Stability of On-line Resource Managers for Distributed Systems under Execution Time Variations

SERGIU RAFILIU, Department of Computer and Information Science, Linköping University, Sweden
PETRU ELES, Department of Computer and Information Science, Linköping University, Sweden
ZEBO PENG, Department of Computer and Information Science, Linköping University, Sweden
MICHAEL LEMMON, Department of Electrical Engineering, University of Notre Dame, IN 46556, USA

Today's embedded systems are exposed to variations in resource usage due to complex software applications, hardware platforms, and impact of the run-time environments. When these variations are large and efficiency is required, on-line resource managers may be deployed on the system to help it 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 distributed systems this problem is particularly hard because applications distributed over many resources generate complex dependencies between their resources. In this paper we develop a mathematical model of the system, and derive conditions that, if satisfied, guarantee stability.

Additional Key Words and Phrases: control theory, stability criterion, adaptive real-time systems, distributed systems.

1. INTRODUCTION

Today's embedded systems, together with the real-time applications running on them, have a high level of complexity. Moreover, such systems very often are exposed to varying resource 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, on-line resource managers may be deployed to control the system's resource usage. Among the goals of such a resource manager is to maximize the resource usage while minimizing the amount of time the system spends in overload situations.

A key question to ask for such systems 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 notion of stability of control systems. In control applications, a controller controls the state of a plant towards a desired stationary point. The 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 system.

In this work, we consider a distributed real-time system running a set of task chains and managers that control the utilization of resources. 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 8).

The theory presented in this paper has similarities with the theory on queueing networks [Kumar et al.(1995); Bramson (2008)]. 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 6.1 has a similar wording with a classical result on queueing networks (see Chapter I from [Bramson (2008)]). However, in this work we deal with robust stability [Zhou et al.(1996)], that is, we bound the behavior of the system in the worst case, while in queueing networks theory stochastic stability [Bramson (2008)] 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 results known from queueing networks theory.

This work is partially supported by the Swedish Foundation for Strategic Research under the Software Intensive program.
Dr. Lemmon acknowledges the partial financial support of the National Science Foundation NSF-CNS-0931195.

DOI:http://dx.doi.org/10.1145/0000000.0000000

2. SYSTEM DESCRIPTION AND CONTRIBUTIONS

In this manuscript we consider a distributed real-time platform formed of a number of resources (processors, networks, etc.) and an application seen as a set of task chains (formed of tasks connected through data dependencies), mapped across the different resources. We assume that the task chains 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 on their resources using any kind of non-idling scheduling policy (different schedulers for different resources). Before being scheduled for execution, jobs accumulate in queues, one for each tasks in the system. We consider that the system possesses a resource manager (also distributed across the different resources) whose job is to adjust task rates subject to the variation in job execution times. In practice, such systems are used for running control applications [Åström et al.(1997)], multimedia applications [Lee et al.(1998)], or web-services [Abdelzaher et al.(2003)].

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

We group and briefly present the main results of this manuscript, namely the system model and the stability criteria, in Section 6. Before discussing stability, we go through a somewhat involved modeling phase where we develop a detailed, non-linear model of our system (Section 7). Our model tracks the evolution in time of the accumulation of execution time on each resource. The accumulation of execution time is the sum of execution times of the queued jobs of all tasks running on a resource. From this model we build a worst-case evolution model that we use when deriving our stability results.

The main challenge that we face is due to the distributed nature of the system since the accumulation of execution times flows between resources and, thus, we may experience very rich behavior patterns that need to be accounted for when determining the worst-case behavior of the system. This is what makes the modeling completely different and the analysis significantly more complex than in our previous work for non-distributed adaptive real-time systems [Rafiliu et al.(2013)]. We illustrate the sometimes counter intuitive behavior, with regards to stability, of distributed systems in Section 8.2 with a simple example. Our worst-case behavior model is that of a linear switching system with random switching, where the system evolves linearly according to one of several branches, but at any time may randomly switch to a different one.

Given this worst-case behavior model we develop three criteria that determine if the system is stable or not. The first two criteria (Sections 8.1 and 8.3) consider the topology and parameters of the system (worst-case execution times and rates, mapping, etc.) and determine if there exist resource managers that can keep the system stable. Although the number of branches in our worst-case behavior model is exponential in the number of resources, we manage to formulate conditions whose complexity is linear in the number of resources in the system.

The last criterion (Section 8.5) describes conditions that a resource manager needs to satisfy in order to keep the system stable. Unlike the previous literature (with the possible exception of [Cucinotta et al.(2010a)]) 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 managers to become stable.

In Section 9 we present a discussion on the meaning, features, and limitations of our model and criteria. We show how to extend our model by considering different adaptation methods (as described in Section 11) and different application models (task trees and graphs instead of chains).

Finally, we end the paper with a set of examples (Section 10), a presentation of the related work done in this area (Section 11), and conclusions (Section 12).

3. PRELIMINARIES

3.1. Mathematical Notations

In this paper we use standard mathematical notations. We describe a set of elements as: $\mathfrak{S} = \{s_i, i \in \mathcal{I}_{\mathfrak{S}}\}$ where $\mathcal{I}_{\mathfrak{S}} \subset \mathbb{Z}_+$ is the index set of $\mathfrak{S}$. We make the convention that if $\mathfrak{S} = \{s_i, i \in \mathcal{I}_{\mathfrak{S}}\}$ is an ordered set, then s_i appears before $s_{i^\star}$ in the set if and only if $i < i^*$, for all $s_i, s_{i^\star} \in \mathfrak{S}$, $s_i \neq s_{i^\star}$. $\mathcal{P}(\mathfrak{S})$ is the power set of $\mathfrak{S}$.

We denote a $n \times m$ matrix as $A = [a_{i,j}]_{n\times m}$ where n is the number of rows and m is the number of columns. We denote with $0_{n\times m}$ the matrix whose elements are all 0 and with I_n the $n \times n$ identity matrix. We denote column vectors as $\vec{v} = (v_1, v_2, \cdots v_n)^T$ or more compactly as $\vec{v} = [v_i]_n$. We denote with 0_n and 1_n the n-dimensional columns vectors formed of all elements equal to 0 and 1 respectively. In a n-dimensional normed space $\mathcal{V}$ we denote the norm of a vector $\vec{v}$ with $|\vec{v}|$.

When we compare two vectors $\vec{v} = [v_i]_n$ and $\vec{u} = [u_i]_n$, we use the notations $\succ$, $\prec$, $\succeq$, and $\preceq$. For example the relationship $\vec{v} \succeq \vec{u}$ means $v_i \geq u_i$, $\forall i \in \{1, 2, \cdots, n\}$. Similarly for the rest of the notations as well.

We recall that a function $\gamma : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ is a $\mathfrak{K}$-function if it is continuous, strictly increasing, and $\gamma(0) = 0$. A function $\gamma : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ is a $\mathfrak{K}_\infty$-function if it is a $\mathfrak{K}$-function and it is unbounded. A function $\beta : \mathbb{R}_{\geq 0} \times \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ is a $\mathfrak{KL}$-function if for each fixed $t \geq 0$ the function $\beta(\cdot, t)$ is a $\mathfrak{K}$-function and for each fixed $s \geq 0$ the function $\beta(s, \cdot)$ is decreasing and $\beta(s, t) \to 0$ as $t \to \infty$.

3.2. Platform and Application

We consider that our distributed platform is composed of an ordered finite set of n resources (e.g. processors and buses) $\mathcal{R} = \{\mathbf{N}_i, i \in \mathcal{I}_{\mathcal{R}}\}$, $\mathcal{I}_{\mathcal{R}} = \{1, \cdots, n\}$. Each resource is characterized by the finite set of tasks that compete for it $\mathbf{N}_i = \{\tau_j, j \in \mathcal{I}_{\mathbf{N}_i}\}$. We only consider mutual-exclusive resources, where only one task may use the resource at any given time. Each resource is, therefore, equipped with a scheduler to schedule the succession of tasks. The scheduler is embedded (as part of the hardware or the software middleware) in the resource.

The application running on this platform is composed of a finite set of acyclic task chains $\mathcal{A} = \{C_p, p \in \mathcal{I}_{\mathcal{A}}\}$, formed of tasks linked together through data dependencies. A task chain is a finite ordered set of tasks: $C_p = \{\tau_j, j \in \mathcal{I}_{C_p}\}$. We characterize each task τ_j in the system, by the maximum execution times that jobs of this task can have $c_j^{\max}$. Each task is part of one task chain and is mapped onto one resource. With each task chain C_p we associate the set of rates at which it can release new instances for execution: $\mathbf{P}_p \subseteq [\rho_p^{\min}, \rho_p^{\max}]$. All tasks of a task chain C_p release jobs periodically and simultaneously, at the variable rate ρ_p chosen from the set $\mathbf{P}_p$. However, jobs cannot be executed before their data dependency is solved.

Each task in the task chain C_p has at most one data dependency to the previous task in C_p. Figure 1 shows an example of a system with 4 resources, where $\mathbf{N}_1, \mathbf{N}_2$, and $\mathbf{N}_3$ are processors and $\mathbf{N}_4$ is a communication link. The application is composed of 4 task chains: $C_1 = \{\tau_1\}$, $C_2 = \{\tau_2, \tau_3, \tau_4, \tau_5, \tau_6\}$, $C_3 = \{\tau_7, \tau_8, \tau_9\}$, and $C_4 = \{\tau_{10}, \tau_{11}, \tau_{12}\}$. The order in which tasks appear in the task chain determines the dependencies between them. Tasks are ordered according to their index. Tasks are mapped to resources as follows: $\mathbf{N}_1 = \{\tau_1, \tau_6, \tau_7\}$, $\mathbf{N}_2 = \{\tau_{10}, \tau_9, \tau_4\}$, $\mathbf{N}_3 = \{\tau_2, \tau_{12}\}$, and $\mathbf{N}_4 = \{\tau_8, \tau_5, \tau_3, \tau_{11}\}$. Observe that we treat both processors and communication links as resources. Because of this, messages sent on communication links are called tasks in our notation.

We denote the set of all tasks in the application with $\mathbf{\Theta} = \bigcup_{p\in\mathcal{I}_{\mathcal{A}}} C_p$, indexed by $\mathcal{I}_{\mathbf{\Theta}}$. Also, we denote with $\mathbf{P} = \prod_{p\in\mathcal{I}_{\mathcal{A}}} \mathbf{P}_p$ the set of all possible rates vectors $\vec{\rho}$. Based on the above definitions and assuming $\xi \notin \mathcal{I}_{\mathbf{\Theta}}$ we define the following mappings:

— $\gamma : \mathcal{I}_{\mathbf{\Theta}} \to \mathcal{I}_{\mathcal{A}}$, $\gamma(j) = p$ if $\tau_j \in C_p$, which is a mapping from tasks to task chains,
— $\nu : \mathcal{I}_{\mathbf{\Theta}} \to \mathcal{I}_{\mathcal{R}}$, $\nu(j) = i$ if $\tau_j \in \mathbf{N}_i$ which is a mapping from tasks to resources,
— $\pi : \mathcal{I}_{\mathbf{\Theta}} \to \mathcal{P}(\mathcal{I}_{\mathbf{\Theta}})$, $\pi(j) = \mathcal{S} \subset \mathcal{I}_{C_p}$ is the set of all indexes of tasks that are predecessors of τ_j in the task chain,

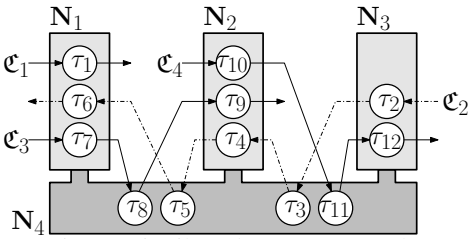


Fig. 1: Distributed system example.

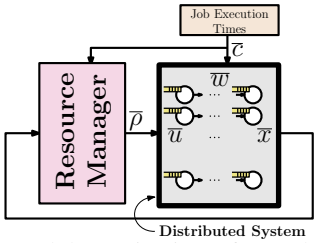


Fig. 2: Control theoretic view of our adaptive distributed system.

— $\varphi : \mathcal{I}_{\mathcal{R}} \to \mathcal{P}(\mathcal{I}_{\mathcal{R}})$, $\varphi(i) = \mathcal{S}_{\mathcal{N}} \subset \mathcal{I}_{\mathcal{R}}$, is the set of all indexes of resources that contain predecessor tasks – of the tasks mapped to $\mathbf{N}_i$ – mapped to them,
— $\mathfrak{p} : \mathcal{I}_{\boldsymbol{\Theta}} \to \mathcal{I}_{\boldsymbol{\Theta}} \cup \{\xi\}$, $\mathfrak{p}(j) = j^\star \neq \xi$ if $\tau_{j^\star}$ is the predecessor of τ_j, or $\mathfrak{p}(j) = \xi$ if τ_j has no predecessor, and

Using the above mappings, the predecessor task of τ_j (assuming τ_j has one) is $\tau_{\mathfrak{p}(j)}$ and the release rate of the task chain, of which τ_j is part of, is $\rho_{\gamma(j)}$. The resource on which the predecessor of τ_j is running is $\mathbf{N}_{\nu(\mathfrak{p}(j))}$.

A job of a task τ_j is characterized by its *execution time* – unknown value, smaller than $c_j^{\max}$; *rate* – value from $\mathbf{P}_j$ prescribed by the resource manager at the job's release time; and *response time* – the interval of time between the release and the finish time of the job. A job can exist in one of the following states:

(1) *Released* when it has been released at the rate of the task chain,
(2) *Ready for Execution* when the job's data dependency was solved, that is, when the corresponding job of the predecessor of this task $\tau_{\mathfrak{p}(j)}$ has finished executing,
(3) *Under Execution* when the job has been partially executed, that is, when it occupied the resource for a portion of its execution time, and
(4) *Finished* when the job has been fully executed.

