q_j^{\mathrm{p}\alpha} - 0|, |c_j^{\max} - c_j^{\min}|, |\rho_j^{\max} - \rho_j^{\min}|\}$$

where:

$$q_j^{\mathrm{p}\alpha} = \frac{h}{c_j^{\min}} \Big(\alpha - \sum_{i \in I} c_i^{\min} \cdot \rho_i^{\min} \Big),$$

and

$$q_j^{\alpha} = \frac{h}{c_j^{\min}} \cdot \max \Big\{ 0, \alpha - 1 \Big\}, \qquad \forall j \in I$$

□

Since $\mathscr{M} = \mathfrak{M}^1$, $\mathscr{M}$ is bounded as well.

The following theorem gives a necessary condition for a system to be stable.

Theorem 2 *A necessary condition for a system to be stable is:*

$$\sum_{i \in I} c_i^{max} \cdot \rho_i^{min} \leq 1 \tag{6}$$

Proof We prove this by contradiction. We allow Equation (6) not to hold and we will show that this leads to an unstable system. If equation (6) does not hold, then we have:

$$\sum_{i \in I} c_i^{\max} \cdot \rho_i^{\min} - 1 = \beta > 0$$

In the case that $c_{i[k]} = c_i^{\max}, \forall k \geq 0$, even if the controller sets the smallest rates ($\overline{\rho}_{[k]} = \overline{\rho}^{\min}$), for all $k \geq 0$, we have from above, and from equations (3), and (4a) that:

$$u_{[k+1]}^D - 1 = u_{[k]}^D - 1 + \beta \quad \Rightarrow \quad u_{[k+1]}^D = u_{[0]}^D + (k+1) \cdot \beta$$

and thus $\lim_{k \to \infty} u_{[k]}^D = \infty$. □

Theorem 2 implies that, in order to achieve stability, there must exist at least a rate vector (the rate vector consisting of the smallest rates for each task), that the controller can choose such that, when jobs have worst case execution times, the contribution to resource demand in each time interval is not more than 1. Otherwise, the queue sizes will continue growing irrespective of the controller and scheduler used, and the resource demand will be unbounded. For the rest of the paper, we will only consider systems that satisfy Theorem 2. We continue by defining the set:

$$\Gamma_{\star} = \Big\{ \overline{\rho}^{\star} \in \mathbf{P} \Big| \sum_{i \in I} c_i^{\max} \cdot \rho_i^{\star} \leq 1, \Big\} \neq \emptyset \tag{7}$$

which is the set of all rate vectors that, if chosen, guarantee that job queue sizes do not continue growing, irrespective of the execution times of the jobs. If the system satisfies Theorem 2, $\Gamma_{\star}$ contains at least one rate vector.

Next, we determine sufficient conditions for the controller, in order to render the system stable.

Theorem 3 *For any constrained system that satisfies Theorem 2 and for any $u^D_\Omega > 1$ a sufficient stability condition is that its controller satisfies:*

$$\overline{\rho}_{[k+1]} \in \begin{cases} \Gamma_\star, & \text{if } u^D_{[k]} \geq u^D_\Omega \\ \mathbf{P}, & \text{otherwise} \end{cases} \tag{8}$$

Proof Ultimately, stability means that queue sizes ($\overline{q}$ and $\overline{q^{\mathrm{p}}}$) have upper bounds. We first proceed on defining the function $V(\overline{x}_{[k]})$ (noted $V_{[k]}$ henceforward) and finding the two function $\varphi_1(d(\overline{x}_{[k]}, \mathscr{M}))$ and $\varphi_2(d(\overline{x}_{[k]}, \mathscr{M}))$ such that equation (2) holds. Considering the model of queue sizes (equations (4a) and (4b)), and considering the worst-case noises in the system (noises are due to inaccuracies in predicting execution times and in measuring queues), we define $V_{[k]}$ as:

$$V_{[k]} = \max\left\{0, \frac{1}{h}\sum_{i\in I} c_i^{\max}\cdot(q^{\mathrm{p}}_{i[k]}+1) + \sum_{i\in I} c_i^{\max}\cdot\rho_{i[k]} - 1\right\} \tag{9}$$

To determine φ_1 we observe that $V_{[k]} \geq \max\{0, u^D_{[k]} - 1\}$ and we construct the set

$$\mathscr{V}_{\alpha_V} = \left\{\overline{x} \in \mathscr{X} \,\middle|\, u^D(\overline{x}) - 1 = \alpha_V\right\} = \mathfrak{M}^{1+\alpha_V}$$

where $\alpha_V > 0$ is an arbitrary constant. We observe that a bound on the largest distance between a state in $\mathscr{V}_{\alpha_V}$ and a state in $\mathscr{M}$ is given by $d^{1+\alpha_V}$, and that:

$$d^{1+\alpha_V} = \frac{h}{\min_{j\in I}\{c_j^{\min}\}}\left(\alpha_V + 1 - \sum_{i\in I} c_i^{\min}\cdot\rho_i^{\min}\right)$$

for all $d^{1+\alpha_V} \geq d^1$. We can also say that $d(\overline{x}_{[k]}, \mathscr{M}) \leq d^{1+\alpha_V}$ and $V(\overline{x}) \geq \alpha_V, \forall \overline{x} \in \mathscr{V}_{\alpha_V}$. Since α_V was chosen arbitrarily and the above expression of $d^{1+\alpha_V}$, as a function of α_V, is invertible, we obtain $\varphi_1(d(\overline{x}_{[k]}, \mathscr{M}))$ in equation (10) which satisfies the conditions in Theorem 1.

$$\varphi_1(d) = \begin{cases} \frac{\min_{j\in I}\{c_j^{\min}\}}{h}\cdot d + \sum_{i\in I} c_i^{\min}\cdot\rho_i^{\min} - 1, & \text{if } d \geq \Omega > d^1 \\ \left(\frac{\min_{j\in I}\{c_j^{\min}\}}{h} + \frac{1}{\Omega}\left(\sum_{i\in I} c_i^{\min}\cdot\rho_i^{\min} - 1\right)\right)\cdot d, & \text{otherwise} \end{cases} \tag{10}$$

To determine φ_2 we construct the set of all states at a distance α_d or less from $\mathscr{M}$:

$$\mathscr{M}_{\alpha_d} = \{\overline{x} \in \mathscr{X} \,|\, d(\overline{x}, \mathscr{M}) \leq \alpha_d\} = \{\overline{x} \in \mathscr{X} \,|\, d(\overline{x}, \overline{y}) = \alpha_d,\ \forall \overline{y} \in \mathscr{M}\}$$

where $\alpha_d > 0$ is an arbitrary constant. We than determine which of the states in $\mathscr{M}_{\alpha_d}$ has the highest value of V. If

$$\overline{y} = \left(\overline{q}, \overline{q^{\mathrm{p}}}, \overline{c}, \overline{\rho}\right)^T \in \mathscr{M} \text{ and}$$
$$\overline{x} = \left(\overline{q} + d\overline{q}, \overline{q^{\mathrm{p}}} + d\overline{q^{\mathrm{p}}}, \overline{c} + d\overline{c}, \overline{\rho} + d\overline{\rho}\right)^T \in \mathscr{M}_d$$

then we have

$$V(\bar{x}) = \max\left\{0, \sum_{i\in I} c_i^{\max}\left(\frac{q_i^{\mathrm{p}} + dq_i^{\mathrm{p}} + 1}{h} + \rho_i + d\rho_i\right) - 1\right\} \tag{11}$$
$$\leq \frac{\alpha_d}{h}\sum_{i\in I} c_i^{\max} + \max\left\{0, \sum_{i\in I} c_i^{\max}\left(\frac{q_i^{\mathrm{p}}\alpha_d + 1}{h} + \rho_i^{\max}\right) - 1\right\}, \forall\, \bar{x} \in \mathscr{M}_{\alpha_d}$$

and since α_d was chosen arbitrarily, we obtain $\varphi_2(d(\bar{x}, \mathscr{M}))$ as being the right half of the inequality (11).

To determine Ω, we construct the set $\mathscr{V}_{u_\Omega^D - 1}$ and we observe that $\Omega = d^{u_\Omega^D}$. It seems that we can only choose u_Ω^D sufficiently large such that $d^{u_\Omega^D} \geq d^1$, however in practice, we can build an equivalent model of the system by applying a linear transformation to the state space, where this condition always holds, thus this is a non-issue.

