NICOLA MANICA

STOCHASTIC ANALYSIS OF A RESOURCE RESERVATION SYSTEM

STOCHASTIC ANALYSIS OF A RESOURCE RESERVATION SYSTEM

International Doctoral School in Information and Communication Technology
XXV Cycle
Department of Information Engineering and Computer Science
University of Trento

Doctoral Dissertation of:
NICOLA MANICA

Advisor:
PROF. LUCA ABENI

January 2013

EXAMINATION COMMITTEE:

Prof. LUCA ABENI	Chair, Advisor
Prof. GIORGIO BUTTAZZO	Member
dott. GREGORIO PROCISSI	Member

Nicola Manica: *Stochastic Analysis of a resource reservation system,* PhD in Information and Communication Technologies - Computer Science Area, © January 2013

PUBLICATIONS

INTERNATIONAL JOURNALS

[J1] L. Abeni, N. Manica, and L. Palopoli, "Efficient and robust probabilistic guarantees for real-time tasks," *Journal of Systems and Software*, 2011.

INTERNATIONAL CONFERENCES

[C1] N. Manica, L. Abeni, and L. Palopoli, "Qos support in the x11 window system," in *Real-Time and Embedded Technology and Applications Symposium, 2008. RTAS'08. IEEE*. IEEE, 2008, pp. 103–112.

[C2] ——, "Reservation-based interrupt scheduling," in *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2010 16th IEEE*. IEEE, 2010, pp. 46–55.

[C3] N. Manica, L. Abeni, L. Palopoli, D. Faggioli, and C. Scordino, "Schedulable device drivers: Implementation and experimental results," *Proceedings of the 6th International Workshop on Operating Systems Platforms for Embedded Real-Time Applications (OSPERT)*, pp. 53–62, 2010.

[C4] N. Manica, L. Abeni, and L. Palopoli, "Reservation-based interrupt scheduling," in *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2010 16th IEEE*, 12-15 2010, pp. 46 –55.

[C5] L. Palopoli, D. Fontanelli, N. Manica, and L. Abeni, "An analytical bound for probabilistic deadlines," in *Real-Time Systems (ECRTS), 2012 24th Euromicro Conference on*. IEEE, 2012, pp. 179–188.

TECHNICAL REPORTS

[T1] L. Palopoli, D. Fontanelli, N. Manica, and L. Abeni, "Closed form approximation for probabilistic guarantees using resource reservations."

[T2] L. Abeni and N. Manica, "Interoperable tracing tools," 2010.

[T3] P. Rallo, N. Manica, and L. Abeni, “Inferring temporal behaviours through kernel tracing,” 2010.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

An unmistakable trend in embedded systems is the growth of *soft real-time* computing. A soft real-time application is one for which deadlines can occasionally be missed, but the probability of this event has to be controllable and predictable.

1.1 SOFT REAL-TIME APPLICATIONS

In the last decade, we have witnessed the emergence of a new design paradigm in real–time computing in which the strict respect of every deadline is replaced by looser performance guarantees. A strong motivation for this is in the growing number of applications that require an "acceptable" timing behaviour but are resilient to occasional timing faults. Such applications are by and large referred to as *soft real–time*. Obvious applications for soft real-time systems can be found in the realm of signal processing, multimedia streaming, or even in control applications. In signal processing, the extraction of features from images can be done using an anytime approach, which produces varying levels of accuracy depending on the time allocated to the application. Audio/video streaming is another classical example of soft-real time: if we stream a movie at 25 frames per second, an occasional loss of a frame is not even perceived by the average user, as far as the anomaly is kept in check. Other unsuspected applications of the soft real-time paradigm have been found in real-time control. Empirical experiences [1] and recent theoretical findings [2] reveal that a moderate occurrence of deadline misses can be easily traded for a more aggressive choice of the task parameters (e.g., shorter activation periods).

For these applications, the execution time or the inter-arrival time have so wide a variance that a design based on the worst case behaviour produces a dramatic under-utilization of the system resources (hardly an affordable choice in many embedded applications). What is more, the strict respect of every deadline is often neither required nor desirable, as long as it can traded for a more efficient utilization of the system. This is true for multimedia applications and, surpris-

ingly, even for a large class of control applications [3]. The rising tide of these soft real–time applications is introducing a sea change in the industry of real–time systems, with classic hard real–time confined to a niche (albeit an important one). The market penetration of soft real–time applications is so relevant that it has stimulated an intense research activity aiming for an adequate satisfaction of their Quality of Service (QoS) goals.

1.2 PROBABILISTIC DEADLINE

For all these applications, the traditional notion of deadline is insufficient *per-se* to formulate Quality of Service requirements. Designers working in the soft real–time domain need effective means to reason about the impact of their scheduling choices on the performance of the system in stochastic terms. A very natural direction is offered by the *probabilistic deadlines* [4]: a deadline is associated with a probability of meeting it, which in turn is related to the scheduling parameters. The classic hard real–time systems can be recovered as special cases (setting the probability to one).

Even if probabilistic analysis can be applied to classic real-time scheduling algorithms, other scheduling approaches such as the resource reservations [5, 4] are commonly regarded as a superior choice for design for they enable a fine-grained control on the amount of CPU reserved to each task. For this and other reasons, the resource reservations algorithm (and their derivatives) have gained visibility and consideration in the real–time scheduling community. Despite this success, these algorithms are not yet supported by efficient and *scalable* tests for probabilistic guarantees, let alone analytic results linking the probability of missing the deadline with the resource allocation.

A feature of paramount importance of reservation-based algorithms is the so called *temporal isolation*: minimum performance guarantees can be offered to each task independently from the behaviour of the other tasks. This property introduces a drastic simplification in system design and is probably the main reason for the increasing popularity of resource reservations. However, to date the probabilistic analysis of reservation based systems cannot be regarded as a mature area. One of the few known facts is that when computation times and inter-arrival times are described by an independent and identically distributed (IID) process, a resource reservation can be modelled by

a discrete–time Markov chain (DTMC) with an infinite number of states [6, 7]. This model is difficult to manage from a numeric standpoint and offers little analytical insight into the system behaviour.

1.3 OBJECTIVES AND CONTRIBUTIONS TO NOVEL RESEARCHES

This work is aimed to close the gap in the research of stochastic real-time analysis related to resource reservation scheduling algorithms.

This dissertation attempts to:

1. give a quick overview of classic real-time analysis
2. analyze the problems related to use the well-known techniques in the context of soft real-time applications:
   - overvalue the assignation of parameters as in hard real-time systems based on worst case execution times
   - time and memory complexity using the known theoretical stochastic analysis
3. propose solutions able to overcome the limitation showed in point 2
4. show some specific examples (theoretical and practical) in which resource reservation lead to advantages.

The novel contributions of this thesis are:

- a new bound to predict the probability of a deadline misses in a resource reservation systems
- a very efficient numeric solution for matrix generated with well-know abstraction models of reservation based on Quasi Birth Death Markov Process
- an analytical solution, with some conservative approximations, for the same models.
- a new model for specific applications, like interrupts.
- experiments using resource reservation in different contexts

The thesis is evolved following two different approaches:

1. the first based on the exact model of reservation, and the contributions is:

- define a new pessimistic bound, efficient in term of computation, able to overcome the problem of complete knowledge of the computation time. The solution is an approximation of the real solution of the model.

2. the second based on an approximation model in which the novel contributions are:
   - presents an exact and numeric efficient solution for the model based on Quasi Birth and Death Markov Process
   - introduces an approximate analytical solution which can be computed with no complexity and which is reversible

These techniques are applicable since the minimum inter-arrival of a request is greater than a server period. Unfortunately exists situations in which this assumption is not feasible. An important example is using resource reservation to scheduling interrupts. In order to consider also this situation, another important novel result of this thesis is:

- to introduce a new model for scheduling interrupts

In addition, some practical examples of using resource reservation are presented.

1.4 THESIS ORGANIZATION

This thesis is organized in eleven Chapters, including the present one, and two Appendixes.
Chapter 2 presents some previous researches in the area of stochastic real-time analysis, showing the scenario in which the presented thesis is posed and Chapter 3 defines some basic concepts of random events, discrete time Markov chain and quasi birth and death Markov process. These definitions are needed to understand the main results of the thesis. Chapter 4 introduces the Constant Bandwidth Server (CBS), the scheduling algorithm used in the following chapters. This algorithm has been chosen for its simplicity and effectiveness, but in practice the analysis abstract from the implementation and any other resource reservation algorithm can be used. In Chapter 5 describes one of the most important results of this work: a pessimistic bound to predict the deadline miss probability in presence of resource reservation. The solution is approximated starting from the exact model. Chapter 6 presents a different approach of the same problem: the

classic model of [4] is seen as Quasi Birth and Death Process, allowing a very efficient numeric solution. In addition, with some conservative approximation, the same matrix can be solved with a simple analytic solution. Even if the two techniques presented can be used with real applications, there are still cases in which the analysis is not suitable. In particular when the applications has an inter-arrival time very small, the model cannot guarantee to have consistent results. For example the problem of scheduling interrupts, presented in Chapter 8, shows that resource reservation could be very useful, but a new model must be developed to have prediction on it. In addition, Chapter 9 presents some experiments with resource reservation, showing the improvement on the performance using this paradigm. Chapter 10 is a general discussion of the work described in this thesis, outlining paths for new and future researches. The two Appendixes show some works related to this dissertation which are developed for a different purpose. By definition, a systems cannot guarantee resource reservation in the presence of communication between applications. Appendix A introduces the concepts of resource reservation in the client-server paradigm.

2

STOCHASTIC REAL-TIME ANALYSIS

Examples of a similar analysis have been presented in the past (both for fixed priority [8, 9, 10] and for dynamic priority [11, 12, 13, 14] scheduling) and have been recently extended to multiprocessor systems [15]. Other scheduling approaches (for example, based on Time Division Multiplexing [16], on modifications of fixed priority scheduling [17], or on splitting tasks in mandatory parts and optional parts [18]) have been analyzed too.

The techniques based on classic fixed priority and earliest deadline first estimate the probability of missing deadlines for the task set considered as a whole: the parameters of a task can influence the termination statistics of the other tasks. This diminishes the potential interest of the method as a synthesis tool.

Traditional design of real-time systems follows a simple but effective paradigm [19]: 1) each task composing the system is characterized by its worst case inter-arrival time and execution time, 2) tasks are scheduled using static or dynamic priorities, 3) the ability for each task to meet its deadlines is guaranteed by a portfolio of algorithmic approaches such as the utilization test [19] or the response time analysis [20]. The enormous popularity of these technique is motivated by their amazing numeric efficiency, which makes them suitable for scalable design of real–time systems or even for admission tests executed on-line in a real–time operating system. Besides, these methods provide analytical over-approximations of the region of parameters [19, 21, 22] associated with a feasible design. These analytical expressions offer a precious insight into the system behaviour and guide the design choices.

A recent work from Mills and Anderson [23] has considered the problem of stochastic analysis for resource reservations on multiprocessor systems. The authors main focus is on the computation of tardiness and response time bounds for the average case. As a by-product of their work, the authors offer a result on the probabilistic deadlines, which is very conservative and is applicable only if deadlines much larger than the period are considered.

Finally, an interesting approach that deserve a mention is the real–time queueing theory [24], which allows one to compute the response time distributions of a set of tasks for different scheduling algorithms when the workload of the tasks is very close to 100% (*heavy traffic assumption*). The heavy traffic assumption and other assumptions on the task model restricts its applicability, but the paper has a strong theoretical interest.

Regarding the problem of assigning parameters for a resource reservations system, in [25] the authors characterize the response time of a served task as a function of the server parameters. In addition they provide a methodology to minimize the average response time optimizing the server parameters.

2.0.1 *Computation time*

In general, most of the approaches recalled above require the computation of the stationary probability distribution of the response times (or of an approximation), and only focus on mathematical equations that, when solved, provide such a distribution as a result. However, less effort is dedicated to how such equations are actually solved. As a result, when non-trivial distributions of the execution times are used, long times and large amounts of memory are required to compute the probabilistic deadlines. This issue makes probabilistic analysis unsuitable for on-line acceptance tests, which result inefficient on ordinary computing architecture and are at a serious risk of being unfeasible on many embedded devices utilising low-cost CPUs and small amounts of memory.

2.0.2 *Exact knowledge of distribution*

Another important limitation is that the *exact knowledge* of the entire distributions of the computation times and of the inter-arrival times of the tasks is required in order to properly estimate the deadline miss probabilities (and it is not possible to estimate the errors and approximations caused by an incomplete or inexact knowledge of the probability distributions). However, the statistics of the task activation parameters are typically collected over an extensive set of execution runs of the task. As a result, even in a long sequence of execution the worst-case condition may never occur and the experimental distribution could be incomplete. Hence, execution times higher than the

measured Worst Case Execution Time (WCET) can happen with a low probability ϵ_c (that can be computed by using statistical techniques). In order for an analysis methodology to be practically applicable, it has to possess a certain degree of robustness with respect to partially known distributions.

3

BACKGROUND AND DEFINITIONS

In this thesis, we consider a set of real-time tasks $\{\tau_i\}$ sharing a *processing unit* (CPU). A real-time task τ_i consists of a stream of jobs $J_{i,k}$. Each job $J_{i,k}$ arrives (becomes executable) at time $r_{i,k}$, and finishes at time $f_{i,k}$ after executing for a time $c_{i,k}$. Job $J_{i,k}$ is traditionally characterised by a deadline $d_{i,k} = r_{i,k} + D_i$, that is respected if $f_{i,k} \leqslant d_{i,k}$, and is missed if $f_{i,k} > d_{i,k}$. In this work, *probabilistic deadlines* [6] are used instead of traditional hard deadlines $d_{i,k}$. A probabilistic deadline (δ_i, p_i) is respected if $P\{f_{i,k} > r_{i,k} + \delta_i\} \leqslant p_i$[1].

Since the tasks parameters will be modelled as stochastic processes, we report here some basic definitions from the theory of random processes that will be used in the thesis. Interested readers are referred to the vast literature in the field for additional details [26].

3.1 RANDOMS EVENTS

A *random experiment* is any experiment whose outcome is uncertain. We define *sample space* Ω the set of all possible outcomes of a random experiment (for instance the sample space for throwing a dice the sample space is given by $\Omega = \{1, 2, \ldots, 5, 6\}$). In this set an *event* is a possible outcome of a random experiment and is associated to a subset of Ω (for instance, if the random experiment is throwing a dice, one possible event can be "outcome greater than 3", which is associated to the subset $\{4, 5, 5\}$). The set of all possible events is said *event space* and is a subset of 2^Ω, the set of all subsets of Ω. We will denote it by the symbol $\mathbb{E}$. On the $\mathbb{E}$ set, it is possible to define an algebra using the ordinary operations defined over sets (inclusion, intersection, union, complement).

The pair $(\Omega, \mathbb{E})$ is said *probability space*. Given a probability space $(\Omega, \mathbb{E})$, the probability $P\{.\}$, is a function from $\mathbf{E}$ to $\mathbf{R}$ such that: 1) $P\{A\} \geqslant 0$, for all $A \in \mathbf{E}$, 2)$P\{\Omega\} = 1$, 3) if the events $A_1, A_2, \ldots, A_n$ are disjoint, then $P\{\bigcup_{i=1}^{n} A_i\} = \sum_{i=1}^{n} P\{A_i\}$. Two events A_1 and A_2 are said independent if $P\{A_1 \cap A_2\} = P\{A_1\} \cdot P\{A_2\}$. The *conditional*

1 An hard real-time task can be described by a $(D_i, 0)$ probabilistic deadline.

probability $P(A|B)$ is defined as $P(A|B) = P(A \cap B)/P(B)$. For two independent events we have $P(A|B) = P(A)$.

3.1.1 *Random Variables and Random Processes*

Given a random experiment and a sample space Ω, a random variable X is a function that associates the possible outcome of an experiment with a real value: $X : \Omega \to \mathbf{R}$. For instance, if our random experiment is tossing a coin, we could decide that that $X(\text{head}) = 0$ and $X(\text{tails}) = 1$. For consistency, in the definition of $X()$ we also require that the set $\{\omega \in \Omega \text{ such that } X(\omega) \leqslant x\}$ for every $x \in \mathbf{R}$ is an event, which we will denote as $[X \leqslant x]$. The cumulative distribution function (CDF) $F(x)$ is defined as: $F(x) = P\{X \leqslant x\}$. Clearly, $F(-\infty) = 0$ and $F(\infty) = 1$. If a random variable X takes on values in a discrete set, it is possible to define the probability mass function (PMF) as $p(x) = P\{X = x\}$. The relation between PMF and CDF is given by $F(x) = \sum_{y \leqslant x} p(y)$.

The definition above easily extends to vectors of random variables $\mathbf{X} = [X_1, X_2, \ldots, X_n]$. In this case the CDF is defined as $F(x_1, x_2, \ldots x_n) = P\{X_1 \leqslant x_1 \wedge X_2 \leqslant x_2 \wedge \ldots \wedge x_n \leqslant X_n\}$. The CDF just defined is also called *joint* CDF. The *marginal* CDF $F_i(x_i)$ is simply obtained by evaluating $F(x_1, x_2, \ldots, x_n)$ on $x_j = +\infty$ for all $j \neq i$. In the same way, it is possible to define a joint PMF $p(x_1, x_2, x_n)$ and the marginal PMF $p_i(x_i)$. The random variables X_1 and X_2 are said *independent* if $F(x_1, x_2) = F_1(X_1) \cdot F_2(X_2)$. Given two variables X_1 and X_2 the *conditional PMF distribution* $p_{X_1|X_2}(x_1 1|x_2)$ is defined as $P\{X1 = x_1|X_2 = x_2\}$ and is given by $p(x_1, x_2)/p_2(x_2)$.

A *random process* (or stochastic process) is a family of random variables defined over a common probability space and indexed by a time variable: $\{X_t|t \in T\}$, where t varies over an index set. A stochastic process with a discrete index set is said a discrete–time random process (or random sequence); for this type of processes it is customary in the literature to use the indexes k, n, i, j (instead of t). In this thesis we will only consider discrete–time random processes. The set of values taken by the variable X_k is said *state space* of the process. If the state space is a discrete set (e.g., a subset of the natural numbers), the stochastic process is often referred to as a *chain*. In some sense, it is possible to think of a random process as an experiment whose outcome is a function of time (either discrete–time or continuous–time). For a fixed k, X_k is clearly a random variable, which can be associated with the CDF $F(x_k; k) = P\{X_k \leqslant x_k\}$. The process is *identically dis-*

tributed if $\forall k_1, k_2$, we have $F(x_{k_1}; k_1) = F(x_{k_2}; k_2)$, i.e., the CDF of X_k do not depend on k (the same applies to the PMF). If we consider the process samples taken at several time instants $k_1, k_2, \ldots, k_n$, we obtain a vector of random variables $X_{k_1}, X_{k_2}, \ldots, X_{k_n}$ associated with the joint CDF $F(x_{k_1}, x_{k_2}, \ldots, x_{k_n}; k_1, k_2, \ldots, k_n)$. A stochastic process is said *independent* if: $F(x_{k_1}, x_{k_2}, \ldots, x_{k_n}; k_1, k_2, \ldots, k_n) = \prod_{k=k_1,\ldots,k_n} F(x_k; k)$ (the same applies to the PMF). A process is said IID if it is both independent and identically distributed.

3.1.2 *Discrete–Time Markov Chains*

A *Discrete–Time Markov Process* (DTMP) is a discrete–time process such that its future development only depends on the current state and not on the past history. This can be stated in formal terms on the conditional PMF: $P\{X_n = x_n | X_1 = x_1, X_2 = x_2, \ldots, X_{n-1} = x_{n-1}\} = P\{X_n = x_n | X_{n-1} = x_{n-1}\}$. A DTMP defined over a discrete state space is said discrete–time Markov chain (DTMC). Given a DTMC, let $\pi_n^{(j)}$ represent the probability $\pi^{(j)}(n) = P\{X_n = j\}$, π_n be the vector $\pi_n = [\pi_n^{(0)}, \pi_n^{(1)}, \ldots]$, $P = [p_{i,j}]$ be a matrix whose generic element $p_{i,j}$ is given by the conditional probability $p_{i,j} = P\{X_n = j | X_{n-1} = i\}$. Starting from an initial probability distribution π_0, the application of the Bayes theorem and of the properties of the Markov Processes allow us to express the evolution of the distribution by the matrix equation $\pi_{n+1} = \pi_n P$. The matrix P is said probability transition matrix. An *equilibrium point* for this dynamic equation is a vector $\tilde{pi}$ such that $\tilde{\pi} = \tilde{\pi} P$.

For a DTMC, a *recurrent state* i is a state such that starting from i, the process eventually returns to state i with probability 1. For a recurrent state i, there exist some $n \geqslant 0$ such that $P\{X_m = i \wedge X_{m+n} = i\} > 0, \forall m$. The *period* d_i of a recurrent state i is defined as the greatest common divider of the set of all number n for which $P\{X_m = i \wedge X_{m+n} = i\} > 0, \forall m$. A state is said *aperiodic* if its period $d_i = 1$. A DTMC is said aperiodic, if all of its states are aperiodic. A DTMC is said *irreducible*, if every state can be reached from any other state in a finite number of steps. It can be shown that if a DTMC is irreducible, then all states are of the same type. So, if one state is aperiodic, so is the entire DTMC.

A very important property of irreducible and aperiodic DTMC is that there exist a single equilibrium $\tilde{\pi} = \tilde{\pi} P$ such that the limiting distributions $\lim_{n\to\infty} \pi_n$ converge starting from any initial probability distribution π_0. This equilibrium is called *steady state distribution*.

A DTMC is called a Quasi Birth Death process (QBDP) if its probability transition matrix P has the following block structure:

$$P = \begin{bmatrix} B_0 & B_1 & 0 & 0 & 0 & \cdots \\ A_{-1} & A_0 & A_1 & 0 & 0 & \cdots \\ 0 & A_{-1} & A_0 & A_1 & 0 & \cdots \\ 0 & 0 & A_{-1} & A_0 & A_1 & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \end{bmatrix}$$

When the matrices are scalars, this structure reduces to the standard Birth Dead Process (BDP).

4

CONSTANT BANDWITH SERVER

As multiple real-time tasks may be concurrently active at the same time, a scheduling mechanism is used to schedule the CPU. To this purpose, we advocate the use of *resource reservations*. Each task τ_i is associated with a reservation (Q_i^s, T_i^s), meaning that τ_i is allowed to execute for Q_i^s (*budget*) time units in every interval of length T_i^s (*reservation period*). The bandwidth B_i allocated to the task is defined as $B_i = Q_i^s/T_i^s$ and it can be thought of as the fraction of CPU time allocated to the task. The particular implementation of the Resource Reservations approach that we consider is the (CBS) [4]. In the CBS, reservations are implemented by means of an Earliest Deadline First (EDF) scheduler. The EDF schedules tasks $\{\tau_i\}$ based on their *scheduling deadlines* $d_{i,k}^s$, which are dynamically managed by the CBS algorithm. When a new job $J_{i,k}$ arrives, the server checks whether it can be scheduled using the last assigned scheduling deadline $d_{i,k-1}^s$. In the affirmative case, the scheduling deadline of the job is initially set to current deadline $d_{i,k}^s = d_{i,k-1}^s$. Otherwise, the initial deadline $d_{i,k}^s$ is set equal to $r_{i,k} + T_i^s$. Every time the job executes for Q_i^s time units (i.e., its budget is depleted), its scheduling deadline is postponed by T_i^s: $d_{i,k}^s = d_{i,k}^s + T_i^s$. This way, the task is prevented from executing for more than Q_i^s units with the same deadline. When the remaining budget q_i arrives to 0 τ_i is said to be depleted, and two different behaviours are possible:

- the budget is immediately replenished to Q_i^s and the scheduling deadline is postponed to $d_i^s + T_i^s$ (so, τ_i remains schedulable).
- τ_i is not schedulable until time d_i^s, when the budget will be replenished and the deadline will be postponed as above (so, it cannot be scheduled until d_i^s). This is known as *hard reservation behaviour*.

As a consequence, each task is reserved an amount of computation time Q_i^s in each server period T_i^s regardless of the behaviour of the

other tasks. This property is called *temporal isolation* and it holds as long as the system satisfies the following *schedulability condition*:

$$\sum_i B_i = \sum_i \frac{Q_i^s}{T_i^s} \leqslant 1. \tag{4.1}$$

The scheduling deadline $d_{i,k}^s$ has, in general, nothing to do with the deadline $d_{i,k}$ of the job: it is simply instrumental to a correct implementation of the resource reservation paradigm. For a full description of the algorithm and of its properties the reader is referred to [4].

5

EFFICIENT AND ROBUST PROBABILISTIC GUARANTEES

In this chapter we introduces a new analysis, able to give a pessimistic bound on the prediction of the probability of deadline miss. The specific scheduling algorithm used to implement the reservation strategy is not important for the analysis, and the only important property is that *task* τ_i *can execute for* Q_i^s *time units every* T_i^s *time units*. A possible scheduling algorithm is presented in Chapter 4.

The distribution of the finishing time can be computed applying standard arguments of the queueing theory. Since such analysis is significantly simplified when the inter-arrival times are multiples of T_i^s, a *conservative approximation* of $I_i(t)$ can be used, which ensures that the inter-arrival times are multiples of T_i^s. A possible definition [7] is

$$T_i(z) = P\left\{r_{i,j+1} - r_{i,j} = zT_i^s\right\} \tag{5.1}$$

where the random variable z represents the inter-arrival times expressed in multiples of T_i^s. To qualify the notion of conservative approximation of a random variable, it is useful to introduce the following relation between random variables [12]:

Definition 1. *Given two random variables* X *and* Y, $X \succeq Y$ *if* $F_x(x) \leqslant F_y(x)$ *for all* x *(where* $F_x(x) = P\{X \leqslant x\}$ *is the Cumulative Distribution Function - CDF - of* X*, and* $F_y(y)$ *is the CDF of* Y*).*

Since considering a shorter inter-arrival time is a conservative approximation, to be conservative $T_i(z)$ should be defined so that $F_{I_i}(x) \succeq F_{T_i}(x)$. A distribution with such a property can be easily computed as

$$T_i(z) = \sum_{t=zT_i^s}^{(z+1)T_i^s-1} I_i(t)$$

(see the cited paper [7], where the condition $F_{I_i}(x) \succeq F_{T_i}(x)$ is expressed as $\forall n, \sum_{t=0}^{nT_i^s} I_i(t) \leqslant \sum_{z=0}^{n} T_i(z)$).

Parts of this Chapter are going to appear in:
L. Abeni, N. Manica, L. Palopoli "Efficient and robust probabilistic guarantees for real-time tasks," *Journal of Systems and Software*

Summing up, the guarantees obtained by using $T_i(z)$ are valid for the original inter-arrival times distribution $I_i(t)$ too, and $T_i(z)$ has inter-arrival times multiple of T_i^s by construction.

When using reservation-based scheduling, each task can be provided with an individual guarantee (without having to consider all the other tasks in the system); hence, from now on a single task τ will be considered and the i index will be dropped (to simplify the notation).

In deterministic real-time analysis the WCET $C = \max_j\{c_j\}$ of task τ is assumed to be known, and even the probabilistic analysis techniques proposed up to now make the same assumption. Indeed, since $U(c)$ is assumed to be fully known, the maximum possible value C for which $U(C) \neq 0$ is assumed known as well. In this chapter, such a constraint about the WCET knowledge is relaxed, $U(c)$ is known up to a maximum value $\bar{C}$ and the WCET $C > \bar{C}$ can be unknown. However, in order to perform some analysis the probability $P\{c > \bar{C}\} = \sum_{c=\bar{C}+1}^{\infty} U(c) = \epsilon_c$ to have an execution time larger than $\bar{C}$ must be known. In this framework. setting $\epsilon_c = 0$ to the traditional model (with a known WCET). On the contrary, choosing $\epsilon_c > 0$ allows the designer to improve the robustness of the analysis. For example, if $U(c)$ is estimated by running N jobs of the task and measuring their execution times, we do not have any guarantee that the worst case situation has been considered. By using statistical techniques it is possible to estimate the confidence that the actual worst case computation time is actually the one resulting from the experiments. A very rough estimation of the probability of having a new run in which the worst case exceed the one found in the first N experiment runs is $1/N$.

As in previous work [7] a stochastic process v_j can be introduced to model the amount of time to be executed after the arrival of the j^{th} job $J_{i,j}$. As shown in the cited paper, v_j evolves according to the following rules:

$$\begin{aligned} v_0 &= c_0 \\ v_{j+1} &= \max\{0, v_j - z_j Q^s\} + c_{j+1} \end{aligned} \tag{5.2}$$

Informally speaking, Equations 5.2 says that the amount of time to be executed after the arrival of the first job is equal to the job's execution time, and the amount of time to be executed after the arrival of the

j^{th} job can be computed by summing the job's execution time to the amount of time to be executed after serving the previous jobs.

The worst-case finishing time of job $J_{i,j}$ can be computed based on the value of ν_j, as $\delta_j = \left\lceil \frac{\nu_j}{Q^s} \right\rceil T^s$; hence, when the probability distribution $V(\nu) = P\{\nu_j = \nu\}$ is known, it is possible to compute the probability $D(\delta) = P\{f_{i,j} - r_{i,j} \leqslant \delta\}$ as $P\left\{ \left\lceil \frac{\nu_j}{Q^s} \right\rceil T^s \leqslant \delta \right\}$.

As shown in the original paper, Equations (5.2) can be used to compute the state transition probabilities

$$P\{\nu_{j+1} = \nu | \nu_j = x\} = \sum_{h=1}^{\infty} U(\nu - \max\{0, x - hQ^s\})T(h) \tag{5.3}$$

which can be written as $\pi(j+1) = M\pi(j)$ where $\pi(j)$ is the vector of state probabilities at step j and M is a properly defined matrix. Then, if $Q^s/T^s > E[U(c)]/(E[T(z)]T^s)$ queueing theory says that a stationary probability vector $\pi = \lim_{j\to\infty} \pi(j)$ exists and can be computed by solving the eigenvector problem $\pi = M\pi$. In previous work, numeric techniques are used to solve such an eigenvector problem and find the stationary probabilities. However, this computation is too expensive to be performed on-line (see Section 5.0.5).

5.0.3 *Conservative Bounds*

This section shows how to compute a conservative bound for $P\{f_j > r_j + \delta\}$ by adapting and extending some known bounds about GI/G/1 queues [27]. Equation (5.2) can be written as follows:

$$\begin{aligned} \nu_0 &= c_0 \\ \nu_{j+1} &= \max\{c_{j+1}, \nu_j - z_j Q^s + c_{j+1}\}. \end{aligned} \tag{5.4}$$

Then, a new random variable $Y_j = c_{j+1} - z_j Q^s$ can be introduced, so that

$$\begin{aligned} \nu_0 &= c_0 \\ \nu_{j+1} &= \max\{c_{j+1}, \nu_j + Y_j\} \end{aligned}$$

$E[c]$ is the expected value of the execution times, and $E[z]$ is the expected value of z

Let $h(y) = P\{Y_j = y\}$ represent the PMF of Y_j; since Y_j is given by the linear combination of two independent variables, $h(y)$ can be computed as follows:

$$\begin{aligned} h(y) = P\{Y_j = y\} &= \sum_{\{c,z|c-zQ^s=y\}} P\{c_j = c \wedge z_j = z\} \\ &= \sum_{z=1}^{\bar{z}} P\{c_j = y + zQ^s \wedge z_j = z\} \\ &= \sum_{z=1}^{\bar{z}} P\{c_j = y + zQ^s\} P\{z_j = z\} \\ &= \sum_{z=1}^{\bar{z}} U(y + zQ^s)T(z) \end{aligned} \tag{5.5}$$

where $\bar{z}$ is the maximum value of z. The problem with the computation of this PMF is that $U(c)$ is not known for values of the argument greater than $\bar{C}$. Hence, it is possible to obtain a bound for this function by truncating the sum in Equation 5.5 to values of z which lead to $y + zQ^s \leqslant \overline{C}$. The resulting truncated version $\tilde{h}(y)$ of the PMF of Y can be computed as:

$$\tilde{h}(y) = \begin{cases} h(y) & \text{for } y \leqslant \underline{y} \\ \sum_{z=1}^{\lfloor \frac{\bar{C}-y}{Q^s} \rfloor} U(y + zQ^s)T(z) & \text{for } \underline{y} \leqslant y \leqslant \overline{y} \end{cases}$$

where $\overline{y} = \bar{C} - Q^s$ and $\underline{y} = \bar{C} - \bar{z}Q^s$. In plain words, $\tilde{h}(y)$ is derived from $h(y)$ by padding with zeros the function $U(c)$ in Equation (5.5) for the values of the argument for which there is no knowledge. As a result, the values of $\tilde{h}(y)$ for $y \leqslant \overline{y}$ do not sum to 1, and the missing probabilities are accumulated in unknown values larger than $\overline{y}$: for $y > \overline{y}$, function $\tilde{h}(y)$ is unknown, but it is possible to compute $\sum_{y=\overline{y}+1}^{\infty} \tilde{h}(y)$.

Lemma 1. *The $\tilde{h}()$ function has the following properties:*

- *for $y \leqslant \underline{y}$, $P\{Y_j \leqslant y\} = \sum_{k=-\infty}^{y} \tilde{h}(k)$*
- *for $\underline{y} < y \leqslant \overline{y}$, $P\{Y_j \leqslant y\} \geqslant \sum_{k=-\infty}^{y} \tilde{h}(k)$*
- *for $y > \overline{y}$, $\sum_{k=-\infty}^{\overline{y}} \tilde{h}(k) \leqslant P\{Y_j \leqslant y\} \leqslant 1$; $\epsilon_y = 1 - \sum_{k=-\infty}^{\overline{y}} \tilde{h}(k)$ is the size of the interval in which $P\{Y_j \leqslant y\}$ (or $P\{Y_i > y\}$) can range (with $P\{Y_j \leqslant y\} \in (1-\epsilon_y, 1)$ and $P\{Y_i > y\} \in (0, \epsilon_y)$).*

In essence, a random variable Y_j' associated with the PMF $\tilde{h}(y)$ is a conservative approximation for: $Y_j' \succeq Y_j$, according to Definition 1. As discussed in the next section, Y_i quantifies the possible load changes on the system. A positive Y_i value indicates that the system load is increasing, while negative values indicate that it is decreasing. Thereby, a conservative approximation for Y_j as proposed in the previous lemma can be used to carry out a a conservative analysis on the probability of respecting the deadline.

5.0.3.1 *Main Result*

Theorem 1. *Let* $T \in \mathbb{N}$ *be a positive integer. If there exists a real constant* $\gamma \in \mathbb{R}$ *with* $\gamma > 1$ *such that*

$$\sum_{y=-\infty}^{\overline{y}} \gamma^y \tilde{h}(y) + \gamma^T \epsilon_y < 1 \tag{5.6}$$

then

$$\forall \delta \leqslant \left\lceil \frac{T}{Q^s} \right\rceil T^s, P\{f_j \leqslant r_j + \delta\} \geqslant 1 - (\sum_{c=0}^{\bar{C}} U(c)\gamma^{-(\lfloor \frac{\delta}{T^s} \rfloor Q^s - c)} + \epsilon_c)$$

The constant T is the maximum value of ν_j, for which PMF bound is considered to be useful. This constant is expressed in amount of execution time to be executed (like ν_j). Therefore, only deadlines smaller than $\left\lceil \frac{T}{Q^s} \right\rceil T^s$ can be analysed. T can be chosen very large and it does not make practical sense to analyse values of ν_j larger than T, because they would result in deadlines missed by a very large amount. γ is computed, for a given T, considering the shape of the distribution $\tilde{h}(y)$ (its intuitive meaning will be clarified in the next section).

Note that Theorem 1 also states that it is possible to compute a bound for $P\{f_j \leqslant r_j + \delta\}$ (with $\delta \leqslant \left\lceil \frac{T}{Q^s} \right\rceil T^s$) even if the values of $U(c)$ for $c > \bar{C}$ are not known. Indeed, only the knowledge of the cumulative probability $\epsilon_c = P\{c > \bar{C}\}$ is needed. The values of the probability $U(c)$ for specific $c > \bar{C}$ has an impact only on the computations of $P\{f_j \leqslant r_j + \delta\}$ for large values of δ ($\delta \leqslant \left\lceil \frac{T}{Q^s} \right\rceil T^s$). Clearly, the partial knowledge of $U(c)$ will introduce some more pessimism in the analysis even for small values of δ, but this is accounted for in the computation of the bound, as shown in Section 5.0.5 (see Figure 5.12). This means that a complete knowledge of the execution

times distribution is not needed, and that probabilistic guarantees can be provided even if $U(c)$ is not fully known.

5.0.3.2 *Proof of the Result*

To prove Theorem 1 it is useful to introduce a new variable w_i, representing the amount of execution time to be executed *immediately before* the arrival of job J_j (whereas v_j represents the amount of execution time to be executed *immediately after* the arrival of job J_j). By definition, the evolution of w_j can be expressed as:

$$\begin{aligned} w_0 &= 0 \\ w_{j+1} &= \max\{0, w_j - z_j Q^s + c_j\} \end{aligned}$$

or, using Y_j, as

$$\begin{aligned} w_0 &= 0 \\ w_{j+1} &= \max\{0, w_j + Y_j\} \end{aligned}$$

which is known as Lindley recursion [28].

An easy relation between v_j and w_j is stated in the following.

Fact 1. *For all jobs* J_j, $v_j = w_j + c_j$

Proof. By induction on j.
Induction base: for $j = 0$, by definition $v_0 = c_0 = 0 + c_0 = w_0 + c_0$.
Inductive step: $v_{j+1} = \max\{0, v_j - z_j Q^s\} + c_{j+1}$. By inductive hypothesis, this is equal to $\max\{0, w_j + c_j - z_j Q^s\} + c_{j+1}$, and by definition this is $w_{j+1} + c_{j+1}$. □

5.0.4 *Discussion*

Notice that Y_j represents the variation between the amount of execution time yet to be served immediately before (or immediately after) two consecutive arrivals. In other words, if Y_j is negative then the amount of "accumulated" execution time v_j decreases; otherwise it increases.

If $Q^s \geqslant \frac{\overline{C}}{\underline{z}}$ (where $\underline{z}$ is the minimum value of z), then $\overline{C} - Q^s \underline{z} \leqslant 0$, so $P\{c_j - Q^s z_j > 0\} \leqslant \epsilon_c \Rightarrow P\{Y_j > 0\} \leqslant \epsilon_c$. Hence, $\forall \gamma > 1, P\{\sum_{y=-\infty}^{\overline{y}} \gamma^y \tilde{h}(y) > 0\} \leqslant \epsilon_c$ (and if $\epsilon_c = 0$ then Y_j is always < 0 and all the jobs finish before the arrival of the next job).

On the other hand, it can be shown that if $Q^s \leqslant \frac{E[c]}{E[z]}$, then it is not possible to find a value of γ such that $\sum_{y=-\infty}^{\overline{y}} \gamma^y \tilde{h}(y) < 1$ (this is

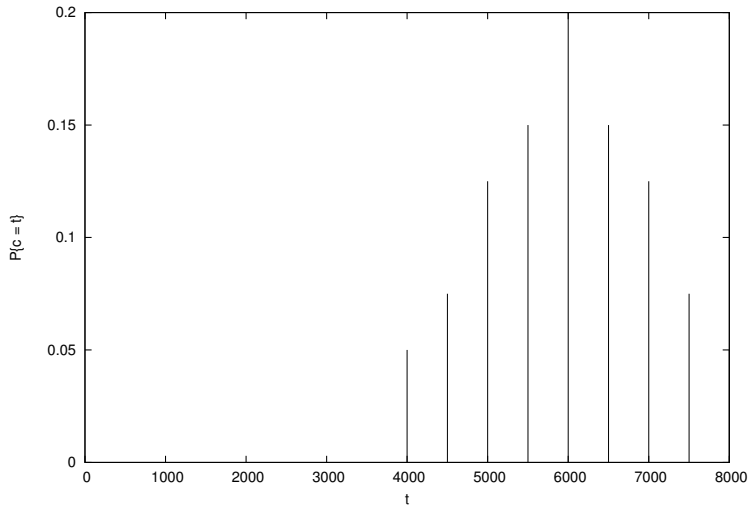

Figure 5.1: PMF of the execution times.

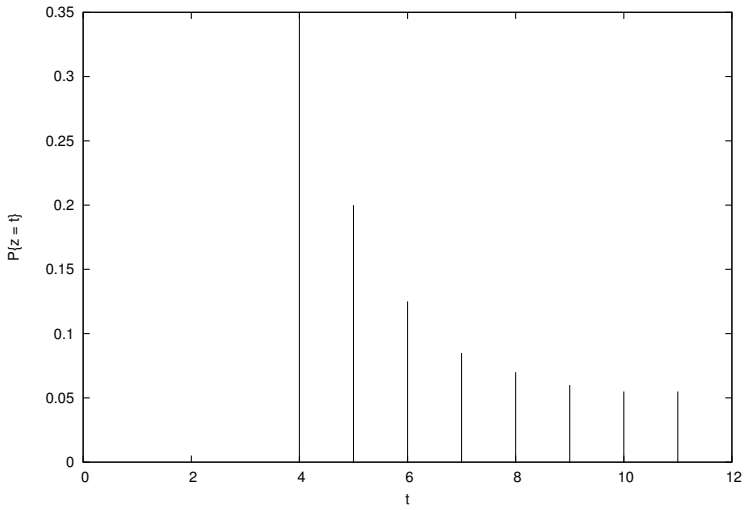

Figure 5.2: PMF of z.

consistent with the fact that a queue with a load $\geqslant 1$ is not stable). As a result, Q^s must be larger than $\frac{E[c]}{E[z]}$.