The tasks in the system are scheduled, on their particular resources, using any scheduler which satisfies the following properties:

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

At a certain moment in time, due to the functioning of the schedulers, at most one job of any task may be under execution. We consider that all jobs which are ready for execution and under execution are accumulated in queues, one for each task in the application. For tasks that have no data dependencies, a job becomes ready for execution whenever it is released. Whenever a job finishes its execution, it is removed from its task's queue. At the same time, the corresponding job of the task's successor becomes ready for execution and, thus, gets added to the successor task's queue. We denote the queue size of each task τ_j, at a certain time moment $t_{[k]}$, with: $q_{j[k]}$, $\forall j \in \mathcal{I}_{\boldsymbol{\Theta}}$. While queue size typically denote the discrete number of jobs waiting to be executed, in our framework, we treat these values as continuous in order to express the fact that one of the jobs in each queue may be partially executed[1] by the scheduler.

3.3. Resource Manager

The system features a resource manager whose goal, among others, is to measure execution times, and then adjust task chain release rates, such that the jobs pending on all resources are executed in a timely fashion. The resource manager is part of the middleware of the system and is, in general, distributed over all resources.

Whenever the resource manager is activated, we assume that it has a worst-case response time of $\Delta < h$, where h is its actuation period, and that it has a worst-case execution time on each resource

[1] A queue size of 2.6 expresses that there are three jobs waiting in the queue, the oldest one having been 40% executed.

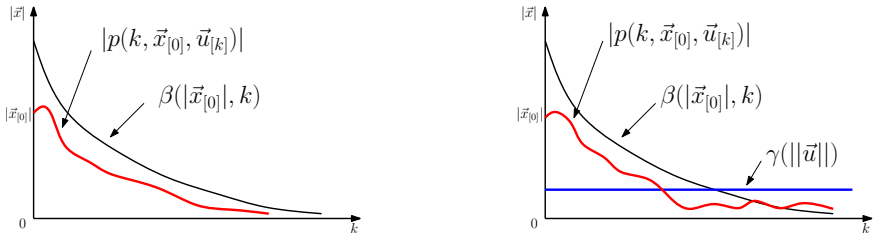


Fig. 3: (a) GAS: The system's state approaches 0_n as time passes, regardless of the initial state. (b)ISS: The system's state reduces at first, but then becomes trapped below a bound determined by the magnitude of the inputs.

of less than Δ. We assume that the resource manager imposes the newly computed task chain rates simultaneously, on all resources, at Δ after it has been activated. All jobs released during the running of the resource manager, are released at the old task chain rates.

4. CONTROL THEORETICAL CONCEPTS

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

We model our distributed real-time system as a discrete-time control system, described by a system of difference equations:

$$\vec{x}_{[k+1]} = f_p(\vec{x}_{[k]}, \vec{u}_{[k]}, \vec{w}_{[k]}) \tag{1}$$

where $\vec{x}_{[k+1]}$ and $\vec{x}_{[k]}$ are the state vectors of the system at the future ($t_{[k+1]}$) and the current ($t_{[k]}$) time moments, $\vec{u}_{[k]}$ is the current input, and $\vec{w}_{[k]}$, is the current perturbation experienced by the system. The model of the resource manager is:

$$\vec{\rho}_{[k]} = f_c(\vec{x}_{[k]}) \tag{2}$$

In our system, input is provided by the resource manager as the task chain rates vector and the perturbations come from the variation in execution times of jobs. Finally, the state of the system is composed of all other time varying quantities that appear in equation (1). This will boil down to the state being represented by the vector of queue sizes of each task in the system. From a modeling perspective, our system can be depicted as in Figure 2, where the tasks and resources form the plant, and the resource manager forms a controller controlling the accumulation of jobs in the system by means of adjusting task rates. The performance of the resource manager and possibly its stability depend on its capacity to accurately predict the changes in the execution times of the jobs released into the system and it capacity to react accordingly.

4.2. Stability of Discrete-Time Dynamical Systems

A discrete-time control system is part of a larger class of systems called discrete-time dynamical systems. A discrete-time dynamical system is a tuple $\{\mathfrak{T}, \mathfrak{X}, \mathfrak{A}, \mathfrak{S}, \mathfrak{U}\}$ where: $\mathfrak{T} = \{t_{[k]} | k \in \mathbb{Z}_+, 0 = t_{[0]} < t_{[1]} < \cdots < t_{[k]} < \cdots\}$ is the discrete set of times at which the system evolves, $\mathfrak{X}$ is the state space, $\mathfrak{A} \subset \mathfrak{X}$ is the set of all initial states of the system ($\vec{x}_{[0]}$), $\mathfrak{U}$ is the bounded set of all inputs and $\mathfrak{S}$ is the set of all trajectories (all solutions of (3) : $\vec{x}_{[k]} = p(k, \vec{x}_{[0]}, \vec{u}_{[k]})$ where $t_{[k]} \in \mathfrak{T}$, $\vec{x}_{[0]} \in \mathfrak{A}$, and $\vec{u}_{[k]} \in \mathfrak{U}$). The state space must be a normed space $(\mathfrak{X}, |\cdot|)$. A dynamical system's state is desired to be 0_n. Because systems are subject to inputs (and perturbations), this condition does not hold. Under these premises a dynamical system is said to be stable if its state remains "close" to 0_n for all input patterns and initial conditions.

For our system we consider the following stability definition (we recall that notations, such as $\mathfrak{K}$ and $\mathfrak{KL}$-functions are described in Section 3.1):

Definition 4.1 (*Global Asymptotic Stability*). A dynamical system S, expressed recursively as:

$$F(\vec{x}_{[k+1]}, \vec{x}_{[k]}, \vec{u}_{[k]}) = 0_n \tag{3}$$

is *global asymptotically stable* (GAS) if there exists a $\mathfrak{KL}$-function β such that for each initial state $\vec{x}_{[0]} \in \mathfrak{A}$ and for each input function $\vec{u} : \mathbb{Z}_+ \to \mathfrak{U}$ we have that:

$$|p(k, \vec{x}_{[0]}, \vec{u}_{[k]})| \leq \beta(|\vec{x}_{[0]}|, k)$$

Definition 4.2 (*Input-to-State Stability*).
A dynamical system S (equation (3)) is *input-to-state stable* (ISS) if there exists a $\mathfrak{KL}$-function β and a $\mathfrak{K}$-function γ such that for each initial state $\vec{x}_{[0]} \in \mathfrak{A}$ and for each input function $\vec{u} : \mathbb{Z}_+ \to \mathfrak{U}$: we have that:

$$|p(k, \vec{x}_{[0]}, \vec{u}_{[k]})| \leq \max\{\beta(|\vec{x}_{[0]}|, k), \gamma(\|\vec{u}\|)\}$$

where $\|\vec{u}\| = \sup\{|\vec{u}_{[k]}|, k \in \mathbb{Z}_+\}$.

A GAS system approaches its equilibrium point regardless of the initial state from where it begins. An ISS system initially approaches its equilibrium point similarly to a GAS system but it stops when its state becomes bounded in a ball of a certain size around its equilibrium point. The size of the ball depends on the magnitude of its input. We illustrate graphically the meaning of these stability concepts in Figures 3a for GAS and 3b for ISS. For a deeper understanding of these two concepts and the link between them we point the reader to works such as [Sontag (2001); Jiang et al.(2001)].

5. PROBLEM FORMULATION

We consider a system consisting of a distributed platform $\mathcal{R}$ with an application $\mathcal{A}$ running on it and a resource manager.

Our goal is to model the behavior of the adaptive distributed real-time system (equation (1)), with a focus on its worst-case behavior in order to derive conditions under which it is guaranteed to be ISS with respect to its perturbations ($\vec{w}$) and GAS with respect to its controlled inputs ($\vec{u}$). We do not concern ourselves with the particular aspect of the resource manager (equation (2)) or its performance when applied to the system. We wish to determine:

— conditions on the system's parameters and structure, that guarantee the existence of a set of resource managers which can keep the system stable, and
— conditions that characterize the set of stable resource managers.

Our stability conditions require that a norm of the state of the system finally becomes bounded in a ball around 0_n, meaning bounded queue sizes and, thus, the existence of real-time properties (e.g. worst-case response times, end-to-end delays, etc.) in the system.

6. MAIN RESULTS

In this section we shall briefly present the main results derived in this manuscript, which are twofold:

(1) a *worst-case behavior model* for the adaptive distributed real-time system, and
(2) stability analysis of this model, comprising: *necessary conditions*, *sufficient conditions*, and *conditions on the behavior of the resource manager*.

These results will be presented in detail in later sections of this manuscript.

6.1. Worst-case Behavior Model

We model the worst-case behavior of an adaptive distributed real-time system as a linear switching system with random switching:

$$\vec{x}_{[k+1]} \leq A_{l_{[k]}}(\vec{x}_{[k]} + (\vec{w}_{[k+1]} - \vec{w}_{[k]}) + (\vec{u}_{[k]} - 1_n)(t_{[k+1]} - t_{[k]})) + \vec{b}_{l_{[k]}} \tag{4}$$

where $l : \mathbb{Z} \to \{0, 1, \cdots, 2^n - 1\}$ is a function determining the branch taken by the system, with:

$$A_{l_{[k]}} \in \{A_0 = 0_{n\times n}, A_1, \cdots, A_{2^n-2}, A_{2^n-1} = I_n\} \qquad \vec{b}_{l_{[k]}} \in \{\vec{b}_0, \vec{b}_1, \cdots, \vec{b}_{2^n-1} = 0_n\}$$

and the state ($\vec{x}_{[k]} = [x_{i[k]}]_n$), input ($\vec{u}_{[k]} = [u_{i[k]}]_n$), and perturbation ($\vec{w}_{[k]} = [w_{i[k]}]_n$) vectors of the system are:

$$\vec{x}_{[k]} = \left[\sum_{j \in \mathcal{I}_{\mathbf{N}_i}} c_j^{\max} \cdot q_{j[k]} + \sum_{j \in \mathcal{I}_{\mathbf{N}_i}} c_j^{\max} \cdot \left(\sum_{j^\star \in \pi(j)} \lceil q_{j^\star[k]} \rceil \right) \right]_n$$

$$\vec{u}_{[k]} = \left[\sum_{j \in \mathcal{I}_{\mathbf{N}_i}} c_j^{\max} \cdot \rho_{\gamma(j)[k]} \right]_n \qquad \vec{w}_{[k]} = \left[\sum_{j \in \mathcal{I}_{\mathbf{N}_i}} c_j^{\max} \cdot \rho_{\gamma(j)[k-1]} \phi_{\gamma(j)[k]} \right]_n$$

The state of the system is an n-dimensional vector where each component $x_{i[k]}$ is the state of a resource $\mathbf{N}_i$, $\forall i \in \mathcal{I}_{\mathcal{R}}$ in the system. $x_{i[k]}$ represents the worst-case accumulation of execution time – existing at $t_{[k]}$ in the system – that needs to be executed by resource $\mathbf{N}_i$ and it is formed of all the released jobs (both ready for execution – the first sum – and not yet ready for execution – the second sum) of tasks mapped to $\mathbf{N}_i$. Since $\vec{x}_{[k]}$ is a linear combination of queue sizes, bounded state will imply bounded queue sizes and, thus, the existence of real-time properties. The input of the system is an n-dimensional vector of worst-case input loads, one for each resource in the system, prescribed by the resource manager. The amount $u_{i[k]}(t_{[k+1]} - t_{[k]})$ represents the worst-case accumulation of execution times released for $\mathbf{N}_i$ (as jobs) during $[t_{[k]}, t_{[k+1]}]$. This amount should be executed by resource $\mathbf{N}_i$ until $t_{[k+1]}$ in order for its state not to grow. The perturbation in the system exists because the input accumulation (described based on the continuous worst-case input load $u_{i[k]}$) is, in fact determined by the discrete number of released jobs. All these accumulations are computed assuming the worst-case execution times for all jobs. The equation (4), therefore, describes the worst-case behavior of the system. When execution times of jobs are at their largest, it takes the longest amount of time for the system to process these jobs, thus leading the largest queue sizes.

The system evolves linearly according to one of 2^n branches, each characterized by a different matrix A_l and vector $\vec{b}_l$, $l \in \{0, \cdots, 2^n - 1\}$. The branches correspond to all combinations of *starving* and *non-starving* resources. At time $t_{[k]}$, resource $\mathbf{N}_i$ is *starving* if it cannot utilize the whole interval of time $[t_{[k]}, t_{[k+1]}]$ to execute jobs. This is because the queues of the tasks mapped to this resource become empty. Thus, $\mathbf{N}_i$ is starving at $t_{[k]}$ if $q_{j[k+1]} = 0$ for all tasks τ_j, $j \in \mathcal{I}_{\mathbf{N}_i}$. At any moment of time, we cannot determine which resources are starving since that requires information about the future, therefore, we assume that the switching between branches is done randomly. The matrices A_l associated with the different branches describe how accumulation of execution times migrates from the non-starving resources into the starving ones. These migrations happen because jobs that finish their execution on a resource $\mathbf{N}_i$ triggers the corresponding jobs (of the successive tasks in the task chain, mapped onto different other resources) to become ready for execution and, thus, accumulation of execution times from $\mathbf{N}_i$ migrates to the other resources, being amplified or attenuated in the process. The amplification/attenuation happens because the corresponding jobs of tasks in a task chain have different execution times. For a branch l where resources $\mathbf{N}_i$ to $\mathbf{N}_j$ are non-starving, the matrix A_l and vector $\vec{b}_l$ have the following form:

$$A_l = \begin{array}{c} \\ 1 \\ 2 \\ \\ i-1 \\ i \\ i+1 \\ \\ j \\ j+1 \\ \\ n \end{array} \begin{array}{c} \begin{array}{cccccccccc} 1 & \cdots & i-1 & i & i+1 & \cdots & j & j+1 & \cdots & n \end{array} \\ \begin{pmatrix} 0 & \cdots & 0 & \alpha_{1,i} & \alpha_{1,i+1} & \cdots & \alpha_{1,j} & 0 & \cdots & 0 \\ 0 & & 0 & \alpha_{2,i} & \alpha_{2,i+1} & & \alpha_{2,j} & 0 & & 0 \\ \vdots & & & \vdots & & & & \vdots & & \\ 0 & & 0 & \alpha_{i-1,i} & \alpha_{i-1,i+1} & & \alpha_{i-1,j} & 0 & & 0 \\ 0 & & 0 & 1 & 0 & \cdots & 0 & 0 & & 0 \\ 0 & & 0 & 0 & 1 & & 0 & 0 & & 0 \\ \vdots & & & \vdots & & \ddots & & \vdots & & \\ 0 & & 0 & 0 & 0 & & 1 & 0 & & 0 \\ 0 & & 0 & \alpha_{j+1,i} & \alpha_{j+1,i+1} & \cdots & \alpha_{j+1,j} & 0 & & 0 \\ \vdots & & & \vdots & & & & \vdots & & \\ 0 & & 0 & \alpha_{n,i} & \alpha_{n,i+1} & & \alpha_{n,j} & 0 & & 0 \end{pmatrix} \end{array}; \quad \vec{b}_l = \begin{pmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_{i-1} \\ 0 \\ 0 \\ \vdots \\ 0 \\ \beta_{j+1} \\ \vdots \\ \beta_n \end{pmatrix}$$

where all α coefficients (common for all matrices) are amplification/attenuation coefficients related with the migration of accumulation of execution times. For any two resources $\mathbf{N}_i$ and $\mathbf{N}_{i^\star}$, $\alpha_{ii^\star}$ is the largest ration between all sets of tasks on these two resources, where task $\tau_{j^\star}$ mapped to $\mathbf{N}_{i^\star}$ is a predecessor of τ_j mapped to $\mathbf{N}_i$. The β coefficients are perturbation coefficients characteristic of each particular branch of evolution of the system, and they appear because of approximation done when constructing the worst-case behavior model.