Next, we proceed on showing that $V_{[k+1]} \leq V_{[k]}, \forall d(\bar{x}_{[k]}, \mathscr{M}) \geq \Omega$. In this case, we have from equations (4a), (4b), (8), and (9):

$$V_{[k+1]} = \max\left\{0, \sum_{i\in I} c_i^{\max}\frac{q_{i[k]} + 1}{h} + \underbrace{\sum_{i\in I} c_i^{\max}\cdot\rho_{i[k+1]} - 1}_{\leq 0}\right\}$$

When the value inside the max function is larger than 0, we observe from equations (4a) and (3) that $q_{i[k]} = q_{i[k]}^{\mathrm{p}} + \rho_{i[k]}\cdot h - e_{i[k]}$, where $e_{i[k]} \geq 0$ is the number of jobs of τ_i executed during the time interval $[t_{[k-1]}, t_{[k]}]$, and $\frac{1}{h}\sum_{i\in I} c_{i[k]}\cdot e_{i[k]} = 1$. $e_{i[k]}$, for all $i \in I$ depend, amongst others, on the scheduler used, and are typically unknown. Regardless of their value, however, inequation (12) holds.

$$V_{[k+1]} \leq \sum_{i\in I} c_i^{\max}\frac{q_{i[k]} + 1}{h} = \sum_{i\in I} c_i^{\max}\left(\frac{q_{i[k]}^{\mathrm{p}} + 1}{h} + \rho_{i[k]}\right) - \frac{1}{h}\sum_{i\in I} c_i^{\max}\cdot e_{i[k]} \leq V_{[k]} \tag{12}$$

To complete the proof, we must show that there exists a value $\Psi > 0$ such that for all states with $d(\bar{x}_{[k]}, \mathscr{M}) \leq \Omega$ we have $d(\bar{x}_{[k+1]}, \mathscr{M}) \leq \Psi$. Since $q_{i[k]} \leq \Omega$, $c_{i[k]} \leq c_i^{\max}$, and $\rho_{i[k]} \leq \rho_i^{\max}$, $\forall i \in I$, we then have

$$u_{[k+1]}^D \leq \sum_{i\in I} c_i^{\max}\left(\frac{\Omega + 1}{h} + \rho_i^{\max}\right) = u_\Psi^D \tag{13}$$

All subsequencent states $\bar{x}_{[k']}$, $k' \geq k+1$ will have $u_{[k']}^D \leq V_{[k+1]} + 1$ if $u_\Psi^D \geq u_\Omega^D$ or, otherwise $u_{[k']}^D \leq u_\Omega^D$. Then we have $\Psi = \varphi_2^{-1}(\max\{u_\Psi^D, u_\Omega^D\})$.

With the above, the proof of Theorem 3 is complete. □

Any controller that satisfies the above theorem guarantees the system to be stable. Observe that the condition (8) is a condition for the controller used in the system. As long as the task execution times and rates are such that $u_{[k]}^D < u_\Omega^D$, the controller is free to chose any rate $\rho_{i[k+1]} \in [\rho_i^{\min}, \rho_i^{\max}]$. The controller will choose rates according to the particular control goals requested by the application (e.g. keeping processor

utilization high, providing QoS guarantees, reducing energy consumption, etc.). Once $u^D_{[k]} \geq u^D_\Omega$ (which means that the system reached a certain critical level of overload), the controller will have to choose rate vectors from the set $\Gamma_\star$, and keep doing so until a time instance $t_{[k']}$, when $u^D_{[k']} < u^D_\Omega$.

8 General Model of the Real-Time System

In this section we extend the previous model of a real-time system, by considering task offsets. This will allow us to consider controller periods which are arbitrary, therefore removing the limitation of the previous model (Section 7). Systems described according to this model will be called *general systems*.

This section is organized as the previous one: we start by giving a model of the system and, then, we analyze its stability.

8.1 Model

We first describe the resource demand in the system:

$$u^D_{[k]} = \sum_{i \in I} c_{i[k]} \cdot \frac{q_{i[k-1]} + \left\lceil \rho_{i[k]} \cdot \max\{0, h - \phi_{i[k-1]}\} \right\rceil}{h} \tag{14}$$

In the above definition the quantity $\left\lceil \rho_{i[k]} \cdot \max\{0, h - \phi_{i[k-1]}\} \right\rceil$ describes the exact number of jobs of each task τ_i released during $[t_{[k-1]}, t_{[k]}]$. The ceiling function appears because this number is an integer value. For the constrained model, we have used $\rho_{i[k]} \cdot h$ in (3) as an approximation.

The following equations give the model of the plant:

$$\sum_{i \in I} c_{i[k+1]} \cdot q_{i[k+1]} = h \cdot \max\{0, u^D_{[k+1]} - 1\} \tag{15a}$$

$$\overline{q^{\mathrm{p}}}_{[k+1]} = \overline{q}_{[k]} \tag{15b}$$

$$\overline{c}_{[k+1]} = f_p(\overline{x}_{[k]}) \tag{15c}$$

$$\phi_{i[k+1]} = \phi_{i[k]} + \frac{1}{\rho_{i[k+1]}} \left\lceil \rho_{i[k+1]} \cdot \max\{0, h - \phi_{i[k]}\} \right\rceil - h \tag{15d}$$

$$\overline{\phi^{\mathrm{p}}}_{[k+1]} = \overline{\phi}_{[k]} \tag{15e}$$

Equations (15a) to (15c) have the same meaning as for the constrained model. Equation (15d) models the evolution of offsets in the system and equation (15e) has the sole role of allowing us to write $u^D_{[k]}$ as a function of the state. To understand equation (15d) we invite the reader to take a look at Figure 3 on page 9. In any interval of time $[t_{[k]}, t_{[k+1]}]$, jobs of any task τ_i are released in the interval $[t_{[k]} + \phi_{i[k]}, t_{[k+1]}]$, if, of course, $t_{[k]} + \phi_{i[k]} \leq t_{[k+1]}$. The number of released jobs is an integer and is $\left\lceil \rho_{i[k+1]} \cdot \max\{0, h - \phi_{i[k]}\} \right\rceil$. The interval of time over which these jobs spread starts at $t_{[k]} + \phi_{i[k]}$, has a length of $1/\rho_{i[k+1]} \cdot \left\lceil \rho_{i[k+1]} \cdot \max\{0, h - \phi_{i[k]}\} \right\rceil$, and will in general finish after $t_{[k+1]}$. By considering that $t_{[k+1]} - t_{[k]} = h$ we obtain $\phi_{i[k+1]}$ as in (15d).

We assume the same model for the controller as in Equation (5). The general model of a real-time system is, thus, given by equations (15a) to (15e) and (5).

Since the number of states has grown, the state space is larger but is again of the form $\mathbb{R}^n$ and we shall again use the Chebyshev norm. Also note that offsets are bounded by $\phi_i \in [0, 1/\rho_i^{\max}]$, $\forall i \in I$.

8.2 Stability

Theorem 4 *For any general system that satisfies Theorem 2 and for any $u_\Omega^D > 1$, a sufficient stability condition is that its controller satisfies:*

$$\overline{\rho}_{[k+1]} \in \begin{cases} \Gamma_\star, & \text{if } u_{[k]}^D \geq u_\Omega^D \\ \mathbf{P}, & \text{otherwise} \end{cases} \tag{16}$$

$$\rho_{i[k+1]} \leq \rho_{i[k]} \quad \text{if } u_{[k]}^D \geq u_\Omega^D, \quad \forall i \in I \tag{17}$$

Proof Compared with the constrained model, we have an extra condition that the controller must satisfy (equation (17)). We want to show that the conditions are sufficient to make $\overline{q}_{[k]}$ and $\overline{q^{\mathrm{p}}}_{[k]}$ bounded.