To better understand how the various scheduling parameters affect the probability distribution of Y_j, consider a simple example with the execution times and inter-arrival times distributed as in Figures 5.1 and 5.2. The resulting PMFs for Y_j have been computed (as explained in the previous section) for different values of Q^s ranging from the minimum possible (1000) to almost the maximum (1900), and the results are displayed in Figure 5.3. From the figure, it can be noticed that increasing Q^s, $h(y)$ is shifted left (meaning that the probability to decrease the amount of remaining computation time is increased).

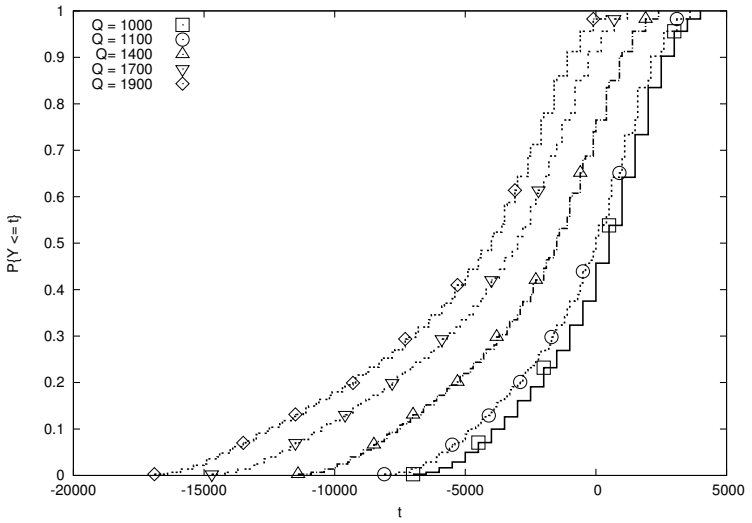

Figure 5.3: CDF of Y_j for various values of Q^s.

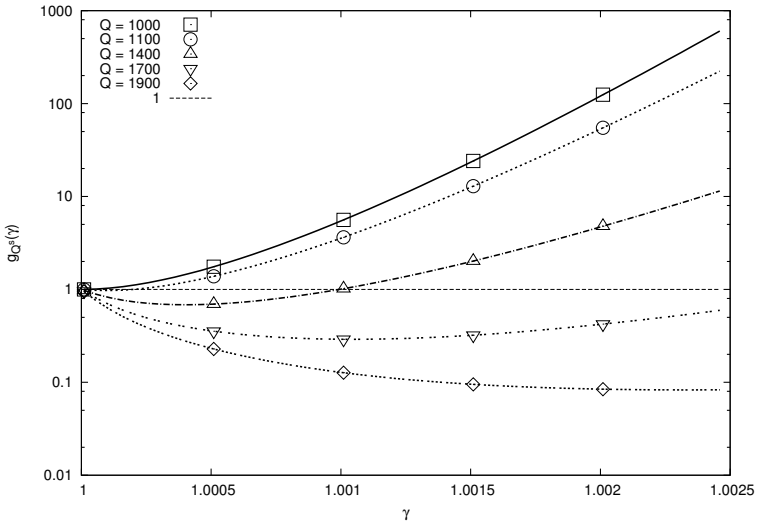

Figure 5.4: $g_{Q^s}(\gamma) = \sum_{t=-\infty}^{\overline{y}} \gamma^t \tilde{h}(t)$ for various values of Q^s.

Remember that to find a pessimistic estimation of the probability to respect a probabilistic deadline, a value of γ satisfying Equation 5.6 must be found. Since in this simplified example we have $\epsilon_c = 0 \Rightarrow \epsilon_y = 0$, the condition is simplified to $\sum_{y=-\infty}^{\overline{y}} \gamma^y \tilde{h}(y) < 1$. To show how the choice of Q^s impacts on the choice of γ, the different functions $g_{Q^s}(\gamma) = \sum_{t=-\infty}^{\overline{y}} \gamma^t \tilde{h}(t)$ for different values of Q^s have been computed, and are displayed in Figure 5.4. It is possible to notice how for $Q^s = 1900$ (close to the hard schedulability condition) the g_{Q^s} is decreasing, and will cross the $g_{1900}(\gamma) = 1$ line only for large values of γ (for $Q^s = 2000$, such a line is never crossed).

5.0.5 *Experimental Results*

The presented analysis technique has been implemented in a set of utilities using fairly portable C code. Dichotomic search is used to find proper values for γ. The resulting library of functions can be used to implement off-line design tools, or on-line admission tests (even in slower CPUs, as it will been shown in this section) and is freely available (downloadable from `http://www.disi.unitn.it/~abeni/gamma-bound.tgz`).

This software has been used to compute the conservative bounds as discussed in this chapter, and to validate them through a comparison with simulations and with the "exact" probability distributions obtained by numerically solving the eigenvector problem [7]. These comparisons have been performed through an extensive set of tests and experiments presented in this section. Such experiments confirmed that the bound is conservative, in perfect accordance with our theoretical expectations.

In a first batch of experiments, the two synthetic PMF distributions for c and z represented in Figures 5.1 and 5.2 have been used. The server period T^s was chosen equal to $T^s = 20000$ µs. Different values for Q^s were considered spanning the interval between 1000 µs (minimum) to 2000 µs (maximum). Some of the bounds obtained for the CDF are shown in Figure 5.5. The bounds have then been compared with the exact CDF and with the empirical distribution obtained from a long simulation run. The workload of the system has been increased by inserting additional real-time tasks (up to utilisation 1) to make sure that the task only receives the reserved computation time (without reclaiming unused bandwidth). Some of the results are reported

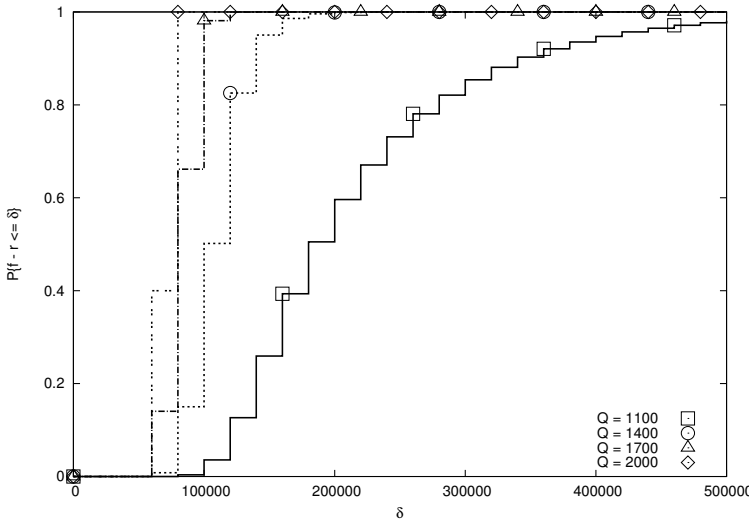

Figure 5.5: Pessimistic bounds of the CDF of the response times for various values of Q^s.

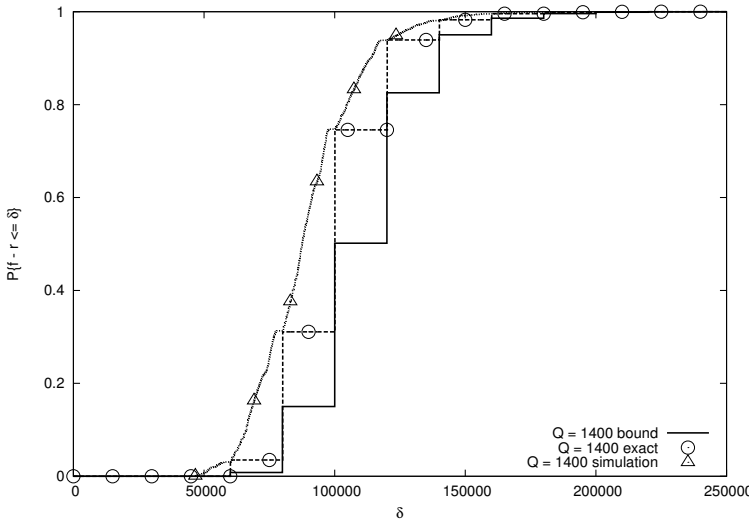

Figure 5.6: Probabilistic deadlines according to various methods, and obtained through simulation. $Q^s = 1400\text{ms}$

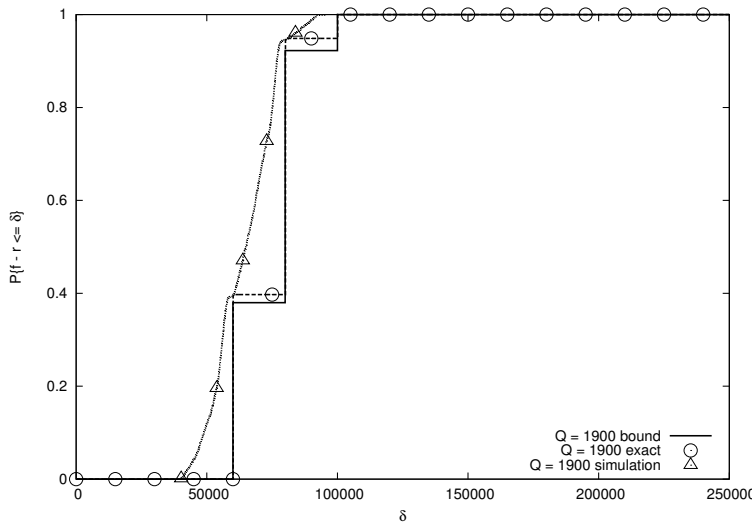

Figure 5.7: Probabilistic deadlines according to various methods, and obtained through simulation. $Q = 1900$

in Figures 5.6, and 5.7, showing that not only is the new bound conservative, but the gap from the exact distribution is relatively narrow.

The worst case response time should be multiple of the server period T^s (since in the worst case the budget Q^s is received at the very end of each server period). However, the empirical distributions obtained from simulation do not exhibit this behaviour (the CDF is not structured as a sequence of step with break points coincident with integer multiples of T^s). This is suggestive of a potential inadequacy of the empirical method for worst case analysis, since it does not seem to capture the worst case patterns.

In a next set of experiments, the performance of the proposed approach has been evaluated by measuring the amount of time needed to compute the bound and comparing it with the amount of time needed to numerically compute the exact CDF. A periodic task with period $P = 200ms$ and a randomly generated PMF of the execution times $U(c)$ (with c varying between $10ms$ and $40ms$), served by a $(30ms, 100ms)$ reservation, has been considered, and the probability $P\{\delta < 100ms\}$ has been computed 100 times (using 100 different PMFs). Each PMF is composed by 300 samples.

The tests have been executed on various systems, characterised by different CPU speeds: a PC based on an Intel Core2 Duo CPU running at 2.60GHz (core2 in the table), a BeagleBoard (an embedded board based on an ARM CPU), a PowerPC 750CX running at 500MHz (ppc in the table), a FoxBoard (an embedded board based on an Etrax

http://www.beagleboard.org

System	Time for Bound	Confidence Interval	Time for Exact Solution	Confidence Interval
core2	0.562ms	0.013ms	807.636ms	12.413ms
ppc	4.775ms	1.317ms	31547.189ms	21.385ms
BeagleBoard	8.817ms	1.436ms	32518.927ms	2.926ms
FoxBoard	2582.697ms	2.106ms	—	—
FLEX	301.9	0.678ms	—	—

Table 5.1: Solution times on different devices.

LX system on chip running at 100MHz), and a FLEX (an embedded board based on a Microchip dsPIC DSC micro-controller). The results are shown in Table 5.1, that presents the average times for finding the bound (including the time needed to compute a correct value for γ, using dichotomic search) and their 95% confidence intervals. The average times needed to compute the exact solution (and their confidence intervals) are also shown for the systems on which it was possible to compute the exact solution.

This experiment shows that using the proposed bound on-line admission control is feasible on almost all of the tested systems, with the only exception is the FoxBoard and the FLEX. Using a resampling of the execution times PMF [12, 29] to reduce its size to 15 samples, the average computation times for the bound are reduced to 135.901ms with a 95% confidence interval of 0.548ms for the FoxBoard (notice that even with the resampled PMF, the time needed to compute the exact solution on a FoxBoard is quite large - more than 10 seconds), and to 22.1ms with a 95% confidence interval of 7.464ms for the FLEX.

Other similar tests have been repeated, with different kinds of tasks, and consistently reported a speedup of at least 400 times.

In order to check the bound on a more realistic example, a video player has been instrumented, measuring the execution times PMF represented in Figure 5.8. Since the video is 25fps (frames per second), the player is modelled as a periodic task with period $P = 40ms$. The proposed technique has been used to compute the CDF of the response times when the player is scheduled by using a $(6ms, 20ms)$ reservation (hence, T^s is half of the task period, and $T(z)$ is a delta function with $T(2) = 1$ and $T(z) = 0$ for $z \neq 2$). The results are reported in Figure 5.9, along with the exact CDF (obtained by numer-

http://foxlx.acmesystems.it
http://www.evidence.eu.com/content/view/114/204

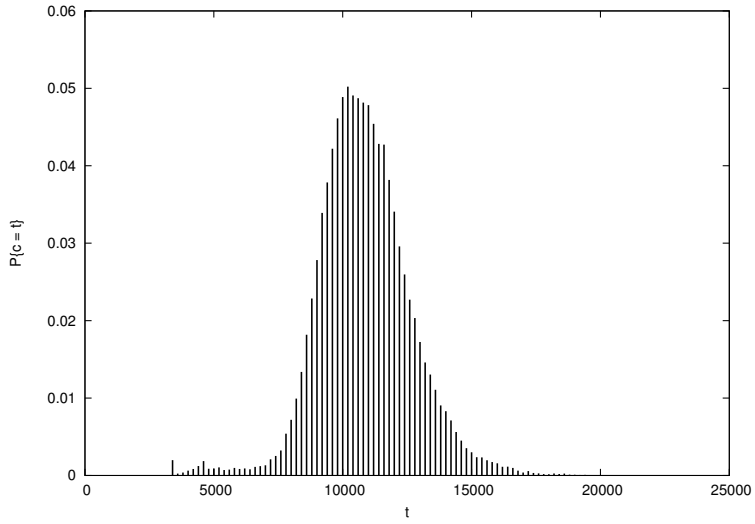

Figure 5.8: $U(c)$ for a video player.

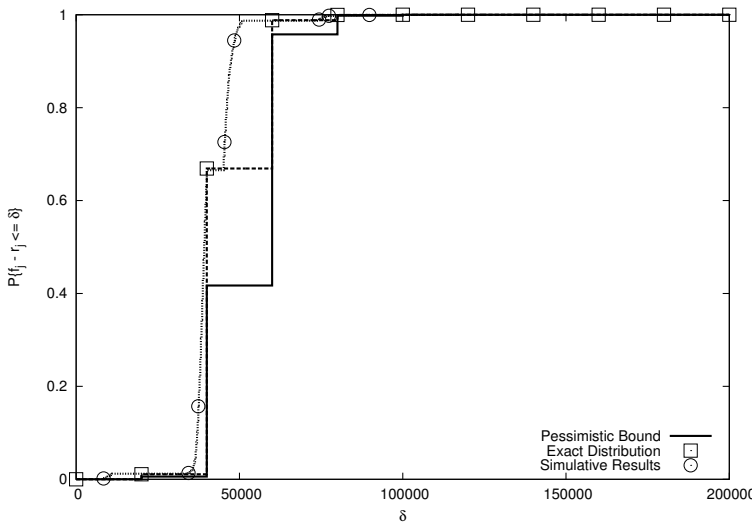

Figure 5.9: Probabilistic deadlines according to various methods, and obtained through simulation.

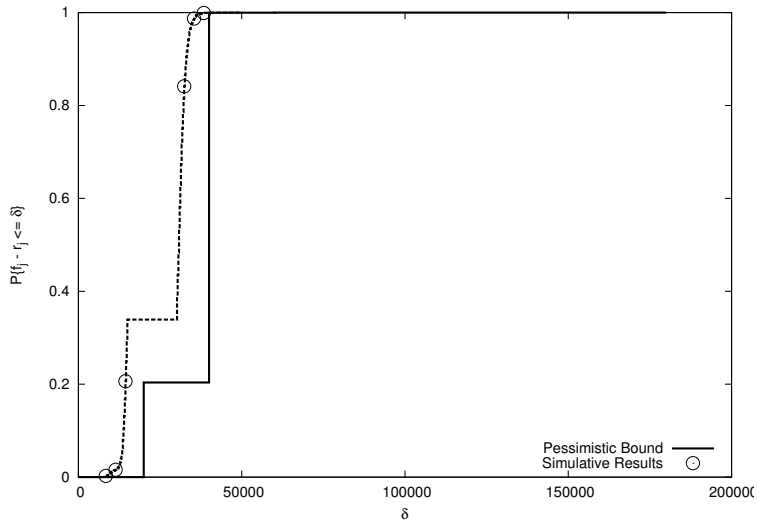

Figure 5.10: Probabilistic deadlines bound vs simulation results. $Q = 10ms$

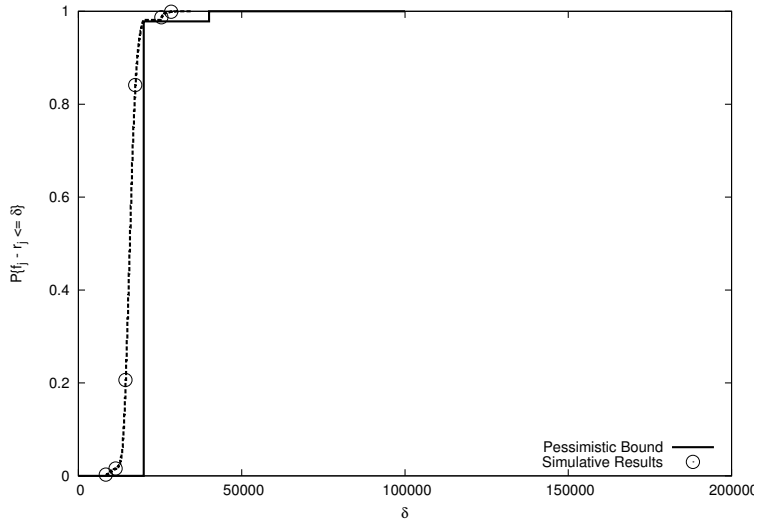

Figure 5.11: Probabilistic deadlines bound vs simulation results. $Q = 15ms$

ically solving the eigenvector problem) and with some empirical distribution, constructed from simulation as discussed in the previous experiment (additional real-time tasks have been inserted to increase the utilisation up to 1). As for the previous example, the bound results correctly conservative (the CDF estimated with the bound remains below the exact one) and the gap from the actual CDF is acceptable. Once again, the CPU time required to compute the bound is between two and three orders of magnitude smaller than from the exact solution. Figures 5.10 and 5.11 compare the conservative bounds with empirical distributions for different values of Q^s, confirming the previous findings.

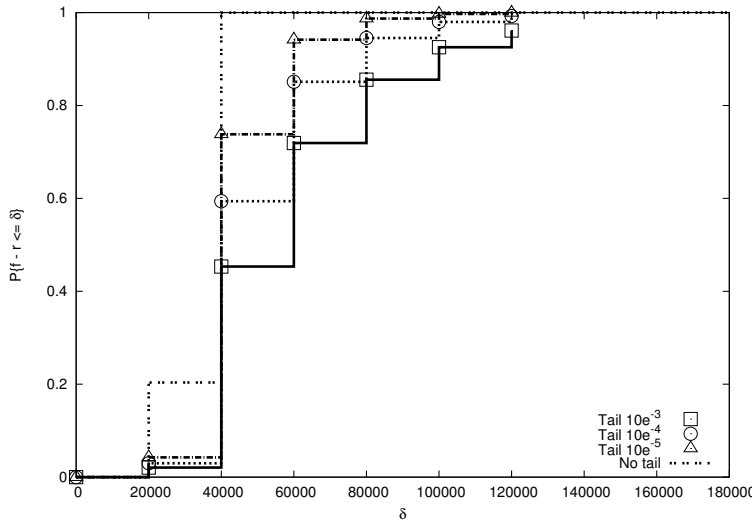

Figure 5.12: Probabilistic deadlines for $Q^s = 10ms$ and various values of ϵ_c.

Figure 5.12 shows the effects of an unknown tail in the PMF of the execution times, by comparing the bound obtained with $Q^s = 10ms$ assuming a complete knowledge of the PMF with the bounds obtained (for the same value of Q^s) assuming $\epsilon_c = 10^{-3}, 10^{-4}, 10^{-5}$. Note that when $\epsilon_c > 0$ the probabilistic deadlines are very conservative but this is the price to be paid to tolerate a partial knowledge of the execution times.

After that, the impact of the server period T^s on the probabilistic deadlines has been evaluated by computing the bound for different values of the server period. T^s affects the final results by changing the degree of pessimism in the generation of $T(z)$ (see Equation 5.1) and by changing the granularity of the CPU time allocation (and hence the size of the steps in the PMF of the response times). For example, consider the inter-arrival times distributed according to the PMF $I(t)$ described in Figure 5.13. The CDFs of z, for various values of T^s (ranging from 1000 to 20000) are displayed in Figure 5.14. The CDF of the response times obtained with a uniform execution time (average 21000) and $Q^s = 0.6T^s$ are shown in Figure 5.15.

Finally, the proposed bounds have been applied to a real-world application: a video tracking task with the execution times shown in Figure 5.16 and period $40ms$. Notice that in this application, $\bar{c} = 53ms$ is larger than the period, so it is not possible to schedule the periodic task without any deadline miss. According to the application's requirements, at least 80% of the deadlines have to be respected, and by applying the proposed analysis with $T^s = 10ms$, it has been possible to verify that with $Q^s = 8.2ms$ such a requirement is respected.

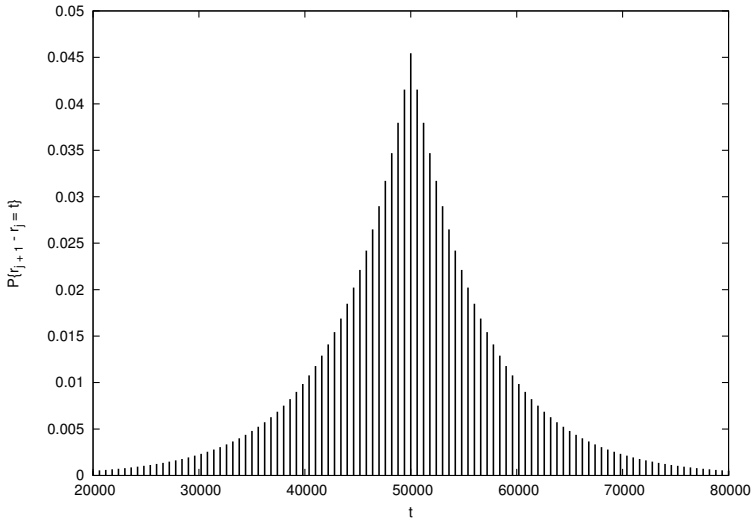

Figure 5.13: $I(t)$ for an example task.

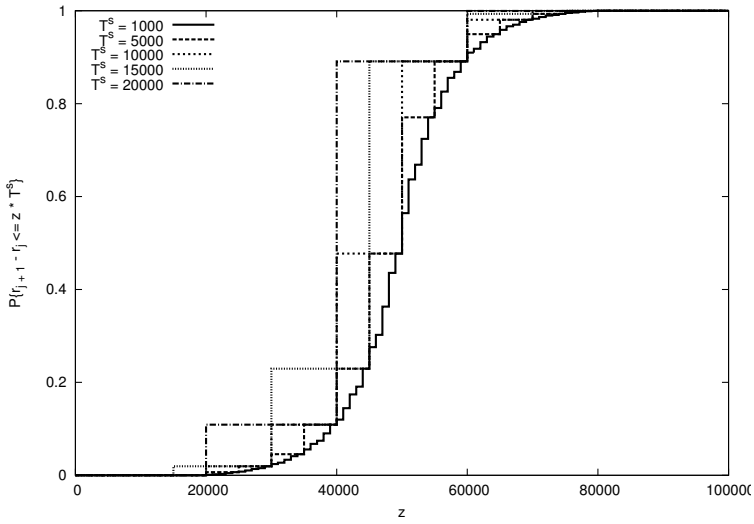

Figure 5.14: CDF of z obtained from $I(t)$ described in Figure 5.13 and $T^s = \{1000, 5000, 10000, 15000, 20000\}$.

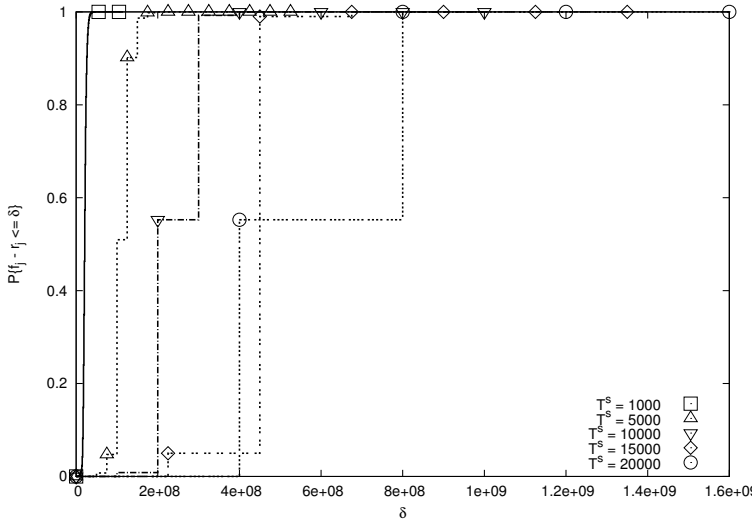

Figure 5.15: Probabilistic deadlines for different values of T^s.

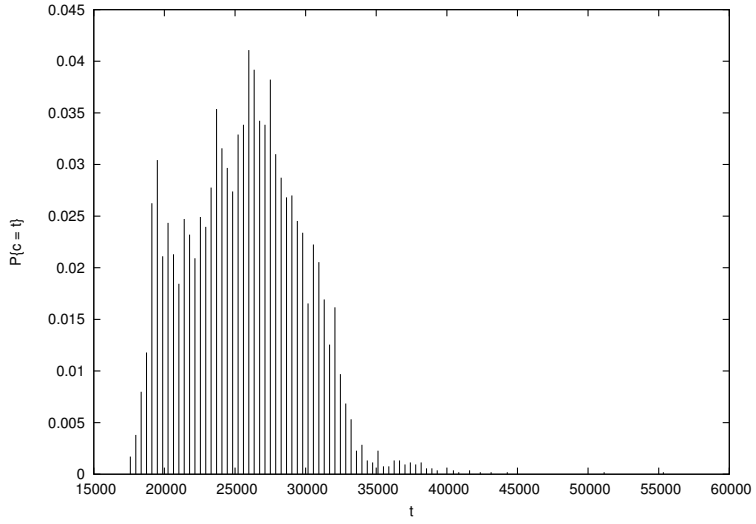

Figure 5.16: $U(c)$ for a video tracking task.

6

ANALYTICAL AND NUMERIC TECHNIQUES FOR PROBABILISTIC GUARANTEES OF SOFT REAL–TIME SYSTEMS USING QUASI–BIRTH–DEATH PROCESSES

In view of the temporal isolation property, each task is guaranteed a minimum share of the processor Q_i^s/T_i^s independently of the behaviour of the other tasks. As a consequence, it is possible to carry out a conservative analysis leading to the computation of a lower bound of the probability of respecting a deadline assuming that the task always receives this minimum (as long as Condition (4.1) is respected). The advantage is that the the behaviour of each task can be studied in isolation. Therefore, we can remove the subscript i meaning that the analysis refers to one specific task.

In this setting, our problem is formulated as follows.

Problem 1. *Given a periodic real–time task with a stochastic computation time characterised by a PMF* $U(c)$*, find conditions on the reservation parameters* (Q^s, T^s) *such that the task respects the probabilistic deadline* (D, p)*.*

A few remarks are in order. First of all, we look for analytical conditions, which can be inverted and offer easy solution for the problem of system design. Second, in order to be safely utilisable, such conditions have to be *sufficient* (although necessity is certainly a desirable additional requirement).

6.1 STOCHASTIC MODEL

6.1.1 *A resource reservation as a Markov Chain*

In the following, we will denote by $F_U(c) = \sum_{h=c_{min}}^{c} U(h)$ the Cumulative Distribution Function (CDF) of the execution time. To simplify the analysis we will also assume that the server period T^s is chosen as an integer sub-multiple of the activation period P: $P = NT^s$. Other choices are certainly possible but make little practical sense.

Let d_k^s denote the latest scheduling deadline used for the execution of job J_k and introduce the symbol $\delta_k = d_k^s - r_k$. The quantity d_k^s

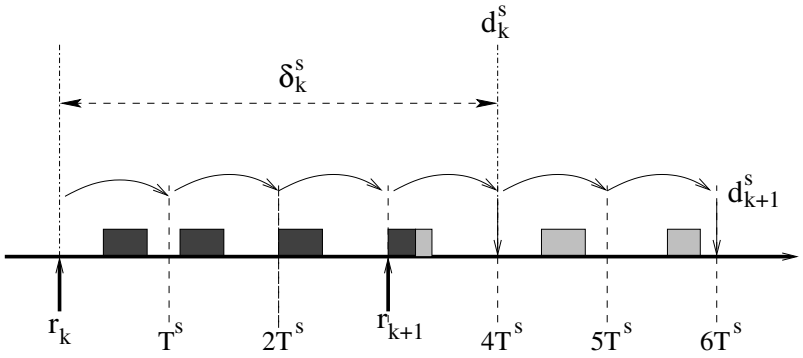

Figure 6.1: Example schedule of a task by a CBS. The two colours denote different jobs.

is an upper bound for the finishing time of the job. Hence, δ_k is an upper bound for the job response time.

Example 1. *Consider the schedule in Figure 6.1. The schedule in the figure considers two adjacent jobs starting at* r_k *and* r_{k+1} *and the reservation period is chosen as one third of the task period. Jok* J_k*, in this case finishes beyond the deadline (which in our periodic model is* r_{k+1}*). More precisely, the last reservation period that it uses (in which its finishing time lies) is upper-limited by the scheduling deadline* d_k^s.

The quantity δ_k takes on values in a discrete set: the integer multiples of T_s and the probability p of meeting the deadline is lower bounded by $P\{\delta_k \leqslant D\}$.

As discussed in previous work [7], we can express the dynamic evolution of δ_k using the following stochastic model:

$$\begin{aligned} \nu_0 &= c_0 \\ \nu_{k+1} &= \max\{0, \nu_k - NQ^s\} + c_{k+1} \\ \delta_k &= \left\lceil \frac{\nu_k}{Q^s} \right\rceil T^s \end{aligned} \tag{6.1}$$

In this case ν_k is a non-measurable variable representing the amount of backlogged execution time that has to be served by the CBS scheduler when a new job arrives.

Since the process $\mathcal{U}$ modelling the sequence c_k of the computation time is assumed a discrete valued and IID random process, the model in Equation (6.1) represents a discrete–time Markov Chain (DTMC) that we define $\mathcal{M}_0$, where the states are determined by the possible values of ν_k and the the transition probabilities by the PMF of the computation time $U(c)$.

This model permits a fine-grained modelling of the behaviour of the reservation, which can be difficult to treat. One possible simplification is to collapse into a single state all the states for which $\delta_k \leqslant D = NT^s$, which correspond to the values of v_k such that $v_k \leqslant NQ^s$. In the modified DTMC $\mathcal{M}$, the state S is defined as

$$S = \begin{cases} 0 & \text{if } v_k \leqslant NQ^s \\ i & \text{if } v_k = NQ^s + i \end{cases}.$$

By using Equation (6.1), the transition probabilities for this DTMC can be written as follows:

$$\begin{aligned} p_{i,j} &= \begin{cases} P\{v_{k+1} \leqslant NQ^s | v_k = i + NQs\}, \text{ if } j = 0 \\ P\{v_{k+1} = j + NQ^s | v_k \leqslant NQ^s\}, \text{ if } i = 0, j \neq 0 \\ P\{v_{k+1} = NQ^s + j | v_k = i + NQs\}, \text{ if } i \neq 0, j \neq 0 \end{cases} \\ &= \begin{cases} P\{c_k \leqslant NQ^s - i\} = F_U(NQ^s - i), \text{ if } j = 0 \\ P\{c_k = j + NQ^s\} = U(j + NQ^s), \text{ if } i = 0, j \neq 0 \\ P\{c_k = NQ^s + j - i\} = U(j - i + NQ^s), \text{ if } i \neq 0, j \neq 0. \end{cases} \end{aligned}.$$

Let $\tilde{\pi}_k$ be the (infinite) vector where the i^{th} element represent the probability associated with the i^{th} state of the DTMC $\mathcal{M}$ after k step of evolution starting from an initial probability vector $\tilde{\pi}_0$. The recursive equation for the evolution of $\tilde{\pi}_k$ is $\tilde{\pi}_{k+1} = \tilde{\pi}_k P$. The objective of our analysis can now be stated in more precise terms: *we aim for the computation of a lower bound for the first element of the steady state probability vector* $\tilde{\pi} = \lim_{k\to\infty} \pi_k$. As long as we are not interested in the distribution of δ_k inside the region $\delta_k \leqslant NQ^s$, collapsing into one state all the values of v_k smaller than NQ^s does not introduce any error because such states do not have influence on the next state ($\max\{0, v_k - NQ^s\} = 0$ in Equation (6.1)).

The probability matrix P resulting from the computation above has the structure in Figure 6.2, where

$$\begin{aligned} a_{n-h} &= p_{i,i+h-2} = U(h - 2 + NQ^s) \\ b_i &= p_{i,0} = F_U(NQ^s - i), \end{aligned}$$

and n is the maximum integer such that $U(NQ^s + h - 2) = 0$ for all $h > n$, and H is the minimum integer such that $U(NQ^s + h - 2) = 0$ for all $h < H$. This structure is recursive: from row $H - 1$ onwards, each row is obtained by shifting the previous one to the right and inserting a 0 in the first position. We now introduce a useful notation for sub-matrices.

Definition 2. *Let* $P = (p_{i,j})$ *be a matrix whose elements are* $p_{i,j}$. *Let* $\alpha = \{i_1, i_2, \ldots, i_n\}$ $\beta = \{j_1, j_2, \ldots, j_m\}$ *two ordered set of indexes. The sub-matrix*

$$P = \begin{bmatrix}
b_0 & a_{n-3} & \dots & a_0 & 0 & \dots & & & & \\
b_1 & a_{n-2} & a_{n-3} & \dots & a_0 & \dots & & & & \\
b_2 & a_{n-1} & a_{n-2} & a_{n-3} & \dots & a_0 & \dots & & & \\
\dots & \dots & \dots & \dots & \dots & \dots & & & & \\
b_{H-1} & a_{n+H-4} & \dots & a_{n-2} & a_{n-3} & \dots & a_0 & \dots & & \\
a_{n+H-2} & a_{n+H-3} & a_{n+H-4} & \dots & a_{n-2} & a_{n-3} & \dots & a_0 & \dots & \\
0 & a_{n+H-2} & a_{n+H-3} & a_{h+H-4} & \dots & a_{n-2} & a_{n-3} & \dots & a_0 & \dots \\
\vdots & \vdots & \vdots & \ddots & & & & & &
\end{bmatrix},$$

Figure 6.2: Structure of the transition matrix

$P_{[\alpha,\beta]}$ *is a matrix whose elements are* p_{i_h,j_t} *for all* $h \in [1, n]$ $t \in [1, m]$ *Likewise, if* π *is a vector, by* $\pi_{[\alpha]}$ *we will denote the sub-vector whose elements are* π_{i_h} *for all* $h \in [1, n]$.

From the properties of our transition matrix we can prove the following result [30].

Theorem 2. *Let* H *the minimum integer such that* $U(NQ^s + h - 2) = 0$ *for all* $h < H$. *Let* F *be defined as* $\max\{n-2, H\}$. *Define* $\alpha(i, F)$ *the set* $\{i, \dots, i + F - 1\}$ *and* $\beta(j, F)$ *the set* $\{j, \dots, j + F - 1\}$. *The transition matrix* P *is block-tri-diagonal with the following structure:*

$$\begin{bmatrix}
C & A_0 & 0 & 0 & \dots \\
A_2 & A_1 & A_0 & 0 & \dots \\
0 & A_2 & A_1 & A_0 & \dots \\
0 & 0 & A_2 & A_1 & \dots \\
\vdots & \vdots & \vdots & \vdots & \ddots
\end{bmatrix} \tag{6.2}$$

where $A_0 = P_{[\alpha(F,F),\beta(0,H)]}$, $A_2 = P_{[\alpha(0,F),\beta(F,F)]}$, $A_1 = P_{[\alpha(F,F),\beta(F,F)]}$, $C = P_{[\alpha(0,F),\beta(0,F)]}$, *are square matrices of order* H.

This structure is known in the literature as a Quasi-Birth-Death Process (see appendix) and lends itself to an efficient numeric solution.

6.1.2 *Numeric Solution*

As long as any element in the diagonal is nonnull the QBDP with transition matrix in Figure 6.2 is aperiodic. Under reasonable conditions on the elements of the matrix, it is also positive recurrent and irreducible. In this case the process has a unique steady state solution for the probability $\tilde{\pi}$ given by the solution of the infinite system of linear equations $\tilde{\pi} = \tilde{\pi} \cdot P, \tilde{\pi} \cdot \mathbf{1} = 1$, where $\tilde{\pi} \cdot \mathbf{1}$ denotes the scalar product between $\tilde{\pi}$ and an infinite vector consisting of unitary elements and P has the structure in Equation (6.2). Let H be the order of the diagonal block A_0, A_1, A_2, C. We partition the vector $\tilde{\pi}$ into sub-vectors $\tilde{\pi}_{[\alpha(hH,H)]}$ of length H (see Definition 2), where $\alpha(hH, H)$ is the

index set $\{hH, \ldots, hH + H - 1\}$. For simplicity, we will denote $\tilde{\pi}_{[\alpha(hH,H)]}$ by $\tilde{\pi}^{(h)}$.

The computation of the stationary probability can be set up in the following recursive way:

$$\begin{aligned} \tilde{\pi}^{(0)}(C - I) + \tilde{\pi}^{(1)}A_2 &= 0 \\ \tilde{\pi}^{(h-1)}A_0 + \tilde{\pi}^{(h)}(A_1 - I) + \tilde{\pi}^{(h+1)}A_2 &= 0 \text{ for } h \geqslant 1 \\ \sum_{h \geqslant 0} \tilde{\pi}^{(h)} \cdot \mathbf{1} &= 1, \end{aligned} \tag{6.3}$$

where $\tilde{\pi}^{(h)} \cdot \mathbf{1}$ denotes the scalar product between the sub-vector $\tilde{\pi}^{(h)}$ and a vector consisting the repetition of H unitary elements. A property of the QBDP is that this recursion can be solved as shown in the following result:

Theorem 3. *[31] If the QBD is positive recurrent, then there exists a non-negative matrix R of order H such that: 1) $\tilde{\pi}^{(h)}$ can be computed as $\tilde{\pi}^{(h+1)} = \tilde{\pi}^{(h)}R$ for $h \geqslant 0$, which is equivalent to*

$$\tilde{\pi}^{(h)} = \tilde{\pi}^{(0)}R^h \tag{6.4}$$

2) $\tilde{\pi}^{(0)}$ is given by

$$\begin{aligned} \tilde{\pi}^{(0)} &= \begin{bmatrix} 0 & 0 & \ldots & 0 & 1 \end{bmatrix} M^+ \\ M &= \begin{bmatrix} & & & & \\ & C + RA_2 - I & & (I-R)^{-1} & \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} \\ & & & & \end{bmatrix} \end{aligned} \tag{6.5}$$

where M^+ denotes the Penrose pseudo-inverse and I denotes the identity matrix. Matrix M can be found by solving the following non-linear matrix equation:

$$R = A_0 + RA_1 + R^2A_2. \tag{6.6}$$

This result offers a straight path to the computation of $\tilde{\pi}$ once we have a solution for Equation (6.6) (and hence to the computation of the probability of meeting the deadline).

The solution of Equation (6.6) can be computed by several algorithms available in the literature [32]. The one that we used is the simple iterative algorithm reported in Algorithm 1. The algorithm can be easily implemented in Matlab or in C using a numeric library (for our implementation we used the Meshach library). Even using a very small tolerance ($\epsilon = 10^{-5}$) the algorithm converges in a few iterations (in the order of 10).

http://www.cs.uiowa.edu/~dstewart/meschach/meschach.html

Algorithm 1 R computation

```
1: procedure ComputeR(A_0, A_1, A_2, ε)
2:     R_t = [0]
3:     R_new = A_0 + R_t A_1 + R_t^2 A_2;
4:     while ||R_t − R_new||_∞ > ε do
5:         R_t = R_new
6:         R_new = A_0 + R_t A_1 + R_t^2 A_2;
7:     end while
8:     R = R_new
9: end procedure
```

6.2 AN ANALYTICAL BOUND

6.2.1 *A conservative simplification*

While the numeric algorithm presented in the previous section is generally applicable, the analytical bound requires a simplification of the DTMC. To be applicable in our context, such simplification has to be "conservative" in the computation of the probabilities. The notion of conservative approximation that we shall adopt here relies on the following order relation:

Definition 3. *Given two random variables* X *and* Y*, with CDFs* $F_x(x)$ *and* $F_y(y)$*,* $X \succeq Y$ *iff* $\forall x\ F_x(x) \leqslant F_y(x)$.