$$\alpha_{ii^\star} = \begin{cases} \max\limits_{\substack{j\in\mathcal{I}_{\mathbf{N}_i} \\ j^\star\in\mathcal{I}_{\mathbf{N}_{i^\star}} \\ \gamma(j)=\gamma(j^\star) \\ j^\star<j}} \left\{\frac{c_j^{\max}}{c_{j^\star}^{\max}}\right\}, & \forall i^\star \in \varphi(i) \\ 0, & \text{otherwise} \end{cases} ; \qquad \beta_i = \sum_{j\in\mathcal{I}_{\mathbf{N}_i}} c_j^{\max}\cdot\pi(j), \qquad \forall i \in \mathcal{I}_{\mathcal{R}}$$

The worst-case behavior model of adaptive distributed real-time systems, will be derived and explained in detail in Section 7.

6.2. Stability Analysis

Stability implies bounded queue sizes. Since all parameters in equation (4) are positive (queue sizes are positive values) the issue of stability reduces to understanding under what conditions the state of the system decreases. In this manuscript we derive three stability results:

(1) *necessary conditions* which consider the behavior of each resource in isolation,
(2) *sufficient conditions* which consider the interaction between resources, and
(3) *conditions on the behavior of the resource manager* which describe what behaviors of the resource manager lead to stable systems.

The first two results determine if there exists a set of resource managers that can keep the system stable. The last result is a characterization of this set. The results are described in the following three theorems:

THEOREM 6.1. *A necessary condition for a system to be stable is:*

$$\sum_{j\in\mathcal{I}_{\mathbf{N}_i}} c_j^{max}\cdot\rho_{\gamma(j)}^{min} \le 1, \ \forall i \in \mathcal{I}_{\mathcal{R}} \tag{5}$$

The proof of this theorem is given in Section 8.1.

COROLLARY 6.2. *If Theorem 6.1 is satisfied, then the set $\Gamma_\star$ defined below is not empty:*

$$\Gamma_\star = \left\{\vec{\rho}^\star \in \mathbf{P} \,\middle|\, \sum_{j\in\mathcal{I}_{\mathbf{N}_i}} c_j^{max}\cdot\rho_{\gamma(\tau_j)}^\star \le 1, \forall i \in \mathcal{I}_{\mathcal{R}}\right\} \neq \emptyset \tag{6}$$

When the resource manager chooses input rate vectors from $\Gamma_\star$ we are guaranteed that all non-starving resources reduce their state (subject to noise). We can observe this in equation (4) where the term $(\vec{u}_{[k]} - 1_n)(t_{[k+1]} - t_{[k]})$ is negative.

However, the condition in Theorem 6.1 is not sufficient as it does not account for the migration of accumulation between resources, where amplification effects (via the α coefficients) can override the reduction in state obtained by setting rate vectors from $\Gamma_\star$. We give an example of such a phenomenon in Section 8.2. To deal with this problem, the following theorem provides sufficient conditions for state reduction to occur:

THEOREM 6.3. *Any adaptive distributed real time system, modeled as in equation* (4)*, that satisfies Theorem 6.1 can be stabilized using rates from the set $\Gamma_\star$ if there exits a point $\vec{p} = [p_i]_n \ge 0_n$, and some $0 < \mu < 1$ such that:*

$$\mu p_i - \left(\sum_{\substack{i^\star=1 \\ i^\star\neq i}}^{n} \alpha_{ii^\star} p_i + \beta_i\right) \ge 0, \qquad \forall i \in \{1, 2, \cdots n\} \tag{7}$$

is satisfied.

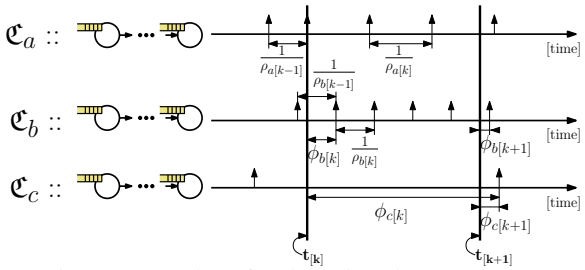


Fig. 4: Examples of task chain release patterns.

The proof of this theorem is given in Section 8.3.

With the two theorems above, we describe the behavior of the resource manager as follows:

THEOREM 6.4. *For any system that satisfies conditions* (5) *and* (7) *(Theorems 6.1 and 6.3) and for any* $\Omega \geq 2\frac{|\vec{w}^{max}|_p}{1-\mu}$*, a sufficient condition for stability is that its resource manager satisfies:*

$$\vec{\rho}_{[k+1]} \in \begin{cases} \Gamma_\star, & \text{if } |\vec{x}_{[k]}|_p \geq \Omega \\ \mathbf{P}, & \text{otherwise} \end{cases}; \qquad \forall k \in \mathbb{Z}_+ \tag{8}$$

The proof of this theorem is given in Section 8.5.

Further results such as bounds on queue sizes and on worst-case response times, together with extensions of the system model to more general systems, are developed in Section 9.

7. MODELING OF THE ADAPTIVE DISTRIBUTED REAL-TIME SYSTEM

In this section we develop the model of the behavior of adaptive distributed real-time systems.We describe the evolution of our system at discrete moments in time $t_{[k]}$, $k \in \mathbb{Z}_+$ when one of the following events happens:

(1) the resource manager activates,
(2) a job of a task τ_i is released at a different rate than the former job of this task, or,
(3) the scheduler on any resource switches to executing jobs of a different task than before.

To help with understanding, we use Figure 4 to illustrate the evolution of an example system with three task chains C_a, C_b, and C_c in between time moments $t_{[k]}$ and $t_{[k+1]}$. The meaning of the task chain rates is simply that the distance between successive jobs of tasks of a task chain C_p is $1/\rho_p$. We can see from C_a that jobs released after $t_{[k]}$ are separated by $1/\rho_{a[k]}$ while jobs released before are separated by $1/\rho_{a[k-1]}$. In the case of C_b we can observe that $t_{[k]}$ does not correspond with the release of new jobs, thus, there exists an offset $\phi_{b[k]}$ between $t_{[k]}$ and the first jobs released after it. From C_c we observe that we might not have any jobs released during $[t_{[k]}, t_{[k+1]}]$.

7.1. System Model

In this section we develop the system of difference equations that form the model of our distributed system.

The evolution of the queues of tasks mapped to a given resource $\mathbf{N}_i$, at the discrete moments in time defined above, depends on: the number of jobs arriving to these queues, the execution times of all these jobs, and how the scheduler on that resource decides to execute jobs from these queues.

We model the evolution of queues of all tasks mapped to a resource together, as an accumulation of execution time that needs to be executed by the resource, as follows:

$$\sum_{j \in \mathcal{I}_{\mathbf{N}_i}} c_{j[k]} q_{j[k+1]} = \max\left\{0, \sum_{j \in \mathcal{I}_{\mathbf{N}_i}} c_{j[k]} q_{j[k]} + \sum_{j \in \mathcal{I}_{\mathbf{N}_i}} c_{j[k]} s_{j[k+1]} - (t_{[k+1]} - t_{[k]})\right\}, \ \forall i \in \mathcal{I}_{\mathcal{R}} \tag{9}$$

We obtain equation (9) where the quantities $q_{j[k]}$, $\forall j \in \mathcal{I}_{\boldsymbol{\Theta}}$ represent the queue sizes of the queues of all tasks in the system, $c_{j[k]}$ represents the (unknown) average execution time for the jobs of τ_j, $\forall j \in \mathcal{I}_{\boldsymbol{\Theta}}$ that will be executed be during $[t_{[k]}, t_{[k+1]}]$, and $s_{j[k+1]}$ represents the number of jobs of τ_j, $\forall j \in \mathcal{I}_{\boldsymbol{\Theta}}$ that become ready for execution during $[t_{[k]}, t_{[k+1]}]$. $\sum_{j \in \mathcal{I}_{\mathbf{N}_i}} c_{j[k]} q_{j[k]}$ is the accumulation of

execution time existing in the queues of the tasks running on resource $\mathbf{N}_i$, at $t_{[k]}$; $\sum_{j\in\mathcal{I}_{\mathbf{N}_i}} c_{j[k]}s_{j[k+1]}$ is the accumulation of execution time of all incoming jobs to these queues, and $\sum_{j\in\mathcal{I}_{\mathbf{N}_i}} c_{j[k]}q_{j[k+1]}$ is the accumulation of execution time remaining in the queues at $t_{[k+1]}$. In between the two consecutive moments of time $t_{[k]}$ and $t_{[k+1]}$, the resource can execute jobs whose execution times amount to at most $t_{[k+1]} - t_{[k]}$. The evolution of the quantities $s_{j[k+1]}$, $\forall j \in \mathcal{I}_{\mathbf{\Theta}}$ is described in the following equation:

$$s_{j[k+1]} = \begin{cases} \lceil q_{j^\star[k]} \rceil + s_{j^\star[k+1]} - \lceil q_{j^\star[k+1]} \rceil, & \text{if } j^\star = \mathfrak{p}(j) \neq \xi \\ \lceil \rho_{\gamma(j)[k]} \max\{0, (t_{[k+1]} - t_{[k]}) - \phi_{\gamma(j)[k]}\} \rceil, & \text{otherwise} \end{cases}, \ \forall j \in \mathcal{I}_{\mathbf{\Theta}} \tag{10}$$

If τ_j has $\tau_{j^\star}$ as predecessor then $s_{j[k+1]}$ represents the number of jobs of $\tau_{j^\star}$ that were executed during $[t_{[k]}, t_{[k+1]}]$ (first branch of the equation). If τ_j has no predecessor, then $s_{j[k+1]}$ represents the number of jobs released by the task chain (second branch of the equation). The quantities $\phi_{p[k]}$, $\forall p \in \mathcal{I}_{\mathcal{A}}$ represent the offsets from $t_{[k]}$ when the new instance of task chains C_p gets released into the system (see also Figure 4) and their evolution in time is given by equation (11).

$$\phi_{p[k+1]} = \phi_{p[k]} + \frac{1}{\rho_{p[k]}} \lceil \rho_{p[k]} \max\{0, (t_{[k+1]} - t_{[k]}) - \phi_{p[k]}\} \rceil - (t_{[k+1]} - t_{[k]}), \ \forall p \in \mathcal{I}_{\mathcal{A}} \tag{11}$$

The quantities $\rho_{p[k]}$, $\forall p \in \mathcal{I}_{\mathcal{A}}$ represent the rates at which task chains release new instances for execution and are the inputs (decided by the resource manager) to our real-time system.

Equations (9) to (11) form a preliminary candidate for the model of our system. For ease of use, we reduce this system in the following way. By observing from equation (11) that

$$\lceil \rho_{p[k]} \cdot \max\{0, (t_{[k+1]} - t_{[k]}) - \phi_{p[k]}\} \rceil = \rho_{p[k]} \cdot (\phi_{p[k+1]} - \phi_{p[k]}) + \rho_{p[k]} \cdot (t_{[k+1]} - t_{[k]})$$

and by introducing the expressions of $s_{j[k]}$ into equation (9) we obtain:

$$\begin{aligned} \sum_{j\in\mathcal{I}_{\mathbf{N}_i}} c_{j[k]}q_{j[k+1]} = max\Big\{0, \quad & \sum_{j\in\mathcal{I}_{\mathbf{N}_i}} c_{j[k]}q_{j[k]} + \sum_{j\in\mathcal{I}_{\mathbf{N}_i}} \Big(c_{j[k]} \sum_{j^\star\in\pi(j)} \lceil q_{j^\star[k]} \rceil - \lceil q_{j^\star[k+1]} \rceil \Big) \\ & - \sum_{j\in\mathcal{I}_{\mathbf{N}_i}} c_{j[k]}\rho_{\gamma(j)[k]}(\phi_{\gamma(j)[k]} - \phi_{\gamma(j)[k+1]}) + \Big(\sum_{j\in\mathcal{I}_{\mathbf{N}_i}} c_{j[k]}\rho_{\gamma(j)[k]} - 1 \Big)(t_{[k+1]} - t_{[k]}) \quad \Big\} \end{aligned} \tag{12}$$

Equation (12), repeated for every resource $\mathbf{N}_i$, $i \in \mathcal{I}_{\mathcal{R}}$, forms the model of our system. The state of this system, at each moment of time, is given by the vector of task queue sizes.

Stability implies that the state evolves in a bounded interval. Because we do not have enough equations in our model (we have one equation for each resource, not one equation for each task) we cannot determine the precise queue size for each queue, but we shall reason about the stability of the system nonetheless. We do so by aggregating the queues of all tasks running on a resource into the accumulation of execution times on that resource and bounded accumulations will imply bounded queues.