We define the function V as in the following equation:

$$\begin{aligned} V_{[k]} &= \max\left\{0, \sum_{i\in I} c_i^{\max} \frac{q_{i[k]}^{\mathrm{p}} + 1 + \left\lceil \rho_{i[k]} \cdot \max\{0, h - \phi_{i[k]}^{\mathrm{p}}\} \right\rceil + 1 - \rho_{i[k]} \cdot \phi_{i[k]}}{h} + \mu - 1\right\} \\ &= \max\left\{0, \frac{1}{h}\sum_{i\in I} c_i^{\max}\left(q_{i[k]}^{\mathrm{p}} + \eta\right) + \sum_{i\in I} c_i^{\max} \cdot \rho_{i[k]} - 1 - \frac{1}{h}\sum_{i\in I} c_i^{\max} \cdot \rho_{i[k]} \cdot \phi_{i[k]}^{\mathrm{p}}\right\} \end{aligned} \tag{18}$$

where

$$\mu = \begin{cases} 0 & \text{, if } \overline{\rho}_{[k]} \in \Gamma_\star \\ \sum_{i\in I} c_i^{\max} & \text{, otherwise.} \end{cases}, \qquad \eta = \begin{cases} 2 & \text{, if } \overline{\rho}_{[k]} \in \Gamma_\star \\ 3 & \text{, otherwise.} \end{cases}$$

At any moment of time $t_{[k]}$, the function V represents the accumulation of execution times, that should have been executed by $t_{[k]}$[1]. Since in each queue we consider the whole execution time of all released jobs, we must remove the part of each job that should be executed after the end of the resource manager's period (this is done by the term $-\frac{1}{h}\sum_{i\in I} c_i^{\max} \cdot \rho_{i[k]} \cdot \phi_{i[k]}^{\mathrm{p}}$).

We want to show that the function V decreases in value when $u_{[k']}^D \geq u_\Omega^D$, $\forall k' \geq k$. This should be true because we only select rate vectors in $\Gamma_\star$ (because of condition (16)) which satisfy equation (6). However, we observe that whenever new rates are computed, they only take effect after their offset. Since the offsets for the different tasks are different, there is an interval of time when some of the tasks have the new rates and some of them have the old rates. We must make sure that also the rate vectors during this interval of time belong to $\Gamma_\star$ as well, or the $V(\cdot)$ function might increase otherwise. We show an example of this in Figure 4 for a system of two tasks.

[1] Strictly speaking, V represents a load value. To obtain the accumulation of execution times, one must multiply the function with h.

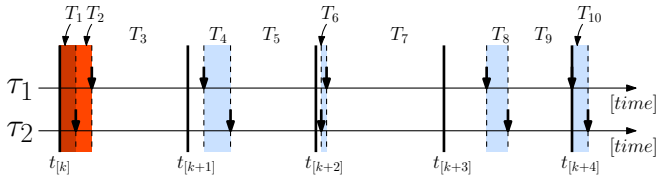


Fig. 4: Behaviour of a system with two tasks, when $u^D_{[k']} \geq u^D_{\Omega}, \forall k' \geq k$.

During intervals of time T_3, T_5, T_7, T_9, ... the accumulation of execution times added in the system is less than the size of the respective intervals, therefore the total accumulation drops. Condition (17) ensures that the same happens over the transition intervals T_4, T_6, T_8, T_{10}, ... The accumulation of execution times from the intervals T_1 and T_2 may be larger than $T_1 + T_2$ because $\overline{\rho}_{[k]} \notin \Gamma_\star$, however, it is bounded by $\sum_{i \in I} c_i^{\max}$. This explains the two cases in the definition of $V(\cdot)$. Otherwise, since the function V is very similar to the one defined in equation (9), all the steps of this proof are very similar with the proof of Theorem 3 and in the interest of brevity we shall not discuss them here.

For determining Ψ, we use a similar reasoning as in Theorem 3 and we obtain that:

$$u^D_{[k+1]} \leq u^D_{[k+1]} \leq \sum_{i \in I} c_i^{\max} \left(\frac{\Omega + 1}{h} + \rho_i^{\max} \right) = u^D_{\Psi} \tag{19}$$

Thus, any general system satisfying the conditions in Theorem 4 is stable. □

We observe that, because of Condition (17), for the general model, the stability criterion is more restrictive than for the constrained model. This is because, including offsets, the general model considers in more depth the transition between different rates, which leads to an additional condition that needs to be satisfied to ensure that this transition is done safely. For both models, we have required that $V(\overline{x}_{[k+1]}) \leq V(\overline{x}_{[k]})$ holds. This means that if the starting point of the system is outside Ω, the system may never reach Ω within a finite amount of time. This is not a problem in practice, because systems usually start with empty queues, thus from within Ω. However, $\Gamma_\star$ can easily be modified such that stronger assumption for V hold. We will further discuss this aspect in Section 13.1.

9 Worst Case Response Time Bound

For any controller that satisfies the stability condition in Theorem 4 there exists a finite response time for each task. The actual value of the response time depends on the concrete scheduling policy and the controller (f_c) used in the system. In this paper we will develop bounds on the worst case response time for tasks considering an EDF scheduler [15] and two classes of controllers. The bounds developed here are different from the well known worst case response times derived in literature for EDF, since our system allows overload situations. The EDF scheduler considers as a

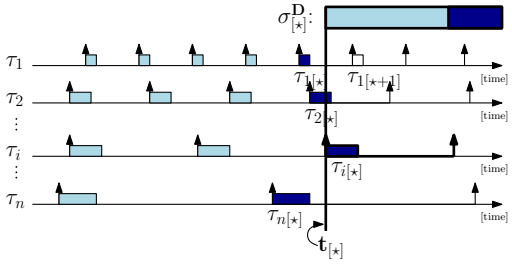


Fig. 5: Accumulation of execution times

working deadline for each job τ_{ij}, the sum of its release time and $1/\rho_{ij}$, where ρ_{ij} is the current job's rate.

For the following analysis, we will consider that the system always starts from a state where the queues are empty.

$$\mathscr{A} = \left\{ \bar{x}_{[0]} \in \mathscr{X} \,\middle|\, q = 0 \right\} \tag{20}$$

In this case $d(\bar{x}_{[0]}, \mathscr{M}) \leq d^1 \leq \Omega$, $\forall \bar{x} \in \mathscr{A}$ holds. According to equations (13) and (19) (for the constrained and general model respectively), u^D_Ψ is the highest resource demand ever achieved in the system. This result is important, since it allows us to bound the overload in the system.

At a certain moment in time $t_{[\star]}$, a new job of a task τ_j is released and we wish to compute its response time. We will denote this job with $\tau_{j[\star]}$. In the system, at $t_{[\star]}$ there already exists a certain number of un-executed jobs and their total accumulation of execution times is:

$$\sigma^D_{[\star]} = \sum_{i \in I} c_{i[\star]} \cdot q_{i[\star]}$$

where $q_{i[\star]}$, $\forall i \in I$ are the queue sizes of each task and $c_{i[\star]}$ are the average execution time for the jobs in the queues (these averages are unknown, but $\bar{c}_{[\star]} \in \mathbf{C}$). Figure 5 illustrates this situation for a system of n tasks. All the jobs depicted in the figure are not yet executed at the moment $t_{[\star]}$, when $\tau_{j[\star]}$ is released. The dark shaded jobs represent the last released jobs of the tasks, before the moment $t_{[\star]}$. The light shaded jobs have been released before the dark shaded ones, and their deadlines are guaranteed to be prior to $t_{[\star]}$. $\sigma^D_{[\star]}$ is the sum of execution times of the light and dark shaded jobs. Since in overload situations EDF executes jobs non-preemptively, in the order of their working deadline, all light colored jobs in the figure will be executed before $\tau_{j[\star]}$, since their deadlines are before $t_{[\star]}$. The execution times of these jobs represent $\sum_{i \in I} c_{i[\star]} \cdot (q_{i[\star]} - 1)$ out of $\sigma^D_{[\star]}$. From the rest of the jobs considered in $\sigma^D_{[\star]}$ (the dark colored ones), the ones with deadlines smaller than that of $\tau_{j[\star]}$ will be executed before it ($\tau_{1[\star]}$ and $\tau_{2[\star]}$ in the figure), and the rest will be executed after $\tau_{j[\star]}$ ($\tau_{n[\star]}$ in the figure). Also there may exist other, not yet released jobs, that have their deadlines prior to the deadline of $\tau_{j[\star]}$ (e.g. $\tau_{1[\star+1]}$ in Figure 5) which also need to be considered. Taking all of this into account, and considering that $\rho_{i[\star]}$, $\forall i \in I$ are the release

rates of all jobs, the response time of $\tau_{j[\star]}$ is:

$$r_{j[\star]} = \sum_{i \in I} c_{i[\star]} \cdot (q_{i[\star]} - 1) + \sum_{i \in I} c_{i[\star]} \cdot \left\lfloor \frac{\frac{1}{\rho_{j[\star]}} + \frac{1}{\rho_{i[\star]}} - \phi_{i[\star]}}{\frac{1}{\rho_{i[\star]}}} \right\rfloor \tag{21}$$

The following theorem gives an upper bound on the response time of the tasks in the system.