When this definition is applied to the evolution of the variable δ_k in Equation (6.1), it plainly means that in the modified system the low values of the δ_k will have a greater probability and so will be the probability of the first element of the probability vector (associated with the deadline).

A conservative approximation can be found by replacing c_k with a new variable c'_k whose distribution is given by:

$$U_\Delta(c') = \begin{cases} 0 & \text{if } c' \mod \Delta \neq 0 \\ \sum_{c=(k-1)\Delta+1}^{k\Delta} U(c') & \text{otherwise,} \end{cases} \tag{6.7}$$

where Δ is chosen as an integer submultiple of Q^s. By using this new variable in Equation (6.1), we can apply the same line of reasoning of Diaz et al. [12] and show that the resulting system is conservative approximation of our system in the sense described above. In particular, we obtain a DTMC whose transition matrix has again the structure in Figure 6.2. In the following we will implicitly refer to a system where the conservative approximation technique just described has been applied for some value of the *scaling factor* Δ.

It can be shown that the model thus obtained is a generalization of the model used in previous work [33, 30], which can be recovered by setting $\Delta = Q^s$.

6.2.2 *Computation of the bound*

As discussed earlier, the steady state probability of meeting the deadline can be found by computing the first element $\tilde{\pi}^{(0)}$ of the $\tilde{\pi}$ that solves the equation $\tilde{\pi} = \tilde{\pi}P$, where P is the infinite transition matrix in Figure 6.2 associated with the DTMC $\mathcal{M}$. Let us consider a new DTMC whose transition matrix is given by:

$$P' = \begin{bmatrix} b_0 & a_{n-3} & a_{n-4} & \dots & a_1 & a_0 & 0 & \dots \\ b_1 & a_{n-2} & a_{n-3} & \dots & a_2 & a_1 & a_0 & \dots \\ 0 & a'_{n-1} & a_{n-2} & \dots & a_3 & a_2 & a_1 & \dots \\ 0 & 0 & a'_{n-1} & a_{n-2} & \dots & a_3 & a_2 & \dots \\ 0 & 0 & 0 & a'_{n-1} & a_{n-2} & \dots & a_3 & \dots \\ \vdots & \vdots & \vdots & \vdots & \ddots & & \ddots & \ddots \end{bmatrix}, \tag{6.8}$$

and $a'_{n-1} = b_1 = a_{n-1} + a_n + \ldots + a_{N+H}$. The underlying idea is very simple. Consider the DTMC associated with matrix P. The terms on the left of the diagonal are transition probabilities toward states with a smaller delay than the current one. By using P' we lump together all these transitions to the state immediately on the left of the current one. For instance, if the current state corresponds to 4 server periods of delay, its only enabled transition to the left will be to the state associated with delay 3. The effect of deleting the transition toward states associated with smaller delays is to slow down the convergence toward small delays, thus decreasing the steady state probability of these states.

Let π represent the steady state probability of this system. We can easily show the following:

Lemma 2. *Let Γ be a random variable representing the state of the DTMC evolving with transition matrix P and Γ' be a random variable describing the state of the DTMC associated with the transition matrix P'. If both DTMC are irreducible and aperiodic, then at the steady state Γ' is a conservative approximation of Γ: $\Gamma' \succeq \Gamma$, according to Definition 3. Therefore, for the first element of the steady state probability, we have $\tilde{\pi}^{(0)} \geqslant \pi^{(0)}$.*

Proof. We prove the Lemma by showing that if for some k we have

$$\forall h, \sum_{i=0}^{h} \tilde{\pi}_k^{(i)} \geqslant \sum_{i=0}^{h} \pi_k^{(i)} \tag{6.9}$$

then at step $k+1$ the same holds:

$$\forall h, \sum_{i=0}^{h} \tilde{\pi}_{k+1}^{(i)} \geqslant \sum_{i=0}^{h} \pi_{k+1}^{(i)}. \tag{6.10}$$

$$P'A = \begin{bmatrix} b_0 & b_0 + a_{n-3} & b_0 + \sum_{i=n-4}^{n-3} a_i & \dots & b_0 + \sum_{i=1}^{n-3} a_i & 1 & 1 & \dots \\ b_1 & b_0 & b_0 + a_{n-3} & \dots & b_0 + \sum_{i=2}^{n-3} a_i & b_0 + \sum_{i=1}^{n-3} a_i & 1 & \dots \\ 0 & b_1 & b_0 & \dots & b_0 + \sum_{i=3}^{n-3} a_i & b_0 + \sum_{i=2}^{n-3} a_i & b_0 + \sum_{i=2}^{n-3} a_i & \dots \\ \vdots & \ddots & \ddots & & \ddots & & & \end{bmatrix}$$

$$PA = \begin{bmatrix} b_0 & b_0 + a_{n-3} & b_0 + \sum_{i=n-4}^{n-3} a_i & \dots & b_0 + \sum_{i=1}^{n-3} a_i & 1 & 1 & \dots \\ b_1 & b_0 & b_0 + a_{n-3} & \dots & b_0 + \sum_{i=2}^{n-3} a_i & b_0 + \sum_{i=1}^{n-3} a_i & 1 & \dots \\ b_2 & b_1 & b_0 & \dots & b_0 + \sum_{i=3}^{n-3} a_i & b_0 + \sum_{i=2}^{n-3} a_i & b_0 + \sum_{i=2}^{n-3} a_i & \dots \\ \vdots & \ddots & \ddots & & \ddots & & & \\ b_{H-1} & b_{H-2} & b_{H-3} & \dots & b_0 + \sum_{i=m}^{n-3} a_i & b_0 + \sum_{i=m-1}^{n-3} a_i & b_0 + \sum_{i=m-2}^{n-3} a_i & \dots \\ b_H & b_{H-1} & b_{H-2} & \dots & b_0 + \sum_{i=m+1}^{n-3} a_i & b_0 + \sum_{i=m}^{n-3} a_i & b_0 + \sum_{i=m-1}^{n-3} a_i & \dots \\ 0 & b_H & b_{H-1} & \dots & b_0 + \sum_{i=m+2}^{n-3} a_i & b_0 + \sum_{i=m+1}^{n-3} a_i & b_0 + \sum_{i=m}^{n-3} a_i & \dots \\ 0 & 0 & b_H & \dots & b_0 + \sum_{i=m+3}^{n-3} a_i & b_0 + \sum_{i=m+2}^{n-3} a_i & b_0 + \sum_{i=m+1}^{n-3} a_i & \dots \\ \vdots & \ddots & \ddots & & \ddots & & & \end{bmatrix}$$

Figure 6.3: $P'A$ and PA matrices, for $b_H = a_{n+H-2}$.

Define $\delta^{(i)} = \pi_k^{(i)} - \tilde{\pi}_k^{(i)}$. Condition (6.9) can be written as:

$$\forall h, \sum_{i=0}^{h} \delta^{(i)} \leqslant 0. \tag{6.11}$$

We first notice that (6.11) can be written as:

$$(\pi_k - \tilde{\pi}_k)A = \delta A \leqslant 0,$$

where

$$A = \begin{bmatrix} 1 & 1 & 1 & \dots & 1 & \dots \\ 0 & 1 & 1 & \dots & 1 & \dots \\ 0 & 0 & 1 & \dots & 1 & \dots \\ \vdots & \vdots & \ddots & \ddots & & \ddots \end{bmatrix},$$

and the $\leqslant$ operator applies element–wise. Likewise, (6.11) can be written in matrix form as:

$$(\pi_{k+1} - \tilde{\pi}_{k+1})A = (\pi_k P' - \tilde{\pi}_k P)A,$$

where $P'A$ and PA are represented in Figure 6.3, for $b_H = a_{n+H-2}$. This

leads to,

$$\begin{aligned}
&\forall h, \sum_{i=0}^{h} \pi_{k+1}^{(i)} = \\
&\quad = \sum_{j=i}^{h+1} \pi_k^{(j)} b_{j-i} + b_0 \sum_{j=0}^{h-1} \pi_k^{(j)} + \sum_{l=0}^{n-3} \left(a_l \sum_{j=0}^{h-n+2+l} \pi_k^{(j)} \right) \\
&\forall h, \sum_{i=0}^{h} \tilde{\pi}_{k+1}^{(i)} = \\
&\quad = \sum_{j=h}^{h+H} \tilde{\pi}_k^{(j)} b_{j-h} + b_0 \sum_{j=0}^{h-1} \tilde{\pi}_k^{(j)} + \sum_{l=0}^{n-3} \left(a_l \sum_{j=0}^{h-n+2+l} \tilde{\pi}_k^{(j)} \right).
\end{aligned} \tag{6.12}$$

Since we want to show that (6.10) holds whenever (6.9) is verified, we can conservatively neglect the elements from $i+2$ to $i+H$ in the first term $\sum_{i=0}^{h} \tilde{\pi}_{k+1}^{(i)}$ in (6.12). Thereby, (6.12) is conveniently written in terms of $\delta^{(i)}$, i.e.,

$$\begin{aligned}
&\forall h, \sum_{i=0}^{h} \left(\pi_{k+1}^{(h)} - \tilde{\pi}_{k+1}^{(h)} \right) \leqslant \\
&\quad \epsilon^{(h)} = \sum_{j=h}^{h+1} \delta^{(j)} b_{j-h} + b_0 \sum_{j=0}^{h-1} \delta^{(j)} + \sum_{l=0}^{n-3} \left(a_l \sum_{j=0}^{h-n+2+l} \delta^{(j)} \right).
\end{aligned} \tag{6.13}$$

The claim is proved if, for every h, (6.9) implies $\epsilon^{(h)} \leqslant 0$. The latter statement, in turn, is true if the following set of linear constraints does not have feasible assignments in $\delta^{(0)}, \delta^{(1)}, \ldots, \delta^{(h)}$:

$$\begin{aligned}
\delta^{(0)} &\leqslant 0 \\
\delta^{(0)} + \delta^{(1)} &\leqslant 0 \\
&\ldots \\
\delta^{(0)} + \delta^{(1)} + \cdots + \delta^{(h+1)} &\leqslant 0 \\
\epsilon^{(h)} &> 0
\end{aligned} \tag{6.14}$$

This equation can be written as:

$$\forall h, \qquad \tilde{A} \begin{bmatrix} \delta^{(0)} \\ \ldots \\ \delta^{(h+1)} \end{bmatrix} \leqslant 0, \quad c^T \begin{bmatrix} \delta^{(0)} \\ \ldots \\ \delta^{(h+1)} \end{bmatrix} > 0 \tag{6.15}$$

where

$$\tilde{A} = \begin{bmatrix} 1 & 0 & 0 & \dots & 0 \\ 1 & 1 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & 1 & 1 & \dots & 1 \end{bmatrix}$$

and

$$c = \begin{bmatrix} 1 \\ \vdots \\ 1 \\ b_0 + \sum_{l=1}^{n-3} a_l \\ \vdots \\ b_0 + \sum_{l=n-4}^{n-3} a_l \\ b_0 + a_{n-3} \\ b_0 \\ b_1 \end{bmatrix},$$

whereas the first $h - n + 2$ are equal to 1. To prove the infeasibility of this assignment we can use Farkas Lemma, which states that the system in Equation (6.15) is unfeasible if and only if the following is feasible:

$$\forall h, \tilde{A}^T y - c = 0,\ y \geqslant 0 \tag{6.16}$$

This condition can be written as

$$\begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 0 & 1 & 1 & \dots & 1 \\ 0 & 0 & 1 & \dots & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & 1 \end{bmatrix} y = \begin{bmatrix} 1 \\ \vdots \\ 1 \\ b_0 + \sum_{l=1}^{n-3} a_l \\ \vdots \\ b_0 + \sum_{l=n-4}^{n-3} a_l \\ b_0 + a_{n-3} \\ b_0 \\ b_1 \end{bmatrix}$$

$$y \geqslant 0$$

which leads to:

$$\begin{aligned}
y^{(h+1)} &= b_1 \\
y^{(h)} &= b_0 - b_1 \\
y^{(h-1)} &= b_0 + a_{n-3} - b_0 + b_1 - b_1 = a_{n-3} \\
y^{(h-2)} &= b_0 + a_{n-3} + a_{n-4} - b_0 + b_1 - b_1 - a_{n-3} = a_{n-4} \\
&\vdots \\
y^{(h-n+2)} &= a_1 \\
y^{(h-n+1)} &= 1 - b_0 - \sum_{l=1}^{n-3} a_l = a_0 \\
y^{(h-n)} &= 1 - b_0 - \sum_{l=0}^{n-3} a_l = 0 \\
y^{(h-n-1)} &= y^{(h-n-2)} = \ldots = 0.
\end{aligned}$$

Since all a_i and b_i are non-negative, the condition of Farkas Lemma is verified if $y^{(h)} = b_0 - b_1 \geqslant 0$, i.e., $b_0 \geqslant b_1$, which is obviously true. This concludes the proof. □

In view of this Lemma, we can concentrate on the system associated to the transition matrix P'. We first notice that the equilibrium condition $\pi P' = \pi$ produces the following:

$$\begin{aligned}
\pi^{(0)} b_0 + \pi^{(1)} a'_{n-1} &= \pi^{(0)}, \\
\pi^{(0)} a_{n-3} + \pi^{(1)} a_{n-2} + \pi^{(2)} a'_{n-1} &= \pi^{(1)}, \\
\pi^{(0)} a_{n-4} + \pi^{(1)} a_{n-3} + \pi^{(2)} a_{n-2} + \pi^{(3)} a'_{n-1} &= \pi^{(2)}, \\
&\ldots \\
\sum_{i=\max(n-2-j,0)}^{n-3} \pi^{(j+i-(n-2))} a_i + \pi^{(j)} a_{n-2} + \pi^{(j+1)} a'_{n-1} &= \pi^{(j)} \\
&\ldots
\end{aligned}$$

We can conveniently rewrite the above condition as the following recursion:

$$\begin{aligned}
\pi^{(1)} &= \sum_{j=1}^{n-2} \alpha_j \pi^0, \\
\pi^{(k)} &= \left(1 + \sum_{j=1}^{n-2} \alpha_j\right) \pi^{(k-1)} - \sum_{j=\max(n-k,1)}^{n-2} \pi^{(k-n+j)} \alpha_j,
\end{aligned} \tag{6.17}$$

where $\alpha_j = \frac{a_{j-1}}{a'_{n-1}} \geqslant 0$. The equalities of (6.17) hold for $\forall k > 1$.

Now we can determine the closed form solution using the following theorem:

Theorem 4. *Consider a QBDP described by the transition probability matrix (6.8), in which both a_0 and a'_{n-1} differ from zero. Assume that the matrix*

$$W = \begin{bmatrix} 0 & 1 & 0 & \dots & 0 & 0 \\ 0 & 0 & 1 & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & 1 & 0 \\ 0 & 0 & 0 & \dots & 0 & 1 \\ 0 & -\alpha_1 & -\alpha_2 & \dots & -\alpha_{n-2} & 1+\sum_{j=1}^{n-2}\alpha_j \end{bmatrix}, \tag{6.18}$$

has distinct eigenvalues. Then, there exists a limiting probability distribution given by

$$\begin{aligned} \pi^{(0)} &= \lim_{k\to+\infty} \pi^{(0)}(k) = \max\{1 - \sum_{j=1}^{n-2}(n-1-j)\frac{a_{j-1}}{a'_{n-1}}, 0\} = \\ &= \max\{1 - \sum_{j=1}^{n-2}(n-1-j)\alpha_j, 0\}, \end{aligned} \tag{6.19}$$

while the generic terms $\pi^{(j)}$, with $j > 0$, are given by (6.17).

The proof of the theorem is deferred to Section 6.2.3. We complete the section with a few remarks.

The first one is on the intuitive meaning of the result just proposed. Consider a DTMC with transition matrix as in Figure 6.2 and assume for simplicity $n = 5$ and $H = 0$. The analytical bound in Theorem 4 is given by:

$$\begin{aligned} \pi^{(0)} &= 1 - 3\alpha_1 - 2\alpha_2 - \alpha_3 \\ &= 1 - 3\frac{a_0}{a_5+a_4} - 2\frac{a_1}{a_5+a_4} - \frac{a_2}{a_5+a_4} \end{aligned}$$

In the computation of the steady state probability $\pi^{(0)}$ we have to consider every possible transition to the right (i.e., increasing the delay) that the system can make. For each of them, we compute the ratio between the probability of taking the transition and the aggregate probability of moving to the left (decreasing the delay). In the final computation each of this ratio has a state proportional to the delay introduced. In our example, a_0 corresponds to three steps to the right and is weighted by the factor 3.

The second remark is on the assumptions of the theorem on the eigenvalues of the matrix W. This assumption is merely technical (it

is instrumental to the proof of the result) and it is not restrictive. In all our examples (both synthetically generated and using data from real applications), it is respected. Artificial examples that violate it could probably be constructed but they are not relevant in practice.

The application of this result to our context can be formalised in the following:

Corollary 5. *Consider a resource reservation used to schedule a periodic task and suppose that the QBDP produced respects the assumption in Theorem 4. Then the probability of respecting the deadline is greater than or equal to:*

$$\pi^{(0)} = 1 - \sum_{j=1}^{n-2} (n-1-j) \frac{U'_\Delta(N+n-j-1)Q_s)}{\sum_{h=0}^{N-1} U'_\Delta(hQ^s)} \tag{6.20}$$

This corollary descends from the following facts: 1) the DTMC described by the transition matrix P in Figure 6.2 is a conservative approximation of the system, 2) Lemma 2 provides an analytically tractable approximation of the DTMC with transition matrix P, 3) Theorem 4 contains the analytical bound.

6.2.3 *Proof of Theorem 4*

The rationale behind this proof of Theorem 4 is the following. First the equilibrium point of the QBDP is expressed as an iterative system. The evolution in the iteration step represents the connection between the probabilities of the different states. Using this representation, we can express all the steady–state probabilities as a function of the probability $\pi^{(0)}$ to stay in first state. Then, $\pi^{(0)}$ is found by imposing that the probabilities sum to 1, coming up with its closed form expression.

Proof. We start noticing that having a_0 and a'_{n-1} different from zero implies that the Markov chain of the QBDP is irreducible and aperiodic. Therefore, it is guaranteed that the probability of the different states converge to a value [34]. Notice, however, that this does not necessarly imply the existence of a steady–distribution (the probability distribution could gradually shift toward increasing values of the state without ever reaching the equilibrium).

Let us first consider the case that the QBDP is also positive recurrent and so it admits indeed a unique steady state distribution. The first step of the proof is then to introduce the following vector:

$$\Pi_0 = \begin{bmatrix} \pi^{(0)} \\ \vdots \\ \pi^{(n-1)} \end{bmatrix},$$

whose dimension is equal to n. It is possible to exploit (6.17) and (6.18) to derive the equilibrium of the QBDP as the dynamic of the following system:

$$\Pi_1 = \begin{bmatrix} \pi^{(1)} \\ \pi^{(2)} \\ \vdots \\ \pi^{(n)} \end{bmatrix} = W\Pi_0, \quad \Pi_j = \begin{bmatrix} \pi^{(j)} \\ \pi^{(j+1)} \\ \vdots \\ \pi^{(n-1+j)} \end{bmatrix} = W\Pi_{j-1} = W^j\Pi_0,$$

from which one has that the sum of the probabilities of the equilibrium of the QBDP is 1 and it is given by

$$\begin{bmatrix} 1 & 0 & 0 & \dots & 0 \end{bmatrix} \sum_{i=0}^{+\infty} \Pi_i = 1. \tag{6.21}$$

The characteristic polynomial of the lower-left *companion form* matrix W reported in (6.18) is simply given by

$$P(\lambda) = \lambda^n - \left(1 + \sum_{j=1}^{n-2} \alpha_j\right) \lambda^{n-1} + \sum_{j=1}^{n-2} \alpha_j \lambda^j, \tag{6.22}$$

from which it is trivially derived that the matrix W has two simple eigenvalues in $\beta_0 = 0$ and $\beta_1 = 1$ and additional $n-2$ eigenvalues β_i. Therefore

$$P(\lambda) = \lambda(\lambda - 1) \prod_{i=2}^{n-1} (\lambda - \beta_i). \tag{6.23}$$

Since each β_i verifies $P(\beta_i) = 0$, the following relation holds

$$\begin{aligned} &\beta_i^{n-1} - \left(1 + \sum_{j=1}^{n-2} \alpha_j\right) \beta_i^{n-2} + \sum_{j=1}^{n-2} \alpha_j \beta_i^{j-1} = 0 \Rightarrow \\ &1 + \sum_{j=1}^{n-2} \alpha_j = \beta_i + \frac{\sum_{j=1}^{n-2} \alpha_j \beta_i^{j-1}}{\beta_i^{n-2}}. \end{aligned} \tag{6.24}$$

Since all the eigenvalues are assumed simple, we can use of the *spectral decomposition* of the matrix W: $W = \sum_{i=0}^{n-1} \beta_i G_i$, where the *spectral projectors* G_i are given by

$$G_i = \frac{V_i L_i}{L_i V_i} = N_i V_i L_i,$$

and L_i and V_i are respectively the left and right eigenvectors associated with the i-th eigenvalue β_i. N_i is the normalization constant needed to satisfy the spectral projectors basic properties, i.e., $G_i G_j = 0$ for $i \neq j$ and $G_i G_i = G_i$. As a consequence,

$$\Pi_1 = W\Pi_0 = \sum_{i=0}^{n-1} \beta_i G_i \Pi_0 = \sum_{i=1}^{n-1} \beta_i N_i V_i L_i \Pi_0,$$

and, in general,

$$\Pi_j = W^j \Pi_0 = \sum_{i=0}^{n-1} \beta_i^j G_i \Pi_0 = \sum_{i=1}^{n-1} \beta_i^j N_i V_i L_i \Pi_0. \tag{6.25}$$

Therefore, by combining (6.25) and (6.21), one has the following relation

$$\sum_{i=1}^{n-1} \sum_{j=0}^{+\infty} \beta_i^j \nu_i^{(0)} N_i L_i \Pi_0 = 1, \tag{6.26}$$

where $\nu_i^{(0)}$ is the first element of the right eigenvector. Given the expression of the matrix W, the left and right eigenvectors associated to β_i, with $i = 1, \dots, n-1$, have the following expression

$$\begin{aligned} L_i &= \begin{bmatrix} 0 & -\frac{\alpha_1}{\beta_i} & \dots & -\frac{\sum_{j=1}^{n-3} \alpha_j \beta_i^{j-1}}{\beta_i^{n-3}} & -\frac{\sum_{j=1}^{n-2} \alpha_j \beta_i^{j-1}}{\beta_i^{n-2}} & 1 \end{bmatrix}, \\ V_i &= \begin{bmatrix} \frac{1}{\beta_i^{n-1}} & \frac{1}{\beta_i^{n-2}} & \dots & \frac{1}{\beta_i^2} & \frac{1}{\beta_i} & 1 \end{bmatrix}^T. \end{aligned} \tag{6.27}$$

In order to prove the theorem, we have to first notice that the existence of the equilibrium implies $|\beta_i| < 1$ for $i = 2, \dots, n-1$. To this end, we rely on the following propositions.

Proposition 1. *The product between the left eigenvector L_i and the initial condition Π_0 is given by*

$$L_i \Pi_0 = \beta_i^{n-2} (\beta_i - 1) \pi^{(0)}.$$

Proof. Consider (6.17) and substitute the constraint given in (6.24) to get

$$\pi^{(n-k)} = \left(\beta_i + \frac{\sum_{j=1}^{n-2} \alpha_j \beta_i^{j-1}}{\beta_i^{n-2}} \right) \pi^{(n-k-1)} - \sum_{j=1}^{n-1-k} \alpha_{j+k-1} \pi^{(j-1)}, \tag{6.28}$$

for $k = 1, \dots, n-2$. Since the explicit form of $L_i \Pi_0$ is

$$L_i \Pi_0 = \pi^{(n-1)} - \sum_{k=1}^{n-2} \frac{\sum_{j=1}^{k} \alpha_j \beta_i^{j-1}}{\beta_i^k} \pi^{(k)}, \tag{6.29}$$

substituting (6.28) in (6.29) for $k = 1$ yields to

$$L_i \Pi_0 = \beta_i \pi^{(n-2)} - \sum_{k=1}^{n-3} \frac{\sum_{j=1}^{k} \alpha_j \beta_i^{j-1}}{\beta_i^k} \pi^{(k)} - \sum_{j=1}^{n-2} \alpha_j \pi^{(j-1)}.$$

Therefore, by recursively substituting (6.28) for $k = 2, \dots, n-2$ into (6.29), one finally get

$$L_i \Pi_0 = \beta_i^{n-2} \pi^{(1)} - \sum_{j=1}^{n-2} \alpha_j \beta_i^{(j-1)} \pi^{(0)},$$

that, recalling (6.17) yields to

$$L_i \Pi_0 = \left(\beta_i^{n-2} \sum_{j=1}^{n-2} \alpha_j - \sum_{j=1}^{n-2} \alpha_j \beta_i^{(j-1)} \right) \pi^{(0)}.$$

Finally, substituting again (6.24), the proof follows. □

Proposition 2. *The initial condition Π_0 is orthogonal to the left eigenvector associated to $\beta_1 = 1$.*

Proof. The proof follows from Proposition 1. □

Proposition 3. *The normalization factor N_i is given by*

$$N_i = \frac{\beta_i^{n-3}}{(\beta_i - 1) \prod_{\substack{j=1 \\ j \neq i}}^{n-2} (\beta_i - \beta_j)}$$

Proof. Before going into the details of the proof, we need the following definitions.

Definition 4. $\mathcal{C}_k = \{c_{(k,i)}\}$, *where $c_{(k,i)}$ is the i-th string of k elements in $\mathcal{B} = \{\beta_2, \dots, \beta_{n-1}\}$, without repetitions. Therefore, $\#\mathcal{C}_k = \binom{n-2}{n-k}$ and $\mathcal{C}_k = \emptyset$ for $k > n-2$ and $k < 1$.*

From Definition 4, follows that $\mathcal{C}_1 = \emptyset$.

Definition 5. *For all $J \in \mathcal{C}_k$, $\prod \beta_J = \beta_{j_1} \beta_{j_2} \cdots \beta_{j_k}$.*

Definition 6. *Even though $\#\mathcal{C}_{n-1} = 0$, we impose $\sum_{J \in \mathcal{C}_{n-1}} \prod \beta_J = 1$.*

Definition 7. $\mathcal{C}_k^{\beta_i}$ *stands for: "the set of strings of length k in $\mathcal{C}_k$ containing the element β_i". Similarly, $\mathcal{C}_k^{\overline{\beta_i}}$ stands for: "the set of strings of length k in $\mathcal{C}_k$ not containing the element β_i".*

Corollary 6. *From Definition 7 follows that* $\mathcal{C}_k = \mathcal{C}_k^{\beta_i} \cup \mathcal{C}_k^{\overline{\beta_i}}$.

We are now in a position to prove this proposition. From (6.23), we have

$$\frac{P(\lambda)}{\lambda} = (\lambda - 1)\prod_{i=2}^{n-1}(\lambda - \beta_i) = \lambda^{n-1} + \sum_{j=1}^{n-1} \mathcal{S}_j(\beta)\lambda^{j-1}, \tag{6.30}$$

where, using Definition 5, Definition 6 and Definition 7,

$$\begin{aligned}\mathcal{S}_j(\beta) = (-1)^{n-j}\Bigg(&\sum_{J\in\mathcal{C}_{n-j}^{\beta_i}}\prod\beta_J + \sum_{J\in\mathcal{C}_{n-j-1}^{\beta_i}}\prod\beta_J + \\ &+ \sum_{J\in\mathcal{C}_{n-j}^{\overline{\beta_i}}}\prod\beta_J + \sum_{J\in\mathcal{C}_{n-j-1}^{\overline{\beta_i}}}\prod\beta_J\Bigg) \triangleq \mathcal{S}_j^{\beta_i}(\beta) + \mathcal{S}_j^{\overline{\beta_i}}(\beta),\end{aligned} \tag{6.31}$$

where β_i is a generic eigenvalue of $\mathcal{B}$. Since

$$\begin{aligned}\beta_i\mathcal{S}_j^{\overline{\beta_i}}(\beta) &= \beta_i(-1)^{n-j}\left(\sum_{J\in\mathcal{C}_{n-j}^{\overline{\beta_i}}}\prod\beta_J + \sum_{J\in\mathcal{C}_{n-j-1}^{\overline{\beta_i}}}\prod\beta_J\right) = \\ &= -\mathcal{S}_{j-1}^{\beta_i}(\beta),\end{aligned} \tag{6.32}$$

it follows that

$$\mathcal{S}_{j-1}^{\beta_i}(\beta) + \beta_i\mathcal{S}_j^{\overline{\beta_i}}(\beta) = 0, \tag{6.33}$$

and

$$\mathcal{S}_{j-1}(\beta) + \beta_i\mathcal{S}_j(\beta) = \mathcal{S}_{j-1}^{\overline{\beta_i}}(\beta) + \beta_i\mathcal{S}_j^{\beta_i}(\beta). \tag{6.34}$$

Moreover, by (6.31) and Definition 6 follows that $\mathcal{S}_1(\beta) = \mathcal{S}_1^{\beta_i}(\beta)$, since $\mathcal{S}_1^{\overline{\beta_i}}(\beta) = 0$.

By (6.22), (6.30), and Definition 6, we have the following relations

$$\begin{aligned}\mathcal{S}_{n-1}(\beta) &= -\left(1 + \sum_{j=1}^{n-2}\alpha_j\right) = -\left(1 + \sum_{i=1}^{n-2}\beta_i\right), \\ \mathcal{S}_j(\beta) &= \alpha_j, \quad j = 1, \dots, n-2.\end{aligned}$$

Therefore, recalling (6.34),

$$\begin{aligned}&\alpha_j\beta_i^{j-1} + \alpha_{j+1}\beta_i^j = \beta^{j-1}\left(\alpha_j + \beta_i\alpha_{j+1}\right) = \\ &= \beta^{j-1}\left(\mathcal{S}_j(\beta) + \beta_i\mathcal{S}_{j+1}(\beta)\right) = \beta^{j-1}\left(\mathcal{S}_j^{\overline{\beta_i}}(\beta) + \beta_i\mathcal{S}_{j+1}^{\beta_i}(\beta)\right),\end{aligned}$$

and, by virtue of (6.33),

$$\sum_{k=1}^{n-2}(n-1-k)\alpha_k\beta_i^{k-1} = \sum_{j=1}^{n-2}\mathcal{S}_j^{\beta_i}(\beta)\beta_i^{j-1}. \tag{6.35}$$

Without loss of generality, consider the $n-1$-th eigenvalue β_{n-1}. Since it is a simple root of $P(\lambda)$, we define the new polynomial

$$P^\star(\lambda) = \frac{P(\lambda)}{\lambda(\lambda-\beta_{n-1})},$$

for which we have

$$P^\star(\lambda) = (\lambda-1)\prod_{i=2}^{n-2}(\lambda-\beta_i),$$

and, using (6.30),

$$P^\star(\lambda) = \lambda^{n-2} + \sum_{j=1}^{n-2}\hat{\mathcal{S}}_j(\beta)\lambda^{j-1},$$

where the $\hat{}$ highlights the fact that the strings are of maximum length $n-3$ instead of $n-2$. By evaluating $P^\star(\lambda)$ for $\lambda = \beta_{n-1}$, we have

$$P^\star(\beta_{n-1}) = \beta_{n-1}^{n-2} + \sum_{j=1}^{n-2}\hat{\mathcal{S}}_j(\beta)\beta_{n-1}^{j-1},$$

and, multiplying both sides by β_{n-1},

$$\beta_{n-1}P^\star(\beta_{n-1}) = \beta_{n-1}^{n-1} + \sum_{j=1}^{n-2}\beta_{n-1}\hat{\mathcal{S}}_j(\beta)\beta_{n-1}^{j-1}.$$

It has to be noted that $\hat{\mathcal{S}}_j(\beta) = \mathcal{S}_{j+1}^{\overline{\beta}_{n-1}}(\beta)$ since for $\mathcal{S}$ the additional symbol β_{n-1} is also considered (hence, $j \to j+1$). Therefore, making use of (6.32), $\beta_{n-1}\hat{\mathcal{S}}_j(\beta) = -\mathcal{S}_j^{\beta_{n-1}}(\beta)$. Therefore, using (6.35),

$$\begin{aligned}\beta_{n-1}P^\star(\beta_{n-1}) &= \beta_{n-1}^{n-1} - \sum_{j=1}^{n-2}\mathcal{S}_j^{\beta_{n-1}}(\beta)\beta_{n-1}^{j-1}\\ &= \beta_{n-1}^{n-1} - \sum_{k=1}^{n-2}(n-1-k)\alpha_k\beta_{n-1}^{k-1}.\end{aligned}$$

To prove the Proposition, we have only to recall the closed form of the left and right eigenvector reported in (6.27), which yields to

$$\begin{aligned} N_i &= \frac{1}{1-\sum_{k=1}^{n-2} \frac{\sum_{j=1}^{k} \alpha_j \beta_i^{j-1}}{\beta_i^k} \frac{1}{\beta_i^{n-1-k}}} \\ &= \frac{\beta_i^{n-1}}{\beta_i^{n-1} - \sum_{k=1}^{n-2} (n-1-k)\alpha_k \beta_i^{k-1}}. \end{aligned}$$

Finally, noticing that by definition

$$\beta_{n-1} P^\star(\beta_{n-1}) = \beta_{n-1}(\beta_{n-1}-1) \prod_{i=2}^{n-2} (\beta_{n-1} - \beta_i),$$

the proof follows. □

We can now state the following proposition.

Proposition 4. *The existence of the equilibrium for the QBDP implies* $|\beta_i| < 1$ *for* $i = 2, \ldots, n-1$.

Proof. If the QBDP has an equilibrium then (6.26) holds true. The unitary eigenvalue $\beta_1 = 1$ does not play any role in the summation of (6.26) in view of Proposition 2. Next, suppose that there exists one or more $|\beta_i| > 1$. From Equation (6.26) it follows that it may be $L_i \Pi_0 = 0$, $N_i = 0$ or $\Pi_0 = 0$. However, the first two conditions are excluded by Proposition 1 and Proposition 3, while using (6.25) it follows that $\Pi_0 = 0 \Rightarrow \Pi_j = 0$, $\forall j$. Therefore,

$$\pi^{(j)} = \lim_{k \to +\infty} \pi^{(j)}(k) = 0, \quad \forall j,$$

and, since the Markov chain is irreducible and aperiodic, the QBDP does not have a unique stationary distribution [34], which contradicts the hypothesis. □

Proposition 4 yields

$$\sum_{i=2}^{n-1} \frac{\nu_i^{(0)} N_i}{1-\beta_i} L_i \Pi_0 = 1, \tag{6.36}$$

which is a more useful version of (6.26). Indeed, (6.26) states that $\Pi_0 = f(\pi^{(0)})$ while (6.36) gives a straightforward way to describe it in closed form, as summarised in the following.

Proposition 5. *The value of* $\pi^{(0)}$ *is given by*

$$\pi^{(0)} = \prod_{i=2}^{n-1} (1-\beta_i).$$

Proof. By substituting into (6.36) the results of Proposition 1 one gets

$$\pi^{(0)} = -\frac{1}{\sum_{i=2}^{n-1} \frac{N_i}{\beta_i}},$$

and then by making use of Proposition 3

$$\pi^{(0)} = -\frac{1}{\sum_{i=2}^{n-1} \frac{\beta_i^{n-3}}{(\beta_i - 1)\prod_{\substack{j=2 \\ j\neq i}}^{n-1}(\beta_i - \beta_j)}}. \tag{6.37}$$

Without loss of generality, let us consider the k-th eigenvalue β_k and its associated term, which can be expressed using the *partial fraction decomposition*, i.e.,

$$\frac{\beta_k^{n-3}}{(\beta_k - 1)\prod_{\substack{j=2 \\ j\neq k}}^{n-1}(\beta_k - \beta_j)} = \frac{(-1)^{n-1}}{\prod_{i=2}^{n-1}(\beta_i - 1)} + \sum_{\substack{i=2 \\ i\neq k}} \frac{\beta_i^{n-3}}{(\beta_k - \beta_i)(\beta_i - 1)\prod_{\substack{j=2 \\ j\neq k \\ j\neq i}}^{n-1}(\beta_i - \beta_j)}.$$

Plugging the partial fraction expansion in (6.37), one gets

$$\pi^{(0)} = -(-1)^{n-1}\prod_{i=2}^{n-1}(\beta_i - 1) = \prod_{i=2}^{n-1}(1 - \beta_i).$$

□

The result of Proposition 5 can suitably rewritten in the following way

$$\prod_{i=2}^{n-1}(1 - \beta_i) = 1 + \sum_{i=1}^{n-2}(-1)^{n-i-1}\sum_{J\in\mathcal{C}_{n-i-1}}\prod\beta_J = 1 + \sum_{i=1}^{n-2}\mathcal{W}_i(\beta),$$

where

$$\begin{aligned}
\mathcal{W}_1(\beta) &= -\mathcal{S}_1(\beta), \\
\mathcal{W}_2(\beta) &= -\left(\mathcal{S}_1(\beta) + \mathcal{S}_2(\beta)\right), \\
&\cdots \\
\mathcal{W}_k(\beta) &= -\sum_{j=1}^{k}\mathcal{S}_j(\beta),
\end{aligned}$$

and, recalling that $\mathcal{S}_j(\beta) = \alpha_j$, for $j = 1, \ldots, n-2$, finally yields to

$$\begin{aligned}\pi^{(0)} &= 1 + \sum_{i=1}^{n-2} \mathcal{W}_i(\beta) = 1 - \sum_{i=1}^{n-2} \sum_{j=1}^{i} \mathcal{S}_j(\beta) = \\ &= 1 - \sum_{i=1}^{n-2} \sum_{j=1}^{i} \alpha_j = 1 - \sum_{j=1}^{n-2} (n-1-j)\alpha_j,\end{aligned} \tag{6.38}$$

as desired.

At this point we have proved that *if the QBDP has an equilibrium*, this is given by (6.38) and by the recursion of (6.17). Now consider the case in which QBDP does not have a stationary distribution at all. In such a case, recalling the block-tridiagonal matrix representation of (6.2), the following theorems are of interest.

Theorem 7. *[34] An irreducible Markov chain has a stationary distribution if and only if all its states are positive recurrent.*

Definition 8. *Assume $A = A0 + A1 + A2$ is irreducible. Then, by the Perron-Frobenius Theorem, there exists a unique vector $\mu > 0$ with $\mu^T \mathbf{1} = 1$ and $\mu^T A = \mu^T$. The vector μ is called the stationary probability vector of A, while $\mathbf{1}$ is a column vector whose elements are all equal to one.*

Theorem 8. *[35] The QBDP is transient if $\mu^T A_2 \mathbf{1} < \mu^T A_0 \mathbf{1}$, null recurrent if $\mu^T A_2 \mathbf{1} = \mu^T A_0 \mathbf{1}$ and positive recurrent if $\mu^T A_2 \mathbf{1} > \mu^T A_0 \mathbf{1}$.*

By Theorem 7, the QBDP does not have an equilibrium if and only if it has at least one state that is transient or null recurrent. Since A is irreducible, one immediately has that $\mu^T = \frac{1}{n}\mathbf{1}^T$, from which it is possible to explicitly compute $\mu^T A_2 \mathbf{1} = \frac{a'_{n-1}}{n}$ and $\mu^T A_0 \mathbf{1} = \frac{1}{n}\sum_{j=1}^{n-2}(n-1-j)a_{j-1}$. From Theorem 8, the QBDP does not have an equilibrium if and only if $\mu^T A_2 \mathbf{1} \leqslant \mu^T A_0 \mathbf{1}$ or, equivalently,

$$\frac{a'_{n-1}}{n} \leqslant \frac{1}{n} \sum_{j=1}^{n-2} (n-1-j)a_{j-1},$$

This is exactly the condition in which Equation (6.38) produces $\pi^{(0)} \leqslant 0$. We finally notice that, even if the QBDP does not have an equilibrium, it is still irreducible and aperiodic, from which follows that a limiting probability exists, which is given by

$$\pi^{(j)} = \lim_{k \to +\infty} \pi^{(j)}(k) = 0, \quad \forall j,$$

An this ends the proof of Theorem 4. □

A useful corollary of the proof above is the following:

Corollary 9. *If the QBDP does not have a stationary distribution, there exist at least one eigenvalue $|\beta_i| > 1$.*

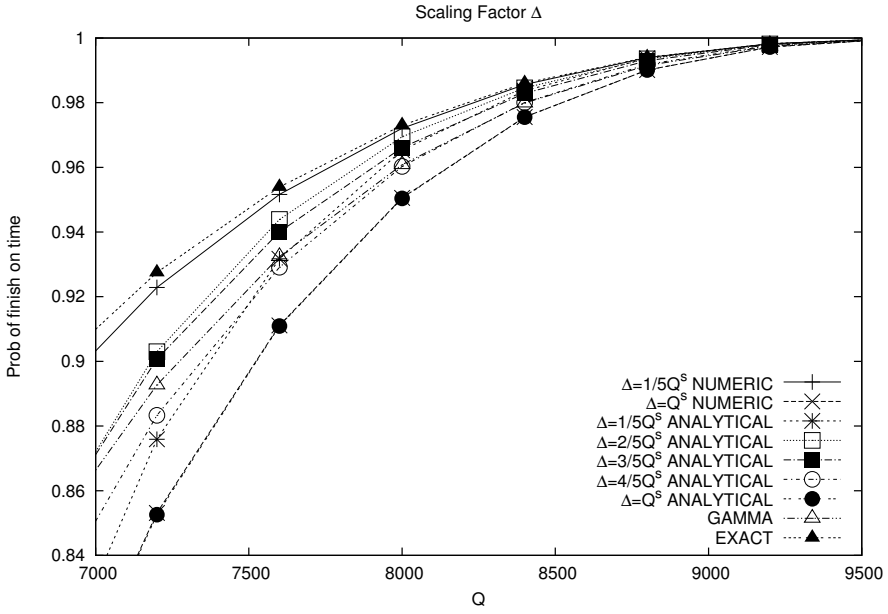

Figure 6.4: Probability of respecting the deadline obtained when using different algorithms to compute the probability to respect a deadline, as a function of the maximum server budget and of the scaling factor.