7.2. Worst-case Behavior of the System

We shall now perform a number of manipulations to the system model in order to make it easier to handle. First, we define a set of notations in order to make the formulas more compact to write. Next we develop an approximated version of the system that describes its behavior in the worst case. This allows us to eliminate the effect of variations in execution times. Finally, we write the whole system model in a matrix form.

In order to simplify the handling of equation (12) we define the following notations, for all resources $\mathbf{N}_i$, $\forall i \in \mathcal{I}_{\mathcal{R}}$:

$$\sigma_{i[k]} = \sum_{j\in\mathcal{I}_{\mathbf{N}_i}} c_{j[k]} \cdot q_{j[k]} \qquad \sigma^{\max}_{i[k]} = \sum_{j\in\mathcal{I}_{\mathbf{N}_i}} c^{\max}_{j} \cdot q_{j[k]}$$

$$\zeta_{i[k]} = \sum_{j\in\mathcal{I}_{\mathbf{N}_i}} c_{j[k]} \cdot \Big(\sum_{j^\star\in\pi(j)} \lceil q_{j^\star[k]} \rceil \Big) \qquad \zeta^{\max}_{i[k]} = \sum_{j\in\mathcal{I}_{\mathbf{N}_i}} c^{\max}_{j} \cdot \Big(\sum_{j^\star\in\pi(j)} \lceil q_{j^\star[k]} \rceil \Big)$$

$$o_{i[k]} = \sum_{j \in \mathcal{I}_{\mathbf{N}_i}} c_{j[k]} \cdot \rho_{\gamma(j)[k-1]} \phi_{\gamma(j)[k]} \qquad o^{\max}_{i[k]} = \sum_{j \in \mathcal{I}_{\mathbf{N}_i}} c^{\max}_{j} \cdot \rho_{\gamma(j)[k-1]} \phi_{\gamma(j)[k]}$$

$$U_{i[k]} = \sum_{j \in \mathcal{I}_{\mathbf{N}_i}} c_{j[k]} \cdot \rho_{\gamma(j)[k]} \qquad U^{\max}_{i[k]} = \sum_{j \in \mathcal{I}_{\mathbf{N}_i}} c^{\max}_{j} \cdot \rho_{\gamma(j)[k]}$$

We can now rewrite equation (12) as:

$$\sum_{j \in \mathcal{I}_{\mathbf{N}_i}} c_{j[k]} q_{j[k+1]} = \max \left\{ 0, \quad \sigma_{i[k]} + \zeta_{i[k]} - \zeta_{i[k+1]} + o_{i[k+1]} - o_{i[k]} + (U_{i[k]} - 1)(t_{[k+1]} - t_{[k]}) \right\} \tag{13}$$

where $\sigma_{i[k]}$ represents the accumulation of execution times on resource $\mathbf{N}_i$ at time moment $t_{[k]}$, $\zeta_{i[k]}$ represents the accumulation of execution times flowing into $\mathbf{N}_i$ from tasks predecessor to the ones running on $\mathbf{N}_i$, the value $U_{i[k]}$ represents the load added to the system during $[t_{[k]}, t_{[k+1]}]$, and $o_{i[k]}$ represents the accumulation of execution times determined by the offsets.

Because of the way we define the time moments $t_{[k]}$, we are sure that if $\phi_{j[k]} \neq 0$ then $\rho_{\gamma(j)[k-1]} = \rho_{\gamma(j)[k]}$ and thus $o_{i[k]} = \sum_{j \in \mathcal{I}_{\mathbf{N}_i}} c_{j[k]} \cdot \rho_{\gamma(j)[k]} \phi_{\gamma(j)[k]}$ as well. All values $\sigma_{i[k]}$, $\zeta_{i[k]}$, $o_{i[k]}$, $U_{i[k]}$ are positive values for all $\mathbf{N}_i$, $i \in \mathcal{I}_{\mathcal{R}}$ and $t_{[k]}$, and are upper bounded by $\sigma^{\max}_{i[k]}$, $\zeta^{\max}_{i[k]}$, $o^{\max}_{i[k]}$, and $U^{\max}_{i[k]}$. We observe that $o^{\max}_{i[k]} \leq \sum_{j \in \mathcal{I}_{\mathbf{N}_i}} c^{\max}_{j}$ for all $t_{[k]}$. Furthermore, since between every consecutive time moments only jobs of one task (for each resource) are being executed, inequality (14) holds, and represents the worst-case behavior of the system (attainable if $c_{j[k]} = c^{\max}_{j}$, $\forall j \in \mathcal{I}_{\boldsymbol{\Theta}}$, $\forall k \in \mathbb{Z}_+$):

$$\sigma^{\max}_{i[k+1]} \leq \max \left\{ 0, \quad \sigma^{\max}_{i[k]} + \zeta^{\max}_{i[k]} - \zeta^{\max}_{i[k+1]} + o^{\max}_{i[k+1]} - o^{\max}_{i[k]} + (U^{\max}_{i[k]} - 1)(t_{[k+1]} - t_{[k]}) \right\}, \qquad \forall i \in \mathcal{I}_{\mathcal{R}}, \forall k \in \mathbb{Z}_+ \tag{14}$$

The state of the system corresponds to $(\sigma^{\max}_{i} + \zeta^{\max}_{i})$ since these quantities contain the task queue sizes, whose evolution is of interest to us.

We observe that each resource $\mathbf{N}_i$ has two modes of operation, corresponding to the two arguments to the max operator in inequation (14):

(1) *starving:* when the queues of tasks running on the resource are empty, and the resource must sit idle ($\sigma^{\max}_{i[k+1]} = 0$), and
(2) *non-starving:* when the queues of tasks running on the resource are non-empty and the resource is working executing jobs.

We rewrite inequation (14) (for each resource $\mathbf{N}_i$, $i \in \mathcal{I}_{\mathcal{R}}$) in such a way as to highlight the state of the system and the two modes of operation:

$$\sigma^{\max}_{i[k+1]} + \zeta^{\max}_{i[k+1]} \leq \begin{cases} \zeta^{\max}_{i[k+1]}, & \text{if } \sigma^{\max}_{i[k+1]} = 0 \\ \sigma^{\max}_{i[k]} + \zeta^{\max}_{i[k]} + o^{\max}_{i[k+1]} - o^{\max}_{i[k]} + (U^{\max}_{i[k]} - 1)(t_{[k+1]} - t_{[k]}), & \text{otherwise} \end{cases} \tag{15}$$

The evolution of the state of the system (associated with $\mathbf{N}_i$) is well described when the resource $\mathbf{N}_i$ is non-starving (the second branch), and it depends on its current state $(\sigma^{\max}_{i[k]} + \zeta^{\max}_{i[k]})$, the load added to $\mathbf{N}_i$ during the time interval $[t_{[tp]}, t_{[k+1]}]$ $(U^{\max}_{i[k]})$ and a random but bounded noise due to the offsets $(o^{\max}_{i[k+1]} - o^{\max}_{i[k]})$. However when $\mathbf{N}_i$ is in starving mode, inequation (15) becomes indeterminate $(\sigma^{\max}_{i[k+1]} + \zeta^{\max}_{i[k+1]} \leq \zeta^{\max}_{i[k+1]}$ when $\sigma^{\max}_{i[k+1]} = 0)$. We deal with this situation by finding an upper bound on $\zeta^{\max}_{i[k+1]}$ which depends on the state of the other resources in the system (specifically, on the resources containing the predecessors of the tasks running on $\mathbf{N}_i$). We develop this upper bound as follows:

$$\zeta^{\max}_{i[k+1]} = \sum_{j \in \mathcal{I}_{\mathbf{N}_i}} c^{\max}_{j} \cdot \Big(\sum_{j^\star \in \pi(j)} \lceil q_{j^\star[k+1]} \rceil \Big) \leq \sum_{j \in \mathcal{I}_{\mathbf{N}_i}} c^{\max}_{j} \cdot \Big(\sum_{j^\star \in \pi(j)} q_{j^\star[k+1]} \Big) + \underbrace{\sum_{j \in \mathcal{I}_{\mathbf{N}_i}} c^{\max}_{j} \cdot \pi(j)}_{\beta_i}$$

$$= \sum_{i^\star \in \varphi(i)} \Bigg(\underbrace{\sum_{\substack{j \in \mathcal{I}_{\mathbf{N}_i} \\ j^\star \in \mathcal{I}_{\mathbf{N}_{i^\star}} \\ \gamma(j)=\gamma(j^\star)}} \frac{c_j^{\max}}{c_{j^\star}^{\max}} (c_{j^\star}^{\max} q_{j^\star[k+1]})}_{\leq \alpha_{ii^\star} \sigma_{i^\star[k+1]}^{\max}} \Bigg) + \beta_i \leq \sum_{i^\star \in \mathcal{I}_{\mathbf{N}_i}} \alpha_{ii^\star} (\sigma_{i^\star[np]}^{\max} + \zeta_{i^\star[k+1]}) + \beta_i$$

$$\leq \sum_{\substack{i^\star \in \varphi(i) \\ \sigma_{i^\star[k+1]}^{\max} \neq 0}} \alpha_{ii^\star} \Big(\sigma_{i^\star[k]}^{\max} + \zeta_{i[k]}^{\max} + o_{i[k+1]}^{\max} - o_{i[k]}^{\max} + (U_{i[k]}^{\max} - 1)(t_{[k+1]} - t_{[k]}) \Big) + \beta_i \tag{16}$$

Plugging in the bound from (16) into inequality (15) gives us a working model of the system for predicting its worst-case behavior. When a resource $\mathbf{N}_i$ is starving, the evolution of its state depends on the other resources in the system which are non-starving ($\mathbf{N}_{i^\star}$, where $i^\star \in \varphi(i)$, and $\sigma_{i^\star[k+1]}^{\max} \neq 0$). We can thus rewrite the inequation for each resource $\mathbf{N}_i$ as having 2^n branches, each one representing a different combination of starving and non-starving resources. These equations can then be aggregate into a matrix form as shown in equation (4). The approximation done in (16), makes this model an upper bound on the worst-case behavior of the system.

At each moment in time $t_{[k]}$, the system evolves according to one of the 2^n branches ($l_{[k]}$). The choice of the branch is unknown to the resource manager as we cannot tell at $t_{[k]}$ which resource will be starving at $t_{[k+1]}$. Such systems are called switching systems with random switching. The branch having A_{2^n-1} and $\vec{b}_{2^n-1}$ represents the case where all the resources are in non-starving mode. The branch having A_0 and $\vec{b}_0$ represents the case where all resources are starving meaning all queues are empty.

7.3. System Model Properties

We discuss here the form and properties of A_l and $\vec{b}_l$. From inequation (16) we can observe that the matrices $A_l = [a_{lij}]_{n\times n}$ and vectors $\vec{b}_l = [b_{li}]_n$ have a particular form. In any branch l, of the 2^n branches, in which the system may find itself at a certain moment in time, all non-starving resources ($\mathbf{N}_i$) evolve depending only on themselves and thus the rows i in A_l have all elements 0 apart from a_{lii} which is 1. All starving resources $i^\star$ evolve depending on the non-starving resources only and thus the rows $i^\star$ in A_l have all elements associated with the starving resources 0. The elements associated with the non-starving resources on the rows $i^\star$ are chosen from the coefficients α. In the vector $\vec{b}_l$ all elements associated with the non-starving resources are 0. The elements associated with the starving resources are taken from the coefficients β. The matrix A_l and vector $\vec{b}_l$ for an example setting where resources i to j are non-starving ($i < j$, $i, j \in \{1, 2, \cdots, n\}$), and resources 0 to $i-1$ and $j+1$ to n are starving has been presented in Section 6.

Associated with each matrices A_l we define the mapping: $S : \{A_0, \cdots A_{2^n-1}\} \rightarrow \mathcal{P}(\{1 \cdots n\})$ where $S(A_l)$ is the set of indexes of all non-starving resources (the indexes of all non-0 columns in A_l).

For each of the above matrices A_l and vectors b_l we have the following two properties:

(1) all matrices A_l are idempotent: $A_l^2 = A_l$, $\forall l \in \{0, \cdots, 2^n - 1\}$, and
(2) $A_l \vec{b}_l = 0_n$, $\forall l \in \{0, \cdots 2^n - 1\}$

7.4. Illustrative Example

In Figure 5 we present an example of a system with two resources ($\mathbf{N}_1$ and $\mathbf{N}_2$) transversed by a task chain (C) of four tasks (τ_1, τ_2, τ_3, and τ_4), two on each resource (τ_1 and τ_4 on $\mathbf{N}_1$, and τ_2 and τ_3 on $\mathbf{N}_2$). The mapping of tasks to resources forms a cycle in the resource graph (a dependency from τ_1 on $\mathbf{N}_1$ to τ_2 on $\mathbf{N}_2$, and a dependency from τ_3 on $\mathbf{N}_2$ to τ_4 on $\mathbf{N}_1$). At each moment in time $t_{[k]}$, the

state, perturbation, and input to the system (when modeling its worst-case behavior) are:

$$\vec{x}_{[k]} = \begin{pmatrix} \underbrace{c_1^{\max} q_{1[k]} + c_4^{\max} q_{4[k]}}_{\sigma_{1[k]}^{\max}} + \underbrace{c_1^{\max} 0 + c_4^{\max}(\lceil q_{1[k]} \rceil + \lceil q_{2[k]} \rceil + \lceil q_{3[k]} \rceil)}_{\zeta_{1[k]}^{\max}} \\ \underbrace{c_2^{\max} q_{2[k]} + c_3^{\max} q_{3[k]}}_{\sigma_{1[k]}^{\max}} + \underbrace{c_2^{\max} \lceil q_{1[k]} \rceil + c_3^{\max}(\lceil q_{1[k]} \rceil + \lceil q_{2[k]} \rceil)}_{\zeta_{2[k]}^{\max}} \end{pmatrix}$$

$$\vec{w}_{[k]} = \begin{pmatrix} c_1^{\max} \rho_{[k-1]} \phi_{1[k]} + c_4^{\max} \rho_{[k-1]} \phi_{4[k]} \\ c_2^{\max} \rho_{[k-1]} \phi_{2[k]} + c_3^{\max} \rho_{[k-1]} \phi_{3[k]} \end{pmatrix} \qquad \vec{u}_{[k]} = \begin{pmatrix} (c_1^{\max} + c_4^{\max}) \rho_{[k]} \\ (c_2^{\max} + c_3^{\max}) \rho_{[k]} \end{pmatrix}$$