Theorem 5 *An upper bound on the worst-case response time of task τ_j in the system can be computed using the following equation:*

$$r_j^{max} = \frac{1}{\rho_j^{min}} + h \cdot (u_{max}^D - 1) + \sum_{i \in I} c_i^{max} \cdot \lfloor \frac{\rho_i^{max}}{\rho_j^{min}} \rfloor \tag{22}$$

Proof The proof follows from equation (21) by observing that:

1. $\sigma_{[\star]}^D = h \cdot (u_{[\star]}^D - 1) \leq h \cdot (u_{\max}^D - 1)$ (from equation (15a), since the system is overloaded);
2. When $u_{\max}^D$ occurs at time instance $t_{[\star]}$, the last released job of task τ_j is released with at most $1/\rho_j^{\min}$ before $t_{[\star]}$.

□

Up to this point, we have concerned ourselves with the scheduler used, and we have determined a formula for the worst-case response time for EDF assuming knowledge of the largest resource demand in the system ($u_{\max}^D$). One should note that the largest resource demand in the system typically depends on the scheduler and the controller (f_c) used in the system. The value computed in equation (19) is an upper bound on the largest resource demand, since it is independent on these parameters. By adding extra constraints on the scheduler or the controller one may be able to tighten this bound. We will now consider two classes of controllers for which we will determine $u_{\max}^D$. The two classes of controllers are:

$$\mathbf{C1} = \left\{ f_c : \mathscr{X} \to \mathbf{P} \middle| \rho_{[k+1]} \in \begin{cases} \{\overline{\rho}^{\min}\}, & \text{if } \Gamma_{[k]}^{\alpha} = \emptyset \\ \mathbf{P}, & \text{otherwise} \end{cases} \right\} \tag{23}$$

$$\mathbf{C2} = \left\{ f_c : \mathscr{X} \to \mathbf{P} \middle| \rho_{[k+1]} \in \begin{cases} \{\overline{\rho}^{\min}\}, & \text{if } \Gamma_{[k]}^{\alpha} = \emptyset \\ \Gamma_{[k]}^{\alpha}, & \text{otherwise} \end{cases} \right\} \tag{24}$$

where $\Gamma_{[k]}^{\alpha}$ is

$$\Gamma_{[k]}^{\alpha} = \left\{ \overline{\rho} \in \mathbf{P} \middle| \sum_{i \in I} c_{i[k]}^p \cdot \frac{q_{i[k]}^{\mathrm{p}} + \lceil \rho_i \cdot \max\{0, h - \phi_{i[k]}\} \rceil}{h} = \alpha,\ \overline{c}_{[k]}^p = f_p(\overline{c}_{[k]}) \right\} \tag{25}$$

and $\alpha > 1$ is an arbitrary constant ($\overline{\phi}^p$ is obtained from equation (15e), according with the chosen $\overline{\rho}$). $\Gamma_{[k]}^{\alpha}$ is the set of all rate vectors which will lead to $u_{[k+1]}^D = \alpha$ (see equation (14)).

The intuition behind the two classes of controllers is the following: **C2** always tries to take decisions such that u^D is kept very aggressively around α. When $u_{[k]}^D \neq \alpha$, the resource demand will be brought back to α as soon as possible ($u_{[k+1]}^D = \alpha$ if the prediction is correct). **C1** is a class of more general controllers, which includes **C2** as a particular case. We first show that both classes of controllers lead to stable systems.

Lemma 2 *Any system for which $f_c \in$ **C1** is stable and* **C2** $\subset$ **C1**.

Proof Since $\Gamma^{\alpha}_{[k]} \subset \mathbf{P}$ and $\{\overline{\rho}^{\min}\} \subset \Gamma_{\star}$, **C2** $\subset$ **C1** follows directly. Also condition (17) is satisfied.

We will now show that for any system having $f_c \in$ **C1**, f_c also satisfies condition (16) for a certain u^D_{Ω}. We want to show that there exists a finite u^D_{Ω} such that, whenever $u^D_{[k]} \geq u^D_{\Omega}$, it also happens that $\Gamma^{\alpha}_{[k]} = \emptyset$. We can then say that u^D_{Ω} is larger than the largest value of $u^D_{[k]}$ for which $\Gamma^{\alpha}_{[k]} \neq \emptyset$. From equation (15a) we have that $\frac{1}{h}\sum_{i\in I} c_{i[k]} \cdot q_{i[k]} = u^D_{[k]} - 1$ when the system is overloaded. From this and (25) we have that the largest value of $u^D_{[k]}$ happens when $\overline{c}_{[k]} = \overline{c}^{\max}$ and $f_p(\overline{c}_{[k]}) = \overline{c}^{\min}$ and is:

$$u^D_{[k]} \leq V_{[k]} + 1 = \frac{1}{h}\sum_{i\in I} c_i^{\max} \cdot q_*$$

where $\overline{q}_*$ is obtained by solving the following linear programming problem:

$$\begin{aligned} &\text{maximize } \sum_{i\in I} c_i^{\max} \cdot q_i \qquad \text{subject to} \\ &q_i \geq 0, \\ &\frac{1}{h} \cdot \sum_{i\in I} c_i^{\min} \cdot q_i + \sum_{i\in I} c_i^{\min} \cdot \rho_i^{\min} = \alpha \end{aligned} \tag{26}$$

where that last constraint enforces that $\Gamma^{\alpha}_{[k]} \neq \emptyset$ (see equation (25)). Since u^D_{Ω} exists, the proof follows. □

Lemma 3 *For any system $\{\mathscr{T}, \mathscr{X}, \mathscr{A}, \mathscr{S}\}$ with $f_c \in$ **C1** and $\mathscr{A}$ defined as in equation* (20)*, the largest possible value of the resource demand is given by*

$$u^D_{max} = \sum_{i\in I} c_i^{max}\big(\frac{q_{*i}+1}{h} + \rho_i^{max}\big) \tag{27}$$

where $\overline{q}_*$ is the solution of the linear programming problem (26)*.*

Proof For any state for which $\Gamma^{\alpha}_{[k]} \neq \emptyset$, $u^D_{[k]} \leq u^D_{\max}$ since $\overline{c}_{[k]} \leq \overline{c}^{\max}$ and $\overline{\rho}_{[k]} \leq \overline{\rho}^{\max}$, $\forall i \in I$.

For any state for which $\Gamma^{\alpha}_{[k]} = \emptyset$ and $\Gamma^{\alpha}_{[k-1]} \neq \emptyset$, we have that the system is overloading ($u^D_{[k]} \geq 1$) and therefore $u^D_{[k]} \leq V_{[k]} + 1 \leq V_{[k-1]} + 1 \leq u^D_{\max}$.

For any state for which $\Gamma^{\alpha}_{[k]} = \emptyset$ and $\Gamma^{\alpha}_{[k-1]} = \emptyset$, there exists a previous time moment $t_{[k-p]} \leq t_{[k-1]}$ such that either $\Gamma^{\alpha}_{[k-p]} \neq \emptyset$ when $u^D_{[k]} \leq V_{[k-p]} + 1$; or $t_{[k-p]} = 1$ and $\Gamma^{\alpha}_{[k-p]} = \emptyset$ when $u^D_{[k]} \leq V_{[1]} + 1 = \sum_{i\in I} c_i^{\max} \cdot \rho_i^{\max}$ (since $q^{\mathrm{p}}_{[1]} = q_{[0]} = 0$). In both cases $u^D_{[k]} \leq u^D_{\max}$. From the above cases, since k is arbitrary, the proof follows. □

Lemma 4 *For any $\{\mathscr{T}, \mathscr{X}, \mathscr{A}, \mathscr{S}\}$ with $f_c \in$ **C2** and $\mathscr{A}$ defined as in equation* (20)*, the largest possible value of the resource demand is given by*

$$u^D_{max} = \max\Big\{\alpha \cdot \max_{i\in I}\Big\{\frac{c_i^{max}}{c_i^{min}}\Big\}, \sum_{i\in I} c_i^{max} \cdot \rho_i^{max}\Big\} \tag{28}$$