6.3 EXPERIMENTAL VALIDATION

The presented approach was validated in two different ways. First, the probabilistic deadline was computed using synthetic distributions, to compare the degree of conservativeness of our bounds w.r.t alternative methods and using different values for the scaling factor Δ (see Equation (6.7)). Second, we evaluated the method in a more realistic situation using traces extracted from real applications.

6.3.1 *Synthetic Distributions*

We considered a periodic task with period $P = 40ms$ and random execution time changing according to different probability distributions. We computed the probability of deadline miss probability by using the numeric method illustrated in Section 6.1.2 (denoted as *QBDM* in the sequel) and the analytical bound (denoted as *ANALYTICAL*) for different values of the scaling factor Δ. For comparison purposes, we also compared the results with the one obtained using pre-existing method [36] (denoted as *GAMMA*) and with the exact solution of the eigenvector problem (denoted to as *EXACT*). As a first example, we

have analysed the behaviour of the system for a beta distribution with parameters $\alpha = 2$ and $\beta = 3$ and for different choices of Q^s The server period was set to $T^s = P/2 = 20ms$ ($N = 2$). As an example, Figure 6.4 shows the different probabilities computed for a We have restricted to a range of Q^s that produces a probability of meeting the deadline greater than 90%. This is because smaller probabilities would make little practical sense for a system required to provide an "acceptable" Quality of Service. A first consideration is that the QBDM method approaches very closely the exact solution for small values of Δ (the plot with $\Delta = \frac{1}{5}Q^s$ is always within 1%). The quality of the QBDM bound decreases monotonically with Δ. This is in accordance with our expectations since resampling with a coarser granularity increases the level of pessimism. Different considerations apply to the analytical bound. In this case, we have two effects contrasting each other. On the one hand, the reduction of Δ decreases the pessimism as for the QBDM method. On the other, it increases the pessimism in view of the simplification described in Section 6.2.1. In this situation it is reasonable to expect a bad performance for $\Delta = Q^s$ and for Δ very small, while the optimal choice stays in between. The comparison with the GAMMA bound does not produce univocal results. The analytical methods and the GAMMA bound outperform each other for different values of Δ and for different choices of Q^s. In this particular example, the analytical solution has worse performance than the GAMMA bound for $\Delta = Q^s$,$\Delta = \frac{4}{5}Q^s$, $\Delta = \frac{1}{5}Q^s$ and better for $\Delta = \frac{2}{5}Q^s$,$\Delta = \frac{3}{5}Q^s$. As a final remark, a reasonably good choice for Δ is in a neighborhoud of $\frac{1}{2}Q^s$ (between $\frac{2}{5}Q^s$ and $\frac{3}{5}Q^s$).

To verify the validity of these consideration in a more general case, we have generated 1000 random distribution functions for the periodic task. We have set the server period to $T^s = 10000\ \mu s$ ($N = 4$), with a budget Q^s ranging in $[1.7\frac{E(c)}{N}; \frac{2E(c)}{N}]$ and a scaling factor Δ ranging in $[0.2Q^s; 1.0Q^s]$. Let p denote the probability computed with the EXACT method and $p'_{\mathcal{A}}$ denote the approximation computed using the algorithm $\mathcal{A}$. We define the approximation absolute error introduced by algorithm $\mathcal{A}$ as $E_{\mathcal{A}} = p - p'_{\mathcal{A}}$. In Figure 6.5 we report the approximation error (for different values of the ratio between the assigned bandwidth Q^s/T^s and the average utilisation) averaged through the 1000 different distributions. In accordance with the results obtained for the single beta distribution, the QBDM method with $\Delta = \frac{1}{5}Q^s$ and the quality of the QBDM method changes monotonically with Δ. For the analytical bound, the comparison with the

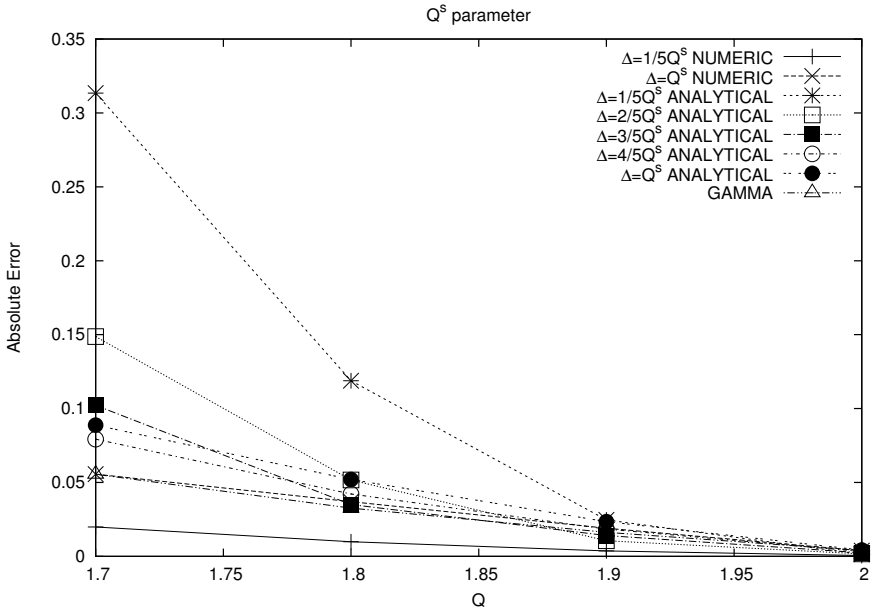

Figure 6.5: Approximation error

GAMMA method is not univocal again (the winner is not the same for all Q^s and for all Δ) and the best choice is in the neighborhoud of $\frac{1}{2}Q^s$. The spread between the different choices of Δ is much smaller than for the single beta distribution. This is because the optimal choice is different for each distribution and the average flattens the results. We believe it interesting to report also the computation time required for the different methods for this set of experiments. The data (average and width of the confidence interval) are reported in Table 6.1. The closed form solution is obviously the fastest: its computation time is at least two order of magnitude below any other method and it is not significantly affected by Δ (for this reason we just report one value). The computation time for the QBDM method, on the contrary, has a strong dependency from the size of the model (which is determined by Δ. However, even for relatively small values of Δ, it remains well below the one of pre-existing methods. The GAMMA bound pays its generally good accuracy and its general applicability with a larger computation time. The EXACT method is reported for the sake of comparison, but in fact it is not applicable for systems of reasonable size.

method	average	95% interval
$\frac{1}{5}\Delta$ NUMERIC	7667.47 µs	[4993.58-10341.35] µs
$\frac{2}{5}\Delta$ NUMERIC	1366.75 µs	[970.75-1762.76] µs
$\frac{3}{5}\Delta$ NUMERIC	611.78 µs	[458.57-764.98] µs
$\frac{4}{5}\Delta$ NUMERIC	356.59 µs	[281.01-432.16] µs
$\frac{5}{5}\Delta$ NUMERIC	241.70 µs	[202.14-281.26] µs
ANALYTICAL	$\tilde{5}$.08 µs	[$\tilde{4}$.99-$\tilde{5}$.16] µs
GAMMA	21333.63 µs	[20626.06-22041.21] µs
EXACT	107191302.70 µs	[101909558.63-112473046.76] µs

Table 6.1: Time computation for the different techniques

6.3.2 *Real application*

To test the method (and the validity of its underlying assumptions) in a real–application, we have considered a vision program that tracks a line a stream captured by a webcam (this is used in a larger control application). The computation was carried out using an embedded board (a Beagle Board). We have collected several execution traces of a reasonable length (about 600 seconds of execution). The application period was $P = 40000$ µs; the server period and the budget were chosen as $T^s = 20000$ µs and $Q^s = 3000$µs. Such traces have been used in the RTSIM scheduling simulator to estimate the experimental probability of meeting the deadline for the assigned choice of parameters. The result will be denoted below as RTSIM-Trace. What is more, we have derived the experimental probabilities and used them in our methods neglecting the correlation structure of the process and assuming it to be IID. As in the experiments reported in the previous section, we applied the QBDM and the ANALYTICAL methods for several choices of Δ and compared them with the GAMMA bound and with the experimental probability. For the sake of completeness, we also computed the experimental probability reported by RTSIM using an IID process having the PMF estimated from the trace (this will be denoted as RTSIM-PMF). The results are shown in Figure 6.6, where we consider two different traces collected in different environmental conditions. For the first trace (top half of the figure) the GAMMA bound produces a very accurate bound, while in the second (bottom half of the figure) the QBDM and ANALYTICAL method have a better performance. As far as the QBDM method

www.beagleboard.org
http://rtsim.sssup.it/

is concerned, we observe the usual monotonicity with respect to Δ, while for the ANALYTICAL method the best choice for Δ is again very close to 50% of Q^s. The plots reported by the ANALYTICAL method and by QBDM overlap for large values of Δ.

Another important remark is on the impact of the assumption that underlie the application of our method, and in particular of the computation process being IID. In this particular example this assumption does not have a strong impact. Indeed, the distance between the experimental probability obtained with the trace (RTSIM–Trace) and with the PMF generated neglecting the correlation structure (RTSIM–PMF) is very small and cannot be appreciated with the scale of the plots. We can hardly claim any generality for this remark. For other applications assuming an IID process can have a significant impact. In our previous work [33] we made a more extensive investigation on this aspect on a large set of applications and the result was that the error introduced is almost always acceptable. Nevertheless, accounting for the correlation structure of the process is certainly an important future direction for research.

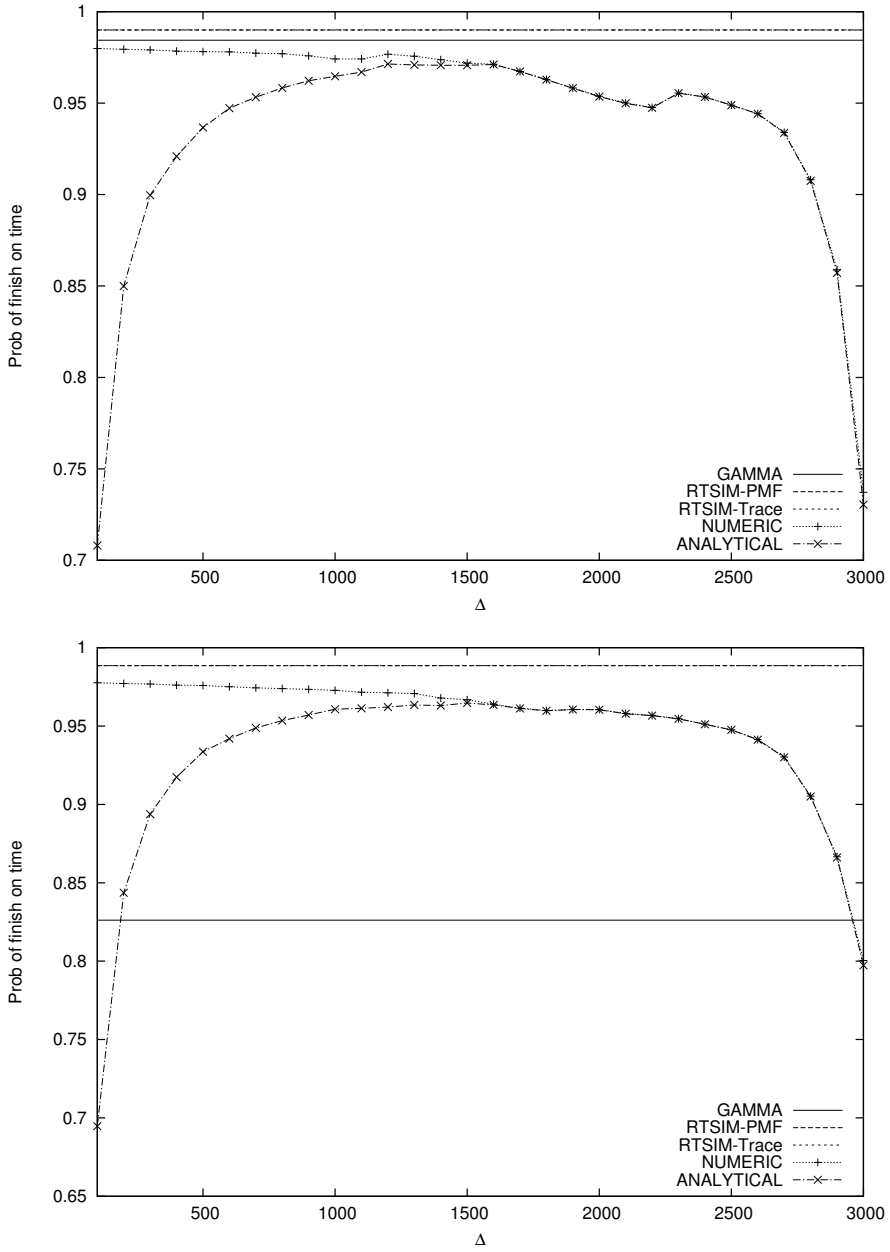

Figure 6.6: Real application

7

DISCUSSION OF MODELS

We already shows some numbers about the difference between the models in Chapter 6, in this chapter we discuss it at higher level. First of all, the methods presented in Chapter 5 and Chapter 6 assume that the processes are IID. Since in real life this assumption can be very difficult to claim, this section try to give some information about that simplification.

7.1 IID: INDEPENDENT AND IDENTICALLY DISTRIBUTED RANDOM VARIABLES

To evaluate the impact of approximating a real process as an IID process, we have tests on 78 executions of a mix of real applications. In particular, we considered video encoding and decoding tasks and video tracking tasks (used in visual control applications). To avoid an overcommitment to a specific architecture, some of the applications were executed on an INTEL based personal computer and some of the applications were executed on an ARM based embedded board (the beagle board: www.beagleboard.org). For each application, we collected the traces of the execution times and derived their distributions, which were used as input data for the execution of the different algorithms. In doing this step, the correlation structure of the process has been discarded and the process has been assumed IID.

Our goal is to evaluate the impact of discarding the correlation structure (independently of the algorithm used for the computation of the probability). To this end, for each application we generated a synthetic application, with IID computation times distributed with the same distribution as the "real" application. The synthetic application was executed and its deadline misses recorded for the different values of the budget. In Figure 7.1, we show the error E^{IID} reported in this way. The plot contains the worst performance and the best performance obtained on the entire set of applications. The former is obtained by the tracking application in a particular configuration; the latter was obtained by an MPEG decoder. The performance of all the tested application lies somewhere in between these two lines. In the

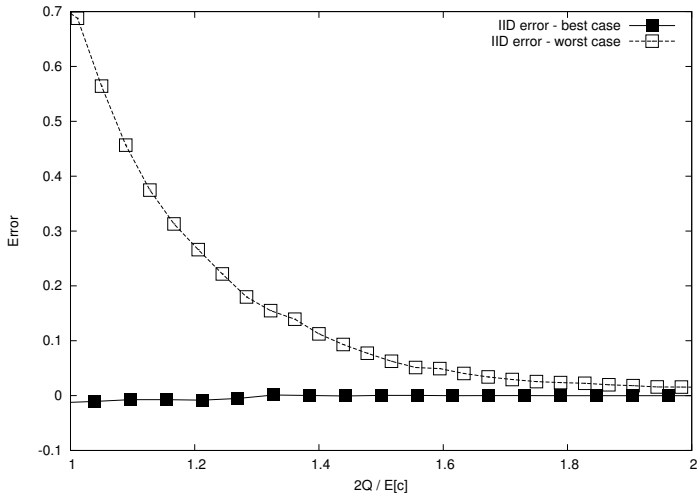

Figure 7.1: Approximation error E^{IID} introduced by the IID assumption (independently by the bound used to estimate the deadline respect probabilities).

method	robust	IID	small arr.	pessimistic	pessimism	comp.
Simulation	N	Y	Y	N	0	-
Exact	N	N	Y	Y	1	4
Gamma	Y	N	N	Y	2	3
QBDM	N	N	N	Y	2	2
Analytical	N	N	N	Y	3	1

Table 7.1: Comparison between methods

x –axis, we report the ratio $2Q^s/E[c]$ between the average utilisation of the task and the assigned bandwidth. For values of the bandwidth close to the average of the utilisation, the IID approximation produces, in the worst case, a very conservative estimation of the probability (error close to 70%). In the best case, the error remains very limited for every value of the bandwidth.

7.2 SUMMARY OF RESULTS

In this section we want to compare the characteristics of the different methods. The schema in Figure 7.2 resumes the evolution of this dissertation. The new methods introduced are compared with the literature methods in Table 7.1 show the different characteristics of the possible methods to predict the behaviour a resource reservation systems. The methods considered are:

- *Simulation*: this analysis is done using the RTSIM simulator,

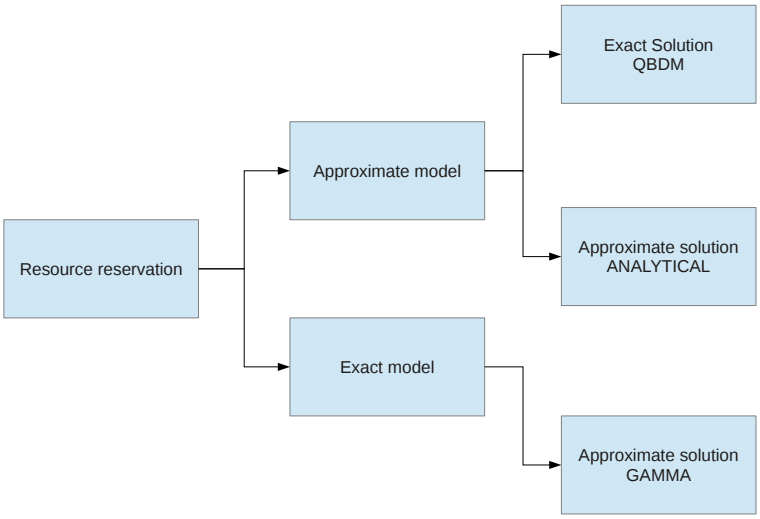

Figure 7.2: Schema of the used approaches

- *Exact*: this analysis in [4],
- *Gamma*: the analysis presented in Chapter 5,
- *QBDM*: the numeric solution presented in Chapter 6,
- *Analytical*: the analytical solution described in Chapter 6.

The parameters are:

- *robust*: the method can be used without the complete knowledge of the execution times. The answer is Yes or No
- *IID*: the method can considers not only PMF but also trace of executions. The answer is Yes or No.
- *small arrivals*: the method can be used for application in which the inter-arrival time is very small, even with burst of arrivals. The answer is Yes or No.
- *pessimistic*: in order to give guarantees, the method is pessimistic respect to the real probability. The answer is Yes or No.
- *degree of pessimism*: the difference between the real probability and the pessimistic probability obtained with the method. Lower is the number, lower is the pessimism in the average case.
- *complexity*: the complexity in term of time and memory. Lower is the number, lower is the complexity.

As expected, from Table 7.1, we can notice that greater is the pessimism of the method, lower is the complexity. This is true in the average case but in the single experiment it is possible and frequent that the analytical solution gives result equal or very similar to the others. For the methods presented in this dissertation, for an offline computation is possible to use all of this methods and simply choose the best one. For online computation, the most indicated methods are the analytical solution, and in case of less strict timing constraints also the QBDM method can be applied.

8

RESERVATION-BASED INTERRUPT SCHEDULING

In this chapter is presented an example in which resource reservation can be very useful to improve the performance of the system. Unfortunately the techniques explained in the previous chapters are not suitable for this example. A new model must be created specific for this application.

8.1 INTRODUCTION

A recent trend in real-time research is to support complex applications for which the restrictive assumptions made by the standard real-time theory are not applicable. For example, when real-time applications heavily interact with the external environment (reading or writing large amount of data from hardware devices), the classical model of real-time tasks as independent activities, activated with a given temporal pattern, and using only one type of resources (typically the CPU) is at risk of losing fundamental details. Indeed, I/O operations (and hence the execution of device drivers) can generate timing problems of unexpected harshness when they are not adequately handled and taken into account.

From the scheduling point of view, device drivers can introduce very long priority inversions, potentially disrupting the temporal guarantees. Therefore, a necessary condition to process device information in real-time system is to execute them inside schedulable entities (typically threads). In the past, this solution has been mainly used in μkernel based systems or in small real-time kernels. More recently, even widely used OS kernel such as Linux offered the possibility of executing interrupt handlers in dedicated threads, by applying a real-time patch to the kernel. The recent adoption of interrupt threads in the main stream Linux kernel (starting from version 2.6.30) raises strong expectations on the future popularity of these technique among developers.

Parts of this Chapter are going to appear in:
N. Manica, L. Abeni, L. Palopoli, "Reservation-based interrupt scheduling," *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2010 16th IEEE*

In this new context, a very important issue is the impact of the scheduling policy on the device drivers performance. As of today, interrupt threads are frequently scheduled with a fixed priority assigned in empirical ways. The result is often unsatisfactory: the system has to be designed with a conservative assignment of resources and temporal guarantees are difficult to provide. Hence, there is a pressing need for theoretical models for interrupt threads scheduling enabling well-founded choices of the scheduling parameters.

The scheduling analysis of interrupt threads cannot be carried out using the standard approaches because many hardware devices have finite queues which impose an upper bound on the number of pending interrupts (e.g., interrupts awaiting the CPU allocation). This situation is not normally considered in the standard real-time scheduling theory. Finite queues are obviously dealt with in the framework of queueing theory [37], but this type of analysis is not directly concerned with real-time constraints.

Besides the difficulty in modelling the scheduling problem, the use of standard scheduling solutions could be inadequate in many situation of interest. For instance Modena and other [38] showed that by using fixed priority it is impossible to guarantee *both* the timing constraints of real-time tasks and the performance of hardware devices. In other words, we can easily find situations in which there does not exist a priority assignment allowing us to respect the constraints of real-time tasks and to maintain an acceptable value of throughput for non real-time applications at once. An example of this kind is discussed in Section 8.3.1.

The adoption of advanced scheduling solution such as reservation-based scheduling [39, 4, 5] allows us to overcome this difficulty, as shown in Section 8.5.2. The basic idea of reservation-based interrupt scheduling is that each interrupt thread can be reserved a fraction of the CPU time. This approach (as compared to fixed priorities) enables a finer grained control of the CPU. In this framework, the scheduling design problem becomes identifying the correct choice of the reservation parameters. Standard techniques for dimensioning these parameters [7] require that the inter-arrival time of the threads be large enough compared to the reservation period (which is one of the parameters to choose). However, in the case of interrupt threads, this assumption fails because the inter-arrival time of interrupts can be very small.

To cope with this problem, two novel analysis techniques (the first one deterministic and the second one stochastic) have been developed to identify a correct choice of the reservation parameters so that the number of pending interrupts can be controlled. Finally, an extensive experimental validation carried out on a real implementation is presented that validates the theoretical analysis and proves that the approach based on interrupt threads and resource reservations is actually viable in real-life situations and it enables a good trade-off between predictability and throughput.

8.1.1 *Related Work*

The effects of device drivers on user space applications have been previously analysed mainly considering network devices and the network subsystem. Some modifications of such subsystem have been previously proposed in order to improve the network throughput, or to eliminate the so called "receive livelock" [40, 41, 42], but, excluding some notable exceptions [43, 44], real-time aspects have not been investigated.

Interrupt handlers have been traditionally executed inside dedicated threads in μkernel environments [45, 46], and the idea of scheduling device drivers have been previously considered also in real-time theory [47]. Reservation-based scheduling has also been proposed in the past as a way to serve interrupt threads [48]. However, none of the papers mentioned above provide a theoretical analysis of a system including interrupt threads. Moreover, they require heavy modifications to the kernel structure, while the approach proposed in this chapter is based on a widely-supported kernel (Linux) plus two fairly maintainable patches [49, 50].

More recently, interrupt threads scheduling in Linux has been evaluated in the context of real-time scheduling [51, 52]. However, the experiments and analysis presented in such works are limited to fixed priorities, and fixed priorities do not seem to provide enough flexibility to properly serve device drivers without affecting real-time tasks [38].

8.2 DEFINITIONS AND BACKGROUND

Real-time systems are traditionally modelled as sets $\Gamma = \{\tau_i, i \in \mathcal{N}\}$ of real-time tasks τ_i, where a real-time task is a stream of jobs (or instances) $J_{i,j}$. Job $J_{i,j}$ becomes ready for execution (arrives) at time $r_{i,j}$, it requires a computation time $c_{i,j}$, and finishes at time $f_{i,j}$. Each job $J_{i,j}$ is also characterised by a deadline $d_{i,j}$ that is respected if $f_{i,j} \leqslant d_{i,j}$, and is missed if $f_{i,j} > d_{i,j}$. Task τ_i is often described by a Worst Case Execution Time (WCET) $C_i = \max_j\{c_{i,j}\}$ and a Minimum Interarrival Time $P_i = \min_j\{r_{i,j+1} - r_{i,j}\}$.

In this chapter, we will use reservation based scheduling: each task τ_i is served by a reservation $RSV_i = (Q_i^s, T_i^s)$, meaning that a time Q_i^s is *reserved* to τ_i in a period T_i^s; Q_i^s is called *maximum budget*, and T_i^s is called *server period*. Resource reservations present the great advantage of providing *temporal isolation* between tasks, meaning that the worst case behaviour of task τ_i is not affected by the other tasks running in the system. As a result, the performance of each task can be analysed in isolation, without considering all the other tasks; this is very important from the analysis point of view, because a system containing IRQ threads can be very complex, and including all the possible interrupts and tasks in the model can make it intractable.

The reservation mechanism used in the section is the Constant Bandwidth Server (CBS) (Chapter 4).

8.3 THE PROBLEM

Based on the system model introduced in Section 8.2, which only considers applications using one hardware resource (the CPU), the schedulability of a real-time task set (i.e., the fact that the tasks' deadlines are respected) can be guaranteed by using a *proper scheduling algorithm* and an *admission test*. The admission test can be carried out in different ways (utilisation-based test, response time analysis, or time demand analysis) and is traditionally based on C_i and P_i. However, this model is simplistic because it does not consider the time consumed by the OS kernel to handle interrupts raised by the hardware devices connected to the system. For example, consider a traditional kernel, in which hardware interrupts are generally served in two phases:

The word "task" is used to identify a schedulable entity, being it a thread or a process.

- a short *Interrupt Service Routine* (ISR) is invoked as soon as an interrupt fires and is responsible for acknowledging the hardware interrupt mechanism, postponing the real data transfer and processing to a longer routine, to be executed later;
- a longer routine (*soft interrupt*, or *bottom half*) is executed later to correctly manage the data coming from the hardware device.

ISRs generally execute with interrupts disabled, while soft interrupts always execute with interrupts enabled and are served when switching from kernel space (where ISRs run) to user space (where user programs are executed). Therefore, soft interrupts can be preempted by ISRs.

Both ISRs and soft interrupts have a higher priority than user tasks, and can "steal" execution time from them. Such "stolen time" can be accounted for in real-time guarantees by modelling it as a blocking time B_i (the admission tests mentioned above can be enhanced to account the blocking times). This implies that a low-priority task can affect the schedulability of high-priority tasks by causing the generation of a large number of hardware interrupts.

This problem is generally solved in real-time kernels by scheduling the interrupt handlers: for example, the Real-Time Preemption patch (RT-preemtp) [49] introduces real-time features in the Linux kernel and transforms ISRs and soft interrupts in kernel threads (the hard IRQ thread and the soft IRQ thread), that are *schedulable entities* handled by the kernel scheduler in the same way as user tasks (so, IRQ threads can have lower priorities than real-time tasks, and can be preempted by them). Hence, some new tasks τ_i are added to the system, representing the IRQ threads which are used to serve hardware interrupts. The arrival times $r_{i,j}$ of these new tasks correspond to the times when the interrupts fire.

This solution can present a slightly higher overhead, and requires a more careful synchronisation, but has the advantage to correctly accounting the handler code in a real-time system (that is, the CPU time required to execute the handler can be correctly accounted in order not to break the system's guarantees).

8.3.1 *A Motivational Example*

The possibility to schedule interrupt handlers (provided by IRQ threads) permits to reduce the interference from hardware devices (by giv-

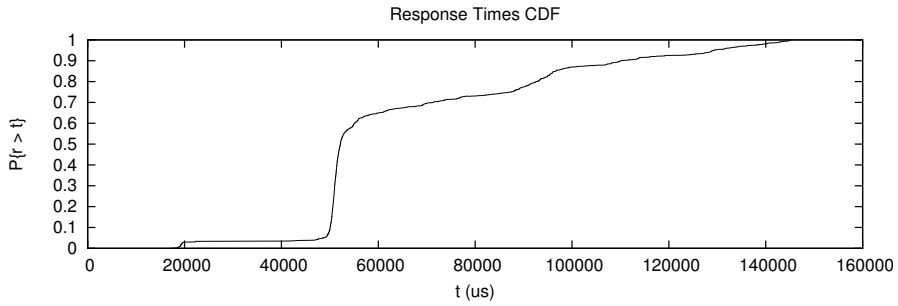

Figure 8.1: CDF of the response times for a $\tau = (20\text{ms}, 50\text{ms})$ real-time task in a standard Linux kernel with high network traffic.

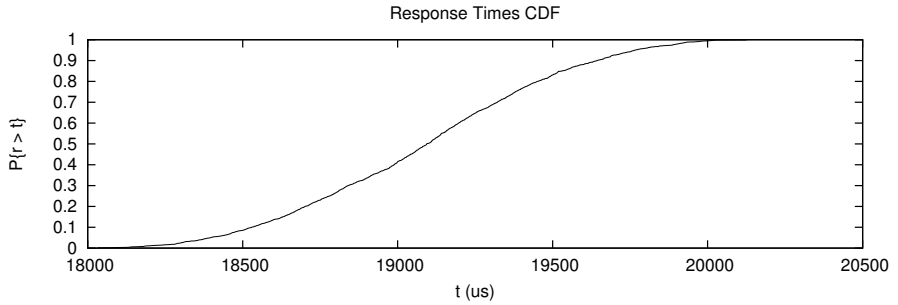

Figure 8.2: CDF of the response times for a $\tau = (20\text{ms}, 50\text{ms})$ real-time task in a Preempt-RT kernel with high network traffic and τ's priority higher than the IRQ thread priority.

ing user-space real-time tasks higher priorities than interrupts). However, identifying a correct scheduling policy for this type of threads is largely an open issue, as discussed in the following example. On a Linux based systems (see Section 8.5 for a more detailed description of the experimental setup), we have executed a periodic real-time task $\tau = (C = 20\text{ms}, P = 50\text{ms})$ along with a non real-time task receiving UDP packets from the network (the `netperf` benchmark has been used for this purpose, for it generates the desired workload and evaluates the network throughput).

When the real-time task runs alone on an unmodified Linux kernel, the maximum measured response time is around 20ms (corresponding to the WCET) as expected. When running the network server alone on an unmodified Linux kernel and using small packets (around 192 bytes), the measured network throughput is around 74Mbps. However, when the two tasks are executed simultaneously the worst-case response time of the real-time task grows to more than 150ms (even if the program receiving UDP packets is executed as a

http://www.netperf.org

non real-time task), as shown in Figure8.1 (representing the Cumulative Distribution Function - CDF - of the response times).

When executing the two tasks on a preempt-RT kernel, it is possible to set the priority of the IRQ threads so that the real-time guarantee for τ is not jeopardised: if the priority of the IRQ threads is less than τ's priority, then the worst-case response time is still very close to $20ms$ (see Figure8.2). However, the network throughput drops to $48Mbps$. As one would expect, the network throughput returns to $74Mbps$ if the priority of the network IRQ threads is raised to a value higher than the priority of τ, but the response time of the latter is in this case out of control. These results confirm the outcome of some previous work [38] in showing that fixed priorities do not allow to find good trade-offs between the real-time performance of user-space tasks and the throughput of hardware devices.

8.3.2 *Reservation-Based Scheduling of Interrupt Threads*

The work presented in this chapter addresses the problem of finding good latency/throughput trade-offs by using reservation-based scheduling for the IRQ threads (in particular, a CBS with hard reservation behaviour is used).

The CBS parameters Q_i^s and T_i^s must be properly dimensioned to avoid dropping too many interrupts. For example, some hardware devices have a limited amount of memory for incoming data, and interrupt handlers are used to get data from this buffer. If too many interrupts are pending, the buffer can overflow and some input data can be lost (in the next sections, a network card which can buffer at most N_c input Ethernet frames will be considered). Hence, the CBS used to schedule the IRQ thread for the device must be dimensioned to control the number of pending interrupts.

A typical strategy for dimensioning a reservation is to choose a reservation period T^s smaller than the minimum inter-arrival time of the served task. Such a choice allows one to model a reservation as a queue and to use standard tools for Markov Chain analysis to compute the probability distribution of the response time. Unfortunately, this approach is not viable in our setting: indeed, two interrupts can even be separated by an interval of time of a few micro-seconds. On the other hand, choosing a value for T^s smaller than $1ms$ leads to an unacceptable scheduling overhead. As discussed next, this consider-

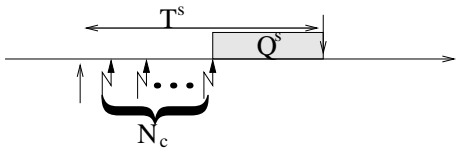

Figure 8.3: Worst case reservation behaviour for periodic interrupt arrivals.

ation induces a different modelling technique than the one typically used in these cases.

8.4 THEORETICAL ANALYSIS

In this section, the problem of correctly dimensioning the CBS parameters for serving an IRQ thread is addressed, starting with a deterministic model and then providing a stochastic analysis of the system. As already explained in Section 8.3, thanks to the temporal isolation property provided by the CBS it is possible to analyse a single IRQ thread in isolation; hence, the analysis can be developed considering one single interrupt type.

8.4.1 *Deterministic Analysis*

In the deterministic case, which is simpler to analyse, interrupts are assumed to fire with a minimum inter-arrival time P, and the interrupt handler is assumed to need a maximum time C to serve an interrupt request. Based on such assumptions, the worst case situation is represented by periodic interrupt requests (with period P), with the interrupt hander serving each one of them in a constant time C.

In this situation, the IRQ thread can be modelled as a periodic task $\tau = (C, P)$ and properly dimensioning a CBS (Q^s, T^s) so that no interrupt is lost is quite simple. For the sake of simplicity and without loss of generality, T^s is set as a multiple of P. In a server period $n = \frac{T^s}{P}$ interrupts arrive, requiring $\frac{T^s}{P}C$ units of time to be served. Hence, the CBS must reserve at least $Q^s = \frac{T^s}{P}C$ unit of time every server period T^s. Moreover, there is an upper bound to the server period, due to the maximum number of pending interrupts N_c that can be queued. In fact, in the worst case the CBS can provide the Q^s time units at the end of the server period, as shown in Figure 8.3; hence, in the first $T^s - Q^s$ time units of the server period the IRQ thread will not execute, and $\frac{T^s - Q^s}{P}$ pending interrupts will be queued. This number

should be smaller than the maximum number of pending interrupts N_c.

As a result, the conditions expressed in Equation 8.1 must be satisfied:

$$\begin{cases} \frac{Q^s}{T^s} \geqslant \frac{C}{P} \\ \frac{T^s - Q^s}{P} < N_c \end{cases} \tag{8.1}$$

As noticed above, this kind of conservative analysis can be applied even if the IRQ thread is not periodic (and/or its execution times are not constant), by substituting C with the WCET of the IRQ thread, and P with the minimum inter-arrival time between interrupts. However, assuming this kind of worst case conditions for the workload can be very pessimistic, often resulting in over-provisioning and in a suboptimal system utilisation.

To evaluate the degree of conservativeness of Equation 8.1 for real-world workloads, a stochastic model (offering a fine grained description of the system evolution) has been developed.

8.4.2 *Stochastic Analysis*

A stochastic analysis of a reservation-based system has already been performed in the past [7, 4, 6], but such previous approaches cannot be used in this context for the reasons explained at the end of Section 8.3. Hence, a new stochastic model has been developed.

The model is constructed assuming that the inter-arrival time between the h^{th} and $(h+1)^{th}$ interrupts is a stochastic process and that the time needed for serving the h^{th} interrupt is also a stochastic process. Both these stochastic processes are assumed to be independent and identically distributed (i.i.d.). However, the stochastic analysis presented in this chapter differs from the previous ones because it is based on a discrete-time model, where all the times are multiple of a small time interval Δ. The system evolution is necessarily observed at each time interval Δ (this difference respect to the previous works allows to handle inter-arrival times smaller than T^s and to count the number of pending jobs).

Applying standard modelling techniques based on Discrete Time Markov Chains (DTMC) it is possible to find the probability of drop-

To be more correct, the resulting stochastic process is semi-Markovian.

ping an interrupt. The state of the IRQ thread can be described by a 5-tuple $S_i = (x, h, z, t, q)$ where:

- x is the number of the pending interrupts;
- h is the time to the arrival of the next interrupt;
- z is the residual execution time of the currently served interrupt;
- t is the time elapsed from the beginning of the last server period;
- q is the residual budged of the CBS.

Each variable is described by an integer number of time intervals Δ, therefore the number of states is numerable. Since the system cannot queue more than N_c pending interrupts, $x \leqslant N_c$. All other variables are also bounded, hence the number of states is finite (i.e., model is finite).

The transitions between the states are driven by 1) the evolution of time (which has an impact on h, z, t), 2) the arrival of a new interrupt (that has an impact on x), 3) the allocation of the CPU by the scheduler (which decreases q, z and, when the current task finishes, may determine a decrease of x). The probability to execute a task served by a CBS at time t is estimated by Equation 8.2

$$Exec(q,t) := \begin{cases} 1 & \text{with probability } \frac{q}{T^s - t} \\ 0 & otherwise \end{cases} \tag{8.2}$$

(remember that q is the current budget and t is the time elapsed from the beginning of the period). At the beginning of a reservation period (when the deadline is postponed), the task has a probability to execute equal to Q^s/T^s. After 1 time unit, the execution probability becomes $(Q^s - 1)/(T^s - 1)$ if the task executed, or $Q^s/(T^s - 1)$ if the task did not execute. Note that if the task does not execute for $T^s - q$ time units, then it is guaranteed to execute in the next time unit ($Exec(q,t)$ gives an execution probability equal to 1). This function models the properties of a hard CBS, so that a task executes for Q^s time units every T^s time units.

The resulting set of states transitions is clearly very large and complex, and due to space constraints it is not possible to fully describe it. However, the most important transitions are described in the following.

The initial state of the system is $S_0 = (0, 0, 0, 0, 0)$. A set of states with $x = -1$ is used to model the situation in which an interrupt is lost (an interrupt arrived when there already are N_c pending interrupts).

An interrupt is completely served (and the corresponding job of the IRQ thread finishes) when z arrives to 0; then, if there is a pending interrupt it can be served. Hence, a new job for the IRQ thread is generated, and its execution time is computed based on the Probability Distribution Function (PDF) of the service times $U(c) = P\{c_{i,j} = c\}$.

In the same way, a new interrupt fires (and becomes ready to be served or pending) when h arrives to 0; in this case, the arrival time of the next interrupt is computed based on the PDF of the inter-arrival times $V(\delta) = P\{r_{i,j+1} - r_{i,j} = \delta\}$.

The CBS budget is recharged at the next scheduling deadline (the end of the current server period) according to the hard reservation behaviour, so when t arrives to T^s its value is reset to 0 and q is recharged to Q^s.

As previously mentioned, the complete model is much more complex than this (there are many state transitions that have not been mentioned above), and the complete transition function between S_i and S_{i+1} is described in Figure 8.9 and Figure 8.10, at the end of this chapter.

The presented model permits to compute the probability to drop an interrupt, which is equal to the steady state probability of the $(-1, h, z, t, q)$ states. Such probability can be computed by finding the steady state probabilities through some numerical tool: in practice, if $\pi(n)$ is a vector containing the state probabilities at time $n\Delta$ (that is, the i^{th} component of $\pi(n)$ is probability that at time $n\Delta$ the system is in the i^{th} state - remember that the states are numerable), then $\pi(n+1) = \pi(n)M$, where M is a transition matrix containing the values from the model presented in Figure 8.9 and 8.10. The searched steady state probabilities are given by $\pi = \lim_{n\to\infty} \pi(n)$, and can be found by numerically solving the equation $\pi = \pi M$ (that is, finding the eigenvector with eigenvalue 1 for the matrix M).