The coefficients α and β for this system are:

$$\alpha_{12} = \max\left\{\frac{c_4^{\max}}{c_3^{\max}}, \frac{c_4^{\max}}{c_2^{\max}}\right\} \qquad \alpha_{21} = \max\left\{\frac{c_2^{\max}}{c_1^{\max}}, \frac{c_3^{\max}}{c_1^{\max}}\right\} \qquad \beta_1 = 3c_4^{\max} \qquad \beta_2 = c_2^{\max} + 2c_3^{\max}$$

This system has $2^2 = 4$ branches according to which it can evolve, with:

$$A_l \in \left\{ \underbrace{\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}}_{A_0}, \underbrace{\begin{pmatrix} 1 & 0 \\ \alpha_{21} & 0 \end{pmatrix}}_{A_1}, \underbrace{\begin{pmatrix} 0 & \alpha_{12} \\ 0 & 1 \end{pmatrix}}_{A_2}, \underbrace{\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}}_{A_3} \right\} \text{ and } \vec{b}_l \in \left\{ \underbrace{\begin{pmatrix} \beta_1 \\ \beta_2 \end{pmatrix}}_{\vec{b}_0}, \underbrace{\begin{pmatrix} 0 \\ \beta_2 \end{pmatrix}}_{\vec{b}_1}, \underbrace{\begin{pmatrix} \beta_1 \\ 0 \end{pmatrix}}_{\vec{b}_2}, \underbrace{\begin{pmatrix} 0 \\ 0 \end{pmatrix}}_{\vec{b}_3} \right\}.$$

8. CONDITIONS FOR STABILITY

Up to this point we have determined that our distributed real-time system can be modeled, as a switching system with random switching. This system evolves according to one of 2^n possible branches, and at every moment in time the branch may switch to a different one. As described in the above section, the system is stable when a norm of its state ($|\vec{x}_{[k]}|$) remains bounded at all times, regardless of the switching pattern the system may employ.

At runtime, the system is controlled by its resource manager. For a given system we may envision a set of resource manager policies that are able to keep the system stable. These policies may differ amongst themselves by their performance[2] but they are all capable of keeping the system stable.

In this section we present the proofs of the stability results (the three theorems) presented in Section 6.2.

8.1. Proof of the Necessary Conditions for Stability

We prove theorem 6.1 by contradiction. We allow equation (5) not to hold and we will show that this leads to an unstable system. If equation (5) does not hold, there exists at least one resource $\mathbf{N}_i$ that will have:

$$\sum_{j \in \mathcal{I}_{\mathbf{N}_i}} c_j^{\max} \cdot \rho_{\gamma(\tau_j)}^{\min} - 1 = \Lambda_i > 0$$

For the worst-case evolution $c_{j[k]} = c_j^{\max}$, $\forall j \in \mathcal{I}_{\mathbf{N}_i}$, $\forall k$ then we can rewrite equation (12) as:

$$\sigma_{i[k+1]} + \zeta_{i[k+1]} - o_{i[k+1]} = \max\left\{\zeta_{i[k+1]} - o_{i[k+1]}, \sigma_{i[k]} + \zeta_{i[k]} - o_{i[k]} + \Lambda_i(t_{[k+1]} - t_{[k]})\right\}$$

It is easy to see from above that $\sigma_{i[k+1]} + \zeta_{i[k+1]} - o_{i[k+1]} \geq \sigma_{i[k]} + \zeta_{i[k]} - o_{i[k]} + \Lambda_i(t_{[k+1]} - t_{[k]})$, $\forall k$. Since $0 \leq o_{i[k]} \leq \sum_{j \in \mathcal{I}_{\mathbf{N}_i}} c_j^{\max}$ for all k, we have that $\lim_{k\to\infty} \sigma_{i[k+1]} + \zeta_{i[k+1]} = \infty$ which implies that there exists at least one task $\tau_{j^\star}$, $j^\star \in \mathcal{I}_{\mathbf{N}_i} \bigcup \left(\bigcup_{j \in \mathcal{I}_{\mathbf{N}_i}} \pi(j) \right)$ such that $q_{j^\star[k]} \xrightarrow{k\to\infty} \infty$. It then follows that $|\vec{x}_{[k]}| \xrightarrow{k\to\infty} \infty$ and the proof is complete.

[2]We do not address the performance of a certain resource manager policy in this paper, as performance may mean different things for different systems and applications.

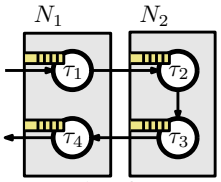


Fig. 5: Example of a system with two resources and one task chain.

Table I: Possible behavior of the system presented in Fig. 5.

t	0	$\frac{1}{5}$	$3N+\frac{4}{5}$	$3N+1$	$6N+\frac{8}{5}$
q_1	$2N$	$2N-1$	0	1	$3N+1$
q_2	0	1	0	0	0
q_3	0	0	$5N+1$	$5N$	0
q_4	0	0	0	1	0

8.2. Instability Example

We shall now present an example showing that, while condition (5) is necessary, it is not sufficient. The example presented here is a variation of the classic Lu-Kumar network (see [Bramson (2008)] chapter 3) from the domain of multi-class queueing networks.

Let us consider the system from Figure 5 for which we have $c_1^{\max} = c_3^{\max} = 1/5$, $c_2^{\max} = c_4^{\max} = 3/5$ and the input rate is $\rho^{\min} = 1$. All these numbers are given in appropriate time units. This system obviously satisfies Theorem 6.1.

Let us assume that the schedulers on each resource are fixed priority based where τ_4 and τ_2 have the highest priority on their respective resources. Let us also assume that the resource manager always chooses the minimum task chain release rate and the system evolves such that execution times are always at their maximum. If at time $t = 0$ $q_1 = 2N$ and all other queues are empty, then the evolution in time of the system is as shown in the Table I. At time $t = 1/5$ a job of τ_1 gets executed and moves to the queue of q_2. From this point on tasks τ_1 on $\mathbf{N}_1$ and τ_2 on $\mathbf{N}_2$ execute jobs in parallel. The jobs executed from q_2 are moved to q_3, but because τ_2 has priority over τ_3, $\mathbf{N}_2$ will execute jobs from q_3 only after q_2 becomes empty. Because the execution time of jobs of τ_1 is smaller than the execution time of jobs of τ_2, jobs from q_1 move to q_2 faster than from q_2 to q_3, thus $\mathbf{N}_2$ only begins executing jobs of τ_3 when the initial jobs in the system have been processed by τ_1 and τ_2. During the execution of these jobs, new jobs arrive into the system and move from q_1 to q_2. Considering the rate at which jobs arrive into the system (1 job every 1 time unit) the queue of τ_2 will finally become empty at time $t = 3N + 4/5$ time units. At this time the queue of τ_3 will now contain $q_3 = 5N + 1$ jobs. One job from q_3 gets executed until $t = 3N + 1$ time units and moves to q_4. From this point on, τ_3 and τ_4 execute jobs in parallel until q_3 empties (because jobs of τ_3 have smaller execution times that jobs of τ_4). On $\mathbf{N}_1$, τ_4 has priority over τ_1 so while q_4 empties, jobs accumulate on q_1. τ_4 has to execute $5N + 1$ jobs, so they will be finished in $3/5 * 5N + 3/5$ time units. During this time $3N + 1$ jobs accumulate in q_1. We can observe that at $t = 6N + 8/5$ the state of the system is similar with the one at $t = 0$. Queues q_2, q_3 and q_4 are empty but the size of q_1 has increased from $2N$ to $3N + 1$ jobs. Since N has been chosen arbitrarily, we can see that the behavior of the system is cyclic and the sizes of q_1 increases progressively with every cycle and, thus, the system is unstable, even if Theorem 6.1 is satisfied.

8.3. Proof of the Sufficient Conditions for Stability

Our system model presented in equation (4) describes the behavior of the system with respect to the input rate vectors, perturbations, and the interaction between the resources. We construct the proof of theorem 6.3 in three steps, first focusing on the stability of a simpler system that only shows the interaction between resources.

(1) We show that the system:

$$\vec{x}_{[k+1]} = A_{l_{[k]}} \vec{x}_{[k]} \tag{17}$$

where the matrices $A_{l_{[k]}}$ are the same as in equation (4), is GAS if there exist $0 < \mu < 1$ and $[p_i]_n \succ 0_n$ such that:

$$\mu p_i - \Big(\sum_{\substack{i^\star=1 \\ i^\star \neq i}}^{n} \alpha_{ii^\star} p_i \Big) \geq 0, \qquad \forall i \in \mathcal{I}_{\mathcal{R}} = \{1, 2, \cdots n\} \tag{18}$$

To do this, we construct the norm:

$$|\vec{x}_{[k]}|_p = \max_{i\in\mathcal{I}_{\mathcal{R}}}\Big\{\frac{|x_i|}{p_i}\Big\} \tag{19}$$

and we show that:

$$|\vec{x}_{[k+1]}|_p \leq |\vec{x}_{[k]}|_p, \quad \forall k \in \mathbb{Z}_+ \tag{20}$$

(2) We show that using norm (19), the system

$$\vec{x}_{[k+1]} = A_{l_{[k]}}\Big(\vec{x}_{[k]} + (\vec{w}_{[k+1]} - \vec{w}_{[k]}) + (\vec{u}_{[k]} - 1_n)(t_{[k+1]} - t_{[k]})\Big) \tag{21}$$

which evolves such that $\vec{x}_{[k]} \geq 0_n$, $\forall k \in \mathbb{Z}_+$ is GAS with respect to $\vec{u}$ and ISS with respect to $\vec{w}$.
(3) We extend the above discussion to the model of our system presented in equation (4).

Step 1**.** If condition (18) is satisfied, since $S(A_l) \subset \mathcal{I}_{\mathcal{R}}$, $\forall l \in \{0, 1, \cdots 2^n - 1\}$, we have that:

$$\sum_{\substack{i^\star=1\\ i^\star\neq i}}^{n} a_{lii^\star}\frac{p_{i^\star}}{p_i} \leq \mu < 1, \quad \forall i \in \mathcal{I}_{\mathcal{R}} \qquad \Rightarrow \qquad \sum_{i^\star\in S(A_l)} a_{lii^\star}\frac{p_{i^\star}}{p_i} \leq \mu < 1, \quad \forall i \in \mathcal{I}_{\mathcal{R}} \setminus S(A_l)$$

For system (17), if at time $t_{[k]}$ we have A_l such that $\mathbf{N}_i$ is starving, then we have that $x_{i[k+1]} = \sum_{i^\star\in S(A_l)} \alpha_{ii^\star} x_{i^\star[k]}$, thus, for any A_l we have:

$$|A_l\vec{x}|_p = \max\left\{\max_{i\in\mathcal{I}_{\mathcal{R}}\setminus S(A_l)}\left\{\frac{\Big|\sum\limits_{i^\star\in S(A_l)} a_{lii^\star}x_{i^\star}\Big|}{p_i}\right\}, \max_{i^\star\in S(A_l)}\left\{\frac{|x_{i^\star}|}{p_{i^\star}}\right\}\right\} \tag{22}$$

For all $\forall i \in \mathcal{I}_{\mathcal{R}} \setminus S(A_l)$ we have the following inequality[3]:

$$\frac{\Big|\sum\limits_{i^\star\in S(A_l)} a_{lii^\star}x_{i^\star}\Big|}{p_i} \leq \sum_{i^\star\in S(A_l)} a_{lii^\star}\frac{p_{i^\star}}{p_i}\frac{|x_{i^\star}|}{p_{i^\star}} \leq \frac{1}{\mu}\sum_{i^\star\in S(A_l)} a_{lii^\star}\frac{p_{i^\star}}{p_i}\frac{|x_{i^\star}|}{p_{i^\star}} \leq \frac{\sum\limits_{i^\star\in S(A_l)} a_{lii^\star}\frac{p_{i^\star}}{p_i}\frac{|x_{i^\star}|}{p_{i^\star}}}{\sum\limits_{i^\star\in S(A_l)} a_{lii^\star}\frac{p_{i^\star}}{p_i}} \leq \max_{i^\star\in S(A_l)}\left\{\frac{|x_{i^\star}|}{p_{i^\star}}\right\} \leq |\vec{x}|_p \tag{23}$$

and thus $|A_l\vec{x}|_p \leq |\vec{x}|_p$. Clearly if there exists a point $[p_i]_n$ such that condition (18) is satisfied, then norm (19) is a norm that satisfies condition (20) for system (17).
Step 2**.** We recursively expand equation (21) as:

$$\vec{x}_{[k+1]} = A_{l_{[k]}}A_{l_{[k-1]}}\cdots A_{l_{[1]}}\vec{x}_{[0]} + \vec{U}_{S[k]} + \vec{\Omega}_{S[k]}$$

where:

$$\begin{aligned}\vec{U}_{S[k]} =& A_{l_{[k]}}(\vec{u}_{[k]} - 1_n)(t_{[k+1]} - t_{[k]}) + A_{l_{[k]}}A_{l_{[k-1]}}(\vec{u}_{[k-1]} - 1_n)(t_{[k]} - t_{[k-1]})\\ &+ \cdots + A_{l_{[k]}}A_{l_{[k-1]}}\cdots A_{l_{[1]}}(\vec{u}_{[0]} - 1_n)(t_{[1]} - t_{[0]})\\ \vec{\Omega}_{S[k]} =& A_{l_{[k]}}(\vec{w}_{[k+1]} - \vec{w}_{[k]}) + A_{l_{[k]}}A_{l_{[k-1]}}(\vec{w}_{[k]} - \vec{w}_{[k-1]}) + \cdots + A_{l_{[k]}}A_{l_{[k-1]}}\cdots A_{l_{[1]}}(\vec{w}_{[1]} - \vec{w}_{[0]})\end{aligned}$$

We have that:

$$|\vec{x}_{[k+1]}|_p \leq |A_{l_{[k]}}A_{l_{[k-1]}}\cdots A_{l_{[1]}}\vec{x}_{[0]} + \vec{U}_{S[k]}|_p + |\vec{\Omega}_{S[k]}|_p$$

Since $\vec{x}_{[k]} \geq 0_n$, and $\vec{U}_{S[k]} \leq 0_n$ (since $\vec{u}_{[k]} \leq 1_n$), $\forall k \in \mathbb{Z}_+$ we have that $A_{l_{[k]}}A_{l_{[k-1]}}\cdots A_{l_{[1]}}\vec{x}_{[0]} \geq A_{l_{[k]}}A_{l_{[k-1]}}\cdots A_{l_{[1]}}\vec{x}_{[0]} + \vec{U}_{S[k]} \geq -\vec{\Omega}_{S[k]}$ and thus:

$$|\vec{x}_{[k+1]}|_p \leq \max\{|\vec{\Omega}_{S[k]}|_p, |A_{l_{[k]}}A_{l_{[k-1]}}\cdots A_{l_{[1]}}\vec{x}_{[0]}|_p\} + |\vec{\Omega}_{S[k]}|_p$$

[3]We use the fact that the weighted average of n elements is smaller/equal than the largest element, if the weights are all positive.