Proof We have two cases to analyze. When $\Gamma^{\alpha}_{[k-1]} \neq \emptyset$ then we have from equations (25) and (24):

$$\sum_{i\in I} c^{p}_{i[k-1]}\big(\frac{q_{i[k-1]}}{h} + \rho_{i[k]}\big) = \alpha$$

and there exist the quantities $u_{i[k]} \in [0,\alpha]$ with $\sum_{i\in I} u_{i[k]} = \alpha$ such that:

$$u_{i[k]} = c^{p}_{i[k-1]}\big(\frac{q_{i[k-1]}}{h} + \rho_{i[k]}\big)$$

From the above it follows that:

$$\begin{aligned} u^{D}_{[k]} \leq V_{[k]} + 1 &= \sum_{i\in I} c^{\max}_{i}\big(\frac{q_{i[k-1]}+1}{h} + \rho_{i[k]}\big) \\ &= \sum_{i\in I} \frac{c^{\max}_{i}}{c^{p}_{i[k-1]}} \cdot u_{i[k]} \leq \alpha \cdot \max_{i\in I}\Big\{\frac{c^{\max}_{i}}{c^{\min}_{i}}\Big\} \end{aligned} \tag{29}$$

On the other hand, when $\Gamma^{\alpha}_{[k-1]} = \emptyset$, there must exist a previous time instance $t_{[k-p]} \leq t_{[k-1]}$ with $\Gamma^{\alpha}_{[k-p+r]} = \emptyset$, $\forall r = \overline{0, p-1}$. In this case there are two possibilities: either $t_{[k-p]} = 1$ when $u^{D}_{[k]} \leq V_{[1]} + 1 = \sum_{i\in I} c^{\max}_{i} \cdot \rho^{\max}_{i}$; or there exists $\Gamma^{\alpha}_{[k-p-1]} \neq \emptyset$. In this second case we have that $u^{D}_{[k]} \leq V_{[k-p]} + 1$ and inequality (29) applies. □

The bound on the worst case response time (for an EDF scheduler) can be calculated using the equation (22) where $u^{D}_{\max}$ is computed according with equation (27) for controllers in **C1**, and equation (28) for controllers in **C2**. These results were determined considering that the system is modeled using the general model. For the constrained model, the classes of controllers **C1** and **C2** can also be defined as in equations (23) and (24), where $\Gamma^{\alpha}_{[k]}$ is:

$$\Gamma^{\alpha}_{[k]} = \Big\{\overline{\rho} \in \mathbf{P}\Big| \sum_{i\in I} c^{p}_{i[k]}\big(\frac{q_{i[k]}}{h} + \rho_{i}\big) = \alpha;\ \overline{c}^{p}_{[k]} = f_{p}(\overline{c}_{[k]})\Big\} \tag{30}$$

By a similar reasoning, these classes of controllers can also be shown to be stable, and their worst case resource demand are the same as for **C1** and **C2** for the general model (equation (27) for **C1** and equation (28) for **C2**).

10 Worst-Case Response Time Example

In the previous sections we have developed models and stability conditions, first for a constrained system and then for a general one. Obviously, eliminating the constraint on the controller period is very important since it allows to adapt controller rates to the particularities of the application and, thus, to improve the quality of management. In order to provide further insight, we will now compare the controllers generated with the two models in terms of the bounds on the worst case response time of the tasks. We will consider the classes of controllers **C1** and **C2** for both models.

We will construct two test-cases, both consisting of two tasks. The first is characterized by small variations of execution times ($c^{\max}_{i}/c^{\min}_{i} = 2$) and rates ($\rho^{\max}_{i}/\rho^{\min}_{i} = 2$) and the fact that the rates of both tasks have the same order of magnitude. The second test-case will have large variations ($c^{\max}_{i}/c^{\min}_{i} = 10$, $\rho^{\max}_{i}/\rho^{\min}_{i} = 10$) and the tasks will have rates of different orders of magnitude.

Table 1: Response times for our examples, considering three different controller periods

Model	Controller	Example 1			Example 2		
		h	$r_1^{\max}$	$r_2^{\max}$	h	$r_1^{\max}$	$r_2^{\max}$
Constrained	**C1**	20	48	53	5000	91200	97050
	C2	20	28	33	5000	46200	52050
General	**C1**	4	19	24	1000	19750	25600
	C2	4	12	17	1000	10200	16050
	C1	2	15	20	100	3550	9400
	C2	2	10	15	100	2100	7950

Example 1 We consider a task set $\Lambda_1 = \{\tau_1, \tau_2\}$, where:

$$\tau_1 = \{\mathbf{P}_1 = [0.5, 1], \mathbf{C}_1 = [0.5, 1]\}$$
$$\tau_2 = \{\mathbf{P}_2 = [0.25, 0.5], \mathbf{C}_2 = [1, 2]\}$$

Example 2 We consider a task set $\Lambda_2 = \{\tau_1, \tau_2\}$ where:

$$\tau_1 = \{\mathbf{P}_1 = [0.01, 0.1], \mathbf{C}_1 = [5, 50]\}$$
$$\tau_2 = \{\mathbf{P}_2 = [0.001, 0.01], \mathbf{C}_2 = [50, 500]\}$$

For the constrained model, as explained in Section 7.1, the controller period must be much larger than the largest task period in the system. Let us consider that this assumption holds if h is no smaller than 5 times the largest task period. For the general model this constraint disappears, and we can choose smaller controller periods as well. In Table I we present the response time bounds $r_i^{\max}$ for our chosen examples, considering three controller periods h:

$$5 \cdot \max_{i \in I} \left\{ \frac{1}{\rho_i^{\min}} \right\}; \ \max_{i \in I} \left\{ \frac{1}{\rho_i^{\min}} \right\}; \ \text{and} \ \min_{i \in I} \left\{ \frac{1}{\rho_i^{\min}} \right\}$$

the first period is used for the constrained model, and the following two for the general one.

We have two observations to make:

1. in both examples the class of more aggressive controllers (**C2**) is able to control the system such that response times are reduced considerably when compared with **C1**; this is due to the fact that **C2** is more aggressive in controlling the resource demand of the system.
2. in Example 1, the best performing controller (in terms of response time), **C2** with $h = 2$, leads to worst case response times that are 21% (for τ_1) and 28% (for τ_2) of the worst case response times with the worst performing controller (**C1** with $h = 20$). In the second example, the performance gap is much more dramatic, and **C2** with $h = 100$ leads to worst case response times that are 2.3% and 8.2% of the worst case response times of **C1** with $h = 5000$. This is largely due to the fact that for the worst performing controllers (modeled according with the constrained model) the period of the controller h must be much larger than the largest period in the system (our assumption from Section 7.1), and in overloaded conditions, a task with high rate will release a high number of jobs, during the interval h, that will only queue up.

The general model removes this constraint and the controllers for this model, can take advantage of much smaller controller periods, in order to reduce response times by very large amounts.

11 Case Study

In this section we will show how to apply our stability criterion for several resource managers. First we develop two simple ad-hoc resource managers and we show how they trivially satisfy our stability criterion and then we show how our stability criterion can be applied to three existing resource managers.

11.1 Stability of Ad-Hoc Resource Managers

In this section we give two examples of resource managers, which might be used to control the resources in a given system, and we shall use them to illustrate the effectiveness of our stability criterion.

Let us assume that we have a system working as described in Section 4.1 for which the designer has knowledge about the expected execution times (noted c_j^e, $j \in I$) of each task in the system. This system produces its desired runtime performance when tasks are running at certain known rates, noted $\overline{\rho}^e$, and these rates do not lead to increasing task queues, if execution times for jobs of tasks are the expected ones. Furthermore, worst-case execution times for all tasks are known, and there exists a set of rates $\overline{\rho}^{\min}$ which satisfy Theorem 2. For this system, one may imagine a resource manager which measures task queue sizes at certain moments in time and, assuming expected execution times, computes the resource demand. If the resource demand is larger than a predefined bound, the task chain rates are switched to $\overline{\rho}^{\min}$, otherwise, they are kept to $\overline{\rho}^e$. We can trivially show that this resource manager leads to a stable system according to Theorem 4. The algorithm for this resource manager is the following:

Algorithm 1 Switching Resource Manager

1: /* measure $q_{j[k]}, \forall j \in I$ */
2: /* compute $u^D_{[k]}$ */
3: **if** $u^D_{[k]} \geq u^D_t$ **then**
4: $\quad \overline{\rho}_{[k+1]} \leftarrow \overline{\rho}^{\min}$
5: **else**
6: $\quad \overline{\rho}_{[k+1]} \leftarrow \overline{\rho}^e$
7: **end if**

The qualitative transfer function for this resource manager is given in Figure 6a. Intuitively it is easy to accept that such a resource manager must lead to a stable system. However, since we deal with a switching control policy, demonstrating the stability of this controller with established methods from control theory is by far not straightforward.