Although M can be a very large matrix, it has a sparse structure (the non-null terms are a small percentage of the elements of the matrix). Therefore, the computation is tractable appropriate tools for sparse matrix (in particular we used the sparse matrices libraries from the Trilinos Project).

http://trilinos.sandia.gov

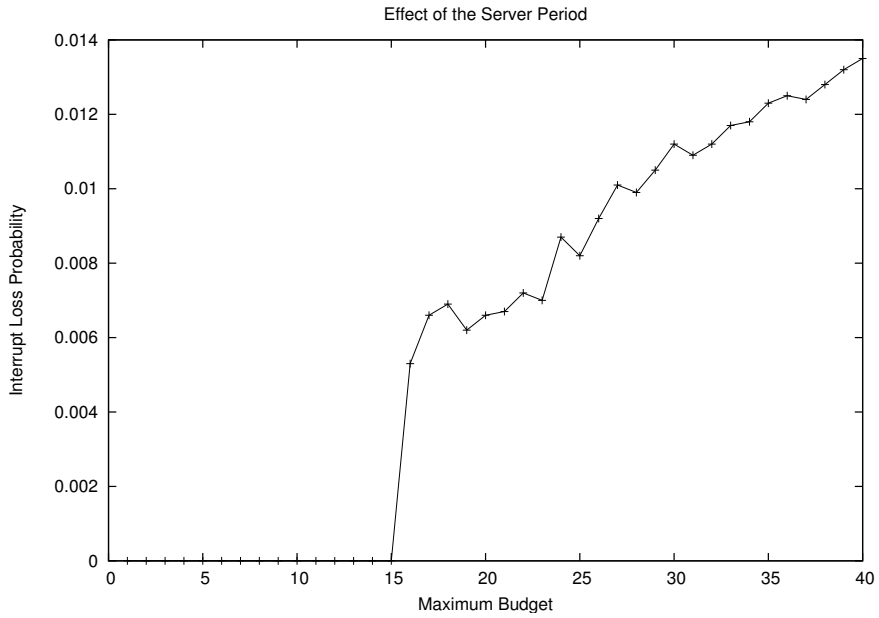

Figure 8.4: Interrupt loss probability for $P = 4$, $C = 2$, $N_c = 4$ and $T^s = 2Q^s$ as a function of Q^s.

8.4.3 *Selected Results*

The proposed model has been numerically solved using different PDFs for the execution and inter-arrival times, and different (Q^s, T^s) settings, verifying that the results are consistent with Equation 8.1.

A first set of results shows that if the PDFs of the execution and inter-arrival times are deterministic, then the results exactly match the deterministic analysis. To achieve this result, the PDF of the execution times has been set to $U(2) = 1$, and the PDF of the inter-arrival times has been set to $V(4) = 1$ (hence, interrupts are periodic with period $P = 4$ and each interrupt needs 2 time units to be served). Then, when solving the eigenvector problem mentioned above it turns out that the probability to drop an interrupt is > 0 if $Q^s/T^s < 0.5$. If, instead, $Q^s/T^s \geqslant 0.5$, things are more interesting and the probability to drop an interrupt depends on the period. Consistently with Equation 8.1, if $Q^s > T^s - N_c P = T^s - N_c \cdot 4$ then such probability is 0. For example, Figure 8.4 shows the probability to drop an interrupt when $T^s = 2Q^s$ and $N_c = 4$ and it is possible to see that interrupts are not lost for $Q^s < 2Q^s - 4 \cdot 4 \Rightarrow Q^s < 16$.

The effect of the maximum number of pending interrupts N_c is also shown in Figure 8.5, which plots the interrupt loss probability for a $(20, 40)$ CBS as a function of N_c. Finally, Figure 8.6 shows how the interrupt loss probability depends on both Q^s and T^s.

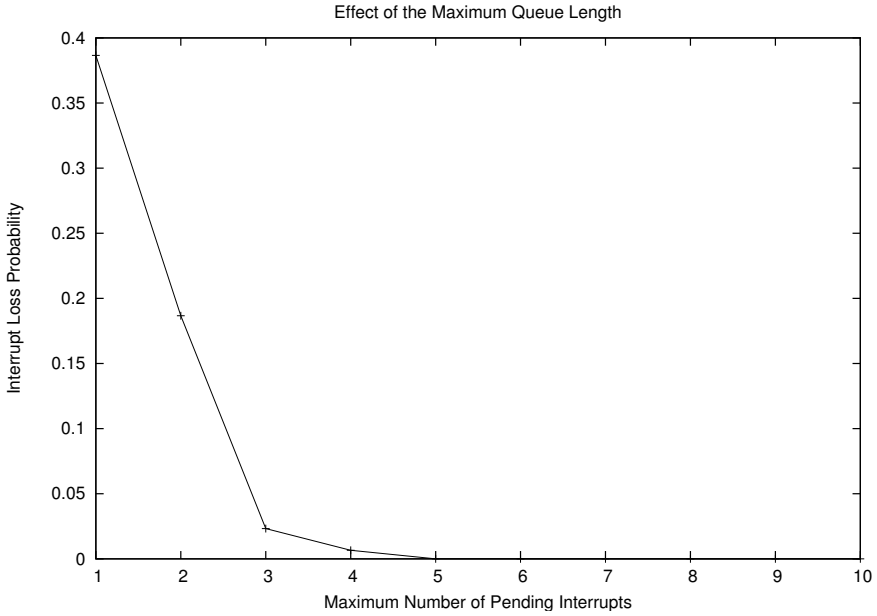

Figure 8.5: Interrupt loss probability for $P = 4$, $C = 2$, $Q^s = 20$, and $T^s = 40$ as a function of N_c.

The advantages of using a stochastic model become even more visible when execution and inter-arrival times are not fixed. For example, consider a simple example with $V(\delta) : V(4) = 0.2, V(5) = 0.8$ and $U(c) : U(2) = 0.3, U(3) = 0.7$ (hence, the maximum load is $3/4 = 0.75$). The maximum amount of pending interrupts is assumed to be $N_c = 3$. According to the deterministic analysis, the reserved fraction of CPU time must be $Q^s/T^s > 0.56250$, and the maximum budget must be $Q^s > T^s - 3 \cdot 4 = T^s - 12$. However, Table 8.1 shows that even if these two conditions are not respected the probability to drop an interrupt can be very low.

8.5 IMPLEMENTATION AND EXPERIMENTS

An implementation of the CBS scheduler for Linux [50] has been used to schedule the IRQ threads in the 2.6.21.4-rt12-cfs-v17 of the RT-Preempt for the Linux kernel. This patch transforms both the ISRs and the bottom halves in kernel threads (the hard IRQ threads and the soft IRQ threads), and the CBS scheduler is used to handle such threads (in the experiments reported in this section, a hard reservation behaviour has been used, to leave some execution time to non-real-time userspace tasks).

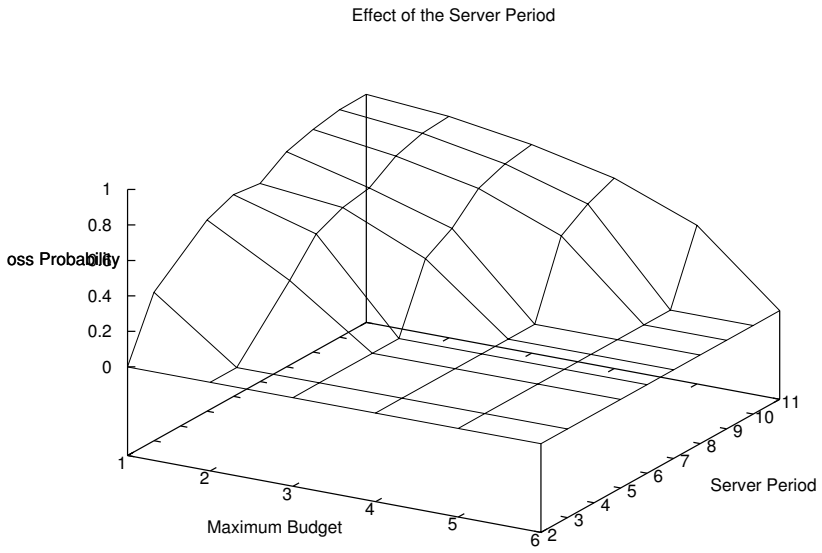

Figure 8.6: Interrupt loss probability for $P = 4$, $C = 2$, and $N_c = 32$ as a function of (Q^s, T^s).

The experiments reported in this section focus on the network card as a hardware device, because it is a good example of a high-throughput device (up to 100Mbps) for which it is possible to control the input load (this is needed for checking the consistence between the theoretical analysis and the experimental results). In particular, the test system used a "National Semiconductor Corporation DP83815" ethernet card (handled by the "natsemi dp8381x" Linux driver), which has an input ring buffer able to store up to 32 ethernet frames. This means that at most 32 ethernet IRQs can be simultaneously pending without dropping any input packet ($N_c = 32$). The test system is based on an Intel Pentium 4 CPU running at 1700MHz, equipped with 256MB of RAM.

To generate a controlled load on the network card, the test machine has been connected to a traffic generator through a cross network cable. The traffic generator is another PC based on RT-Preempt, which is able to send UDP packets according to a specified distribution of the inter-packet times.

8.5.1 *Validation of the Theoretical Model*

First of all, the theoretical analysis developed in Section 8.4 has been validated through a set of experiments, by using the traffic generator

to send fixed-size UDP packets to the test system according to a specified PDF of the inter-packet times, shown in Table 8.2. The amount of time needed by the IRQ thread to receive an ethernet frame has then been measured by using accurate CPU accounting and measuring the execution time for receiving a large number of ethernet frames. A large number of runs revealed that the amount of CPU time needed by the IRQ thread to receive an ethernet frame does not depend on the frame size and is between 22µs and 25µs.

Then, the number of network card IRQs in the test machine has been measured by reading `/proc/interrupts` (to check if it matched the number of ethernet frames sent by the traffic generator), and the number of correctly received packets has been measured by using the `netstat` command. Based on these two measurements, it has been possible to compute the fraction of lost ethernet frames (equal to the fraction of dropped ethernet IRQs), displayed in the second column of Table 8.3.

The stochastic model has then been used to compute the packet loss probability (using a PDF of the execution times consistent with the numbers measured above), and the results are displayed in the third column of Table 8.3. These results show that there is a good match between the theoretical model and the experiments, and that the stochastic model is pessimistic (it always gives a probability to drop an interrupt that is slightly higher than the one measured in the real experiments). This last property is important for providing stochastic real-time guarantees. Note that the deterministic model can be used by considering the minimum inter-arrival time between interrupts (100µs) and the maximum interrupt service time (25µs). The results obtained by using Equation 8.1 indicate that only the $(300\mu s, 1000\mu s)$ configuration results in no lost interrupt, but do not provide any information about the amount of lost interrupts. On the other hand, the stochastic model shows that some other configurations can also be useful (for example, $(3000\mu s, 10000\mu s)$.

8.5.2 *Back to the Motivational Example*

The analysis presented in the previous section can be used to assign the reservation parameters so that the problem with `netperf` and small network packets showed in Section 8.3.1 is solved. After estimating the PDF of the inter-arrival times of the packets generated by `netperf` and of the computation time, we used the stochastic model

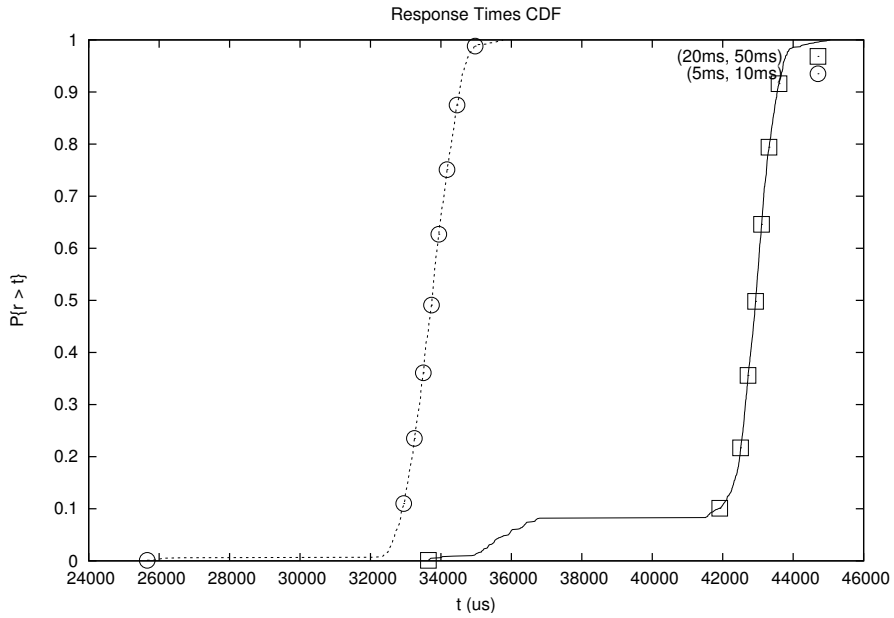

Figure 8.7: CDF of the response times for a $\tau = (20\text{ms}, 50\text{ms})$ real-time task in a standard Linux kernel with high network traffic, when serving τ and the IRQ thread with CBS.

to compute the packet loss probability when a $(4\text{ms}, 10\text{ms})$ CBS is used for the network hard IRQ thread. The resulting probability was 0. The experiments confirmed this result, showing a network throughput of 74Mbps (which is equal to the network throughput measured when the network hard IRQ thread is scheduled with the maximum fixed priority). In the same experiment, we used a $(20\text{ms}, 50\text{ms})$ CBS to serve the periodic real-time task, and the worst case response time has been measured as 46ms (which is consistent with the expectations: since $Q^s = 20\text{ms}$ is larger than the WCET, the worst case response time is expected to be less or equal than $T^s = 50\text{ms}$[4]).

The worst case response time of the periodic task can be reduced by using a smaller server period and reserving a larger fraction of the CPU time to the periodic task. To verify this hypothesis, the experiment has been repeated scheduling the periodic task with a $(5\text{ms}, 10\text{ms})$ CBS, and using a $(2\text{ms}, 10\text{ms})$ CBS for the IRQ thread (a fraction of the CPU time has been left unused for the `netperf` server, or for non real-time activities or other IRQ threads). The stochastic model allowed to compute a probability to drop an interrupt equal to 17.8%, and the throughput measured in the experiment was 65Mbps (a drop rate of about 12% w.r.t. the previous 74Mbps). The difference between the expected 17.8% and the measured 12% is probably due to the slight divergence of the IRQ thread from the theoretical

model, since it used to serve more than one interrupt at time, with a polling mechanism. The expected worst case response time was less than $\lceil C/Q^s \rceil T^s = 40ms$, and indeed a worst case response time of $36ms$ has been measured.

Figure 8.7 shows the CDFs of the real-time task's response times measured in the two experiments described above (using a $(20ms, 50ms)$ CBS or a $(5ms, 10ms)$ CBS to serve the real-time task).

Summing up, the previous two examples show how the CBS can be used to find latency/throughput trade-offs, and the proposed theoretical model can be used to analyse the CBS parameters assignments. Practical experiments on real hardware show a reasonable consistence with the theoretical analysis.

8.5.3 *Controlling the System Performance*

After verifying the correctness of the theoretical model, some additional experiments have been performed to verify that the proposed approach allows to control the real-time and I/O performance. This evaluation has been performed by first verifying that if $\sum_i Q_i^s/T_i^s \leqslant 1$ then the worst case response times of a task τ_i can be controlled by modifying its CBS parameters Q_i^s and T_i^s, and are not affected by the other tasks running in the system and by the I/O load. For this purpose, some periodic tasks $\tau_i = (C_i, P_i)$ have been run, measuring the worst case response times. Then, `netperf` has been used to evaluate the network throughput between the traffic generator and the test system when the IRQ threads are scheduled through the CBS (and to check that the worst case response times of the periodic tasks are not affected). The test focused on the network hard IRQ thread (running the ISR for the network card) and UDP traffic, and the results for 2 different packet sizes (1024 bytes and 512 bytes) are displayed in Figure 8.8.

From the figure it is possible to notice that when the reserved CPU time is enough the network throughput using IRQ threads and CBS is comparable with the network throughput of a standard Linux kernel (quite near to the theoretical maximum). Hence, the proposed approach does not affect the network performance too badly. It is also worth noting that the network throughput is linearly proportional to Q^s/T^s, and that the scaling factor depends on the packet size. This last result is consistent with the fact that each ethernet frame needs an almost-constant time to be processed, independently from the size,

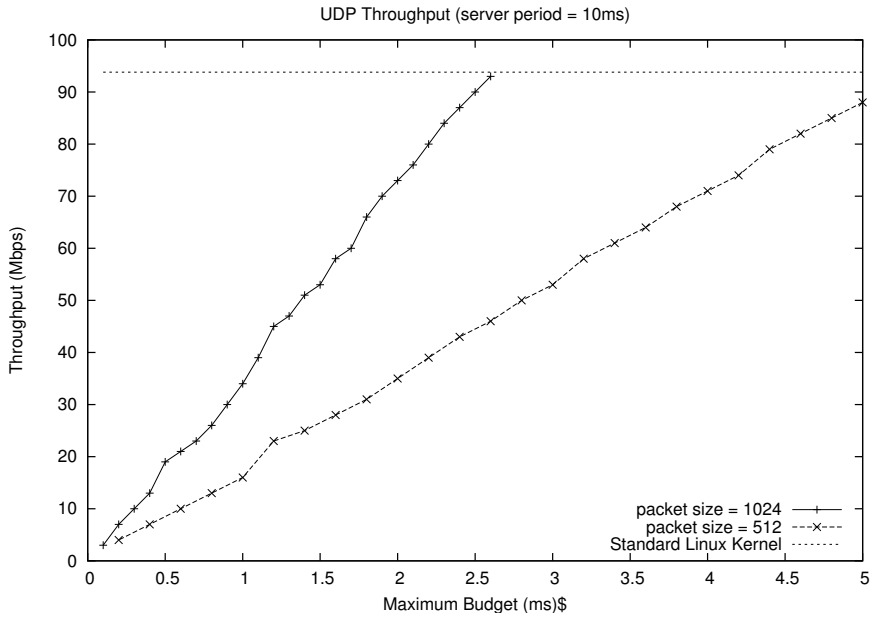

Figure 8.8: Network throughput as a function of the maximum budget Q^s ($T^s = 10ms$).

as previously measured (in particular, note that the slope of the curve for packet size = 512 is about half of the other curve, showing that the CPU time needed by the IRQ thread is about double).

In the following experiment, a stream of UDP packets has been sent from the traffic generator to the test system, with a fixed inter-packet time $P = 100\mu s$, measuring the times when the packets were received by a user-space real-time application running on the test system. The difference between the reception times of two consecutive packets (expected to be $100\mu s$) depends on the (Q^s, T^s) scheduling parameters of the IRQ thread. The results showed that if the fraction of CPU time reserved to the ethernet IRQ thread is not enough, then the times between the reception of two consecutive packets can be very large. But if enough CPU time is reserved, then the inter-packet times are very stable (more stable than for an unpatched Linux kernel). The maximum inter-packet times are reported in Table 8.4.

8.6 CONCLUSIONS AND FUTURE WORK

This chapter presented the application of reservation-based scheduling (more specifically, the CBS algorithm) to interrupt threads. Two analysis techniques have been presented for correctly dimensioning

the reservation parameters, and the analysis has been validated through a set of experiments on a real-time version of the Linux kernel.

As a future work, the stochastic model will be simplified (the current model is quite complex, and the number of states and transitions can probably be reduced), and a continuous-time model will be tested as an alternative. Moreover, it will be extended to analyse more complex situations (chains of interrupt threads, etc...). Additional experiments will also be performed using different hardware and multiple cooperating resources.

$$
s_{i+1} := \begin{cases}
(1, j, 0, 0, Q^s) & (0,0,0,0,0) \text{ with prob } V(j) \ (1)\\
(0, h-1, 0, t+1, q) & (0, >1, z, t, q) \ (2)\\
(N_c - 1, h-1, 0, t+1, q-1) & (-1, >1, 1, <T^s - 1, q) \text{ with prob } P\{Exec(q,t) = 1\} \ (4)\\
(N_c, h-1, z, t+1, q) & (-1, >1, 1, <T^s - 1, q) \text{ with prob } P\{Exec(q,t) = 0\} \ (5)\\
(-1, j, z-1, t+1, q-1) & (-1, 1, >1, <T^s - 1, q) \text{ with prob } P\{Exec(q,t) = 1\}V(j) \ (6)\\
(-1, j, z, t+1, q) & (-1, 1, >1, <T^s - 1, q) \text{ with prob } P\{Exec(q,t) = 0\}V(j) \ (7)\\
(N_c, h-1, z-1, t+1, q-1) & (-1, >1, >1, <T^s - 1, q) \text{ with prob } P\{Exec(q,t) = 1\} \ (8)\\
(N_c, h-1, z, t+1, q) & (-1, >1, >1, <T^s - 1, q) \text{ with prob } P\{Exec(q,t) = 0\} \ (9)\\
(N_c, j, 0, t+1, q-1) & (-1, 1, 1, <T^s - 1, q) \text{ with prob } P\{Exec(q,t) = 1\}V(j) \ (10)\\
(-1, j, z, t+1, q) & (-1, 1, 1, <T^s - 1, q) \text{ with prob } P\{Exec(q,t) = 0\}V(j) \ (11)\\
(N_c - 1, h-1, 0, t+1, q-1) & (-1, >1, 0, <T^s - 1, q) \text{ with prob } P\{Exec(q,t) = 1\}U(1) \ (12)\\
(N_c, h-1, k-1, t+1, q-1) & (-1, >1, 0, <T^s - 1, q) \text{ with prob } P\{Exec(q,t) = 1\}U(k) \ (13)\\
(N_c, h-1, z, t+1, q) & (-1, >1, 0, <T^s - 1, q) \text{ with prob } P\{Exec(q,t) = 0\} \ (14)\\
(N_c, j, 0, t+1, q-1) & (-1, 1, 0, <T^s - 1, q) \text{ with prob } P\{Exec(q,t) = 1\}U(1)V(j) \ (15)\\
(-1, j, k-1, t+1, q-1) & (-1, 1, 0, <T^s - 1, q) \text{ with prob } P\{Exec(q,t) = 1\}U(k)V(j) \ (16)\\
(-1, j, z, t+1, q) & (-1, 1, 0, <T^s - 1, q) \text{ with prob } P\{Exec(q,t) = 0\}V(j) \ (17)\\
(N_c - 1, h-1, 0, 0, Q^s) & (-1, >1, 1, T^s - 1, q) \text{ with prob } P\{Exec(q,t) = 1\} \ (18)\\
(N_c, h-1, z, 0, Q^s) & (-1, >1, 1, T^s - 1, q) \text{ with prob } P\{Exec(q,t) = 0\} \ (19)\\
(-1, j, z-1, 0, Q^s) & (-1, 1, >1, T^s - 1, q) \text{ with prob } P\{Exec(q,t) = 1\}V(j) \ (20)\\
(-1, j, z, 0, Q^s) & (-1, 1, >1, T^s - 1, q) \text{ with prob } P\{Exec(q,t) = 0\}V(j) \ (21)\\
(N_c, h-1, z-1, 0, Q^s) & (-1, >1, >1, T^s - 1, q) \text{ with prob } P\{Exec(q,t) = 1\} \ (22)\\
(N_c, h-1, z, 0, Q^s) & (-1, >1, >1, T^s - 1, q) \text{ with prob } P\{Exec(q,t) = 0\} \ (23)\\
(N_c, j, 0, 0, Q^s) & (-1, 1, 1, T^s - 1, q) \text{ with prob } P\{Exec(q,t) = 1\}V(j) \ (24)\\
(-1, j, z, 0, Q^s) & (-1, 1, 1, T^s - 1, q) \text{ with prob } P\{Exec(q,t) = 0\}V(j) \ (25)\\
(N_c - 1, h-1, 0, 0, Q^s) & (-1, >1, 0, T^s - 1, q) \text{ with prob } P\{Exec(q,t) = 1\}U(1) \ (26)\\
(N_c, h-1, k-1, 0, Q^s) & (-1, >1, 0, T^s - 1, q) \text{ with prob } P\{Exec(q,t) = 1\}U(k) \ (27)\\
(N_c, h-1, z, 0, Q^s) & (-1, >1, 0, T^s - 1, q) \text{ with prob } P\{Exec(q,t) = 0\} \ (28)\\
(N_c, j, 0, 0, Q) & (-1, 1, 0, T^s - 1, q) \text{ with prob } P\{Exec(q,t) = 1\}U(1)V(j) \ (29)\\
(-1, j, k-1, 0, Q) & (-1, 1, 0, T^s - 1, q) \text{ with prob } P\{Exec(q,t) = 1\}U(k)V(j) \ (30)\\
(-1, j, z, 0, Q) & (-1, 1, 0, T^s - 1, q) \text{ with prob } P\{Exec(q,t) = 0\}V(j) \ (31)\\
(x-1, h-1, 0, 0, Q^s) & (N_c, >1, 0, T^s - 1, q) \text{ with prob } P\{Exec(q,t) = 1\}U(1) \ (32)\\
(x, h-1, k-1, 0, Q^s) & (N_c, >1, 0, T^s - 1, q) \text{ with prob } P\{Exec(q,t) = 1\}U(k) \ (33)\\
(x, h-1, z, 0, Q^s) & (N_c, >1, 0, T^s - 1, q) \text{ with prob } P\{Exec(q,t) = 0\} \ (34)\\
(x, j, 0, 0, Q^s) & (N_c, 1, 0, T^s - 1, q) \text{ with prob } P\{Exec(q,t) = 1\}U(1)V(j) \ (35)\\
(-1, j, k-1, 0, Q^s) & (N_c, 1, 0, T^s - 1, q) \text{ with prob } P\{Exec(q,t) = 1\}U(k)V(j) \ (36)\\
(-1, j, z, 0, Q^s) & (N_c, 1, 0, T^s - 1, q) \text{ with prob } P\{Exec(q,t) = 0\} \ (37)\\
(x-1, h-1, 0, 0, Q^s) & (N_c, >1, 1, T^s - 1, q) \text{ with prob } P\{Exec(q,t) = 1\} \ (38)\\
(x, h-1, z, 0, Q^s) & (N_c, >1, 1, T^s - 1, q) \text{ with prob } P\{Exec(q,t) = 0\} \ (39)\\
(x, j, 0, 0, Q^s) & (N_c, 1, 1, T^s - 1, q) \text{ with prob } P\{Exec(q,t) = 1\}V(j) \ (40)\\
(-1, j, z, 0, Q) & (N_c, 1, 1, T^s - 1, q) \text{ with prob } P\{Exec(q,t) = 0\}V(j) \ (41)\\
(x, h-1, z-1, 0, Q^s) & (N_c, >1, >1, T^s - 1, q) \text{ with prob } P\{Exec(q,t) = 1\} \ (42)\\
(x, h-1, z, 0, Q^s) & (N_c, >1, >1, T^s - 1, q) \text{ with prob } P\{Exec(q,t) = 0\} \ (43)\\
(-1, j, z-1, 0, Q^s) & (N_c, 1, >1, T^s - 1, q) \text{ with prob } P\{Exec(q,t) = 1\}V(j) \ (44)\\
(-1, j, z, 0, Q^s) & (N_c, 1, >1, T^s - 1, q) \text{ with prob } P\{Exec(q,t) = 0\}V(j) \ (45)\\
(x, h-1, z, 0, Q^s) & (0, >1, z, T^s - 1, q) \ (46)\\
(x+1, j, z, 0, Q^s) & (0, 1, z, T^s - 1, q) \text{ with prob } V(j) \ (47)\\
(x-1, h-1, 0, 0, Q^s) & (<N_c, >1, 0, T^s - 1, q) \text{ with prob } P\{Exec(q,t) = 1\}U(1) \ (48)\\
(x, h-1, k-1, 0, Q^s) & (<N_c, >1, 0, T^s - 1, q) \text{ with prob } P\{Exec(q,t) = 1\}U(k) \ (49)\\
(x, h-1, 0, 0, Q^s) & (<N_c, >1, 0, T^s - 1, q) \text{ with prob } P\{Exec(q,t) = 0\} \ (50)\\
(x, j, 0, 0, Q^s) & (<N_c, 1, 0, T^s - 1, q) \text{ with prob } P\{Exec(q,t) = 1\}U(1) \ (51)\\
(x+1, j, k-1, 0, Q^s) & (<Nc, 1, 0, T^s - 1, q) \text{ with prob } P\{Exec(q,t) = 1\}U(k)V(j) \ (52)\\
(x+1, j, z, 0, Q^s) & (<N_c, 1, 0, T^s - 1, q) \text{ with prob } P\{Exec(q,t) = 0\}V(j) \ (53)\\
(x-1, h-1, 0, 0, Q^s) & (<N_c, >1, 1, T^s - 1, q) \text{ with prob } P\{Exec(q,t) = 1\} \ (54)\\
(x, h-1, z, 0, Q^s) & (<N_c, >1, 1, T^s - 1, q) \text{ with prob } P\{Exec(q,t) = 0\} \ (55)\\
(x, j, 0, 0, Q^s) & (<N_c, 1, 1, T^s - 1, q) \text{ with prob } P\{Exec(q,t) = 1\}V(j)) \ (56)\\
(x+1, j, z, 0, Q^s) & (<N_c, 1, 1, T^s - 1, q) \text{ with prob } P\{Exec(q,t) = 0\}V(j)) \ (57)
\end{cases}
$$

Figure 8.9: The stochastic model (1/2).

$$
s_{i+1} := \begin{cases}
(x, h-1, z-1, 0, Q^s) & (<N_c, >1, >1, T^s-1, q) \text{ with prob } P\{Exec(q,t)=1\}\ (58)\\
(x, h-1, z, 0, Q^s) & (<N_c, >1, >1, T^s-1, q) \text{ with prob } P\{Exec(q,t)=0\}\ (59)\\
(x+1, j, z-1, 0, Q^s) & (<N_c, >1, >1, T^s-1, q) \text{ with prob } P\{Exec(q,t)=1\}V(j)\ (60)\\
(x+1, j, z, 0, Q^s) & (<N_c, >1, >1, T^s-1, q) \text{ with prob } P\{Exec(q,t)=0\}V(j)\ (61)\\
(x-1, h-1, 0, t+1, q-1) & (N_c, >1, 0, <T^s-1, q) \text{ with prob } P\{Exec(q,t)=1\}U(1)\ (62)\\
(x, h-1, k-1, t+1, q-1) & (N_c, >1, 0, <T^s-1, q) \text{ with prob } P\{Exec(q,t)=1\}U(k)\ (63)\\
(x, h-1, z, t+1, q) & (N_c, >1, 0, <T^s-1, q) \text{ with prob } P\{Exec(q,t)=0\}\ (64)\\
(x, j, 0, t+1, q-1) & (N_c, 1, 0, <T^s-1, q) \text{ with prob } P\{Exec(q,t)=1\}U(1)V(j)\ (65)\\
(-1, j, k-1, t+1, q-1) & (N_c, 1, 0, <T^s-1, q) \text{ with prob } P\{Exec(q,t)=1\}U(k)V(j)\ (66)\\
(-1, j, z, t+1, q) & (N_c, 1, 0, <T^s-1, q) \text{ with prob } P\{Exec(q,t)=0\}V(j)\ (67)\\
(x-1, h-1, 0, t+1, q-1) & (N_c, >1, 1, <T^s-1, q) \text{ with prob } P\{Exec(q,t)=1\}\ (68)\\
(x, h-1, z, t+1, q) & (N_c, >1, 1, <T^s-1, q) \text{ with prob } P\{Exec(q,t)=0\}\ (69)\\
(x, j, 0, t+1, q-1) & (N_c, 1, 1, <T^s-1, q) \text{ with prob } P\{Exec(q,t)=1\}V(j)\ (70)\\
(-1, j, z, t+1, q) & (N_c, 1, 1, <T^s-1, q) \text{ with prob } P\{Exec(q,t)=0\}V(j)\ (71)\\
(x, h-1, z-1, t+1, q-1) & (N_c, >1, >1, <T^s-1, q) \text{ with prob } P\{Exec(q,t)=1\}\ (72)\\
(x, h-1, z, t+1, q) & (N_c, >1, >1, <T^s-1, q) \text{ with prob } P\{Exec(q,t)=0\}\ (73)\\
(-1, j, z-1, t+1, q-1) & (N_c, 1, >1, <T^s-1, q) \text{ with prob } P\{Exec(q,t)=1\}V(j)\ (74)\\
(-1, j, z, t+1, q) & (N_c, 1, >1, <T^s-1, q) \text{ with prob } P\{Exec(q,t)=0\}V(j)\ (75)\\
(x, h-1, z, t+1, q) & (0, >1, 0, <T^s-1, q)\ (76)\\
(x+1, j, z, t+1, q) & (0, 1, 0, >T^s-(\frac{q}{Q}T^s), q) \text{ with prob } V(j)\ (77a)\\
(x+1, j, z, 0, Q) & (0, 1, 0, <T^s-(\frac{q}{Q}T^s), q) \text{ with prob } V(j)\ (77b)\\
(x-1, h-1, 0, t+1, q-1) & (<N_c, >1, 0, <T^s-1, q) \text{ with prob } P\{Exec(q,t)=1\}U(1)\ (78)\\
(x, h-1, k-1, t+1, q-1) & (<N_c, >1, 0, <T^s-1, q) \text{ with prob } P\{Exec(q,t)=1\}U(k)\ (79)\\
(x, h-1, 0, t+1, q) & (<N_c, >1, 0, <T^s-1, q) \text{ with prob } P\{Exec(q,t)=0\})\ (80)\\
(x-1, h-1, 0, t+1, q-1) & (<N_c, >1, 1, <T^s-1, q) \text{ with prob } P\{Exec(q,t)=1\}\ (81)\\
(x, h-1, z, t+1, q) & (<N_c, >1, 1, <T^s-1, q) \text{ with prob } P\{Exec(q,t)=0\}\ (82)\\
(x, h-1, z-1, t+1, q-1) & (<N_c, >1, >1, <T^s-1, q) \text{ with prob } P\{Exec(q,t)=1\}\ (83)\\
(x, h-1, z, t+1, q) & (<N_c, >1, >1, <T^s-1, q) \text{ with prob } P\{Exec(q,t)=0\}\ (84)\\
(x+1, j, z-1, t+1, q-1) & (<N_c, 1, >1, <T^s-1, q) \text{ with prob } P\{Exec(q,t)=1\}V(j)\ (85)\\
(x+1, j, z, t+1, q) & (<N_c, 1, >1, <T^s-1, q) \text{ with prob } P\{Exec(q,t)=0\}V(j)\ (86)\\
(x, j, 0, t+1, q-1) & (<N_c, 1, 1, <T^s-1, q) \text{ with prob } P\{Exec(q,t)=1\}V(j)\ (87)\\
(x+1, j, z, t+1, q) & (<N_c, 1, 1, <T^s-1, q) \text{ with prob } P\{Exec(q,t)=0\}V(j)\ (88)
\end{cases}
$$

Figure 8.10: The stochastic model (2/2).

Table 8.1: Probability to drop an interrupt for various (Q^s, T^s) assignments.

Q^s	T^s	$\frac{Q^s}{T^s} \geqslant 0.75$	$Q^s \geqslant T^s - 12$	P{Drop}
3	4	Yes	Yes	0.0
6	8	Yes	Yes	0.0
9	12	Yes	Yes	0.0
12	16	Yes	Yes	0.0
15	20	Yes	Yes	0.0
18	24	Yes	Yes	0.0
21	28	Yes	Yes	0.0
2	4	No	Yes	0.0027
4	8	No	Yes	0.0062
6	12	No	Yes	0.0102
8	16	No	Yes	0.0142
10	20	No	Yes	0.0215
12	24	No	No	0.0244
14	28	No	No	0.0269
4	6	No	Yes	$4.11e-16$
8	12	No	Yes	$2.12e-15$
12	18	No	Yes	0.0058
16	24	No	Yes	0.0056
20	30	No	Yes	0.0055
24	36	No	No	0.0054
4	7	No	Yes	$2.011e-08$
8	14	No	Yes	$5.29e-07$
12	21	No	Yes	0.0064
16	28	No	No	0.0060
20	35	No	No	0.0056
24	42	No	No	0.0063

Table 8.2: PDF of the inter-packet times.

$\delta(\mu s)$	$P\{r_{i+1} - r_i = \delta\}$
100	0.312393
110	0.684386
120	0.001791
130	0.000210
140	0.000240
150	0.000290
160	0.000420
170	0.000090
180	0.000070
190	0.000110

Table 8.3: Probability to drop an interrupt, according to the theoretical model and to experimental measurements.

$Q^s(\mu s)$	$T^s(\mu s)$	Experimental Results	Stochastic Model
100	1000	0.55	0.5899
200	1000	0.18	0.1997
300	1000	0	0
1000	10000	0.63	0.6717
2000	10000	0.27	0.3105
3000	10000	0	0.0007

Table 8.4: Maximum measured inter-packet times as a function of the scheduler.

CBS Parameters	Maximum Inter-Packet
$(1ms, 10ms)$	52593μs
$(1.5ms, 10ms)$	42922μs
$(2ms, 10ms)$	29869μs
$(3ms, 10ms)$	489μs
$(10ms, 100ms)$	323716μs
$(20ms, 100ms)$	147064μs
$(30ms, 100ms)$	491μs
No CBS	656μs

9

PRACTICAL EXAMPLES

In this chapter we present two possible examples in which resource reservation has an undisputed advantage respect to fixed priority scheduling. One examples is the interrupt scheduling and one is the scheduler inside the X server.

9.1 SCHEDULABLE DEVICE DRIVERS: IMPLEMENTATION AND EXPERIMENTAL RESULTS

An important issue when designing real-time systems is to control the kernel latencies introduced by device drivers. This result can be achieved by transforming the interrupt handlers into schedulable entities (threads). This section shows how to schedule such threads (using resource reservations) so that both the performance of real-time tasks and the device throughput can be controlled. In particular, some tools based on a kernel tracer (`Ftrace`) are used to collect timing information about the IRQ threads, and a novel reservation-based scheduler for Linux (`SCHED_DEADLINE`) is used to schedule them. An implementation of the proposed technique is validated through an extensive set of experiments, using different kinds of resources and of realistic applications.

9.1.1 *Introduction*

One of the prominent issues in the design of modern real-time operating systems is accounting for interference of device drivers on the real-time tasks. Indeed, the interference possibly generated by a device driver (which contributes to the so called "Kernel latencies") introduces some unbounded blocking times that can compromise the schedulability of task sets deemed schedulable by the formal analy-

Parts of this Chapter are going to appear in:
N. Manica, L. Abeni, L. Palopoli, D. Faggioli, C. Scordino, "Schedulable Device Drivers: Implementation and Experimental Results," *International Workshop on Operating Systems Platforms for Embedded Real-Time Applications*

sis techniques (because the worst case blocking times are unknown, hence it is not possible to account for them in the formal analysis).

A straightforward strategy to address this problem is to give real-time tasks higher priorities than device drivers. However, this is not possible on general purpose systems, where interrupt handlers and device drivers are not schedulable entities (and are executed with a higher priority than all the user-space tasks). For this reason, recent developments in the Linux kernel allow transforming the interrupt handlers (both Interrupt Service Routines — ISRs — and Bottom Halves) into kernel threads, the so called IRQ threads (note that in the past similar solutions have been mainly used in µkernel based systems [45, 46] or in proprietary real-time kernels such as LynxOS). This functionality was originally developed for the Preempt-RT real-time kernel [49, 53], and enables a control on the amount of interference from device drivers suffered by real-time tasks [51, 52, 38].

Once interrupt handlers have been transformed into schedulable entities, the problem remains open of identifying the best scheduling algorithm that can be used to serve the newly introduced threads. For example, Manica et al. [54, 55] have provided a clear evidence that using resource reservations [5] to schedule the interrupt handlers (IRQ threads, in Preempt-RT) allows the designer to find appropriate trade-offs between the response time of real-time tasks and the device throughput (this is important when the device is used by real-time tasks). However, to the best of our knowledge, most experiments and tests with advanced scheduling solutions have been performed only using prototypical schedulers or experimental Operating Systems [56, 43]. Only recently has a Linux scheduler based on resource-reservation has been proposed to the kernel community [57]. Such a scheduler exports an API that can be easily used to schedule kernel threads implementing the device drivers. Additionally, most of the previous work has focused on network devices [47, 48, 44, 51, 55] paying little or no attention to other types of devices (e.g., disks). Finally, another limitation of previous results is that they are mostly collected on artificial task sets.

This section takes a step forward to show that the results collected with experiments based on prototypical schedulers can be repeated: 1) with a scheduler likely to become main line in the near future, 2) using different kinds of resources, 3) with realistic applications rather than with artificial task sets.

As a last contribution, a set of tools based on the *Ftrace* kernel tracing facility is used to collect the stochastic distribution of the execution time and of the inter-arrival time of device drivers. This way it is possible to apply design techniques that enable an appropriate dimensioning of the scheduling parameters.