We want to show that the sequence $\left\{|\vec{\Omega}_{S\,[k]}|_p\right\}_k$ is bounded. We observe that:

$$\begin{aligned}\vec{\Omega}_{S\,[k]} &= A_{l_{[k]}}(\vec{w}_{[k+1]} - \vec{w}_{[k]}) + A_{l_{[k]}}\vec{\Omega}_{S\,[k-1]} \\ &= A_{l_{[k]}}(\vec{w}_{[k+1]} - \vec{w}_{[k]}) + A_{l_{[k]}}A_{l_{[k-1]}}(\vec{w}_{[k]} - \vec{w}_{[k-1]}) + A_{l_{[k]}}A_{l_{[k-1]}}\vec{\Omega}_{S\,[k-2]} = \cdots\end{aligned}$$

Let us analyze the behavior of the sequence in two extreme cases:

(1) if the matrices $A_{l_{[k]}}$ are such that $S(A_{l_{[k]}}) \subseteq S(A_{l_{[k-1]}})$, $\forall k \in \mathbb{Z}_+$, then one can verify that $A_{l_{[k]}}A_{l_{[k-1]}} = A_{l_{[k]}}$, which implies that $\vec{\Omega}_{S\,[k]} = A_{l_{[k]}}(\vec{w}_{[k+1]} - \vec{w}_{[1]})$ and then $|\vec{\Omega}_{S\,[k]}|_p \leq |\vec{w}^{\max}|_p$, $\forall k \in \mathbb{Z}_+$;

(2) if the matrices $A_{l_{[k]}}$ are such that $S(A_{l_{[k]}}) \cap S(A_{l_{[k-1]}}) = \emptyset$, $\forall k \in \mathbb{Z}_+$, then one can verify – by successive application of norm (19) to $\vec{x}$, $A_{l_{[k]}}\vec{x}$, $A_{l_{[k]}}A_{l_{[k-1]}}\vec{x}$, ... as in (22) and by repeating the reasoning from inequality (23) – that $|A_{l_{[k]}}A_{l_{[k-1]}}\vec{x}|_p \leq \mu|\vec{x}|_p$, $|A_{l_{[k]}}A_{l_{[k-1]}}A_{l_{[k-2]}}|_p \leq \mu^2|\vec{x}|_p$, ..., $\forall\vec{x}$ and then $|\vec{\Omega}_{S\,[k]}| \leq |\vec{w}^{\max}|_p \sum_{r=1}^{k}\mu^r \leq \frac{|\vec{w}^{\max}|_p}{1-\mu}$.

In the general case, some of the components of the vector $\vec{\Omega}_{S\,[k]}$ will behave as in the first case while the others will behave as in the second case and we have that $|\vec{\Omega}_{S\,[k]}|_p \leq \max\{|\vec{w}^{\max}|_p, \frac{|\vec{w}^{\max}|_p}{1-\mu}\} = \frac{|\vec{w}^{\max}|_p}{1-\mu}$, $\forall k \in \mathbb{Z}_+$ and thus bounded. Since $|\vec{\Omega}_{S\,[k]}|_p \leq \frac{|\vec{w}^{\max}|_p}{1-\mu}$ we have that (21) is GAS with respect to $\vec{w}$ and ISS with respect to $\vec{u}$, and thus stable.

Step 3**.** Our system model (4) may be rewritten in homogeneous coordinates as:

$$\begin{pmatrix}\vec{x}_{[k+1]}\\ 1\end{pmatrix} = \begin{pmatrix}A_{l_{[k]}} & \vec{b}_{l_{[k]}}\\ 0,0,\cdots 0 & 1\end{pmatrix}\left(\begin{pmatrix}\vec{x}_{[k]}\\ 1\end{pmatrix} + \begin{pmatrix}(\vec{u}_{[k]} - 1_n)(t_{[k]} - t_{[k-1]})\\ 0\end{pmatrix} + \begin{pmatrix}\vec{w}_{[k+1]} - \vec{w}_{[k]}\\ 0\end{pmatrix}\right) \tag{24}$$

One may simply verify that $\begin{pmatrix}A_l & \vec{b}_l\\ 0,0,\cdots 0 & 1\end{pmatrix}$, $l \in \{0,1,\cdots 2^{n-1}\}$ are idempotent and, thus, the above system represents the system (21) written in one dimension higher. For this system the norm (19) becomes $|(\vec{x},1)^T|_p = \max\{|\vec{x}|_p, \frac{1}{p_{n+1}}\}$ and condition (18) becomes $\mu p_i - \left(\sum_{\substack{i^\star=1\\ i^\star\neq i}}^{n}\alpha_{ii^\star}p_i + \beta_1 p_{n+1}\right) \geq 0$, $\forall i \in \mathcal{I}_{\mathcal{R}} = \{1,2,\cdots n\}$, where $p_{n+1} > 0$. We can conveniently choose $p_{n+1} = 1$ and, thus, our system model written in equation (4) can be stabilized if condition (7) is satisfied and the behavior of the system is such that the following inequation holds:

$$|\vec{x}_{[k+1]}|_p \leq \max\left\{|\vec{x}_{[k]}|_p, 1, \frac{|\vec{w}^{\max}|_p}{1-\mu}\right\} + \frac{|\vec{w}^{\max}|_p}{1-\mu}$$

8.4. Sufficient Conditions for a System of Two Resources

We shall now exemplify how to apply the above conditions on the example presented in Section 7.4. The conditions are:

$$\begin{cases}\alpha_{21}p_1 + \beta_2 - \mu p_2 \leq 0\\ \alpha_{12}p_2 + \beta_1 - \mu p_1 \leq 0\end{cases}$$

This system has the solutions: $p_1 \geq \frac{\beta_1+\alpha_{12}\beta_2/\mu}{\mu-\alpha_{12}\alpha_{21}/\mu}$ and $p_2 \geq \frac{\beta_2+\alpha_{21}\beta_1/\mu}{\mu-\alpha_{12}\alpha_{21}/\mu}$ when the condition $\mu-\alpha_{12}\alpha_{21}/\mu > 0$. Ultimately, the example system from Figure 5 is stable if $\alpha_{12}\alpha_{21} < \mu^2 < 1$. The meaning of this condition is that when accumulation of execution times moves circularly through the system e.g. from $\mathbf{N}_1$ to $\mathbf{N}_2$ and then back again to $\mathbf{N}_1$, the final accumulation on $\mathbf{N}_1$ is less then the initial one. For the particular case presented on page 14, we have $\alpha_{12} = \max\{\frac{3}{5}/\frac{1}{5}, \frac{3}{5}/\frac{3}{5}\} = 3$ and $\alpha_{21} = \max\{\frac{3}{5}/\frac{1}{5}, \frac{1}{5}/\frac{1}{5}\} = 3$. We then have $\alpha_{12}\alpha_{21} = 9 > 1$, thus the system is unstable.

8.5. Stability Analysis

In this section we derive condition on the behavior of the resource manager, that will render the whole adaptive real-time system, modeled in Section 7.1, stable.

We have shown in the above section that when choosing rate vectors from $\Gamma_\star$, a norm of the state of the system reduces below $2\frac{|\vec{w}^{\max}|_p}{1-\mu}$, thus, while $\vec{x}_{[k]} \geq \Omega$, the state norm will reduce until it becomes lower than Ω.

When the state of the system is such that $|\vec{x}_{[k]}|_p < \Omega$ the state norm may increase, but the increase is limited to the number of jobs that can enter the system in between two resource manager actuations. Assuming the highest task graph rates ($\vec{\rho}^{\max}$) the norm of the state may grow until it reaches the bound:

$$|\vec{x}_{[k]}|_p \leq \Psi = \Omega + \max_{i \in \mathcal{I}_\mathcal{R}} \left\{ \frac{\sum_{j \in \mathcal{I}_{\mathbf{N}_i}} \lceil c_j^{\max} \rho_{\gamma(j)}^{\max} h \rceil}{p_i} \right\} \tag{25}$$

at which point the state norm reduces again. Thus the system controlled by any resource manager that satisfies (8) is stable.

9. DISCUSSION

In this section we further explain the meaning and implications of our results, how to apply them, and their limitations.

In this work we have given a mathematical model (equation (12)) of the evolution in time of a system composed of a number of resources and a number of task chains distributed across them. These task chains release jobs at variable rates, decided by a resource manager (equation (2)). Because we only consider minimal assumptions regarding the behavior of the schedulers (see Section 3) on all resources, we only model the evolution of the queues in an aggregated fashion (we describe how the task queues of tasks running on each resource evolve together).

With the model that we developed, our goal is to describe its worst-case behavior and determine if the modeled system is stable in that case. Stability implies bounded task queues which means that no jobs are dropped and bounded real-time properties such as worst-case response times for tasks and end-to-end delays for task chains. In order to describe the worst-case behavior of the system, we bring forth notations such as the accumulation of execution times on each resource and we proceed to rewrite and build several upper-bound approximations of our model (equations (13) to (15)) until we find a worst-case behavior model that we can work with (equation (4)).

With the worst-case behavior model we proceed to determine stability properties and we present three results: necessary conditions for stability (Theorem 6.1), sufficient conditions stability (Theorem 6.3), and, finally, conditions on the resource manager controlling the system (Theorem 6.4).

The necessary conditions observe the individual behavior of each resource and determine if there exist rate vectors that can keep the accumulation of execution times on each resource bounded. The outcome is the set $\Gamma_\star$ of all these rate vectors. These conditions, however, are not enough because they do not take into account the interaction between the resources (interaction determined by the mapping of task chains to resources).

The sufficient conditions consider the interaction between the resources and determine if choosing actuation rates from the set $\Gamma_\star$ indeed guarantees the non-increase of the state of the system.

The necessary and the sufficient conditions in Theorems 6.1 and 6.3 respectively, refer to the properties of the system (resources and tasks) and indicate if it can be guaranteed that there exists any resource manager that keeps the system stable. If those conditions are satisfied, Theorem 6.4 formulates a condition which, if satisfied by the resource manager, guarantees that the system will remain stable.

To determine if a particular real-time system can be controlled in a stable way, we have to perform the checks from Theorems 6.1 and 6.3. All of these checks are performed off-line.

There are n necessary conditions to check (one for each resource in the system) and each is linear in the number of tasks running on that particular resource (see equation (5)). This makes the complexity of checking the necessary conditions to be linear in the number of tasks in the system, and thus easy to perform computationally.

The sufficient conditions require the determination of the coefficients $p_1, \ldots, p_n$ and μ. For a given μ we can determine $[p_i]_n$ by solving a system of n linear equations. The coefficient μ may be determined using a binary search on top of this. The complexity of the sufficient conditions is the

complexity of binary search multiplied by the complexity of the simplex algorithm (polynomial in the average-case).

Theorem 6.4 can be used in different ways:

(1) *To determine if an existing resource manager is stable*. This is done by determining if a certain threshold Ω, on the norm of the state, exists.
(2) *To help build custom, ad-hoc resource managers which are stable*. This is done by designing the resource manager around a preselected threshold Ω.
(3) *To modify an existing resource manager to become stable*.

We shall give several simple examples of resource managers in Section 10.

9.1. Link with Timing Analysis

As mentioned previously, an upper bound on the norm of the state of the system determines upper bounds on the queue sizes of each task and maximum queue sizes determine worst-case response times of each task. We can thus see our work as a form of timing analysis for adaptive distributed real-time systems.

To determine upper bounds on the worst-case response times for the tasks in our system one may proceed as follows: it is common for computer systems to start with task queues of size 0, thus making the ultimate bound Ψ (equation (25)) a bound on the largest norm of the state of the system. Remembering the definition of our norm (equation (19)) and the definitions of the state and accumulation of execution time (Section 7.1), we may bound each task queue size in the system to $q_j \leq \left\lceil \Psi \frac{p_{\gamma(j)}}{c_j^{\max}} \right\rceil, \forall j \in \mathcal{I}_{\mathbf{\Theta}}$, and assuming that all the jobs in the queue were released at their slowest rate, we see that the worst-case response time of task τ_j is $wcrt(\tau_j) = \frac{1}{\rho_{\gamma(j)}^{\min}} \left\lceil \Psi \frac{p_{\gamma(j)}}{c_j^{\max}} \right\rceil + \max_{j^\star \in \pi(j)} \{wcrt(\tau_{j^\star})\}$.

9.2. Further Extensions

We discuss two types of limitations here. First we discuss the limitation of our model being restricted to applications composed of task chains (where every task in the chain has at most one predecessor and one successor) rather than the more general acyclic task graphs. We have presented our theoretical results considering this model, for simplicity. However, what we have used in fact, is only that each task has a single predecessor. Thus all results trivially extend to task trees (each task has at most one predecessor but may have many successors) as well. Extending the model to general task graphs (where each task may have several predecessors, as well) is more complex. In that case, the concept of task queues disappears and is replaced by queues on each dependency link between two tasks (thus there are more input queues to each task and a job of that task is ready for execution the moment it exists in all input queues).