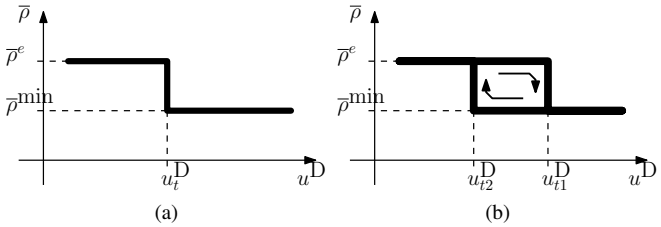


Fig. 6: Qualitative transfer function ($\overline{\rho}_{[k+1]} = f(u^D_{i[k]})$) of the two resource managers.

For the above described resource manager, it might happen, in practice, that the resource demand always remains around the bound, and there is a lot of switching between the two sets of task rates. To mitigate this problem the algorithm can be modified as in Algorithm 2, where $u^D_{t2} < u^D_{t1}$. The qualitative transfer function of this

Algorithm 2 Switching Resource Manager with Hysteresis

```
1: /* measure q_{j[k]}, ∀j ∈ I */
2: /* compute u^D_{[k]} */
3: if u^D_{i[k]} ≥ u^D_{t1} and ρ̄_{[k+1]} = ρ̄^e then
4:     ρ̄_{[k+1]} ← ρ̄^min
5: else
6:     if u^D_{i[k]} ≤ u^D_{t2} and ρ̄_{[k+1]} = ρ̄^min then
7:         ρ̄_{[k+1]} ← ρ̄^e
8:     else
9:         /* do nothing */
10:    end if
11: end if
```

resource manager is given in Figure 6b and we can observe that it exhibits hysteresis. This is an even harder problem to analyze using established methods. However, it is still trivial to check, using our stability criterion, that this resource manager leads to a stable system according to Theorem 4.

11.2 Stability of Existing Resource Managers

In this section, we take three resource management policies, presented in previous literature, and determine if they lead to stable real-time systems. We will consider the *QRAM* algorithm described in [1], and the *corner-case* and *QoS derivative* algorithms described in [16].

The *QRAM* and *corner-case* algorithms work in similar ways. They consider as a resource the processor utilization and, when the control algorithms are executed, they initially select task rates such that the perceived resource demand (computed based on estimations of future job execution times, and task queues) is minimum ($\overline{\rho}^{\min}$). Then, if resources are still available in the system, the algorithms select tasks whose

rates are increased until all available resources are used. These tasks are selected such that the value of some quality-of-service function is maximized. If no resources are available, the new task rates are $\overline{\rho}^{\min}$. From this behavior we can observe that *QRAM* and *corner-case* satisfy Theorem 4 and, therefore, these resource managers lead to stable systems. More specifically, they belong to the class **C2**, introduced in Section 9.

The *QoS derivative* algorithm works in a different way. At the beginning it determines the resource demand in the system, assuming current rates. Then, the manager solves a convex optimization problem, whose goal is to select new task rates such that some quality-of-service function is maximized, with the constraint that the resource demand with the new rates is equal with the amount of available resources. If a feasible solution (a solution that satisfies the constraint) exists, the system is stable. However, if the constraint cannot be satisfied by any $\overline{\rho} \in \mathbf{P}$, it cannot be demonstrated that the selected rates will satisfy Theorem 4. A straight forward approach would be to test the solution of the optimization to determine if it is feasible, and if not, to select $\overline{\rho} \in \Gamma_\star$. However, for the convex optimization to find a feasible solution, it is required that the starting point satisfies the constraint[2], otherwise no feasible solution will be found and, in practice, the straight forward approach always sets rates in $\Gamma_\star$, which will keep the processor load unnecessarily low.

Our solution is to modify the *QoS derivative* algorithm to start from a feasible point (if one exists):

Algorithm 3 Modified *QoS derivative* Algorithm

Input: $c_{i[k]}$, $q_{i[k]}$, $\rho_{i[k]}$, h

1: /* here we assume that $u^D_{[k]}$ is computed according to Equation (14) */
2: $u^D_i \leftarrow \frac{1}{h} \cdot c_{i[k]} \cdot q_{i[k]} + c_{i[k]} \cdot \rho_{i[k]}$
3: $\Delta u \leftarrow u - u^D_{[k]}$
4: **while** $i < n$ and $\Delta u \neq 0$ **do**
5: /* change $\rho_{i[k]}$ to ρ_{i*} such that the absolute value of Δu is reduced */
6: $\Delta u \leftarrow \Delta u - c_{i[k]} \cdot \rho_{i[k]} + c_{i[k]} \cdot \rho_{i*}$
7: $i \leftarrow i + 1$
8: **end while**
9: $\overline{\rho}_{[k+1]} \leftarrow$ *QoS derivative*$(\overline{\rho}_*)$
10: **return** $\overline{\rho}_{[k+1]}$

The algorithm changes the initial rates to a new vector $\overline{\rho}_*$ (line $4-8$). The original manager is then called (line 9 in the algorithm) with this rate vector, as a starting point If there are $\overline{\rho} \in \mathbf{P}$ for which the constraint is satisfied, then the algorithm has a feasible solution, otherwise, the algorithm will set $\overline{\rho}_{[k+1]} = \overline{\rho}^{\max}$ when the system is underloading and $\overline{\rho}_{[k+1]} = \overline{\rho}^{\min}$ when the system is overloading. From this behavior we can observe that the *QoS derivative* resource manager, with the above modification, is also stable and belongs to **C2**. Also note that if quality-of-service is not an

[2] In addition to this the Karush-Kuhn-Tucker matrix must be non-singular at the starting point, but it can be shown that this condition holds for any $\overline{\rho} \in \mathbf{P}$. See [20] Sec. 10.2 for an in depth treatment of these conditions.

issue, the above algorithm, with line 9 changed to $\overline{\rho}_{[k+1]} \leftarrow \overline{\rho}_*$ is also guaranteed to be stable and belonging to **C2**.

12 Stability Examples

In this section we will show several graphical examples illustrating stable and unstable systems. The examples consist of two tasks, whose jobs are scheduled with EDF. The behavior of these systems, however, is representative for any real-time system accepted by this framework.

Example 3 Let us consider the system $\Lambda_3 = \{\tau_1, \tau_2\}$ where:

$$\tau_1 = \{\mathbf{P}_1 = [0.015, 0.1], \mathbf{C}_1 = [5, 50]\}$$
$$\tau_2 = \{\mathbf{P}_2 = [0.0015, 0.01], \mathbf{C}_2 = [50, 500]\}$$

We can observe that it does not satisfy Theorem 2 since:

$$\sum_{i \in I} c_i^{\max} \cdot \rho_i^{\min} = 50 \cdot 0.015 + 500 \cdot 0.0015 = 1.5.$$

As a consequence, no controller will be able to keep the system stable in the worst case.

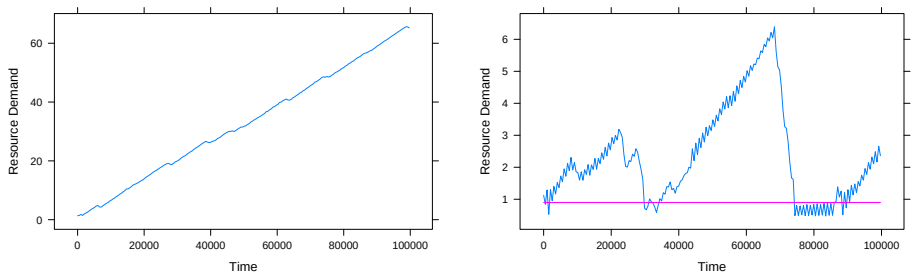
Let us consider that this system has a resource manager which keeps the task rates constant at the minimum rates and runs with a period of 500 time units. Let us further consider that the execution times for the jobs of the tasks are distributed such that the expected execution time for jobs of τ_1 is $c_1^e = 40$ and for jobs of τ_2 is $c_2^e = 400$ time units. For this case, the resource demand, as a function of time, is plotted in Figure 7a. We can observe that this system is not stable because the resource demand grows as time progresses. This is to be expected since the expected load in the system is:

$$\sum_{i \in I} c_i^e \cdot \rho_i^{\min} = 40 \cdot 0.015 + 400 \cdot 0.0015 = 1.3 > 1$$

Of course, it is possible for certain runtime scenarios to be stable, despite the system being unstable in the worst case. For instance if execution times are distributed such the the average execution times are $c_1^e = 30$ and $c_2^e = 300$ time units, the system will be stable. The behaviour of this system is plotted in Figure 7b. Although as $t_{[k]} \rightarrow \infty$ the system is stable and achieves an average utilization of 0.9, we can see that for several intervals of times the resource demand of the system is substantially higher and increasing. This happens because for these intervals the execution times of jobs of tasks are higher than the expected ones and the system may be subjected to poor performance.