9.1.2 *Scheduling the IRQ Threads*

This section briefly recalls some basic concepts about resource reservations, and about assigning proper reservation parameters to the interrupt threads. It also introduces the reservation-based scheduler for Linux (named `SCHED_DEADLINE`) that has been used for scheduling the interrupt threads.

9.1.2.1 *Reservation-Based Scheduling*

The basic idea of reservation-based scheduling is that each task is reserved an amount Q of CPU time (named *maximum budget*) every T time units (T is called *reservation period*). Such a strategy can be implemented by using various scheduling algorithms. The particular reservation algorithm used in this section is the Constant Bandwidth Server (CBS)(See Chapter 4 or [4]), which, contrary to different scheduling algorithms of the same kind, is well behaved with both regular and periodic tasks and with aperiodic and dynamically changing tasks.

The CBS algorithm assigns each task a *scheduling deadline*, and schedules processes and threads using an Earliest Deadline First (EDF) policy (i.e., the task with the earliest scheduling deadline is selected first for execution). When a task wakes up, the CBS checks if its current scheduling deadline can be used; otherwise, a new scheduling deadline is generated (as $d = t + T$, where t is the wakeup time). The scheduling deadline is then postponed by T ($d = d + T$) every time that the task executes for Q time units (if having a work conserving algorithm is not important, the task is removed from the runqueue until time $d - T$).

An interesting feature of the reservation-based schedulers is that they provide *temporal isolation* among tasks. This means that the temporal behaviour of a task is not affected by the behaviour of the other tasks in the system: if a task requires a large execution time, it cannot affect the schedulability of the other tasks, or monopolise the proces-

sor. This is a basic property needed for scheduling real-time tasks on general-purpose operating systems.

9.1.2.2 *The Linux SCHED_DEADLINE Policy*

In the recent versions, the official Linux kernel has introduced a new scheduling framework that replaces the old O(1) scheduler. This framework contains an extensible set of *scheduling classes*. Each scheduling class implements a specific algorithm and schedules tasks with a specific policy.

Currently, two scheduling classes are available in the Linux kernel:

- `sched_fair`, which implements the *"Completely Fair Scheduler"* (CFS) algorithm, and schedules tasks having `SCHED_OTHER` or `SCHED_BATCH` policies. Tasks are run at precise weighted speeds, so that each task receives a "fair" amount of processor share.
- `sched_rt`, which implements a POSIX fixed-priority real-time scheduler, and handles tasks having `SCHED_FIFO` or `SCHED_RR` policies.

As explained in previous papers [57], using these scheduling policies with tasks characterised by temporal constraints might be problematic, mainly because the standard API used in general purpose kernels like Linux does not allow to associate temporal constraints (e.g., deadlines) to the tasks. In fact, although it allows to assign a share of processor time to a task, there is no way to specify that the task must finish the execution of a job before a given time. Using CFS, moreover, the time elapsed between two consecutive executions of a task is not deterministic and cannot be bound, since it depends on the number of tasks running in the system at that time.

For these reasons, very recently, a new scheduling class based on resource reservations has been implemented and proposed to the kernel community. The project, formerly known as `SCHED_EDF`, changed name to `SCHED_DEADLINE` after the request of the kernel community.

This class adds the possibility of scheduling tasks using the CBS algorithm, without changing the behaviour of tasks scheduled using the existing policies. Figure 9.1 depicts schematically the Linux scheduler extended with the `SCHED_DEADLINE` scheduling class (note that scheduling classes have increasing priorities from left to right).

`SCHED_DEADLINE`. The code is open and available at `http://gitorious.org/sched_deadline`.

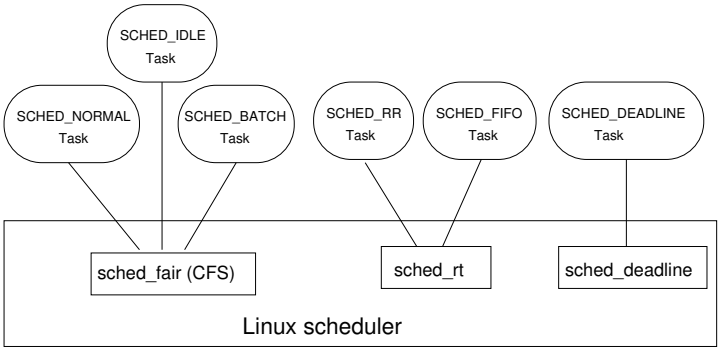

Figure 9.1: Linux scheduler with SCHED_DEADLINE.

The implementation does not make any restrictive assumption on the characteristics of the tasks. Thus, it can handle periodic, sporadic and aperiodic tasks. It is aligned with the current mainstream kernel, and it relies on standard Linux mechanisms to natively support multicore platforms and to provide hierarchical scheduling through a standard API.

MAIN CHARACTERISTICS OF THE IMPLEMENTATION In the implementation, red-black trees are used for ready queues to enable efficient handling of events such as earliest deadline task scheduling, new task activation, task blocking/unblocking, etc. One run-queue per each CPU is used to avoid contention and achieve high scalability even on large systems. Moreover, it is enriched with the following features:

- support for bandwidth reclaiming, to make the scheduler work conserving without affecting guarantees;
- capability of synchronising tasks with the scheduler;
- support for resource sharing similar to priority inheritance (already present in the kernel for fixed priority real-time tasks);
- support for standard Linux mechanisms for debugging and tracing the scheduler behaviour and for specifying per-user policies and limitations;
- capability of sending signals to the tasks on budget overruns and scheduling deadline misses;
- support for bandwidth management throughout admission control, both system-wide and for separate groups of tasks.

USER LEVEL API An user-level application can exploit the services provided by the SCHED_DEADLINE scheduling class by means of some new system calls and a new data structure that accommodates additional scheduling parameters. The new data structure is called sched_param_ex and comprises the following fields:

- temporal parameter of the task — i.e., sched_runtime and sched_deadline which will be Q and T of its reservation, respectively;
- sched_flag for controlling some aspects of the scheduler behaviour. More precisely, (i) whether or not a task wants to be notified about budget overruns and/or scheduling deadline misses and (ii) whether or not a task wants to exploit some kind of bandwidth reclaiming;
- some other fields left there for backward compatibility or future extensions (sched_priority and sched_period).

The most important system calls added are:

- sched_setscheduler_ex (and a couple of others), that manipulates sched_param_ex;
- sched_wait_interval to synchronise the task with the scheduler. This means a task can ask to be put to sleep until either its next deadline or whenever it will be possible to receive its full budget again.

The security model adopted is very similar to the one already in use in the kernel for fixed priority real-time tasks — i.e., it is based on user permissions and capabilities and it can be affected by standard UNIX security mechanism, like rlimits. Controls exist for managing the fraction of CPU time usable by the whole EDF scheduler as well as to a group of EDF tasks, but they are not described here for space reasons.

9.1.2.3 *Assigning the Reservation Parameters*

The reservation parameters (Q, T) can be dimensioned by performing a deterministic or a stochastic analysis of the interrupt behaviour [55]. The deterministic case is simpler to analyse, and allows to dimension the reservation so that no interrupt is lost (at the cost of some overestimation of the reserved CPU time). First of all, if P is the minimum inter-arrival time between two consecutive interrupts and C is the

maximum amount of time needed to serve an interrupt, then Q and T must be assigned according to Equation 9.1

$$\frac{Q}{T} \geqslant \frac{C}{P} \tag{9.1}$$

In other words, the fraction of CPU time reserved to the IRQ thread should be greater than or equal to the fraction of CPU time needed by the IRQ thread for executing.

However, some hardware devices have an upper bound N_c on the number of pending interrupts (interrupts that have not been processed yet), and if an interrupt fires when N_c interrupts requests are already pending, then the interrupt is lost even if Equation 9.1 is respected. As discussed in our previous work [55], this problem ban be addressed by producing the following condition that has to be respected to avoid losing interrupts:

$$\frac{T - Q}{P} < N_c \tag{9.2}$$

The same paper also presented a stochastic analysis instrumental to the correct dimensioning of the CBS parameters: in this case, instead of considering the worst-case times P and C, the interrupt inter-arrival times and the execution times of the interrupt handlers are modelled as stochastic variables. As a result, the probability to drop an interrupt can be computed.

Both approaches require a precise characterisation of the workload generated by IRQ threads. Hence, the need for the tracing mechanism described in the next section.

9.1.3 *Inferring the IRQ Parameters*

According to Section 9.1.2.3, if the probability distributions of the inter-arrival and execution times of an IRQ thread are known, then it is possible to schedule such thread with a (Q, T) reservation so that no interrupt is lost (note that if no interrupt is lost then the device can achieve its maximum throughput). Hence, to assign the maximum budget Q and the reservation period T to an IRQ thread it is necessary to know the *IRQ parameters* (that is, the probability distributions of the inter-interrupt times and of the times needed to serve an interrupt).

Such probability distributions can be measured by using the *Ftrace* tracer provided by the Linux kernel and by properly parsing its traces.

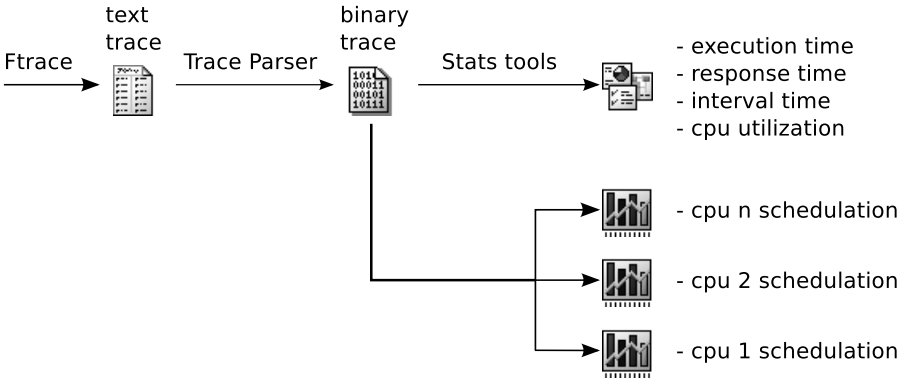

Figure 9.2: General architecture of the tool

In the proposed approach, this is done through a set of tools organised in a pipeline, as shown in Figure 9.2 (more details about the used tracing tools are available in a technical report [58]).

9.1.3.1 *The Tracing Pipeline*

The kernel traces produced by Ftrace can be used to extract various information about tasks' timings, so that their temporal behaviour can be inferred.

The first stage of the pipeline (the trace parser) transforms the textual traces exported by Ftrace in an internal format, which is used by the other tools in the pipeline. This step is needed because Ftrace exports traces in the form of text files, whose format can change from one kernel version to another, containing redundant and unneeded information (this happens because the Ftrace format has been designed to be easily readable by humans). Hence, the textual traces produced by Ftrace are parsed and transformed in a more compact, kernel-independent, binary format which is used as input by the next stages of the pipeline. Such stages are composed by a second set of tools that can:

- parse the internal format to gather statistics about execution times, inter-arrival times, response times, and utilisation;
- generate a chart displaying the CPU scheduling;
- infer some of the tasks temporal properties, identifying (for example) periodic tasks.

In this context, the presented tools are used to extract the probability distributions of the execution and inter-arrival times of the IRQ threads.

Table 9.1: Statistics collected for some periodic tasks. Times are in µs.

Task	Execution Time				Inter-Arrival Time				Response Time			
	Avg	Std Dev	Max	Min	Avg	Std Dev	Max	Min	Avg	Std Dev	Max	Min
Task 1	2991	273	8953	2956	5993	303	10720	11	3182	555	5986	2960
Task 2	553	66	6025	544	2997	10	3002	2991	556	229	6027	546
Task 3	2941	51	5859	2919	7993	24	9049	6938	3683	397	7285	2927

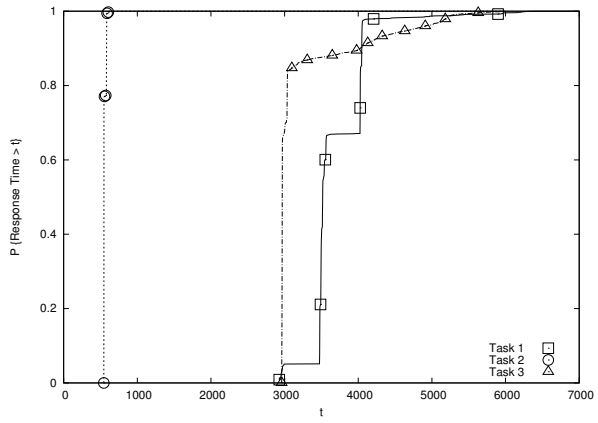

Figure 9.3: CDFs of the response times for 3 periodic tasks.

The various tools composing the pipeline communicate through standard Unix FIFOs (named pipes) and can be combined in different ways, to collect different kinds of information. For example, a tool which periodically displays important statistics for selected tasks (similarly to the standard "`top`" utility) can be inserted into the pipeline. In this work, the collected values are generally saved to files to be processed off-line later, but in other situations they can also be summarised by some statistics that are saved instead of the raw sequence of values, to save some disk space.

Since connecting the different tools in a correctly working pipeline (creating all the needed FIFOs, etc.) can sometimes be difficult, some helper scripts have been developed.

9.1.3.2 *Examples*

The first possible usage of the proposed tools is to visually analyse the scheduler's behaviour, to check its correctness or to understand the reason for unexpected results. An example about this usage will be presented in Section 9.1.4. If, instead, a statistics module is used in the last stage of the pipeline, it is also possible to collect some information for performance evaluation. For example, some statistics for some periodic tasks have been collected and shown in Table 9.1.

Table 9.2: Inter-Packet times as measured in the sender. Times are in µs.

Test	Average	Std Dev	Max	Min
T1	1190	29	1569	1040
T5	5198	22	5278	5058
T10	10195	22	10277	10062
T50	50207	27	50298	50081
T100	100207	25	100290	100093

Table 9.3: Inter-Packet times as measured in the receiver. Times are in µs.

Test	Average	Std Dev	Max	Min
T1	1207	1011	14336	0
T5	5212	1019	6144	4096
T10	10210	271	12288	8192
T50	50229	1023	51200	49152
T100	100204	530	100352	98304

The Cumulative Distribution Functions (CDFs) of the response times for the three tasks, as measured using a different output module, are displayed in Figure 9.3. Note that all the results presented up to now can be obtained by just changing the final stage of the processing pipeline.

As explained, in this work the presented tools are used to collect timing information about IRQ threads. However, before performing such measurements, it is important to test the reliability of this information. For this purpose, some experiments have been performed by considering the network IRQ threads: a stream of periodic UDP packets has been sent between two computers, measuring the inter-packet times in the sender (Table 9.2) and in the receiver (Table 9.3). Note that, as expected, the values in Table 9.3 almost match the values in Table 9.2: the only noticeable difference is test T1, in which the inter-packet times on the receiver have a large maximum value and 0 as a minimum value. This is probably due to some delayed scheduling of the receiver task. After these initial measurements, the proposed tools have been used to extract the inter-arrival times of the network IRQ thread, summarised in Table 9.4. By comparing Table 9.2 and Table 9.4, it is possible to verify that the average inter-activation times of the network IRQ thread in the receiver are consistent with the average inter-packet times in the sender. The maximum times also match, while the minimum times present some differences. In partic-

Table 9.4: Inter-Arrival times for the network IRQ thread. Times are in µs.

Test	Average	Std Dev	Max	Min
T1	1210	32	1424	59
T5	5222	117	5385	63
T10	10264	60	10353	10093
T50	50832	627	50353	50082
T100	100424	9342	100313	76

Table 9.5: Statistics about the execution times of the IRQ thread. Times are in µs.

Test	Average	Std Dev	Max	Min
T1	15	5	63	9
T5	19	1	68	18
T10	14	1	29	13
T50	16	2	28	15
T100	21	3	23	12

ular, in tests T1, T5 and T100 the minimum inter-arrival time for the network IRQ thread in receiver is much smaller than the minimum inter-packet time in the sender. A more detailed analysis revealed that this is probably due to some non UDP packets (ICMP or ARP) which are not directly generated by the test program in the sender machine (hence, they are not periodic and they are not listed in Table 9.2). In any case, the comparison between Table 9.2 and Table 9.4 seems to confirm the correctness of the collected data.

Some information about the IRQ thread execution times (needed to perform some kind of performance analysis of the system) are shown in Table 9.5, and some examples of distribution functions obtained using these tools will be presented in Section 9.1.4.

9.1.4 *Experimental Results*

The effectiveness of the proposed approach has been tested by an extensive set of experiments. In particular, the performance of the Linux `SCHED_DEADLINE` policy and the influence of the scheduling parameters have been evaluated when the new scheduling class is used to:

- schedule real-time tasks sets
- schedule IRQ threads

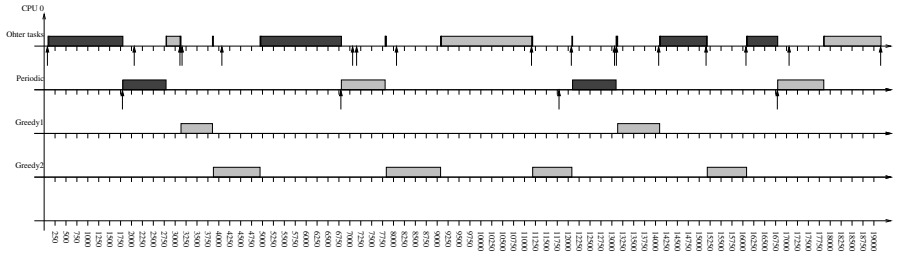

Figure 9.4: `SCHED_DEADLINE` serving a periodic task and two CPU hungry (greedy) tasks.

- schedule hybrid task sets composed of both real-time tasks and IRQ threads.

The next subsections will report the results obtained for each of these cases.

9.1.4.1 *Using SCHED_DEADLINE*

To see the new `SCHED_DEADLINE` policy in action, consider a periodic task (with period 5ms) and two greedy tasks (task which never block, and try to consume all the CPU time) scheduled by two CBSs $(1\text{ms}, 10\text{ms})$ and $(1\text{ms}, 4\text{ms})$. Figure 9.4 shows a segment of the schedule produced by the tools presented in Section 9.1.3.

On the other hand, Figure 9.5 shows how the CBS scheduler implemented by `SCHED_DEADLINE` is more effective in handling multiple time sensitive applications than the fixed priority policy `SCHED_FIFO` provided by the standard Linux kernel. A simple video player based on GTK is used as a real-time task, and two player's instances reproduce the "Big Buck Bunny" trailer either (i) with two different `SCHED_FIFO` priority or (ii) within two CBSs, $(12.5\text{ms}, 40\text{ms})$ and $(25\text{ms}, 40\text{ms})$.

Since the frame rate of the video is 25 frames per second (fps), the expected time interval between two consecutive frames (named *Inter-Frame Time - IFT* - from now on) is supposed to be $1000/25 = 40\text{ms}$. In this experiment, two instances of the video player are executed in parallel, using different scheduling algorithms and priorities. As it clearly emerges from the figure, when `SCHED_FIFO` is used, the player execute with higher priority correctly reproduces the stream, and the IFTs are constant around 40ms (average 39573.0µs, standard deviation 4725.1); however, the low priority instance has poor performance, to the point where playback stops completely at frame 310 (this is the

http://www.gtk.org
http://www.bigbuckbunny.org

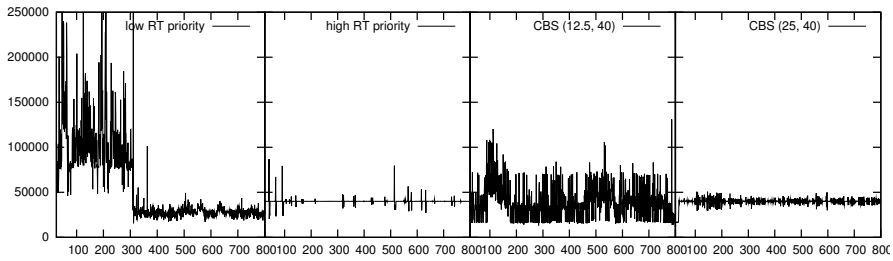

Figure 9.5: Inter-frame times for two instances of the video player when executing under `SCHED_FIFO` (with different priorities) or `SCHED_DEADLINE`. (within different reservations)

meaning of the peak in the graph) and starts again only when the other instance finished.

When the reservation-based approach enabled by `SCHED_DEADLINE` is used, instead, both the instances are able to proceed and the performance they achieve are proportional to the fraction of CPU time they can use. This is shown in the right side of the figure and by the fact that IFT average and standard deviation are, respectively, 39082.0µs, 5735.3 for $(25, 40)$ and 39517.0µs, 19455.0 for $(12.5, 40)$.

9.1.4.2 *Controlling the Device Throughput*

The next set of experiments has been performed to check the effects of scheduling the disk IRQ thread with a (Q, T) reservation, for different values of Q and T.

First of all, the disk throughput has been measured by using the `hdparm` command and disabling the disk caches. The results of this experiment showed that the disk throughput only depends on the fraction of CPU time Q/T reserved to the disk IRQ thread, and is not affected by the specific values of Q and T. This result seems to contradict the condition expressed by Equation 9.2, and is probably due to the fact that the disk controller has a large buffer (i.e., N_c is very large).

Figure 9.6 shows the disk throughput as a function of Q/T (confirming that the throughput is proportional to the fraction of CPU time reserved to the IRQ thread), while Figures 9.7 and 9.8 show the probability distributions of the interrupt inter-arrival and execution times. According to such probability distributions, the maximum utilisation of the disk IRQ thread is about 0.41096, while the average utilisation is about 0.0021833. By comparing these data with results shown in Figure 9.6, it is possible to see that deterministic analysis is too pessimistic and highly overestimates the needed amount of time: in

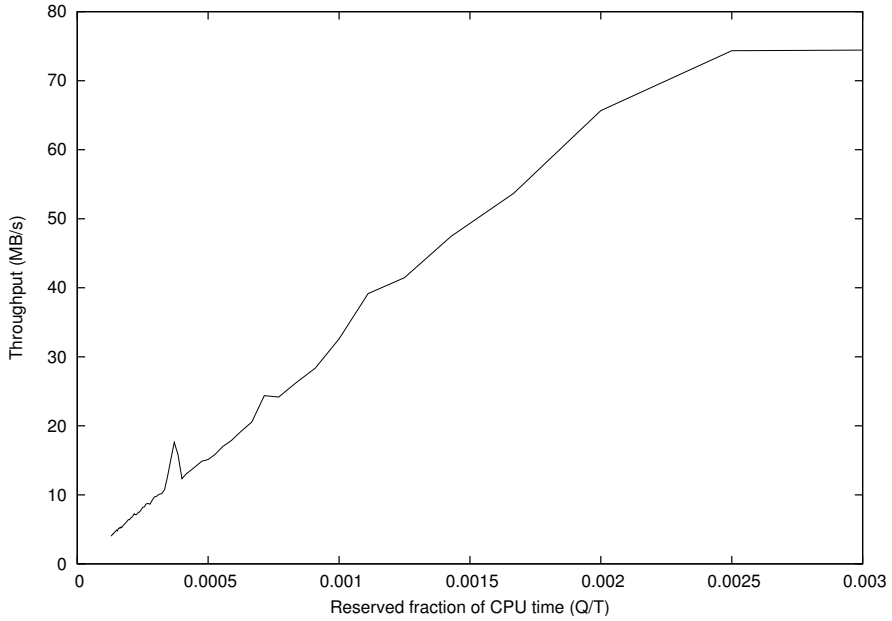

Figure 9.6: Disk throughput (as measured by hdparm) when the disk IRQ thread is scheduled by a CBS, as a function of the reserved fraction of CPU time.

fact, `hdparm` measures the maximum throughput when $Q/T = 0.003$, which is only a little bit more than the average utilisation. By looking at Figures 9.7 and 9.8 again, it is clear that the worst case conditions (leading to the 0.41096 utilisation) are due to a long tail in the execution times probability distribution and to a small amount of small inter-arrival times with low probability, hence they are very unlikely. This explains why a fraction of reserved CPU time which is very close to the average utilisation is sufficient for achieving full utilisation. This consideration motivates future investigations on the application of stochastic analysis technique [55]. This research activity is currently under way.

Note that in this experiment the utilisation of disk IRQ thread is quite low, and only a small fraction of the CPU time had to be reserved to it to control the hard-disk performance. Such a low CPU utilisation caused by the disk IRQ thread is due to the usage of DMA when performing disk accesses. Such a mechanism (the DMA) allows to reduce the amount of CPU time needed by the IRQ thread, but can cause some other kind of interference (that cannot be modelled as a task in schedulability analysis) on real-time tasks due to bus contention. By disabling the DMA, all the interference is due to

When running `hdparm` with the disk IRQ thread scheduled with the maximum fixed priority, the throughput resulted to be about 75MB/s.

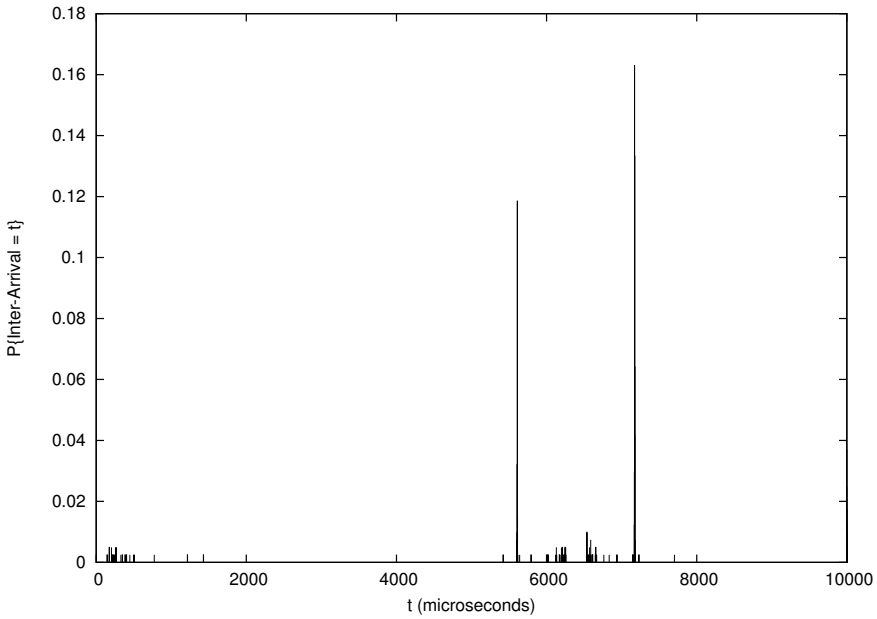

Figure 9.7: PMF of the inter-arrival times for the disk IRQ thread.

the IRQ thread, and can be properly accounted for in the schedulability analysis. Hence, the experiments have been repeated with DMA disabled (these experiments also allows to better understand what happens when the IRQ thread consumes more time); the results are reported in Table 9.6. Each line in the table is the average of the results of 20 repeated tests on a UP machine when DMA is disabled; the average utilisation for the disk IRQ thread is about 0.66, with a minimum utilisation of 0.57 and a maximum of 0.93. As expected, the throughput without DMA is much lower than the throughput achieved when using DMA, and it proportionally grows with the fraction of CPU time reserved to the interrupt thread. The maximum throughput (100% of the throughput measured when the disk IRQ thread is scheduled with a fixed priority) is achieved when $Q/T = 0.95$. Again, the throughput seems to only depend on the Q/T ratio, and not on the reservation period T: in other words, the average throughput achieved when using a $(2ms, 100ms)$ reservation is the same achieved when using a $(20ms, 1000ms)$ reservation.

After evaluating the "raw" disk performance through `hdparm`, the next experiments have been run to evaluate the performance of more complex read operations, involving multiple system calls and file system access. The operation involved is a simple `cat` of a large file (about 44MB) redirecting output to /dev/null. The total time for the operation has been measured, disabling disk caches and DMA. Sev-

Table 9.6: Disk IO-throughput when IRQ thread is scheduled with deadline scheduler.

Test	Average
$(2ms, 100ms)$	1.9858%
$(4ms, 100ms)$	4.23484%
$(6ms, 100ms)$	6.38374%
$(20ms, 1000ms)$	2.16843%
$(40ms, 1000ms)$	4.46488%
$(60ms, 1000ms)$	6.49811%
$(10ms, 100ms)$	10.6726%
$(20ms, 100ms)$	21.5825%
$(40ms, 100ms)$	41.8182%
$(60ms, 100ms)$	62.4476%
$(80ms, 100ms)$	82.5952%
$(90ms, 100ms)$	92.9371%
$(95ms, 100ms)$	100%
$(100ms, 1000ms)$	11.778%
$(200ms, 1000ms)$	23.261%
$(400ms, 1000ms)$	44.4056%
$(600ms, 1000ms)$	65%
$(800ms, 1000ms)$	84.5455%
$(900ms, 1000ms)$	93.007%
$(950ms, 1000ms)$	100%

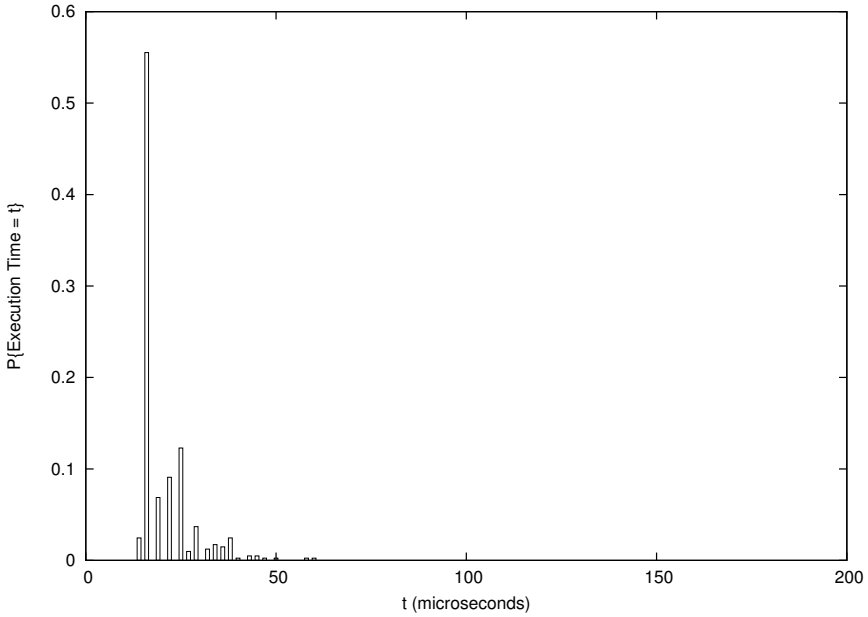

Figure 9.8: PMF of the execution times for the disk IRQ thread.

Table 9.7: Time needed to perform a file copy, when the disk IRQ thread is scheduled with different parameters.

Test	Average	Std Dev	Max	Min
No Reservations	16.89s	0.12s	17.05s	16.67s
(30ms, 100ms)	52.85s	0.87s	55.22s	52.36s
(40ms, 100ms)	39.52s	0.61s	41.25s	39.27s
(50ms, 100ms)	31.49s	0.12s	31.74s	31.40s
(60ms, 100ms)	26.23s	0.03s	26.30s	26.19s
(70ms, 100ms)	22.58s	0.20s	23.14s	22.47s
(80ms, 100ms)	19.77s	0.19s	20.31s	19.69s
(90ms, 100ms)	17.59s	0.04s	17.66s	17.55s

eral runs have been repeated, and the results are reported in Table 9.7. The experiments were performed in a pretty old machine, and the operation lead to a very big interrupt workload, loading the CPU up to about 100% of the CPU time. The first line of the table reports the time needed for the operation when the default scheduler is used (that is, `SCHED_RT` is used for IRQ threads) in a machine with no other load. In the following lines `SCHED_DEADLINE` is used and the time falls down as the reserved fraction of CPU grows.

Note that by modifying the amount of reserved CPU time it is possible to control the amount of time needed for executing the `cat` command. In particular, the throughput (computed as the ratio between

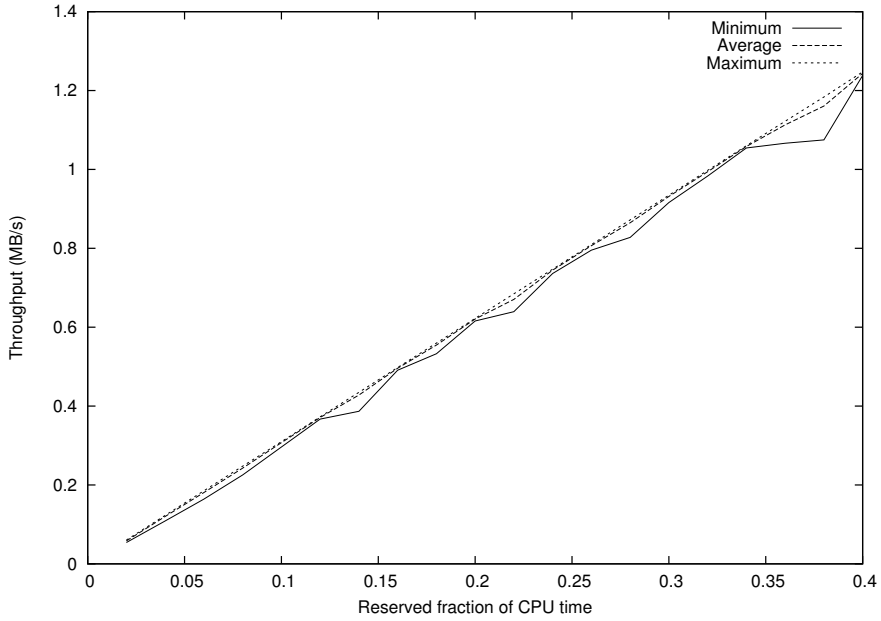

Figure 9.9: Throughput when reading a large file, as a function of the reserved CPU time.

the file size and the time needed to cat it) is proportional to Q/T, as shown in Figure 9.9.

Finally, the time needed to read a large file has been analysed by measuring the system time, the user time, and the total time used by the task performing the read operation (disabling the disk caches so that the experiment is more deterministic and repeatable). The size of the file involved in this experiment was about 80MB. As expected, the total time needed to read the file resulted to be much larger than the sum of the system time and the user time, because the task is often blocked waiting data from the disk (so, the task performing the read operation spends most of the time in the wait state. Most of the CPU time is consumed by the disk IRQ thread, and is not visible in the statistics of the user task performing the read operation). Moreover, the amount of user time and system time used by the task resulted to be very small, and did not depend on the reservation parameters (the user time was around 2.5ms, and the system time was around 100ms). On the other hand, the total time (shown in Figure 9.10) resulted to be proportional to T/Q (because the disk throughput is proportional to Q/T), and (again), disk interrupts did not suffer by Equation 2.

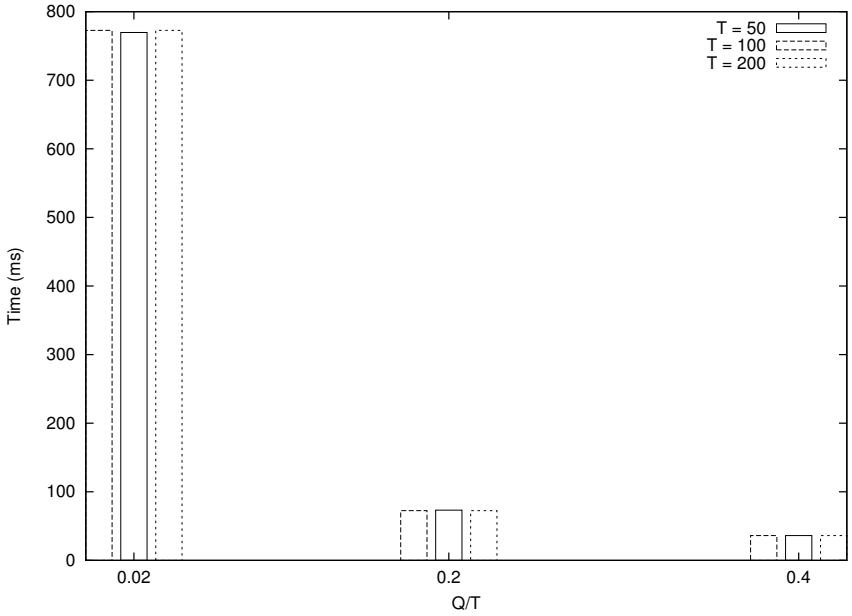

Figure 9.10: Total time needed to read a large file.

9.1.4.3 *Latency/Throughput Trade-Offs*

Finally, some experiments have been performed to show how the proposed approach allows one to control both the device throughput and the real-time performance of user-space tasks. The video player presented above has been used as a real-time task, and the network device has been considered (using `netperf` to generate network load and to measure the network throughput [55]).

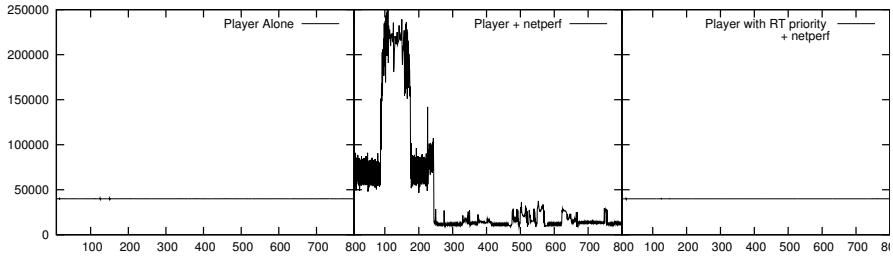

Figure 9.11: Inter-frame times for the video player when executed alone and concurrently with `netperf`, with different priorities.

An instance of `netperf` has then been activated concurrently with the video player, and the experiment has been repeated scheduling the video player as `SCHED_OTHER` and as `SCHED_FIFO` with a priority higher than the network IRQ threads. The results, displayed in Figure 9.11, show that when the player executes alone it is able to correctly reproduce the video (the inter-frame times are constant around

Table 9.8: Network throughput achieved by using different reservation parameters for the video player and for the network hard IRQ.

Test	Player CBS	net IRQ CBS	Throughput
Test1	(29ms, 40ms)	(9ms, 100ms)	59.75Mbps
Test2	(28ms, 40ms)	(12ms, 100ms)	65.43Mbps
Test3	(26ms, 40ms)	(13ms, 100ms)	70.83Mbps
Test4	(25ms, 40ms)	(14ms, 100ms)	76.14Mbps
Test5	(20ms, 40ms)	(18ms, 100ms)	88.55Mbps

40ms), while when some concurrent network load is created, the inter-frame times increases by a large amount (and video playback is not continuous). Finally, if the player is scheduled with a priority higher than the priority of the network IRQ threads, then it is again able to work correctly, but in this case **the network throughput measured by netperf drops from** 88Mbps **to about** 58Mbps.

A trade-off between real-time performance for the player and high network throughput can be found by using reservation-based scheduling. To this purpose, the tool presented in Section 9.1.3 can be used to collect the probability distributions of the execution and inter-arrival times of the network IRQ threads, and these data can be used as shown in Section 9.1.2.3.

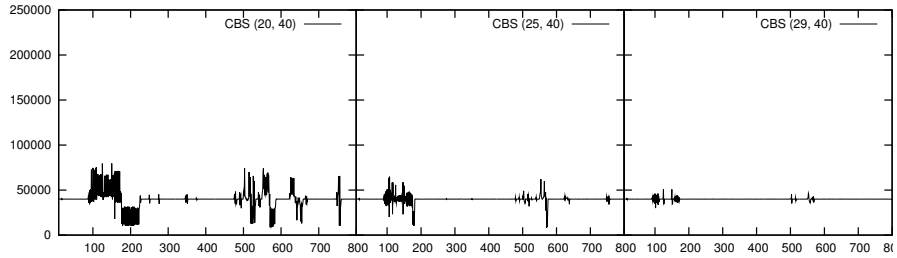

Figure 9.12: Inter-frame times for a video player when executed alone and concurrently with `netperf`, using different kinds of reservations.

Based on such analysis, the reservation parameters shown in Table 9.8 have been used, obtaining the network throughput shown in the last column of the table. The evolution of the inter-frame times in the player for the most interesting cases is shown in Figure 9.12. As it is possible to perceive by looking at the table and at the figure, resource reservations really allow to find latency/throughput trade-offs, as previously claimed:

- if the player is reserved enough CPU time (29ms every period of 40ms), then the inter-frame times are stable and near to 40ms (see the right side of Figure 9.12). However, in this case it is possible to reserve only a small fraction of the CPU time to the network IRQ thread, and the network throughput is low (about 60Mbps);
- if enough CPU time is reserved to the network IRQ thread (18ms every 100ms), then the maximum network throughput can be achieved). But in this case it is possible to reserve only 20ms of CPU time every 40ms to the player, and the inter-frame times increase (see the left side of Figure 9.12). Note, however, that the maximum inter-frame times are still under 80ms (compare this situation with the middle of Figure 9.11);
- some compromises can be found: for example, scheduling the player with a $(25\text{ms}, 40\text{ms})$ reservation and the network IRQ thread with a $(14\text{ms}, 100\text{ms})$ reservation it is possible to have reasonable inter-frame times with a good network throughput (about 86% of the maximum).

9.1.5 *Conclusions and Future Work*

This section reported the results of some experiences with device drivers scheduling in real-time systems. In particular, some of the presented experiments show how recent developments in the Linux kernel can be exploited to schedule the IRQ threads so that both their interference on real-time tasks and the device throughput can be controlled.