Let us now move on to the second type of limitations. In this paper we have used task chain rate changes as the method for adaptation of the system to various loads. This is useful e.g. for systems where the task chains represent controllers for external plants which may run with different control rates and higher rates improve the control quality. We shall discuss here extending the model for two other types of adaptations:

— adaptations by changing the resources' speed (using dynamic voltage and frequency scaling techniques), useful in low power systems, and
— adaptations by job dropping/admission, useful in web and multimedia applications

When dealing with adaptations by changing the resources' speed, let us assume constant task chain rates ρ_p, $p \in \mathcal{I}_{\mathcal{C}_p}$. We consider the worst-case execution times of tasks computed at the slowest speed of each resource, and we will model the act of increasing the speed of a resource as an increase in the load it can hold (as opposed to a decrease in the execution times of tasks running on that resource, e.g. doubling the speed of $\mathbf{N}_i$ means that $\mathbf{N}_i$ will handle a load of $\sum_{j \in \mathcal{I}_{\mathbf{N}_i}} c_{j[k]} \rho_{\gamma(j)} \leq 2$). We then have for each resource a set of loads L_i associated with the speeds it can run at and the system model from (4) becomes:

$$\vec{x}_{[k+1]} \leq A_{l_{[k]}}(\vec{x}_{[k]} + (\vec{w}_{[k+1]} - \vec{w}_{[k]}) + (\vec{u}_{[k]} - [L_{i[k]}]_n)(t_{[k+1]} - t_{[k]})) + \vec{b}_{l_{[k]}} \tag{26}$$

Table II: The parameters of the systems in Examples 1 to 4

Example	$c_1^{\max}$	$c_2^{\max}$	$c_3^{\max}$	$c_4^{\max}$	$\rho^{\min}$
1	6	6	6	6	1/10
2	2	6	2	6	1/10
3	5	4	4	3	1/10
4	2	4.95	2	4.95	1/10

The vector $[L_{i[k]}]_n$ is the input of this system, while $\vec{u}_{[k]}$ becomes a constant. In this setting, the necessary conditions describe the set of vectors $[L_{i[k]}]_n$ (the set of resource speeds) that can handle the worst-case load in the system ($\Gamma_\star^{speed}$). The sufficient conditions and the conditions on the resource manager retain their meaning and their proofs after the obvious modifications due to the different method of actuation. In the proof of the sufficient conditions, we use the fact that $\vec{u}_{[k]} - [L_{i[k]}]_n \leq 0_n$ when choosing $[L_{i[k]}]_n$ from $\Gamma_\star^{speed}$ instead of $\vec{u}_{[k]} \leq 1_n$ when choosing rates from $\Gamma_\star$.

When dealing with job dropping/admission we consider again constant rates. The inputs in the system are the number of jobs dropped from each task[4] and represent the amount of load that is dropped from each resource $d_{i[k]}$. The necessary conditions then become $\sum_{j\in\mathcal{I}_{\mathbf{N}_i}} c_j^{\max}\rho_{\nu(j)} - 1 \leq d_i$ and describe all combinations of dropped jobs that must occur in order to keep the system stable, in the worst-case scenario. In between two moments in time $t_{[k]}$ and $t_{[k+1]}$, the quantity $d_i(t_{[k+1]} - t_{[k]})$ describes the accumulation of execution times that must be dropped on resource $\mathbf{N}_i$. Of course, this dropped accumulation may come as jobs of only one task and the system must allow for that. With our new inputs d_i the worst-case behavior of the system becomes:

$$\vec{x}_{[k+1]} \leq A_{l_{[k]}}(\vec{x}_{[k]} + (\vec{w}_{[k+1]} - \vec{w}_{[k]}) + (\vec{u}_{[k]} - [d_{i[k]}]_n - 1_n)(t_{[k+1]} - t_{[k]})) + \vec{b}_{l_{[k]}} \tag{27}$$

As before, the sufficient conditions and the conditions on the resource manager retain their meanings and proofs.

Indeed, all three actuation methods, that we have described so far, could be used simultaneously, without affecting the basic structure of the model and of the results.

10. EXAMPLES

In this section we present several example systems in order to further illustrate the meaning of our theorems and how to apply them in practice. We shall present simulation results for 7 example systems, all of which have the topology of the example presented in Section 7.4 (see Figure 5) but with different system parameters (execution times, and rates) and different resource managers.

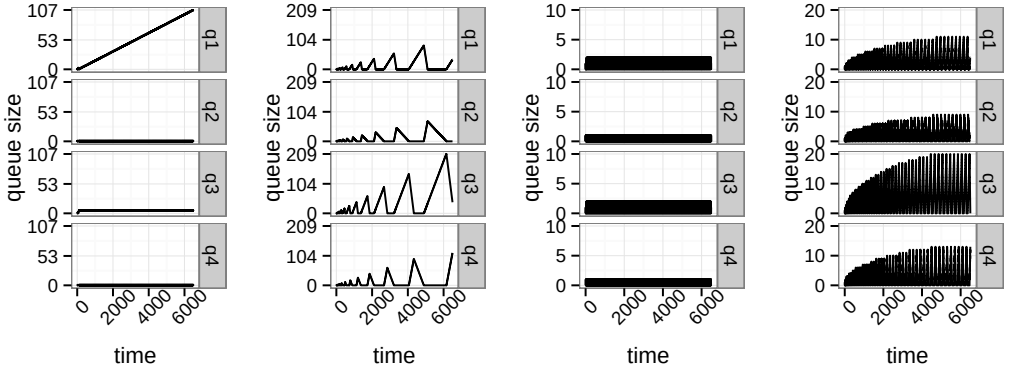
The first 4 examples employ a resource manager which always keeps the rates at their minimum. This allows us to explore different behaviors associated with satisfying or not satisfying the necessary and the sufficient conditions for stability. For these examples, at simulation, we only consider their worst case behavior (when all jobs execute with their worst-case execution times) and we plot the evolution in time of the task queue sizes.

The last 3 examples present the case of a systems which satisfies both Theorems 6.1 and 6.3, running with different resource managers. We analyze for stability the resource managers in these examples and we plot the evolution in time of the queue sizes of each tasks and the utilization on the resources.

The parameters of interest for the first 4 examples (the worst-case execution times of tasks and the minimum rate of the task chain) are presented in Table II. *Example* 1 presents a system that cannot be stabilized because it doesn't satisfy the necessary conditions. We can observe that by performing the tests in Theorem 6.1: $\rho^{\min}(c_1^{\max} + c_4^{\max}) = 1.2 > 1$ for $\mathbf{N}_1$ and $\rho^{\min}(c_2^{\max} + c_3^{\max}) = 1.2 > 1$. The behavior of this system is depicted in Figure 6a where we observe the unbounded growth of q_1, thus showing that the system is unstable.

Example 2 presents a system which satisfies the necessary conditions from Theorem 6.1 but not the sufficient ones in Theorem 6.3. This is the example presented on page 14. Here we can see that $\alpha_{21} = \max\{\frac{c_2^{\max}}{c_1^{\max}}, \frac{c_3^{\max}}{c_1^{\max}}\} = 3$ and $\alpha_{21} = \max\{\frac{c_4^{\max}}{c_2^{\max}}, \frac{c_4^{\max}}{c_3^{\max}}\} = 3$ so that $\alpha_{12}\alpha_{21} = 9 > 1$, thus not satisfying

[4] when a job of a task τ_j is dropped, the corresponding jobs of all succeeding tasks in the task graph are also dropped.



(a) *Example* 1. (b) *Example* 2. (c) *Example* 3. (d) *Example* 4.

Fig. 6: Evolution of queue sizes for the systems in Examples 1 to 4.

ALGORITHM 1

```
1: function P_CONTROLLER
2:     /* measure U_{i[k]}, ∀i ∈ I_R */
3:     err_[k] ← U^ref − max_i{U_{i[k]}}
4:     ρ⃗_[k+1] ← ρ⃗^ref + K_p err_[k] 1_n
5:     return ρ⃗_[k+1]
6: end function
```

ALGORITHM 2

```
1: function GUARDED_P_CONTROLLER
2:     /* measure U_{i[k]}, ∀i ∈ I_R */
3:     /* measure |x⃗_[k]|_p */
4:     if |x⃗_[k]|_p < Ω then
5:         return P_Controller()
6:     else
7:         return ρ⃗^min
8:     end if
9: end function
```

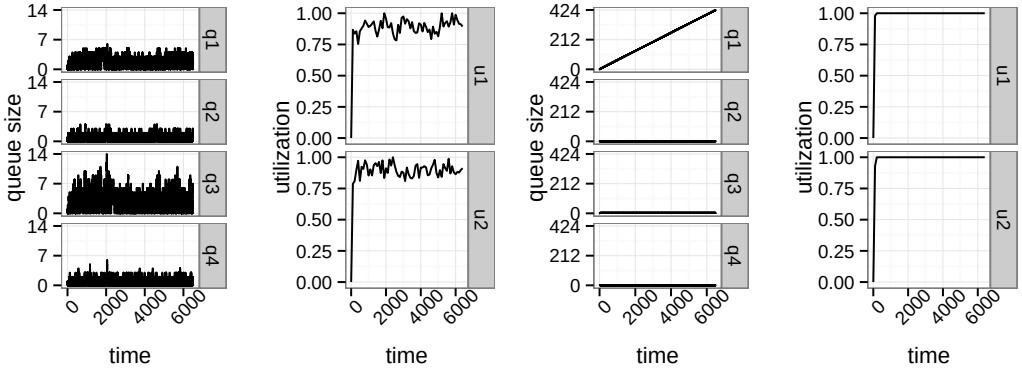
the sufficient conditions from Theorem 6.3. The behavior of this system is presented in Figure 6b and we can observe the queue sizes increasing and decreasing in a oscillatory fashion. However, each successive increase is larger than the previous one, thus showing that the system spirals out of control.

Example 3 presents a system that satisfies both the necessary and the sufficient conditions. This system can be stabilized by, for instance, always choosing the minimum rate. Its evolution is presented in Figure 6c and we can observe that the queue sizes are bounded for all queues in the system.

Example 4 presents a more interesting system, which shows an oscillatory but stable behavior, although while satisfying Theorem 6.1, it does not satisfy Theorem 6.3. This shows the sufficient, but not necessary nature of Theorem 6.3.

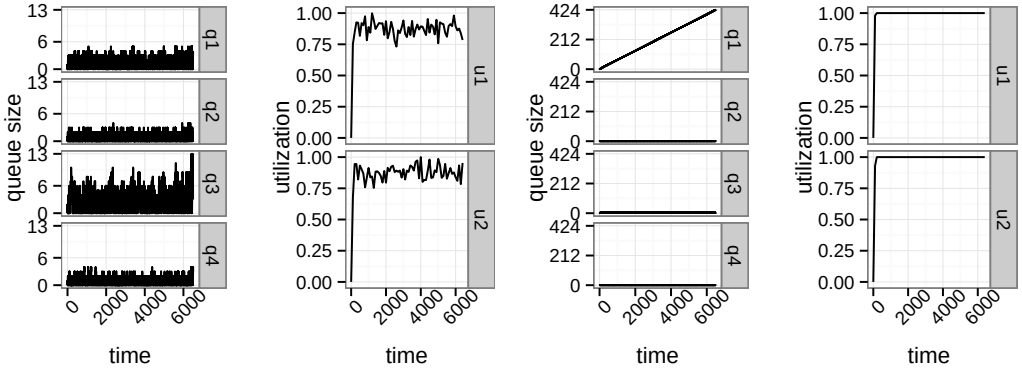
For the last three examples we present several simple resource managers and we show how to analyze them. All these resource managers control a system with the following parameters: $c_1 \in [0.5, 5]$, $c_2 \in [0.4, 4]$, $c_3 \in [0.4, 4]$, $c_4 \in [0.3, 3]$, and $\rho \in [0.1, 1]$. The resource managers actuate with a period $h = 100$ time units. This system has the same worst-case behavior as the one presented in *Example* 3 and, thus, satisfies Theorem 6.1, with $\Gamma_\star = \{\rho | \rho \in [0.1, 0.125]\}$, and Theorem 6.3, with $\mu = 0.882$ and $(p_1, p_2, p_3)^T = (100, 100, 1)$. Thus, it is possible to design resource managers that keep this system stable. In these examples we assume that high utilization on resources and low values for the queue sizes of the system are a desirable feature for our system.

Let us assume that the most desirable task chain rate in the system is $\rho^{ref} = 0.21$. Figure 7a presents the behavior of the system when execution times of jobs vary randomly in their intervals and the rate of the task chain it fixed at ρ^{ref}. Since $\rho^{ref} \notin \Gamma_\star$, however, the system is unstable in the worst-case as we can see in Figure 7b. For the worst-case simulation we considered that execution times are constantly maximal. In order to achieve stability, let us consider the resource manger presented in Algorithm 1, which is a simple proportional controller that reacts to the maximum utilization on the resources. This resource manager chooses rates that deviate from ρ^{ref} depending on the error measured between the maximum utilization and U^{ref}.



(a) Average-case behavior. (b) Worst-case behavior.

Fig. 7: Behavior of the system when $\rho_{[k]} = \rho^{ref}$.



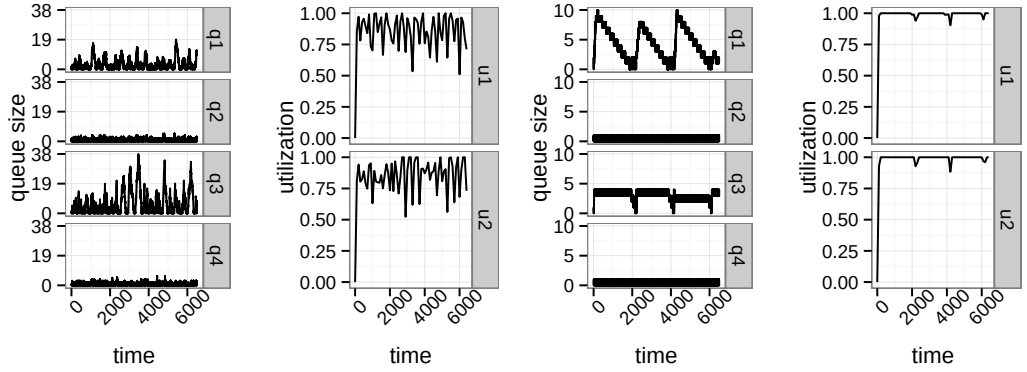
(a) Average-case behavior. (b) Worst-case behavior.