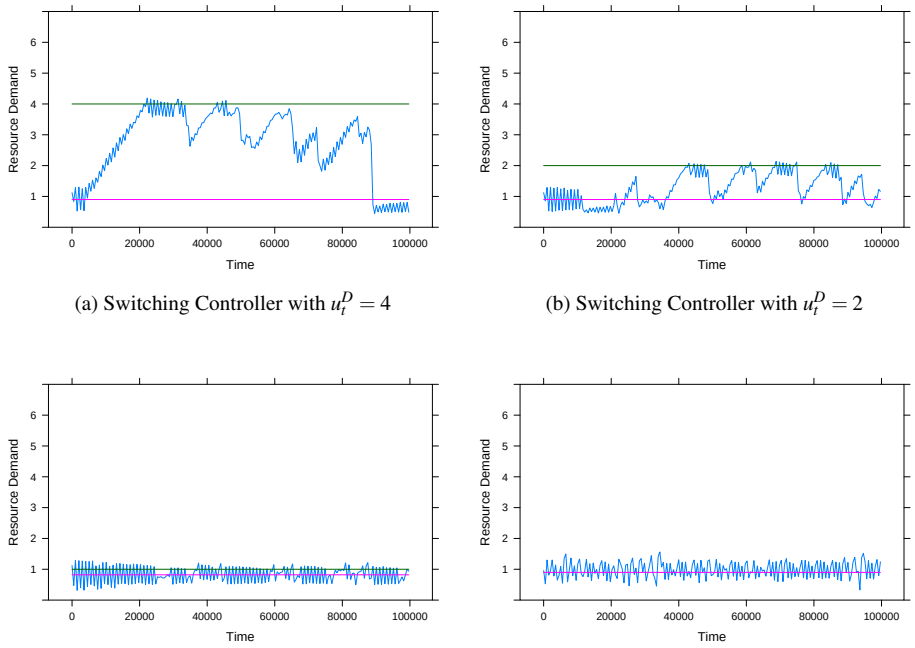
Example 4 Let us now consider the system $\Lambda_4 = \{\tau_1, \tau_2\}$ where:

$$\tau_1 = \{\mathbf{P}_1 = [0.008, 0.1], \mathbf{C}_1 = [5, 50]\}$$
$$\tau_2 = \{\mathbf{P}_2 = [0.0008, 0.01], \mathbf{C}_2 = [50, 500]\}$$



(a) Overloaded System (b) Underloaded System

Fig. 7



(a) Switching Controller with $u_t^D = 4$ (b) Switching Controller with $u_t^D = 2$

(c) Switching Controller with $u_t^D = 1$ (d) *QoS derivative* Controller

Fig. 8

which satisfies Theorem 2 since $50 \cdot 0.008 + 500 \cdot 0.0008 = 0.8$. Let us further consider that the execution times of the jobs of these tasks are distributed such that their expected execution times are $c_1^e = 40$ and $c_2^e = 400$ time units. Let us consider five resource managers that control the system, as follows:

1. a manager which chooses rates randomly such that the expected rates are $\rho_1^e = 0.015$ and $\rho_2^e = 0.0015$,
2. the resource manager presented in Algorithm 1 with $\overline{\rho}^e = (0.0125, 0.001)^T$ and a threshold of $u_t^D = 4$,
3. the resource manager presented in Algorithm 1 with $\overline{\rho}^e = (0.0125, 0.001)^T$ and a threshold of $u_t^D = 2$,
4. the resource manager presented in Algorithm 1 with $\overline{\rho}^e = (0.0125, 0.001)^T$ and a threshold of $u_t^D = 1$, and
5. the *uniform QoS* resource manager described in Algorithm 3 with $\Gamma_{[k]}^{\alpha} = \Gamma_{[k]}^{0.9}$ (We remind the reader that the *uniform QoS* resource manager belongs to the **C2** class of resource managers, described in Section 9 by equations (24) and (25)).

In all cases, the resource managers actuate with a period of 500 time units and use an execution time prediction function which always predicts the expected values.

For the first controller we can observe that the expected load in the system is $0.015 \cdot 40 + 0.0015 \cdot 400 = 1.2 > 1$, therefore the system behaves similar to the previous example, as in Figure 7a.

The system behavior for the next three resource managers (cases 2, 3, and 4 above) is illustrated in Figures 8a, 8b, and 8c respectively. Their behavior is similar because in their average case they keep a resource demand of $0.0125 \cdot 40 + 0.001 \cdot 400 = 0.9$. However, we can observe the effect of the different thresholds. The resource demand overshoot is lower for lower thresholds but for too low thresholds the system acts too aggressively and undershoot occurs more frequently in the system.

The last resource manager is an algorithm of the class **C2** with a chosen $\Gamma_{[k]}^{\alpha} = \Gamma_{[k]}^{0.9}$, that is, it tries to keep $u_{[k]}^D = 0.9$ (see Section 9). In the average case this resource manager does the same as the previous three cases, but due to the more evolved nature of the control algorithm it produces better (smaller) over- and under-shoot characteristics. The behaviour of this resource manager is illustrated in Figure 8d.

13 Discussion

The presentation up to this point has been terse and mathematically involving and has served the purpose of deriving our main result presented in Theorem 2, Theorem 3 and Theorem 4. In this section we address potential limitations of these results, and we show how they can be extended in various ways. Also we discuss our choice of stability.

13.1 Ultimate Uniform Boundedness

The concept of stability used in this paper is *Uniform Boundedness* which means that there is a final bound in the system (Ω in Theorem 1), independent of the starting point $\overline{x}_{[0]}$, such that all trajectories tend towards it and finally become bounded in a larger ball of size Ψ around the stability region $\mathscr{M}$ (see Figure 9). However, this may happen only when $t_{[k]} \to \infty$, because we allow in Theorem 1 that $V(\overline{x}_{[k+1]}) \leq V(\overline{x}_{[k]})$. If

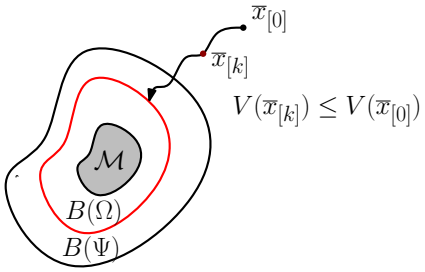


Fig. 9: Illustration of the Uniform Boundedness stability concept.

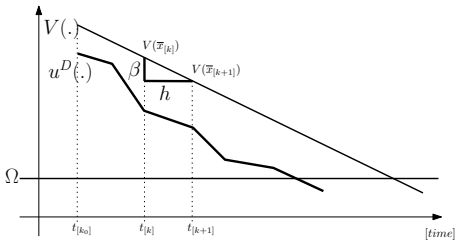


Fig. 10: Behavior in time of the $V(.)$ and $u^D(.)$ functions.

the system evolves such that $V(\overline{x}_{[k+1]}) = V(\overline{x}_{[k]})$, then it never reaches Ω or Ψ. This is, in general, not a problem, because systems typically start with empty queues, and all these starting points are within the ball of size Ψ. If, for a certain application, this is a problem, one is forced to use the stronger concept of *Ultimate Uniform Boundedness* instead. This concept requires that there exists a constant $\beta > 0$, β bounded away from 0, such that Theorem 1 holds with the modification that $V(\overline{x}_{[k+1]}) \leq V(\overline{x}_{[k]}) - \beta$. In the interest of brevity we do not go into the details of the proof, however, one may show that a modified version of Theorem 4 where, instead of $\Gamma_\star$, we use:

$$\Gamma^\beta = \left\{ \overline{\rho}^\star \in \mathbf{P} \,\middle|\, \sum_{i \in I} c_i^{\max} \cdot \rho_i^\star \leq 1 - \beta \right\}$$

is a stability criterion for our system, in the stronger sense. The set Γ^β is obviously a subset of $\Gamma_\star$ and of course it must be non-empty if we want to use it as a replacement for $\Gamma_\star$ in the stability criterion given in Theorem 4. The choice of β is, thus, limited to the range: $\beta \in (0, \beta^{\max}]$ where:

$$\sum_{i \in I} c_i^{\max} \cdot \rho_i^{\min} = 1 - \beta^{\max} \qquad \Rightarrow \qquad \beta^{\max} = 1 - \sum_{i \in I} c_i^{\max} \cdot \rho_i^{\min}$$

Let us now describe the meaning of β with the help of Figure 10. In our analysis we have defined $V(\bar{x}_{[k]})$ as the worst case behavior of $u^D(\bar{x}_{[k]})$ so if at time $t_{[k_0]}$ these values are known and above Ω, we can expect that the system's evolution from $t_{[k_0]}$ onward is as presented in the figure. We can see that between any two successive time instances $t_{[k]}$ and $t_{[k+1]}$ the drop in $V(.)$ is at least[3] β. By knowing β and the initial overload, one may be able to determine how long it will take for the system to stabilize. Since a reduction in resource demand is linked with reduction in response times, this factor also determines how fast response times of jobs drop. In general, the system reduces its resource demand when the state is outside the ball $B(\Omega)$ but inside $B(\Psi)$. In this case, the resource demand (and also response times) in the system is reduced in at most $(\Psi - \Omega) \cdot h/\beta$ time. A large value of β will then mean fast reduction in overload, but since a system, in general, does not exibit its worst case behavior for a long time, the fast reduction of resource demand may lead to underloading instead.