The proposed solution is based on the Linux Preempt-RT kernel (which transforms interrupt handlers into schedulable entities), a new reservation-based scheduler, and some tools based on Ftrace that can be used to infer the timing information needed to correctly assign the scheduling parameters.

As a future work, the effectiveness and usability of the stochastic analysis for IRQ threads will be investigated by considering different workloads and resources. This will probably require to simplify the Markov model used in the stochastic analysis, so that it can be applied more easily.

9.2 QOS SUPPORT IN THE X11 WINDOW SYSTEMS

In this section, we consider the problem of providing QoS guarantees to the execution of applications using the X11 window system. In particular, we offer a system level analysis of the issues encountered when using X11 to serve real-time applications. By using a tracer developed for the purpose we analyse in depth the internal behaviour of the system. The result of the analysis puts on display the adverse effect played by a non real-time scheduler on the performance of time-sensitive applications. Based on this analysis, we propose an alternative solution based on the CBS scheduler and prove its effectiveness by an extensive set of experiments on real hardware.

9.2.1 *Introduction*

In the past few years, we have observed a clearly established trend toward the use of computer based devices for multimedia applications. The growing commercial fortune of such networked applications as IPTV, YouTube and video servers is a clear indicator of a paradigm shift in the way most people use their personal computers and, generally speaking, their computer based devices. The gain of using computers for this class of applications is evident in terms of flexibility and cost effectiveness. A computer can be used to run multiple and heterogeneous applications at once. Moreover, it is easy to upgrade the software to support new multimedia compression standards and sophisticated sound technologies, as soon as they become available.

This huge potential though poses formidable challenges to the designers of Operating Systems and of network protocols. Indeed, a multimedia application is inherently time-sensitive: uncontrolled fluctuations in latency and frame-rate defy the patience of any user who expects to watch TV or talk to VoIP phone with a Quality of Service (QoS) comparable to the one experienced with traditional dedicated hardware solutions. To this regard, resource sharing is known to play an adverse role since it introduces scheduling delays that are not easily predictable when designing the application (since they depend on the workload of the system). In contrast, what we need is that a

---

Parts of this Chapter are going to appear in:
N. Manica, L. Abeni, L. Palopoli, "QoS support in the X11 Window Systems," *Real-Time and Embedded Technology and Applications Symposium, 2008. RTAS'08. IEEE*

time-sensitive application receives a dedicated fraction of a shared resource in time, regardless of the behaviour of other applications. This property is called *temporal protection* [59] and it is the natural complement of memory protection, which prevents interference in the concurrent access of a set of applications to a limited memory. For years, researchers have been confronted with the challenging effort of designing scheduling algorithms that feature temporal protection. A first important class of algorithms designed to this purpose approximates the Generalised Processor Sharing concept of a *fluid flow* allocation, in which each application using the resource marks a progress proportional to its weight (for example, see some Proportional Share algorithms [60]). Similar are the underlying principles of a family of algorithms known as Resource Reservations [61, 5, 4, 62], which associate to each application a pair (Q, P) guaranteeing that it receives at least Q units of time every P.

In this context a relatively marginal importance has been attached to the problem of building a window system for real-time applications. This lack of attention is not, in our opinion, well deserved. Indeed, for instance, it is of little use to have a very fine grained allocation of the CPU time for a MPEG player if the projection of the movie into the window can be stalled by a non real-time application scrolling a huge amount of textual data in a different window. On the other hand, designing a real-time window system (RTWS) is surely to be considered a challenging activity because a difficult balance has to be found between contrasting needs. Since the most commonly used window systems are based on a client/server paradigm, the first problem is the so called priority inversion [63]. Mainstream window systems (e.g., X11) execute graphical primitives in an order that is irrespective of the real-time priority of the tasks formulating the request. Thereby, real-time tasks can incur a blocking time from lower priority tasks, for which it is difficult or impossible to find an upper bound. Blocking times can be reduced by using appropriate scheduling mechanisms but they are lower-bounded by the length non-interruptible graphical primitives. Another non-trivial problem is that, in order to provide real-time guarantees to the clients, we have to take into account that these requests have to be scheduled in the slots of CPU execution time allocated to the server. Finally, real-time policies are commonly regarded to reduce the system throughput and this is hardly acceptable when a window system is used to

manage heterogeneous applications, which only in part have timing constraints.

A first remarkable attempt to cope with these challenging issues is called Artifact [64], and it was developed in the early nineties as part of the real-time Mach project. Even if the authors use a single thread to manage all real-time request (potentially introducing priority inversion), they alleviate the problem by restricting to a small set the graphical primitives usable by real-time tasks. In this way the priority inversion is limited, but the authors themselves concede that a more satisfactory solution can only be found using different approaches (e.g., a multithreaded approach).The authors also consider the problem of coordinated scheduling of server and client by creating *scheduling models* for client and server upon a connection request, subject to a global admission test. In a more recent proposal called DOPE [65] (based on the DROPS kernel),the authors use a time-triggered thread to refresh the widgets belonging to real-time applications. An interesting idea is the introduction of resource managers that map basic resources to higher level ones, e.g., DOPE maps CPU cycles and main memory to refresh bandwidth. A different viewpoint is taken in EWS, a window system built in the context of the EROS kernel [66]. In this case the authors make a strong point that a WS with fast graphical primitives may provide good real-time performance without the need for an adequate scheduling support. In our evaluation, this statement is arguably true only when the system is not heavily loaded by a *large number* of client requests.

The proposals reported so far have a very important commonality: they are based on research systems, built from scratch to the purpose of displaying issues of interest, or to show the effectiveness of a specific solution. Research on CPU scheduling followed a similar path in the past, and the development of specialised real-time kernels, aimed at demonstrating some novel technique or scheduling algorithm, has been popular for some years producing solutions such as Nemesis [62], Real-time Mach [67], Hartik [68], YARTOS [69]. However, the absence of a strong connection with main-stream technologies such as Windows or Linux has severely hindered maintenance and porting across different hardware platform, reducing the impact of these proposals and ultimately causing their obsolescence. Therefore, there has been a paradigm shift toward architectural constructions tacked on general purpose kernels (typically Linux). As a result, the real-time performance of such general purpose operating systems

as the Linux kernel has improved impressively [70], to a point where today Linux is considered as a viable solution for both real-time research and industrial embedded software.

In this spirit, we believe that research on RTWSs should also move from writing research systems from scratch to modifying commonly used solutions; for this reason, we propose to introduce limited changes to the X11 server to make it suitable to real-time applications. The first contribution of this section is a very thorough analysis of the X server to clearly identify the scenarios in which it fails to provide an adequate support to real-time applications. To this end, we have constructed a tracing tool that allows us to expose the timing behaviour of the X-server in different workload conditions. The conclusion of the first part of this work is that the effects of the scheduling policy are indeed quite heavy on real-time performance, and their importance outweighs the problems generated by the length of the non-interruptible primitives (which is being reduced by the new generation of accelerated graphical cards). This experimental observation is in perfect accordance with a similar conjecture formulated in [71]. In this case, case the authors propose a modification of the X11 server to support fixed priority scheduling. In our evaluation, this solution is not sufficient to lend robustness to the temporal behaviour of such soft real-time applications as continuous media, for which, as noted above, temporal protection plays a prominent role. Therefore, we have built a solution based on a particular flavour of the resource reservation algorithms, the constant bandwidth server (CBS).

9.2.2 *The Problem*

The window system traditionally used in Unix-like systems is based on a client-server paradigm, where an *X server* acts as a manager for the video and for the input devices (keyboard and mouse). The X server forwards input events to some *client applications* and executes the *requests* received from the clients drawing on the screen.

Unfortunately, the X server is not aware of real-time requirements of time-sensitive clients, its only design goal being to maximise the global throughput. Indeed, it is possible to observe that the refresh rate of each window decreases with the number of active windows. This is hardly acceptable for real-time applications, for which a fixed refresh rate (independent of the workload of the server) is required. For instance, in a media player, the users perceives this effect as a

Workload level on the server	average ocbench period	standard deviation
low	12005 µs	182 µs
heavy	35606 µs	28475 µs

Table 9.9: ocbench periods with lightly loaded or overloaded X server.

slowdown of the movie whenever some clients require a heavy operation.

In terms of the real-time scheduling theory, this problem can be classified as a **priority inversion**. The video card is accessed by a resource manager (the X server) that serves the different requests in an order that is not dictated by their real-time priorities. The duration of this priority inversion can be very long (potentially unbounded). The practical consequence is that even if the CPU is not overloaded and all the applications are properly scheduled by the CPU scheduler, time-sensitive applications can still be unable to perform their graphical operations with the correct timing. For example, a media player is not able to play a movie at the correct frame rate even if it is scheduled with the highest priority on the CPU (or if the CPU load - as measured by some utilities like `top` - is low).

To expose the problem and study it, we used an X testing application, called `ocbench` [72]. Ocbench is a very simple application which periodically updates a spinning wheel on the screen, using a very small amount of CPU time (so, it generates a large number of X requests that must be served in a timely fashion, but it creates a very low CPU workload). The application simply consists of an infinite loop that sleeps for a fixed amount of time ($10ms$ in our tests) and then refreshes the image on the screen, and it lends itself to an immediate performance assessment. Indeed, when `ocbench` undergoes the interference of other X applications the user immediately notices a slowdown of the spinning wheel motion. In order to have a quantitative evaluation of this effect, we instrumented the code, by inserting a call to `gettimeofday()` right before the execution of the redraw operation. So, the progress of the measured times can be used as an indicator of the performance of the X server. The ideal evolution of this quantity is given by the $k(a + T)$, where k is the activation number, $T = 10ms$ is the periodicity of the requests and $a \approx 2ms$ is the duration of the redraw operation. To introduce interference on the execution of `ocbench`, we used the `x11perf` application (a standard test for the X server performance) that creates a large volume of X requests.

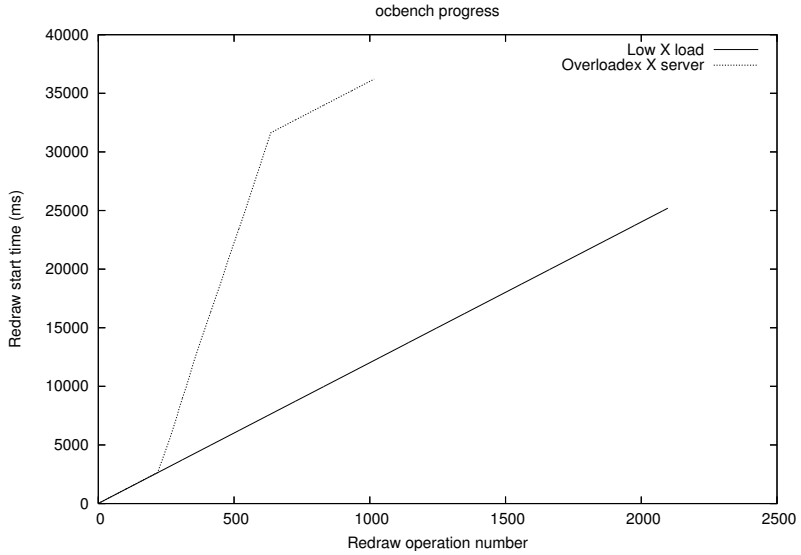

Figure 9.13: Evolution of the ocbench periods, with lightly loaded or overloaded X server.

Table 9.9 shows the average and the standard deviation of the differences between two consecutive time readings of the instrumented `ocbench`. In particular, the first row refers to an execution of ocbench on a lightly loaded X server, whereas in the second one we considered the disturbing effects of an instance of `x11perf -getimagexy100` executing in background,.

As shown in the table, when the X server is not loaded the average distance between the start of two consecutive redraw operations is about 12ms, and the variance is about 0.182ms (in close accordance with the ideal behaviour). However, when the X server is heavily loaded by the x11perf application, the average distance increases and becomes much less stable (the variance increases to 28ms). This behaviour is well depicted in Figure 9.13, which plots the evolution of the time readings as a function of the activation of `ocbench`. In case of a low workload the progress is exactly linear. In case of heavy workload, we have a piecewise linear plot. In particular, in correspondence with the 200th activation of `ocbench`, the slope becomes steeper (meaning a slower progress) due to the activation of `x11perf`. When `x1perf` is terminated (650th activation of `ocbench`) the slope becomes again equal to 12ms.

To cast some light on this big variation in `ocbench` speed noticed in the second case, we measured the time needed by X to serve the redraw requests, inclusive of the execution time of the operation and of the scheduling delays. The result is plotted in Figure 9.14, in which

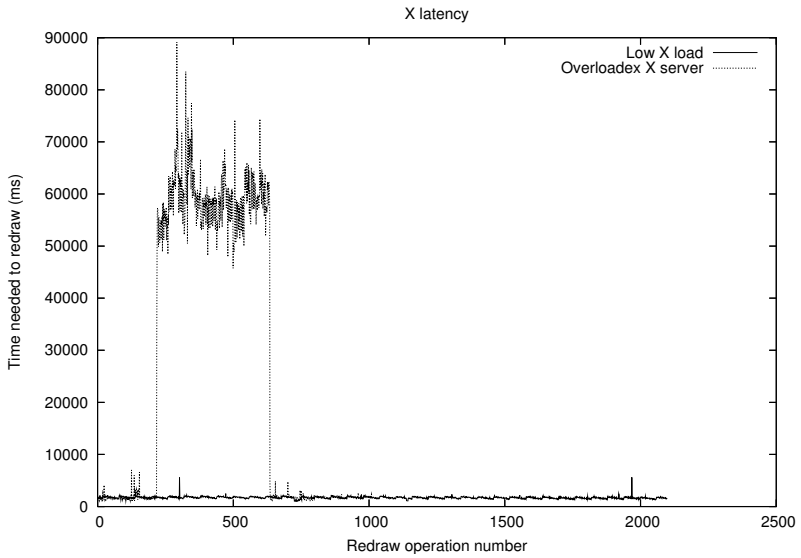

Figure 9.14: Evolution of the time needed by X to serve a request (X latency), with lightly loaded or overloaded X server.

the time required to complete the requests scales by orders of magnitude wen `x11perf` is activated.

9.2.2.1 *Problem Analysis*

As a first step to solve the priority inversion problems, we performed an in-depth analysis on the way the X server schedules the requests of its clients. To do this, we have written a patch of the X server that introduces a tracing mechanism to record the arrival of requests from the clients, their executions, and the times when each request is terminated. In particular, the following events are traced (each event is described by an *event type*, the *event time*, and the *client id*):

- **begin:** indicates the starting time for a trace;
- **end:** end of the trace;
- **creation:** a new client connects to the server;
- **destruction:** a client terminates, or disconnects from the server;
- **activation:** a new request from a connected client arrives (and is inserted into the queue of ready clients);
- **deactivation:** a request has been served, and is terminated;
- **dispatch:** the server starts to serve a request from a client (selected by the server's scheduler).

Figure 9.15: ocbench trace with non-loaded X server.

Figure 9.16: ocbench trace with overloaded X server.

This tracer can be used to identify the sequence of events generated by the X server when it serves a client: for example, Figure 9.15 shows the trace of an instance of `ocbench` served by a lightly loaded X server. From the figure it is easy to see that the program periodically generates a burst of requests (corresponding to a redraw operation), which are immediately served by the X server (in about 2.5ms / 3ms). As a result, `ocbench` can execute at a constant rate, and all the timing constraints are clearly respected. Note that each burst starts about 10ms after the end of the previous one, as expected (`ocbench` sleeps 10ms between two redraw operations).

Figure 9.16, on the other hand, shows a trace of `ocbench` when it competes with `x11perf` (that creates a heavy workload on the X server as discussed above). Right after being started, `x11perf` runs a *calibration* phase, which in this trace ends at time 260ms. During this phase, `x11perf` does not significantly affect the time execution of `ocbench`, which runs smoothly as in Figure 9.15. Around time 260, the calibration phase terminates, and `x11perf` starts to overload the X server: as a result, `ocbench` requests are not served in time. This phenomenon is easily detectable on the trace dedicated to `ocbench`; the bursts are no longer executed with regularity and some of them are delayed by a long time.

This effect has a clear explaination: since the scheduling mechanism adopted by X11 is a variant of the classical round-robin used in traditional time sharing systems, an application able to generated a large number of requests like `x11perf` is able to engage the server for arbitrarily long times starving other graphical applications. As shown next, the scheduling algorithm proposed in this thesis allows us to radically alleviate this problem.

9.2.2.2 *Interactions with the CPU Scheduler*

The test applications used in the previous examples do not consume CPU time other than the one required for submitting request to X, so

their real-time performance is only affected by the X scheduler (and the CPU scheduler contained in the kernel does not really matter, because the CPU load is low).

In more realistic situations things will likely be more complex, and there will be stronger interactions between the CPU scheduler and X scheduler: a request from an X client will only be served when the CPU scheduler selects the X server for execution, *and* the X scheduler selects the client. This means that performing real-time guarantees for X clients could potentially be quite difficult.

However, we believe that the complex system composed by the CPU scheduler, the X scheduler and the X clients' requests can be modelled as a hierarchical system. As a consequence, if both the X scheduler and the CPU scheduler provide predictable performance their combined effect can be analysed by applying the hierarchical scheduling theory [73, 74, 75]. This approach permits to simplify the analysis of the system, by considering the two schedulers in isolation, and composing their real-time guarantees. Since CPU schedulers have been studied at long in the past, and various theoretical frameworks for hierarchically composing scheduling guarantee exist in literature, in this section we only focus on the behaviour of the X scheduler in isolation.

9.2.3 *A Possible Solution*

As shown in the previous section, the absence of an appropriate scheduling mechanism inhibits the use of X11 for real-time clients. On the other hand, fixed priority mechanisms based on classical real-time scheduling theory [19] like the one adopted in linux-SRT [71] have evident shortcomings when the task set encompasses both real-time and non real-time tasks. In particular, a design based on the worst case requirements of the tasks can be overly conservative. On the other hand, if we use a design based on the average resource requirements, a high priority real-time task consuming more than expected can cause deadline misses in other (unrelated) real-time tasks [59]. This problem is well known in the context of CPU scheduling, and a number of different solutions has been proposed. In particular, we based our work on a scheduling algorithm, called the Constant Bandwidth Server (CBS) that implements the *Resource Reservations* paradigm. In the development of this scheduler, we considered the X server as an *Open System* [76]. An Open System is a system in which applications dynami-

cally enter and exit the system in an unpredictable (or hard to predict) way. Tasks (X clients, in this case) can dynamically be activated at any time and are characterised by variable execution times, so the scheduler cannot make any restrictive assumption on the characteristics of the task set.

Resource Reservations [61] have emerged as an effective technique to support time-sensitive applications on general purpose operating systems (GPOS). This technique provides support for time-sensitive applications by allowing the integration of classical real-time techniques, developed to meet timing constraints on real-time operating systems (RTOSs), with the general-purpose allocation strategies used on GPOSs. In particular, CPU reservations have been traditionally implemented by using a dedicated aperiodic server (the Deferrable Server [77]) to serve each reserved task [61, 5]. Unfortunately, this implementation strategy generates some scheduling anomalies when tasks block and unblock dynamically [77]. This particular problem has been solved by the CBS algorithm [4], which uses dynamic priorities to correctly cope with dynamic aperiodic arrivals. Therefore the CBS can provide a predictable QoS to both periodic and aperiodic real-time activities [6].

Since X requests are generally non periodic (for example, in Figure 9.15 we can see that requests arrive in bursts), the CBS appeared as a very natural choice for our reference implementation. However, the original version of the algorithm is fully preemptive. Thereby, it cannot be directly used in the X server (in which each request is substantially uninterruptible). For the sake of clarity, we will first briefly recall the original algorithm. Then we will review the most important features of the X server architecture (to discuss how our solution has been implemented). Finally, we will discuss the modification required to the CBS algorithm to adapt it to the X server.

9.2.3.1 *The Constant Bandwidth Server*

In the sequel an *X client* issuing a request to the server will be considered as a real-time task and referenced as τ_i; when receiving requests from multiple clients, the X server selects the request to be served according to some parameters and variables.

Each client τ_i is characterised by two parameters Q_i and T_i (we say that τ_i is associated to a reservation $RSV_i = (Q_i, T_i)$), meaning that it is *reserved* a time Q_i in a period T_i. According to the original (fully preemptive) CBS algorithm, when a request from client τ_i is

served, the time needed by the X server for serving such request is accounted by decreasing a variable q_i called *budget*, and clients are scheduled based on their *scheduling deadlines* d_i^s (the client with the earliest scheduling deadline is scheduled). When a request from client τ_i arrives to the X server at time t, we say that τ_i is activated. On the first activation of τ_i, q_i is initialised to Q_i and d_i is initialised to $t + T_i$, and when q_i arrives to 0 τ_i is said to be depleted. On depletion, two different behaviours are possible:

- the budget is immediately replenished to Q_i and the scheduling deadline is postponed to $d_i^s + T_i$ (so, the client remains schedulable). This is known as *soft reservation behaviour*;
- the client is not schedulable until time d_i^s, when the budget will be replenished and the deadline will be postponed as above (so, the client cannot be scheduled until d_i^s). This is known as *hard reservation behaviour*.

The CBS is guaranteed to execute respecting the real-time properties of the tasks inasmuch as the following condition is respected:

$$\sum_i \frac{Q_i}{T_i} \leqslant 1 \tag{9.3}$$

For a further discussion on the CBS algorithm and on its properties, the reader is referred to the Chapter 4 or the original paper [4].

9.2.3.2 *The X Server Architecture*

The X server is a single thread application, in which a single flow of execution cyclically intercepts input events and receive clients' requests, selects the action to process, and processes it. While a multi-threaded server (creating a thread per client) can easily delegate to the CPU scheduler (in the OS kernel) the selection of the client to be served, a single-threaded server like X must explicitly contain a scheduler for this purpose. On the hand, retaining the single threaded structure allows us to reduce the modifications required on the X server to implement the real-time scheduler and facilitates porting across the different versions of X.

The X server is logically structured in four layers - an OS dependent layer (OS), a device independent layer (DIX), a device dependent (DDX), and an Extension interface - and the scheduler is located in the DIX layer.

As said, the X server is implemented according to an event-based paradigm, with a main loop waiting for events, scheduling an event to be served, and serving it. There are three different kinds of events:

- connections from a new client;
- input events from the user (mouse click, ...);
- requests from an already connected client (clients' activations).

Events are served in a non-preemptable way, and the selection of the event to be served is performed by the scheduling function `SmartSchedule()`, which implements a variation of the round robin algorithm. So, when an event representing a request from client τ_i is selected (τ_i is scheduled), it executes until completion; moreover, to improve the throughput when τ_i scheduled it can execute a burst of requests (and not only one). In particular, a global variable called `isItTimeToYield` is used by the server to decide when new scheduler invocation are needed. In the standard implementation of X, `isItTimeToYield` is set when there are no more requests from a client (the client is deactivated), or when a maximum number of requests have been served.

9.2.3.3 *Implementing the CBS on the X server*

A modified (non fully-preemptable) version of the CBS scheduler has been implemented in the X server as an extension that can be enabled at compile time. When the feature is enabled, our implementation replaces the `SmartSchedule()` function with a CBS scheduler. First of all, we extended the information maintained for every client to store the CBS parameters and runtime variables:

- rt_period: the reservation period T_i;
- rt_deadline: the scheduling deadline d_i^s;
- rt_capacity: the current budget q_i;
- rt_maxcapacity: the maximum budget Q_i.

Then, we modified the X scheduler introducing a real-time queue (RTQ) ordered by scheduling deadlines, in which requests from clients associated to a CBS are stored. If there are requests in the RTQ, then the first one (i.e., the one having the earliest scheduling deadline) is selected, otherwise the original X scheduler is used.

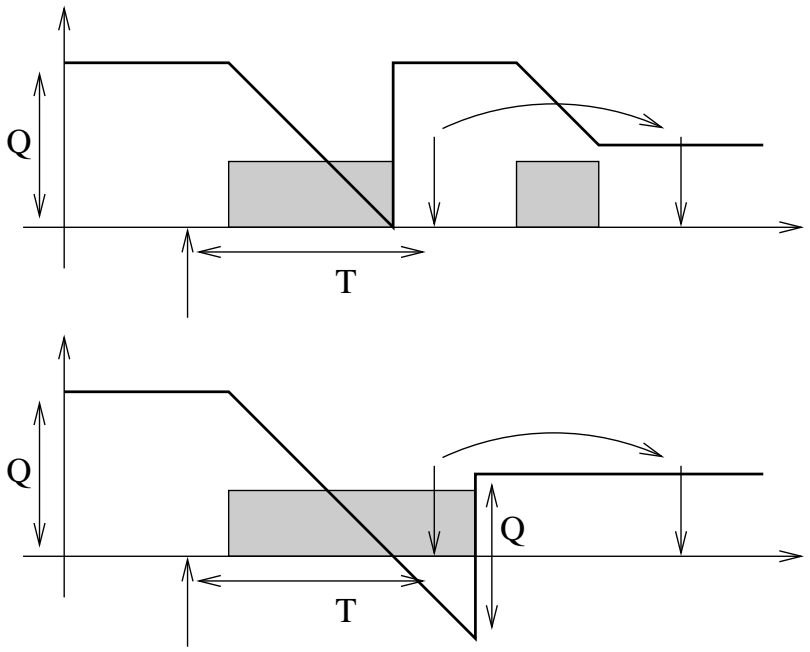

Figure 9.17: Original and modified accounting and replenishment mechanisms.

Notice that since events are served in a non-preemptable way, the accounting can only be performed after serving an event, so q_i is updated when the request has been completely served. This means that after serving long requests q_i can become negative (if the maximum time C needed to serve a request was known in advance, we could consider q_i exhausted if $q_i < C$, but such assumption is not reasonable in an open system). If after performing the accounting $q_i \leqslant 0$, then

- $q_i = q_i + Q_i$, $d_i^s = d_i^s + T_i$, and the RTQ is reordered (soft reservation behaviour), or
- the client is removed from the RTQ, and will be re-inserted only at time d_i^s when budget and scheduling deadline will be updated as above (hard reservation behaviour)

Note that since q_i can become negative, the replenishment is performed by setting $q_i = q_i + Q_i$, and not $q_i = Q_i$. Finally, when the scheduling deadline is postponed the RTQ must be reordered, and a different client is selected for service (note that this only happens after finishing to serve a request).

Figure 9.17 shows the effects of the modified accounting mechanism: the upper part depicts the behaviour of the original accounting and replenishment rules (for soft CBS) , while the the lower part

shows the modified rules in action. When a new request arrives, it is assigned a deadline equal to the arrival time plus T; after some time the request is scheduled and its budget starts to decrease. In the original algorithm, when the budget arrives to 0 it is recharged to Q (remember that we are considering the soft incarnation of the CBS algorithm) and the deadline is postponed by T. Note that since the deadline is postponed, the client can be preempted (in the example, it is actually preempted, and is scheduled again only after some time). In the X implementation of the CBS, since the request is not preemptable it cannot be interrupted when the budget arrives to 0, and the request runs to completion. When it is finally served, the budget is negative, and is recharged by Q, postponing the deadline by T as above. Finally, note that the non-preemtpability of the requests can be modelled through the concept of *blocking time*: each request can cause a blocking time as long as the maximum time needed by X to serve it. So, if an upper bound B for the time needed by the X server to serve a request is known, it is possible produce an update of the admission test in Equation 4.1:

$$\forall i, \sum_{j=1}^{i} \frac{Q_j}{T_j} + \frac{B}{T_i} \leqslant 1$$

(see [78] for more details).

Clients can manage their CBS parameters (Q_i, T_i) by using three new functions:

- `XRTInitialize()`: initialise the RT structure and check if this extension is installed;
- `RTSetProperty()`: transform the client in real-time (CBS) and set the reservation parameters
- `RTGetProperty()`: return some real-time information about the client

As a final remark, this scheduler could be plugged into the structure of X11 with a small effort. All these functions have been developed as X extensions strictly conforming to the guidelines provided by the Xorg foundation (that is, by properly extending the `xext` protocol and by implementing the functions in the Extension Layer) and without interfering with the normal functionalities of the window systems. Therefore, it is possible to rely on the hardware support for a plethora of graphic cards offered by the most commonly used X11

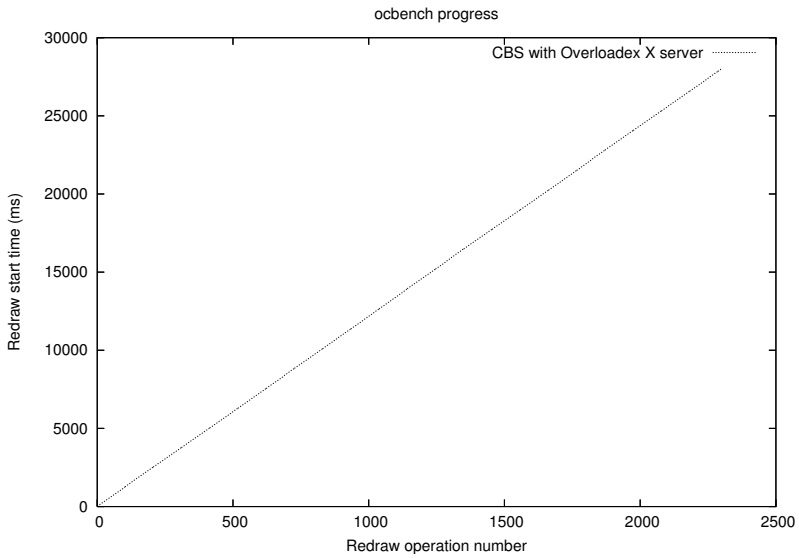

Figure 9.18: ocbench progress with lightly loaded or overloaded X server.

servers (Xfree86 and xorg) and to use the legacy applications without any porting effort (the API of the Window Systems has been totally unaffected).

9.2.4 *Experimental Results*

To test the effectiveness and the efficiency of the proposed solution, we performed an extensive set of experiments on a real implementation of our scheduler, considering different types of X clients (both real-time and not). All the experiments have been run on a standard PC based on an Intel core duo CPU at 1.66GHz equipped with an ATI Radeon X1300 graphic card and 1GB of RAM. The system is running Ubuntu Festy with a standard 2.6.20 Linux kernel, and the X server from current git.

9.2.4.1 *Serving Time-Sensitive Applications*

The objective of a first set of experiments was to verify that the CBS scheduler implemented in the X server properly addresses the problems exposed in Section 9.2.2. To this end, we considered the same situation depicted in Figure 9.13, in which a time-sensitive application (`ocbench`) is scheduled while an instance of `x11perf` overloads the X server and a measure of the system time is taken for each `ocbench` activation right before executing the redraw operations. Contrary to what we did before, the `ocbench` application is scheduled using a

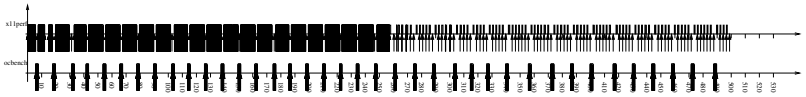

Figure 9.19: trace of an ocbench scheduled by a hard CBS, with overloaded X server.

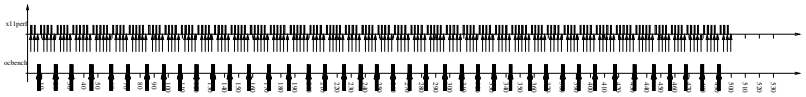

Figure 9.20: trace of an ocbench scheduled by a soft CBS, with overloaded X server.

CBS server, both in the hard and in the soft version, with parameters $(3ms, 10ms)$. The result is reported in Figure 9.18 and the difference with the results displayed in Figure 9.13 is evident. In this case even after the activation of `x11perf`, which occurs after 200 `ocbench` cycles the time marks a perfectly proportionate progress in time. The speed of this progress is only affected by the choice of parameters for the CBS.

Since the amount of X time reserved to `ocbench` is enough for properly serving its requests, the hard and soft CBS schedulers provide the same performance, so the figure displays only one set of results. The correctness of the timing behaviour is confirmed by Figures 9.19 and 9.20, which represent the traces obtained when scheduling ocbench with a hard or soft CBS. As it is possible to see, reserving a correct bandwidth to the client allows the server to fulfill all requests with the proper timing.

9.2.4.2 *Impact on Throughput*

One of the problems encountered when applying real-time techniques in general-purpose systems is that they often have a bad impact on the throughput of non real-time applications. To show that the CBS implementation presented in this section does not suffer from this

	Q = 1/6T		Q = 1/3T		Q = 1/2T		Q = 2/3T		Q = 5/6T	
	Avg	StdDev	Avg	StdDev	Avg	StdDev	Avg	StdDev	Avg	StdDev
T = 30	67	1.14	132	3.64	193	2.40	264	2.3	318	2.48
T = 60	66.2	0.96	133	0.55	192	1.78	251	1.58	313	1.34
T = 90	66	0.55	128	1.51	189	2.04	253	1.30	310	1.48
T = 120	65.8	1.01	126	1.22	186	2.28	248	1.92	310	1.82
T = 150	63.7	1.12	126	1.34	187	2.28	248	2.49	306	2.30
T = 180	62.5	0.99	126	1.64	188	2.34	245	3.36	308	2.00
T = 210	62.8	1.04	125	1.08	187	2.30	246	2.60	305	4.1833

Table 9.10: x11perf throughput when scheduled by a CBS with different parameters.

problem, a second batch of experiments has been performed to gauge the effects of the CBS on the throughput of non time-sensitive applications. To this end, the throughput (number of operations per second) achieved by instance of `x11perf` served by a CBS has been measured (relying on the numbers reported by the `x11perf` program to compute the throughput).

First of all, `x11perf` has been scheduled through a soft CBS, obtaining an average throughput of 360 operations per second with a standard deviation of 1.73, while the throughput obtained using the standard X scheduler is 357 operations per second with standard deviation 2.28. We performed several experiments of this type obtaining consistent results: in all cases the CBS did not worsen the throughput (sometimes the results with the soft CBS were even better than the standard X11 scheduler).

Note that serving non time-sensitive applications with a CBS allows to have some degree of control on the applications' throughput. Hence, a third batch of experiments was designed to show the ability of the CBS to control the fraction of time devoted by the X server to serve requests from a specified client. In this case we used a hard CBS to schedule an instance of `x11perf` with different scheduling parameters. For each choice of parameters Q and T me measured the throughput achieved through 5 trials. The results (average and standard deviation) are shown in Table 9.10 and Figure 9.21. The achieved throughput increases proportionally to the bandwidth Q/T. It is worth noting that the performance obtained with the soft CBS was remarkably better (360 operations per second, as measured in the previous experiment). This is perfectly consistent with our expectations, since the hard CBS is used exactly to the purpose of allocating a hard bound to the time dedicated to the application (e.g., to avoid starvation of other applications running in background).

9.2.4.3 *Scheduling a Media Player with the CBS*

To show how the CBS can improve the performance of more complex time-sensitive applications, some of the previous experiments have been repeated using a media player. To this purpose, a media player based on FFMPEG and GTK/GDK has been ran together with an instance of `x11perf` (used to create a high load on the X server). The

This surprising result is probably due to the fact that scheduling `x11perf` with a CBS gives him a higher priority over non real-time clients, such as the window manager remember that `ocbench` is a very simple application, designed to be only used as a benchmark

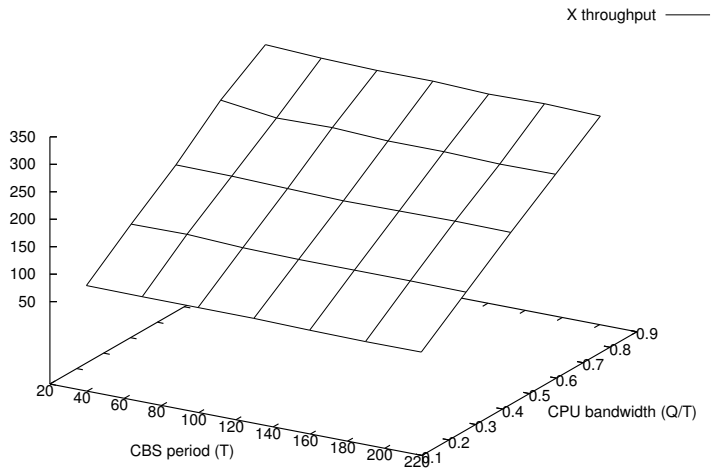

Figure 9.21: x11perf throughput when scheduled by a CBS with different parameters, in 3D.

player has been configured for not skipping video frames, and the Inter-Frame time (defined as the difference between the display times of two consecutive frames) has been used as a measure of the QoS perceived by the user.

Figure 9.22 shows the Inter-Frame times obtained when playing a video at 25 frames per second (fps) when using the standard X scheduler (this player will be referred as nrt player), and when serving the player with a properly dimensioned CBS (rt player). Until arond frame 110, the X server is not overloaded and all the frames from both the player instances are displayed in time (note that the Inter-Frame times are arond $40\text{ms} = 1/25$). Then, an instance of `x11perf` is started around frame 110, and overloads the X server causing a large increase in the Inter-Frame times experienced by the nrt player. The rt player instead, is not affected by the `x11perf` load and its Inter-Frame time remain stable all the time. During `x11perf` execution, the video frames from the nrt player are not displayed in time, and are queued by the X server; when `x11perf` stops, such frames are displayed at a high speed until the X queue is empty and the Inter-Frame times return to 40ms.

9.2.5 *Conclusions*

In this section, we analysed the performance of the X11 server when it is used to serve real-time requests, showing some anomalies caused

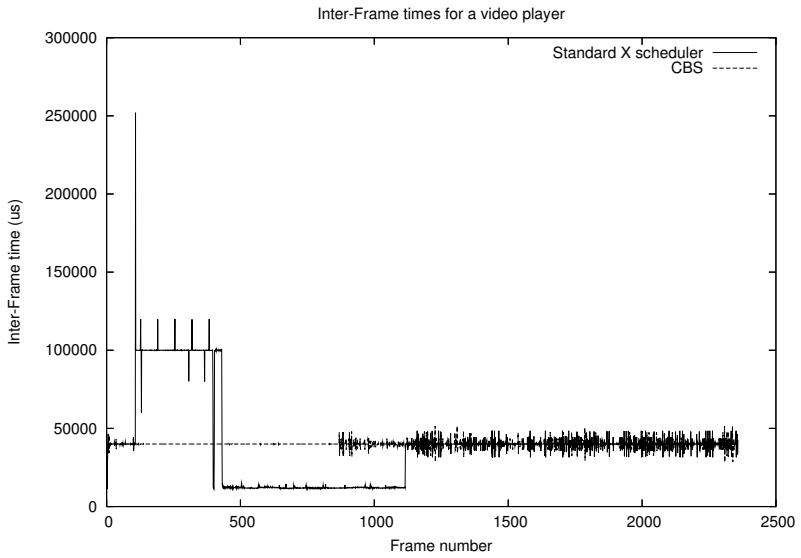

Figure 9.22: video player served by the standard X scheduler and by a CBS.

by the priority inversions that stem from a substantially round-robin mechanism in managing a shared resource (the video card), which do not account for real-time priorities.

To solve this problem, we devised our own scheduling solution based on a CBS server. Advantages of our approach are: 1) the containment of priority inversion, 2) temporal isolation between the different applications, 3) easy portability across different X versions, 4) negligible effects on the global throughput.

As a future work, we plan to study the interactions between the X scheduler and the CPU scheduler contained in the kernel, and to formally analyse the complex hierarchical system composed by the X server and its clients. As far as architectural aspects are concerned, we will test our solution on a wide class of applications to cover the whole gamut of graphical primitives and propose it as a full-fledged alternative to the standard scheduler.

10 CONCLUSION

In this thesis, we have considered the problem of probabilistic guarantees for soft real–time periodic tasks scheduled through a resource reservation.

The new method explained in Chapter 5 for analyzing soft real-time systems through probabilistic deadlines, is faster and is robust against uncertainties in the execution times distribution.

After that we have shown that the evolution of the system can be also modelled as a Quasi Birth Death Process. The probability of respect the deadline amounts to the computation of the steady state probability of this process. We have shown how this is possible both by numeric means achieving different performance/accuracy tradeoffs. We have also shown the computation of an analytical bound and offered a validation of these results by experiments and simulations.

In addition, to complete the analysis, we described a model able to model the problem of scheduling interrupts, which is a particular situation in which the inter-arrival times are very small.

Finally we have shown some practical examples of using the resource reservation.

10.1 FUTURE RESEARCH WORK

As a future work, a new strategy for finding a proper value of γ (based on discrete Fourier transform) will be investigated, and the bounds will be compared with some closed-form solutions that can be obtained in some special cases. Respect to the QBDM we aim to further refine the accuracy of our analytical bound and apply the proposed approach in different contexts.

To complete the Figure 7.2, the next step is to find a way to compute the exact solution starting from the exact model.

A

ANALYSIS OF CLIENT/SERVER INTERACTIONS IN A RESERVATION-BASED SYSTEM

This appendix presents a theoretical schedulability analysis of client/server communication in a reservation-based system. The inheritance mechanism previously implemented in a reservation-based system (based on the `SCHED_DEADLINE` Linux patch, which implements the Constant Bandwidth Server (CBS) algorithm in the Linux kernel) is improved to support predictable client/server communications, and the modified `SCHED_DEADLINE` has been used to run an extensive set of experiments showing the effectiveness of the proposed approach and analysis.