Fig. 8: Behavior of the system from *Example* 5.

Example 5 considers a system which employs the resource manager presented in Algorithm 1 where the proportional coefficient is $K_p = 0.2$. Figure 8 presents the system's behavior. In the average-case, the behavior of the system is slightly better than when choosing fixed rate at ρ^{ref} (queue size are slightly smaller while keeping the same utilization in the system). However, this system surprises us by being unstable in the worst-case. This happens because of the limitations of the resource utilization as a good measure of the state of the system. Since utilization saturates at 1, the measured error, saturates at $U^{ref} - 1 = -0.1$. The smallest rate this controller can choose is $\rho^{ref} - 0.2 * (-0.1) = 0.19 \notin \Gamma_\star$ thus it doesn't satisfy our stability conditions from Theorem 6.4.

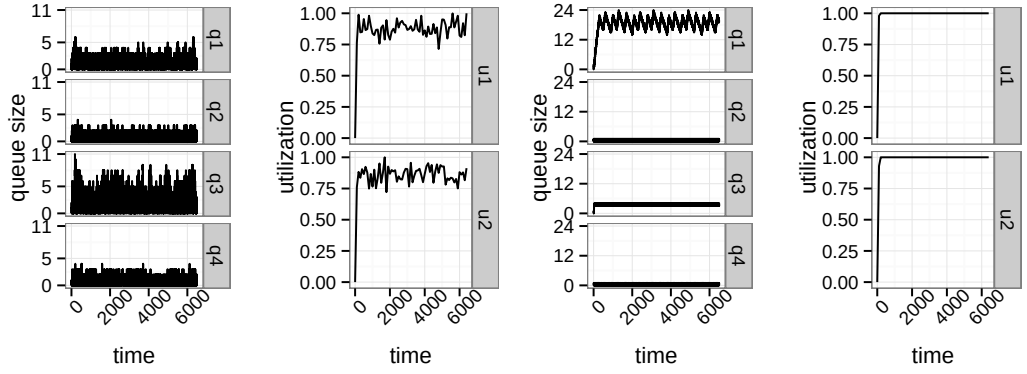
Example 6 attempts to correct the above behavior by choosing a better K_p coefficient. Since we need that $\rho^{ref} + K_p * (-0.1) \leq 0.125$ we obtain that $K_p \geq 0.85$. For this example we choose $K_p = 0.9$ and we present its behavior in Figures 9. From Figure 9b we can observe that the system remains stable in the worst-case. However, from Figure 9a we observe that the average-case performance of the system drops (queue sizes reach larger values than before). This happens because a large K_p coefficient implies too strong reactions to measured error in the system.

Example 7 presents a way to meet both requirements for average-case performance and worst-case stability, by employing the resource manager from Algorithm 2, which is a straight forward modification of Algorithm 1 by monitoring the norm of the state of the system and allowing the system to behave as in Algorithm 1 when the norm is below a bound, while selecting $\rho^{\min} = 0.1$



(a) Average-case behavior. (b) Worst-case behavior.

Fig. 9: Behavior of the system from *Example* 6.



(a) Average-case behavior. (b) Worst-case behavior.

Fig. 10: Behavior of the system from *Example* 7.

otherwise. For this system we select $K_p = 0.2$ and $\Omega = 2\frac{|o^{\max}|_p}{1-\mu} = 1.35$ ($o^{\max} = (8, 8, 0)^T$ for this system). The average-case and worst-case behaviors of this system are presented in Figures 10a and 10b respectively, and we can observe the desired behavior that we were after.

11. RELATED WORK

There exists a vast literature regarding resource utilization control, targeting different types of real-time systems. To understand it, and how it relates to our work, let us divide it into four classes based on the kind of adaptations proposed (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) according 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 [Lee et al.(1998); Buttazzo et al.(1998); Buttazzo et al.(2002); Buttazzo et al.(2004); Marioni et al.(2007); Rafiliu et al.(2010); Lu et al.(2002); Abdelzaher et al.(2003); Cervin et al.(2002)].
(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 voltage and frequency scaling). Methods such as [Marioni et al.(2007); Cucinotta et al.(2010b)] 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 [Abdelzaher et al.(2003)].
(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 [Palopoli et al.(2009); Cucinotta et al.(2010a); Liu et al.(2007); Yao et al.(2008); Cervin et al.(2003)] 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. 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. When discussing the stability of adaptive real-time systems, the issue of handling overload situations arises. By overload we mean that more jobs arrive per unit of time than they can be processed by the resource. Only methods from the first three classes can handle these overloads as they adjust the incoming job flow or the capacity of the resource to preserve equilibrium between demand and capacity. The methods from the fourth class are complementary methods that help improve the performance of the system.

Lee et al.(1998) proposed the QRAM algorithm. 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. 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.

Buttazzo et al.(1998) introduced the elastic model where each task's rate can change within a certain interval. Rates change when a task is added to or removed from the system. Further work deals with unknown and variable execution times [Buttazzo et al.(2002)], optimal control, when the applications are controllers [Buttazzo et al.(2004)], and when dealing with energy minimization [Marioni et al.(2007)]. 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.

In our previous work [Rafiliu et al.(2010)], we have proposed a number of control algorithms to deal with execution time variations in a uniprocessor system.

Lu et al.(2002) described a framework for feedback control scheduling, where the source of nondeterminism is execution time variation, and the actuation method is admission control and task rate change. The authors treat overloads by actuating based on utilization and deadline miss ratio. Both measures, however, saturate at 100% and thus are limited in their capability of describing overloads in the system.

In [Cervin et al.(2002)] 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 [Abdelzaher et al.(2003)] 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. It is well suited for accurately describing situations where overloads occur. 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. Non-empty queue sizes are characteristic of overloaded systems, which means that this approach may not be acceptable for systems where it is important that end-to-end delays are kept small.

Palopoli et al.(2009) consider resource-reservation schedulers and propose a feedback based technique for adjusting the quotas of tasks in reaction to task execution time variations. In [Cucinotta et al.(2010b)] the authors use a two-level feedback control approach where the inner loop adjusts scheduler parameters and the outer loop provides changes in power mode changes (through voltage and frequency scaling). In [Cucinotta et al.(2010a)] tasks share several resources and the quotas of tasks on all resources are determined together, in order to minimize end-to-end delays.

In [Cervin et al.(2003)] 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.

In [Liu et al.(2007); Yao et al.(2008)] the authors, presented a Recursive Least Squares based controller to control the utilization of a distributed system by means of rate adjustment. The authors consider distributed systems where tasks are schedulable if the utilization on each resource is kept at or below certain bounds. In [Liu et al.(2007)] the load on one resource is influenced by the load on the other resources via some coefficients which are estimated on-line, while in [Yao et al.(2008)] the model of the system is learned on-line.

In all works regarding adaptive systems, the issue of stability arises. Stability has slightly different meaning for the different works, but it always includes the idea that a system is stable when it is able to avoid situations where jobs released for execution can accumulate in an unbounded way, without being executed in a timely fashion. The above related works deal with stability in several ways: by proposing methods derived from control theory [Lu et al.(2005); Cervin et al.(2003); Liu et al.(2007); Yao et al.(2008); Abdelzaher et al.(2003); Palopoli et al.(2009); Cucinotta et al.(2010a)], by developing custom analysis to the proposed methods [Lee et al.(1998); Buttazzo et al.(1998); Buttazzo et al.(2002); Buttazzo et al.(2004); Marioni et al.(2007)], and by empirical simulation [Lu et al.(2002); Cervin et al.(2002); Rafiliu et al.(2010)]. All these works give specific solutions (methods for adaptivity together with their stability analysis) to specific types of problems, however, they are not well suited to answer basic stability questions for generic adaptive distributed real-time system.

In this paper we do not give a specific method for adaptation as in the related works. Instead we build a framework for modeling and analyzing generic real-time systems, with the goal of assuring worst-case stability of the system. We focus on making our framework as generic as possible in order for it to be appropriate for a large class of real-time systems. The stability analysis presented in this paper is done for adaptation mechanisms in the first class (*Job Flow Adaptation*), but we later show how it can be extended for methods from classes 2 and 3. The modeling of the system's time behavior presented here has similarities with previous works on schedulability analysis [Lehoczky et al.(1989); Baruah et al.(1990)].

We do not address adaptation mechanisms from the last class as they have limited capacity for handling overload situations. If the system finds itself in a situation where the incoming load is above 1 for extended periods of time, methods such as [Palopoli et al.(2009); Cucinotta et al.(2010a); Liu et al.(2007); Yao et al.(2008); Cervin et al.(2003)] 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,their goal being to improve system performance rather than assuring worst-case stability. This view is in agreement with the one presented in [Liu et al.(2007)] and [Cucinotta et al.(2010b)].

12. CONCLUSIONS

In many real-time system where variations in execution times are present and adaptation to these variations, in order to improve performance, is required, we face the issue of determining weather the adaptation mechanism is stable with respect to the timing properties of the system. Stability means that the state of the system remains bounded, implying bounded real-time properties such as worst-case response times and end-to-end delays. In this paper we have developed a model for adaptive distributed real-time systems, and used it to derive comprehensive conditions, on the system and the resource manager controlling it, that test the stability of the system. We have shown how

these results can be extended in various ways and we gave several examples regarding the meaning of the conditions and how to build stable resource managers.

Although beyond the scope of this paper, the framework presented here can also be used in an exploratory fashion in the design of distributed real-time systems, in order to: determine the necessary amount of adaptation for stability (by using Theorem 6.1), optimize the mapping of task to resources (by using Theorem 6.3), and determine system requirements for stability (e.g. queue size, by using Theorem 6.4).

References

T. F. Abdelzaher, J. A. Stankovic, C. Lu, R. Zhang, and 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).

K. J. Åström and B. Wittenmark. Computer-Controlled Systems. Prentice Hall, 1997.

S.K. Baruah, L.E. Rosier, and R.R. Howell. Algorithms and complexity concerning the preemptive scheduling of periodic real-time tasks on one processor. The Journal of Real-Time Systems, 2 (1990).

M. Bramson. Stability of Queueing Networks. Springer (2008).

G. C. Buttazzo, G. Lipari, and L. Albeni. Elastic Task Model for Adaptive Rate Control. In Proceedings of the IEEE Real-Time Systems Symposium, pp. 286, (December 1998).

G. C. Buttazzo and L. Albeni. Adaptive Workload Management through Elastic Scheduling. Journal of Real-Time Systems, vol. 23, pp. 7-24 (July 2002).

G. C. Buttazzo, 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).

A. Cervin, J. Eker, B. Bernhardsson, and K. E. Årzén. Feedback-Feedforward Scheduling of Control Tasks. Real-Time Systems, vol. 23, pp. 25-53 (July 2002).

A. Cervin and 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).

T. Cucinotta and L. Palopoli. QoS Control for Pipelines of Tasks Using Multiple Resources. IEEE Transactions on Computers, vol. 59, pp. 416-430 (2010).

T. Cucinotta, L. Palopoli, L. Abeni, D. Faggioli, and G. Lipari. On the Integration of Application Level and Resource Level QoS Control for Real-time Applications. IEEE Trans. Industrial Informatics (TII) vol. 6, issue 4, pp. 479-491, (2010).

Z. P. Jiang, and Y. Wang. Input-to-State Stability for Discrete-Time Nonlinear Systems. Automatica, vol. 37, issue 6, pp. 857-869 (2001).

E. Kreyszing. Introduction to Functional Analysis with Applications. John Wiley & Sons., Inc.

P. R. Kumar, S. Meyn. Stability Of Queueing Networks and Scheduling Policies. IEEE Trans. Automatic Control (1995).

C. Lee, J. Lehoczky, R. Rajkumar, and D. Siewiorek. On Quality of Service Optimization with Discrete QoS Options. In proceedings of Real-Time Technology and Applications Symposium, pp.276, (1999).

J.P. Lehoczky, L. Sha, and Y. Ding. The Rate-Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behavior. In Proceedings of the 10th IEEE Real-Time Systems Symposium, pp. 166172, (1989).

X. Liu, X. Zhu, P. Padala, Z. Wang, and 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).

C. Lu, J. A. Stankovic, S. H. Son, and G. Tao. Feedback Control Real-Time Scheduling: Framework, Modeling, and Algorithms. Real-Time Systems, vol. 23, pp. 85-126 (2002).

C. Lu, X. Wang and X. Koutsoukos. Feedback Utilization Control in Distributed Real-Time Systems with End-to-End Tasks. IEEE Transactions on Parallel and Distributed Systems, vol. 16, pp. 550-561 (2005).

M. Marioni and G. C. Buttazzo. Elastic DVS Management in Processors With Discrete Voltage/Frequency Modes. IEEE Transactions on Industrial Informatics, vol. 3, pp. 51-62 (February 2007).

L. Palopoli, T. Cucinotta, L. Marzario, and G. Lipari. AQuoSA – adaptive quality of service architecture. Journal of Software–Practice and Experience, vol. 39, pp. 1-31 (2009).

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).

S. Rafiliu, P. Eles, and Z. Peng. Stability of Adaptive Feedback-based Resource Managers for Systems with Execution Time Variations. Real-Time Systems Journal, vol 49, pp. 367-400 (2013).

E. D. Sontag. The ISS Philosophy as a Unifying Framework for Stability-Like Behavior. Lecture Notes in Control and Information Sciences, vol. 256, pp. 443-467 (2001).

J. Yao, X. Liu, M. Yuan, and 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).

K. Zhou, J.C. Doyle, and K. Glover. Robust and Optimal Control. New Jersey: Prentice Hall, vol. 40 (1996).