[3] The values $V(\bar{x}_{[k]})$ and $V(\bar{x}_{[k+1]})$ depend on the state of the system. If the system did not have its worst-case behavior between $t_{[k]}$ and $t_{[k+1]}$, then the drop in $V(.)$ is higher than β.

The designer of the system must then choose an appropriate value for β, according with the goals of the system.

13.2 Alternative Metrics

The stability criterion that we derive in this paper is bound to the metric used here, namely the resource demand (equation (14)) which is a load metric. However, one may wish to use different metrics, for instance metrics based on throughput, jitter, or end-to-end delays. To this end we provide the following lemma.

Lemma 5 *In Theorem 4, the metric u^D may be changed by any other function $\rho : \mathscr{X} \to \mathscr{R}_+$ provided that there exist two strictly increasing functions $f_1, f_2 : \mathscr{R}_+ \to \mathscr{R}_+$ with $\lim_{x\to\infty} f_i(x) = \infty$, $i = \overline{1,2}$ such that:*

$$f_1\Big(u^D(\bar{x})\Big) \leq \rho(\bar{x}) \leq f_2\Big(u^D(\bar{x})\Big), \forall \bar{x} \in \mathscr{X} \tag{31}$$

Proof The first half of inequality (31) guarantees that $\rho(\bar{x})$ grows if $u_i^D(\bar{x})$ grows: $\lim_{d(\bar{x},\mathscr{M})\to\infty} \rho(\bar{x}) = \infty$. The second half of the inequality allows us to construct the $V_\rho(\bar{x})$ function necessary for the stability proof, as $V_\rho(\bar{x}) = f_2(V(\bar{x}))$. This completes the proof. □

14 Conclusions

In many real-time systems with variations in execution times, it is important to regulate the utilization of system resources at runtime. An important issue at design time is to verify that the real-time system is stable when using a certain adaptive resource manager. Stability means that the resource demand is bounded under all runtime scenarios. We have developed a model for real-time systems and used it to derive comprehensive conditions that resource managers must comply with, in order to render the system stable. For the derived models we also derived bounds on the response times of tasks. We have applied our results to existing resource managers to verify their correctness in terms of stability.

Our stability criterion applies to uniprocessor adaptive real-time systems which employs task rate adaptation. It is composed of an exact test to determine if the system is at all stabilizable (Theorem 2), and a sufficient test on the resource manager to determine if it stabilizes the system (Theorem 4). The criterion has been designed to be easy to apply in a variety of contexts due to the generality of our model and the simplicity of our conditions. As future work we wish to extend this model to include distributed adaptive real-time systems comprised of task graphs mapped on a heterogeneous set of resources that use various scheduling policies, and allowing for different methods of adaptation.

References

1. C. Lee, J. Lehoczky, R. Rajkumar, D. Siewiorek. *"On Quality of Service Optimization with Discrete QoS Options."* In proceedings of Real-Time Technology and Applications Symposium, pp.276, 1999.
2. G. C. Buttazo, G. Lipari, L. Albeni. *"Elastic Task Model for Adaptive Rate Control."* In Proceedings of the IEEE Real-Time Systems Symposium, pp. 286, December 1998.
3. G. C. Buttazo, L. Albeni. *"Adaptive Workload Management through Elastic Scheduling."* Journal of Real-Time Systems, vol. 23, pp. 7-24, July 2002.
4. G. C. Buttazo, M. Velasco, P. Marti and G. Fohler. *"Managing Quality-of-Control Performance Under Overload Conditions."* In Proceedings of the Euromicro Conference on Real-Time Systems, pp. 53-60, July, 2004.
5. M. Marioni, G. C. Buttazo. *" Elastic DVS Management in Processors With Discrete Voltage/Frequency Modes."* IEEE Transactions on Industrial Informatics, vol. 3, pp. 51-62, February, 2007.
6. C. Lu, J. A. Stankovic, S. H. Son, G. Tao. *"Feedback Control Real-Time Scheduling: Framework, Modeling, and Algorithms."* Real-Time Systems, vol. 23, pp. 85-126, 2002.
7. L. Palopoli, T. Cucinotta, L. Marzario, G. Lipari. *"AQuoSA – adaptive quality of service architecture."* Journal of Software–Practice and Experience, vol. 39, pp. 1-31, 2009.
8. T. Cucinotta, L. Palopoli. *"QoS Control for Pipelines of Tasks Using Multiple Resources."* IEEE Transactions on Computers, vol. 59, pp. 416-430, 2010.
9. J. Combaz, J. C. Fernandez, J. Sifakis, L. Strus. *"Symbolic Quality Control for Multimedia Applications."* Real-Time Systems, vol. 40, pp. 1-43, October, 2008.
10. X. Liu, X. Zhu, P. Padala, Z. Wang, S. Singhal. *"Optimal Multivariate Control for Differentiated Services on a Shared Hosting Platform."* In Proceedings of the Conference on Decision and Control, pp. 3792-3799, December 2007.
11. J. Yao, X. Liu, M. Yuan, Z. Gu. *"Online Adaptive Utilization Control for Real-Time Embedded Multiprocessor Systems."* In Proceedings of the International Conference on Hardware/Software Codesign and System Synthesis, pp. 85-90, 2008.
12. T. F. Abdelzaher, J. A. Stankovic, C. Lu, R. Zhang, Y. Lu. *"Feedback performance Control in Software Services – Using a Control-Theoretic Approach to Achieve Quality of Service Guarantees."* IEEE Control Systems Magazine, vol. 23, pp. 74-90, June 2003.
13. A. Cervin, J. Eker, B. Bernhardsson, K. E. Årzén. *"Feedback-Feedforward Scheduling of Control Tasks."* Real-Time Systems, vol. 23, pp. 25-53, July, 2002.
14. A. Cervin, J. Eker. *"The Control Server: A Computational Model for Real-Time Control Tasks."* Proceedings of the 15th Euromicro Conference on Real-Time Systems, July 2003.
15. C. L. Liu, J. W. Layland. *"Scheduling algorithms for multiprogramming in hard-real-time environment."* Journal of ACM, pp. 40-61, 1973.
16. S. Rafiliu, P. Eles, and Z. Peng. *"Low Overhead Dynamic QoS Optimization Under Variable Execution Times."* Proceedings of 16th IEEE Embedded and Real-Time computing Systems and Applications (RTCSA), pp. 293-303, 2010.
17. S. Rafiliu, P. Eles, and Z. Peng. *"Stability Conditions of On-line Resource Managers for Systems with Execution Time Variations."* Proceedings of the 23rd Euromicro Conference on Real-Time Systems, 2011.
18. A. N. Michel, L. Hou, D. Liu. *"Stability of Dynamical Systems: Continuous, Discontinuous, and Discrete Systems."* Birkhäuser Boston, 2008.
19. E. Kreyszing. *"Introduction to Functional Analysis with Applications."* John Wiley & Sons., Inc, 1989.
20. S. Boyd, L. Vandenberghe. *"Convex Optimization."* Cambridge University Press, 2008.
21. P. R. Kumar, S. Meyn. *"Stability Of Queueing Networks and Scheduling Policies."* IEEE Trans. Automatic Control, 1995.
22. M. Bramson. *"Stability of Queueing Networks,"* Springer, 2008.