A.1 INTRODUCTION

Resource Reservation is an effective mechanism for enforcing temporal protection (or temporal isolation) between real-time tasks. This means that the worst case response time of a real-time task does not depend on the other applications running in the system, and allows to provide real-time guarantees considering each task "in isolation" (without having to care about all of the other tasks running in the system). For real-time systems, temporal protection is as important as the *memory protection* (or address-space protection) feature provided by general-purpose Operating System (OS) kernels.

While traditional reservation mechanisms only provide isolation between independent tasks, some form of inheritance mechanism has been introduced for extending temporal isolation to groups of tasks interacting through shared resources accessed in mutual exclusion [79, 80, 81]. More recently, similar inheritance mechanisms have been developed to support different kinds of interactions, such as interactions with device drivers [82] or message passing [83, 84, 82, 36], which is the most common programming model used in modern OSs (for example, constructing pipelines of tasks, or using the client/server paradigm). In particular in Unix-like systems, many services are based on the client/server (message-based) paradigm: for example, graphical input/output often happens through a server (the `X`

server), using a client/server model. This appendix addresses the problem of analysing the schedulability of client/server interactions (based on Remote Procedure Calls) when reservation-based scheduling is used.

The original Resource Reservations abstraction [5] (and, in particular, CPU reservations [39]) considered independent tasks, and the schedulability analysis has been only extended to consider shared resources [79, 85, 80]. In this appendix, client/server interactions between tasks are considered.

In particular, the usage of the BandWidth Inheritance (BWI) [80] mechanism proposed and implemented (but not analysed) in [36], is analysed. Informally speaking, the BWI idea is quite simple: when a task is blocked waiting for other tasks, it can share its reservation with such tasks. A similar algorithm, named Proxy Execution [81], is based on the concept of Group Scheduling and uses proxy tasks instead of reservation inheritance to obtain analogous results. The approach proposed in this appendix is also similar to [86], but allows a better schedulability analysis. Similar inheritance mechanisms have also been proposed in other contexts [82, 84, 83], but a formal schedulability analysis is still missing. While the analysis presented here focuses on [36], it can be adapted to the other implementations mentioned above.

The schedulability analysis presented in this appendix is different from the original BWI analysis (which considered multiple tasks accessing a shared resource in mutual exclusion) because it considers client/server interactions.

A.2 DEFINITIONS AND BACKGROUND

Real-time systems are traditionally modelled as a set $\Gamma = \{\tau_i\}$ of real-time tasks τ_i. In this appendix, the term "task" is used to identify a schedulable entity, being it a thread or a process.

Each real-time task τ_i is modelled as a stream of *jobs* $J_{i,j}$, which become ready for execution (arrive) at time $r_{i,j}$, require a computation time $c_{i,j}$, and finish at time $f_{i,j}$. Job $J_{i,j}$ is also characterised by a deadline $d_{i,j}$ that is respected if $f_{i,j} \leqslant d_{i,j}$, and is missed if $f_{i,j} > d_{i,j}$.

If $r_{i,j+1} - r_{i,j} = P_i$ is constant, then τ_i is said to be periodic, with period P_i. The Worst Case Execution Time (WCET) of task τ_i is defined as $C_i = \max_j\{c_{i,j}\}$.

http://www.x.org

Finally, the response time $\rho_{i,j}$ of job $J_{i,j}$ is defined as $\rho_{i,j} = f_{i,j} - r_{i,j}$. The performance of a real-time task can be described by the Cumulative Distribution Function (CDF) of the tasks' response times: $P\{f_{i,j} - r_{i,j} \leqslant \rho\}$.

This appendix considers sets of tasks scheduled by a *reservation-based* scheduler: each task τ_i is associated to a resource reservation (a CPU reservation, in this case) $RSV_i = (Q_i^s, T_i^s)$, meaning that the scheduler reserves to τ_i, an amount Q_i^s of execution time (called *maximum budget*) every period T_i^s (*server period*).

Resource reservations provide *temporal isolation* between tasks, meaning that the worst case behaviour of each task τ_i is not affected by the other tasks running in the system. As a result, the performance of each task can be analysed in isolation, without considering all the other task and simplifying the theoretical analysis of the system.

Resource reservations can be implemented by using a large number of scheduling algorithms, and the reservation mechanism used in the appendix is the Constant Bandwidth Server (CBS) (See Chapter 4).

The CBS algorithm guarantees that no scheduling deadline will ever be missed (that is, $\forall i$, q_i^s will arrive to 0 before time d_i^s) if the following schedulability condition $\sum_i Q_i^s/T_i^s \leqslant 1$ is respected. In this appendix, it is assumed that this schedulability condition is respected.

A reservation-based scheduler provides temporal isolation between tasks assuming that they are independent (that is, they do not interact nor share resources). However, if tasks share some resources (accessed in mutual exclusion) or interact in some other way (for example, through message passing) it is not possible to provide temporal isolation anymore. This appendix analyses the interactions happening when a message passing mechanisms is used (in particular, client/server interactions). In particular, task τ_i that can act as a client requiring some service from a different (even non real-time) task τ_s, acting as a server. These interaction happen by using a Remote Procedure Call (RPC) mechanism:

- Client τ_i sends a message to the server τ_s, and blocks waiting for a reply message from τ_s
- If τ_s is busy, the request is queued (using a priority-based queue, as described in the next sections) and will be processed later
- As soon as τ_s is ready, it gets the highest priority request from the requests' queue and processes it
- After processing the request, τ_s replies to τ_i

- When τ_i receives the reply message from τ_s, it unblocks and can finish job $J_{i,j}$.

A.3 RPC AND BANDWIDTH INHERITANCE

In a client-server architecture, the execution of the client is influenced by the execution of the server: if the server cannot execute, then the client will be blocked waiting from the results provided by the server.

In general, this problem happens when a high priority task τ_i blocks waiting for a lower priority task τ_j (in case of CBS scheduling, $d_i^s < d_j^s$). For example, τ_i and τ_j share a resource $\mathcal{R}$ accessed in mutual exclusion, and τ_i tries to access it while the resource is locked by τ_j. The same problem also happens when τ_i is a client sending a request to a server τ_j and blocking until it receives a reply from the server, according to the RPC mechanism described in Section A.2.

While τ_i is blocked waiting for τ_j, another medium priority task τ_k (with priority between τ_i and τ_j: in case of CBS, $d_i^s < d_k^s < d_j^s$) can be ready for execution and preempt τ_j, delaying indefinitely τ_i. The amount of time for which lower priority tasks (τ_j and τ_k, in this case) are scheduled instead of τ_i is called *blocking time*, and in a real-time system it is important to provide an upper bound to the blocking time suffered by each task.

The situations described above are the classical example of the *priority inversion* problem, that is well known in real-time literature and can result in an unbounded increase in the blocking time for high priority real-time tasks such as τ_i. The effects of priority inversion can be reduced and controlled in various ways: for example, by using Priority Inheritance (PI) or Priority Ceiling [63], or some similar mechanisms such as the Stack Resource Protocol (SRP) [87].

The PI protocol is one of the simplest ones, and is based on two rules: when a higher priority task τ_i wants to access a critical section locked by a lower priority task τ_j, τ_i blocks and the lower priority task τ_j inherits the priority of τ_i. When τ_j unlocks the resource exiting the critical section, the priorities return the same as before the acquisition of the lock. When using reservation-based scheduling, the PI approach can be extended by inheriting not only the task's priority (the scheduling deadline, if CBS is used) but also the current budget. This is the main idea of the BandWidth Inheritance protocol [80] (BWI): when a high priority task τ_i blocks waiting for a lower priority task τ_j, τ_j not only inherits the priority of τ_i (the scheduling deadline

d_i^s in the case of CBS scheduling) but also consumes budget q_i^s of the blocked task.

The BWI algorithm has been recently used in combination with the RPC mechanism (as described in Section A.2) [36]:

- when, during the execution of job $J_{i,j}$, the client τ_i sends a message to the server τ_s and blocks waiting for a reply message, the CBS RSV_i serving τ_i is inherited by τ_s (in other words, τ_s can be scheduled based on d_i^s and its execution time can be accounted to q_i^s)
- τ_s receives τ_i's request, and processes it (eventually executing with the priority of d_i^s and consuming q_i^s)
- after processing the request, τ_s replies to τ_i
- when τ_i receives the reply message from τ_s, it unblocks and RSV_i return to τ_i (so, τ_s cannot be scheduled by using RSV_i's budget and scheduling deadline).

Some experimental evaluation showed BWI to be effective for scheduling client/server interactions in reservation-based systems, but a formal schedulability analysis has not been presented yet.

A.4 SCHEDULABILITY ANALYSIS

In this section, a schedulability analysis for the scheduling strategy presented in Section A.3 is developed, based on the following definitions and observations.

Definition 9. C_s *is the maximum amount of time needed by a server* τ_s *to process a request from a client.*

Definition 10. RSV_i *is the CBS originally associated to task* τ_i

Definition 11. $\mathcal{S}(i,t)$ *is the set of CBSs serving task* τ_i *at time* t

Definition 12. $\mathcal{T}(i,t)$ *is the set of tasks served by CBS* RSV_i *at time* t

Definition 13. *A CBS* RSV_i *is said to be* active *at time* t *if one of the tasks in* $\mathcal{T}(i,t)$ *can be scheduled at time* t.

Observation 1. *In a client/server system,* $\mathcal{S}(i,t)$ *can contain more than* 1 *CBS only if* τ_i *is a server*

Observation 2. *If* $\exists j : r_{i,j} \leqslant t \leqslant f_{i,j}$ *(a job* $J_{i,j}$ *of task* τ_i *is active), then* $\mathcal{T}(i,t)$ *contains exactly* 1 *non-blocked task.*

Proof. At time $r_{i,j}$, $\mathcal{T}(i,t) = \{\tau_i\}$ and τ_i is not blocked. If τ_i blocks waiting for a reply from server τ_s, then RSV_i is inherited by τ_s, so $\mathcal{T}(i,t) = \{\tau_i, \tau_s\}$. Now, τ_i is blocked and τ_s is active, hence the number of non-blocked tasks in $\mathcal{T}(i,t)$ is still 1. □

Observation 3. *In a single processor system, CBS* RSV_i *is active at time* t *iff* $r_{i,j} \leqslant t \leqslant d_{i,j}$. *One of the tasks in* $\mathcal{T}(i,t)$ *is actually scheduled if* RSV_i *has the earliest scheduling deadline* d_i^s *in the system. In other words, there is no priority inversion between the CBSs.*

This means that if a job of τ_i is started and not finished yet, then RSV_i can be scheduled. If τ_i is blocked (waiting for the completion of an RPC), and RSV_i has the earliest scheduling deadline, then it can be used to serve some other task (the server). Its budget q_i^s can be consumed even if τ_i is blocked. Hence, dimensioning $T_i^s \leqslant P_i$ and $Q_i^s \geqslant C_i$ is not enough to guarantee that τ_i will respect all of its deadlines (Lemma 1 of [4] showed that such an assignment is sufficient if τ_i does not block).

Notice that the previous observation is valid only on single processor systems: on a multi processor system, there can be situations in which τ_i is blocked waiting for the server response, d_i^s is the earliest deadline on its CPU, but the server is scheduled on a different CPU through a CBS RSV_j having $d_j^s < d_i^s$. In this case, RSV_i cannot be used to schedule τ_i nor the server (because the server cannot be simultaneously scheduled on two different CPUs), and suffers a blocking time. Because of this, from this point a single processor system is assumed in the analysis.

Definition 14. *The Inherited time* I_i *is the maximum amount of budget of* RSV_i *consumed when* τ_i *is blocked.*

Using the concepts defined above, it is now possible to guarantee the respect of all the deadlines of a task, as explained in the following lemma.

Lemma 3. *Task* τ_i *is schedulable by a CBS* $RSV_i = (Q_i^s, T_i^s)$ *if* $Q_i^s \geqslant C_i + I_i$ *and* $T_i^s \leqslant P_i$.

Proof. By contradiction, assuming that τ_i is not schedulable. Task τ_i is not schedulable if one of its jobs $J_{i,j}$ misses its deadline: $f_{i,j} > d_{i,j} = r_{i,j} + D_i > r_{i,j} + T_i^s$. Since each job is guaranteed to finish before d_i^s (because $\sum_i Q_i^s/T_i^s \leqslant 1$), this means that $d_i^s > r_{i,j} + T_i^s$. Let $J_{i,k}$ be the first job finishing with $d_i^s > r_{i,k} + T_i^s$. Since $J_{i,k-1}$ finished with

$d_i^s \leqslant r_{i,k-1} + T_i^s$, $r_{i,k} \geqslant r_{i,k-1} + P_i$, and $T_i^s \leqslant P_i$ (by hypothesis), we have $r_{i,k} \geqslant d_i^s$ and a new scheduling deadline $d^s = r_{i,k} + T_i^s$ is generated (recharging the budget to $Q_i^s \geqslant C_i + I_j$).

Since $J_{i,k}$ finishes with $d_i^s > r_{i,k} + T_i^s$, d_i^s has been postponed during the time interval $[r_{i,k}, f_{i,k}]$. This means that in such an interval an amount of CBS budget larger than Q_i^s has been consumed. However, $J_{i,k}$ can consume at most C_i time units of the CBS budget, and at most I_i units of CBS budget are consumed by the server because of inheritance. Hence, $C_i + I_i > Q_i^s$, contradicting the hypothesis. □

Hence, if the inherited time for a task τ_i is known, it is possible to dimension its CBS RSV_i in order to respect all the deadlines. At this point, the problem is how to compute I_i. If τ_i is the only task sending requests to the server τ_s, such a computation is pretty easy, as shown by the following lemma.

Lemma 4. *If τ_i is the only task using server τ_s, then $I_i = C_s$.*

Proof. When τ_i sends a request to τ_s, the server is not executing and the request is immediately processed. To process this request, τ_s inherits τ_i's CBS RSV_i, so it executes with the priority given by d_i^s and consumes the budget q_i^s. In the worst case, τ_s will need an amount of time C_s for serving the request, so it will consume at most C_s time units from q_i^s; hence, $I_i = C_s$. □

If τ_s serves requests coming from multiple clients, I_i can be computed by using Theorem 10, which needs the following definition for its proof.

Definition 15. *A server τ_s is said to be* busy *if it is serving a request. A* busy interval *for server τ_s is a time interval (t_1, t_2) which has the following properties:*

- $\forall t \in (t_1, t_2), \tau_s$ *is busy at time* t
- $\forall t_1' < t_1, (t_1', t_2)$ *is not a busy interval (that is,* $\exists t \in (t_1', t_1) : \tau_s$ *is not busy at time* t*).*

Theorem 10. *If server τ_s serves clients' requests according to the scheduling deadlines of the clients' CBSs, and if every client τ_i sends a request when $q_i^s > 2C_s$, then $I_i = 2C_s$.*

Proof. Consider a client τ_i sending a request at time t: the proof is by induction on the number N of requests arrived in the busy interval (t_1, t_2) including t ($t_1 \leqslant t \leqslant t_2$). **Inductive base**: If client τ_i sends a

request to the server when the server is idle ($t = t_1 \Rightarrow N = 0$), then it can immediately serve the request (using RSV_i's scheduling deadline and budget). This is the same situation as in Lemma 4, hence $I_i = C_s$. Since the request is sent when $q_i^s > 2C_s$, q_i^s is not exhausted and d_i^s is not postponed.

Inductive step: If the property holds when client τ_i's request is the N^{th} request in the busy period, then it holds when the request is the $N+1^{th}$ request in the busy period. Since τ_i's request is not the first request in the busy period, when it arrives the server is already working on a different request, which cannot be preempted. Consider two cases: 1) the currently served request is from higher priority (earlier scheduling deadline) client, and 2) the request which is currently served is from a lower priority client. In case 1), RSV_i is inherited by the server, but it is not used (because τ_s is serving a request from a higher priority client, hence $\mathcal{S}(s,t)$ includes CBSs with scheduling deadlines shorter than d_i^s). Since the property holds for the previous request in the busy period (the N^{th} request) by inductive hypothesis, the scheduling deadline currently used by τ_s is not postponed, hence RSV_i does not become the highest priority CBS in $\mathcal{S}(s,t)$ until the requests from higher priority tasks are completed. Hence, $I_i = C_s$. In case 2), τ_s inherits RSV_i which can become the highest priority CBS in $\mathcal{S}(s,t)$. As a result, up to C_s time units of RSV_i's budget can be consumed to serve the current (lower priority) request (since requests are served in a non-preemptable way). After such a request is served, the situation is the same as in case 1). Hence, $I_i = C_s + C_s = 2C_s$ □

The results from Theorem 10 can be used in two ways: to properly **assign the scheduling parameters** (Q_i^s, T_i^s) of all the clients τ_i using server τ_s so that no deadline is missed in these tasks, or to **enforce a check on the budget before starting an RPC**, so that the interference time of each task has an upper bound.

In the first case, if $\forall \tau_i$ using τ_s, $T_i^s = P_i$ and $Q_i^s \geqslant C_i + 2C_s$, then it is guaranteed that all the deadlines of all these clients are respected. Temporal protection with other tasks not using τ_s is guaranteed, but temporal protection between the clients accessing τ_s is not guaranteed: if a client τ_j executes for more than its expected WCET C_i, it can compromise the schedulability of other clients τ_i accessing the same server.

In the second case, the condition of Theorem 10 (each client sends a request to the server when $q_i^s > 2C_s$) is enforced by the RPC mechanism: before sending a request to the server, q_i^s is checked, and

if $q_i^s < 2C_s$ the deadline is postponed (recharging the budget) so that the condition is respected. Another option would be to block the client until the CBS budget is large enough. Notice that in this case all the deadlines of task τ_i are respected if $Q_i^s \geqslant C_i + 2C_s$, but no assumptions on the maximum budgets of the other clients are needed.

A.5 IMPLEMENTATION

The Linux implementation [36] is based on the `SCHED_DEADLINE` patch [57], which implements the CBS algorithm in the Linux scheduler (by introducing a new `SCHED_DL` scheduling class). While the implementation details can be found in the original paper, this section just recalls the two system calls that have been added to support BWI in case of various kinds of interactions between tasks (for example, RPC, but also other kinds of message passing): `bwi_give_server()` and `bwi_take_back_server()`.

The `bwi_give_server()` system call is used by a client task τ_i to perform the inheritance, by associating an additional task τ_s (the server, in case of client/server interactions) to the client's CBS (as a consequence, the server's priority might be updated, and the server might become a `SCHED_DL` task if it was not): $\mathcal{T}(i,t) = \mathcal{T}(i,t) \cup \{\tau_s\}$.

The `bwi_take_back_server()` system call, instead, is used by τ_i to end the inheritance, by removing τ_s from $\mathcal{T}(i,t)$ (in case of client/server interactions, a client invokes `bwi_take_back_server()` when waking up from an RPC. As a result, the server task's priority is changed back to the original value).

Note that if applications perform RPCs through a system library (for example, `xlib` is used by clients to interact with the X server), then such a library can be modified to call `bwi_give_server()` and `bwi_take_back_server()`. In this way, *unmodified* user applications can transparently take advantage of BWI when accessing a server.

In this work, the `bwi_give_server()` system call has been modified to check if the current budget q_i^s of S_i is larger than $2C_s$, and to postpone the scheduling deadline (and increase q_i^s by Q_i^s) if this condition is not respected: $q_i^s = q_i^s + Q_i^s; d_i^s = d_i^s + T_i^s$. In this way, the the hypotheses of Theorem 10 are always satisfied and the RPC becomes predictable.

A.6 EXPERIMENTAL RESULTS

The implementation described in [36] has been modified as described in Section A.5 and used to perform some experiments on a real system, showing the effectiveness of the proposed approach and the correctness of the analysis. The experiments are based on a server and multiple clients, communicating through UDP sockets. Some other periodic or CPU-hungry tasks which do not interact with the server and the clients have also been added to the system, to make it more realistic and to increase the system workload. Hence, the programs used in the tests are a server, some client, some periodic tasks and some CPU hogs. The *server* program listens on a UDP socket; when it receives an UDP packet, it processes such a packet (spending a variable amount of time, with worst case C_s), and sends back a response to the client. Requests are processed by a dedicated thread, so during this time the server can continue to receive requests and insert them in a priority queue, but since requests are non-premptable the server cannot start to serve a new request until the previous is complete. The deadline of the requesting client is used as a priority to order the pending requests.

Clients are periodic tasks with period P_i. Each job of these tasks executes for a given amount of time (the preamble), then sends a request to the server and waits for the response. After receiving the response from the server, the job executes for some time (the postamble) and then it finishes (the task waits for the next activation).

The "classic" (non interacting) *periodic tasks* execute for a predefined amount of time and then wait for the next activation.

Finally, *CPU hog* tasks, try to consume 100% of the CPU time.

A large number of experiments with different number of tasks and different tasks parameters has been performed, and all the results were consistent with the presented analysis. Here, the results of some simple experiments (which are easier to understand, and can better show the properties of the proposed approach) are presented and discussed. The clients and the periodic tasks are associated to different CBSs, while the server and the *CPU hog* are scheduled using the standard Linux scheduling policy, so they can execute only when CBSs are not active.

In the first example, two clients communicate with a server having $C_s = 5ms$. By default, the server is scheduled by using the `SCHED_OTHER` Linux scheduling policy (non real-time), and a CPU hog task

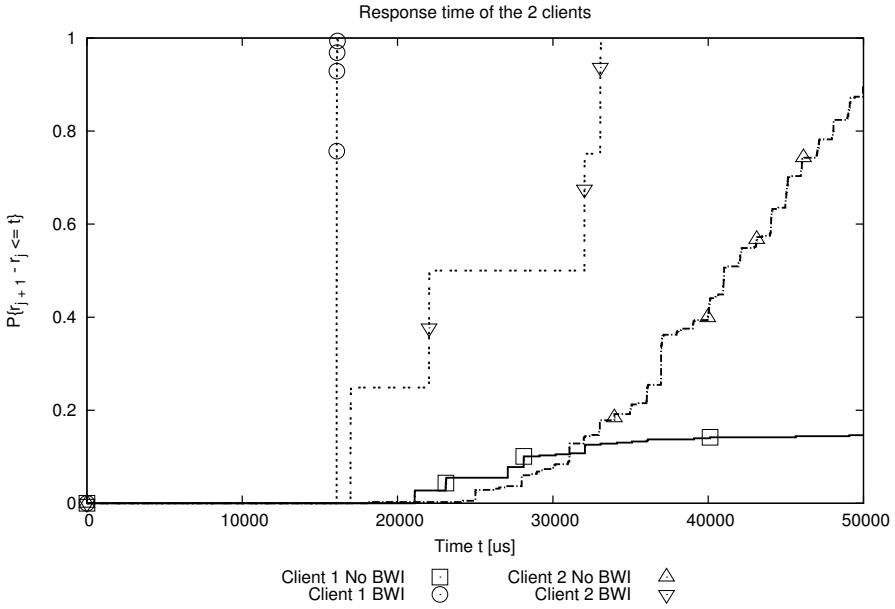

Figure A.1: First example: Client τ_1: deadline $40ms$, Client τ_2: deadline $50ms$.

competes with it. The client τ_1 executes for $4ms$ before sending a request to the server, and executes for $7ms$ after receiving a reply from the server (so, its WCET is $C_1 = 4ms + 7ms = 11ms$). The task's period is $P_1 = 40ms$. The client τ_2 executes for $1ms$ before sending a request to the server, and executes for $11ms$ after receiving the server's reply (so, its WCET is $C_2 = 1ms + 11ms = 12ms$). The tasks' period is $P_2 = 50ms$. According to theorem 10, Q_1^s should be at least $C_1 + 2C_s = 11ms + 2 * 5ms$, hence in the test τ_1 has been scheduled by a CBS $RSV_1 = (21ms, 40ms)$. Similarly, τ_2 is associated to $RSV_2 = (22ms, 50ms)$.

The resulting response times obtained with and without using BWI as proposed in this appendix have been measured, and the CDFs of such times are shown in Figure A.1. By looking at the figure it is possible to notice that since the CBS parameters are properly dimensioned when using BWI all the deadlines are respected (all the response times are less than $40ms$ for τ_1, and less than $50ms$ for τ_2). When BWI is not used, instead, the probability to miss a deadline is pretty high even if each task is reserved enough time.

In a second example, a periodic task τ_3 with $C_3 = 5ms$ and $P_3 = 100ms$ associated to a $RSV_3 = (5ms, 100ms)$ CBS is added to the system. The CDFs of the response times of the three tasks are displayed in Figure A.2, showing that τ_3 is not disturbed by the execution of the other tasks in the system (all its deadlines are respected, as expected).

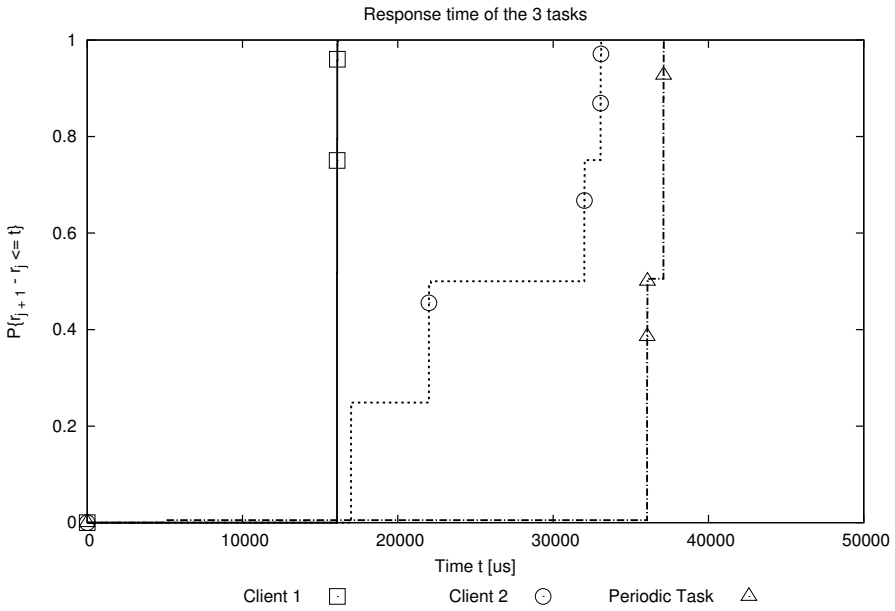

Figure A.2: Second example: a periodic task τ_3 (not interacting with the other tasks) is added to the system.

Summing up, this experiment shows that there is temporal protection between the periodic task (not interacting with the other tasks in the system) and the group of tasks formed by the clients and their server.

In the third example, the system contains three clients τ_1, τ_2, and τ_3, with the parameters described in Table A.1. As it possible to notice from the table, all of the tasks respect the condition necessary for the Theorem 10, so no missed deadlines are expected when BWI is used. Figure A.4 shows the CDF of the response times of the three clients when BWI is used, and confirms this expectation. Figure A.3 presents the CDFs obtained repeating the same experiment without BWI, and show that in this case the response times diverge.

The fourth example shows that the temporal isolation between the tasks in the system is still respected even if the maximum budgets of the various tasks are not dimensioned according to Theorem 10 (because the kernel checks the current budget of a task before performing an inheritance operation). To this purpose, an additional periodic task has been added to the system, and the scheduling parameters of the various tasks have been assigned as shown in Table A.2. The computation time of the server is equal to 15ms. As it is possible to notice from the table, the maximum budget reserved to client τ_2 is not enough to guarantee that all the deadlines of this task are respected (it is $Q_2^s = 31\text{ms}$, while it should be at least $Q_2^s = 5 + 2 * 15 + 11 = 41\text{ms}$).

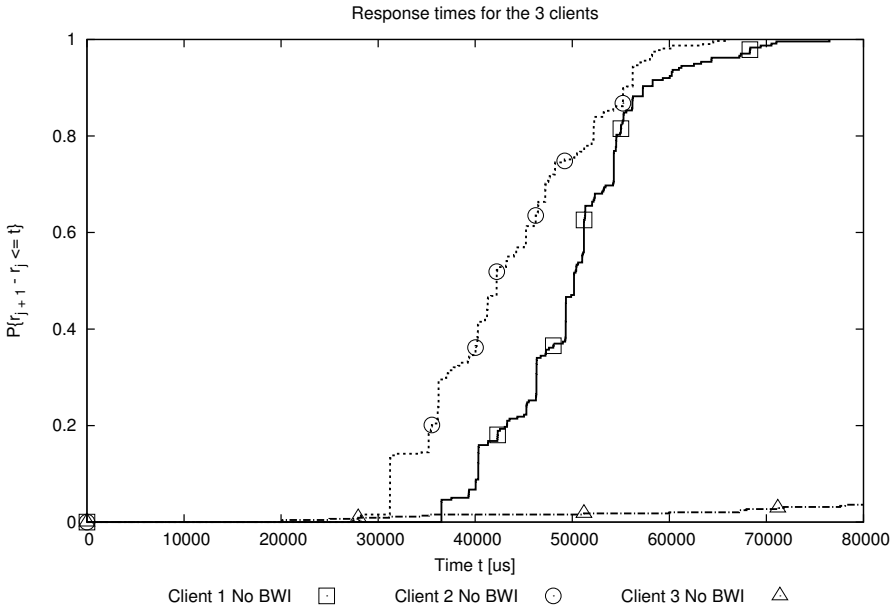

Figure A.3: Third experiment: three clients, no BWI.

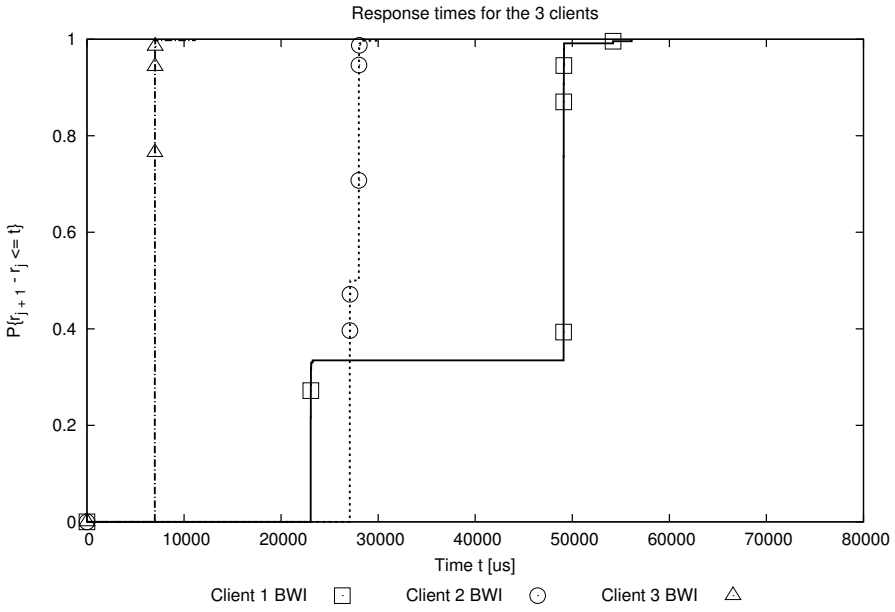

Figure A.4: Third experiment: three clients, with BWI.

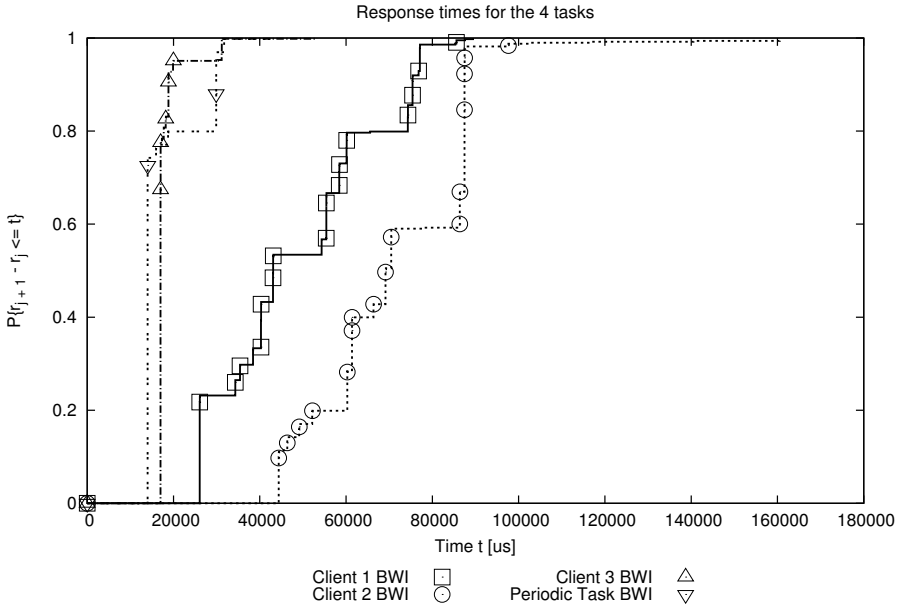

Figure A.5: Fourth experiment: three clients, with client τ_2 not having enough maximum budget. In this case, BWI is used: note that τ_2 is the only task missing some deadlines.

However, there should be temporal isolation between τ_2 and the other tasks running in the system, hence the other tasks are not expected to miss their deadlines. The response times obtained when using BWI are distributed as shown in Figure A.5: note that the plot for client τ_2 has a long tail, arriving to 160ms. This means that there is a non 0 probability for some job of τ_2 to have a response time as large as 160ms, and since the task's period is $P_2 = 120ms$ some deadlines for τ_2 are missed (as expected). By looking at the plots for the other tasks it is also possible to notice that according to the figure τ_2 is the only task in the system missing some deadlines (again, this is consistent with the expectations).

Summing up, this experiment shows that if the kernel checks $q_i^s \geqslant 2C_s$ before starting an RPC, then there is temporal isolation even between clients accessing the same server (provided that C_s is known and correctly estimated). Finally, the same experiment has been repeated without using BWI, and the periodic task (not performing any RPC) resulted to be the only one respecting its deadlines. This shows that even if the scheduling parameters of some tasks are not correctly assigned, BWI still helps in improving the response times.

Table A.1: Task set for the third example.

Client	Preamble	Postamble	Q_i^s	P_i
τ_1	4ms	7ms	21ms	80ms
τ_2	5ms	11ms	26ms	60ms
τ_3	1ms	1ms	12ms	40ms

Table A.2: Task set for the fourth example. Client τ_2 is not reserved enough time to respect all of its deadlines.

Client	Preamble	Postamble	Q_i^s	P_i
τ_1	4ms	7ms	41ms	140ms
τ_2	5ms	11ms	31ms	120ms
τ_3	1ms	1ms	32ms	100ms

A.7 CONCLUSIONS AND FUTURE WORK

This appendix analysed the schedulability of tasks interacting according to the client/server paradigm and scheduled by a reservation-based scheduler. The predictability of the tasks' interactions is increased by using the BWI algorithm, and by modifying the RPC mechanism in order to check the current budget of a client before sending requests to the client. As a result of the analysis, it is possible to assign the reservations' parameters in order to guarantee the respect of the deadlines of a single task, or of a group of tasks. Temporal isolation is guaranteed between tasks that do not interact, but also between tasks accessing the same server via RPC (thanks to the modifications to the RPC mechanism).

This solution has been implemented in the Linux kernel, has been used to perform various experiments in order to verify the correctness of the analysis and to see how the proposed approach works in practice.

Work is currently in progress to extend the proposed analysis to multi-processor systesm (as already noticed in Section A.4, the analysis presented in this appendix is only valid on single processor systems). When there is more than 1 CPU, it is not guaranteed that a CBS does not suffer any blocking time (on single processor systems, if BWI is used there can be an inherited time I_i "stolen" from the budget q_i^s, but there is no blocking time). In particular, issues like the following can occur:

- Client τ_i, running on CPU m, starts an RPC with server τ_s at time t_1 and blocks waiting for the servers' reply. At time t_1, the current budget of RSV_i is q_i^s
- Server τ_s is executing on CPU m, and serving a different client τ_j with $d_j^s < d_i^s$. Hence, RSV_i is inherited, but not used by τ_s. So, d_i^s is the earliest deadline CBS on CPU n, but RSV_i cannot execute (and its budget is not decreased
- At time t_2, τ_s can finally serve τ_i's request, and starts executing with the priority given by d_i^s and consuming q_i^s. However, since q_i^s has not been decreased in the meanwhile it can be that $q_i^s/(d_i^s - t_2) > Q_i^s/T_i^s$. Hence, RSV_i can compromise the schedulability of other CBSs running on CPU m.

The Multiprocessor BWI algorithm [88] addresses this issue by using busy waiting to avoid the blocking time. In this way, q_i^s is decreased while τ_i is waiting for τ_s and τ_s cannot use RSV_i for executing. This idea can be adapted to be used in case of client/server interactions, and the proposed analysis can be modified accordingly. This is currently under development, and mainly requires two modifications to the previous analysis: accounting of the busy waiting in the interference time (or as an increase to the WCET), and analysis of the FIFO queueing used by multiprocessor BWI (this is partly already done in the multiprocessor BWI appendix).

The busy waiting introduced by multiprocessor BWI can be removed by accepting that a CBS RSV_i cannot be scheduled even if τ_i has an active job. In this case, when at time t_2 RSV_i can be used again to schedule a task, q_i^s has to be adjusted in order to avoid causing blocking times on other tasks. This can be done by setting $q_i^s = \min\{q_i^s, Q_/^s T_i^s (d_i^s - t_2)\}$.

BIBLIOGRAPHY

[1] L. Palopoli, L. Abeni, G. Buttazzo, F. Conticelli, and M. Di Natale, "Real-time control system analysis: an integrated approach," *Real-Time Systems Symposium, 2000. Proceedings. The 21st IEEE*, pp. 131–140, 2000.

[2] D. Fontanelli, G. L., and L. Palopoli, "Adaptive reservations for feedback control," in *Decision and Control (cdc2010), 2010 Proc. of 48th IEEE Conference on*, Atlanta, Georgia, Dec. 2010.

[3] D. Fontanelli, L. Greco, and L. Palopoli, "Deterministic and stochastic qos provision for real-time control systems," in *Proc. of 17th Real—time and Embedded Technology and Applications Symposium (RTAS11)*, Chicago, USA, April 2011.

[4] L. Abeni and G. Buttazzo, "Integrating multimedia applications in hard real-time systems," in *Proceedings of the IEEE Real-Time Systems Symposium*, Madrid, Spain, December 1998.

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

[6] L. Abeni and G. Buttazzo, "Qos guarantee using probabilistic dealines," in *Proceedings of the IEEE Euromicro Conference on Real-Time Systems*, York, England, June 1999.

[7] ——, "Stochastic analysis of a reservation-based system," in *Proceedings of the 15th International Parallel and Distributed Processing Symposium.*, San Francisco, California, April 2001.

[8] T.-S. Tia, Z. Deng, M. Shankar, M. Storch, J. Sun, L.-C. Wu, and J. W.-S. Liu, "Probabilistic performance guarantee for real-time tasks with varying computation times," in *Proceedings of the IEEE Real-Time Technology and Applications Symposium*, Chicago,Illinois, January 1995, pp. 164–173.

[9] M. K. Gardner and J. Liu, "Analyzing stochastic fixed-priority real-time systems," in *Proceedings of the 5th International Confer-*

ence on Tools and Algorithms for Construction and Analysis of Systems (TACAS99). Springer, March 1999, pp. 44–58.

[10] L. Cucu and E. Tovar, "A framework for the response time analysis of fixed-priority tasks with stochastic inter-arrival times," *ACM SIGBED Review - Special issue: The work-in-progress (WIP) session of the RTSS 2005*, vol. 3, no. 1, pp. 7–12, January 2006.

[11] J. L. Diaz, D. F. Garcia, K. Kim, C. G. Lee, L. Lo Bello, J. M. López, S. L. Min, and O. Mirabella, "Stochastic analysis of periodic real-time systems," in *Real-Time Systems Symposium, 2002. RTSS 2002. 23rd IEEE*. IEEE, 2003, pp. 289–300.

[12] J. L. Diaz, J. M. López, M. Garcia, A. M. Campos, K. Kim, and L. Lo Bello, "Pessimism in the stochastic analysis of real-time systems: Concept and applications," in *Real-Time Systems Symposium, 2004. Proceedings. 25th IEEE International*. IEEE, 2005, pp. 197–207.

[13] K. Kim, J. L. Diaz, L. Lo Bello, J. M. López, C. G. Lee, and S. L. Min, "An exact stochastic analysis of priority-driven periodic real-time systems and its approximations," *IEEE Transactions on Computers*, vol. 54, no. 11, pp. 1460–1466, 2005.

[14] G. A. Kaczynskit, L. Lo Bello, and T. Nolte, "Deriving exact stochastic response times of periodic tasks in hybrid priority-driven soft real-time systems," in *Proceedings of the IEEE Conference on Emerging Technologies and Factory Automation (ETFA 2007)*, September 2007.

[15] A. Mills and J. Anderson, "A stochastic framework for multiprocessor soft real-time scheduling," in *Proceedings of the 16th IEEE Real-Time and Embedded Technology and Applications Symposium*. IEEE, April 2010, pp. 311–320.

[16] D.-I. Kang, R. Gerber, and M. Sakena, "Performance-based design of distributed real-time systems," in *Proceedings of the IEEE Real-Time Technology and Applications Symposium*, June 1997, pp. 2–13.

[17] A. K. Atlas and A. Bestavros, "Statistical rate monotonic scheduling," in *Proceedings of the IEEE Real-Time Systems Symposium*, Madrid, Spain, December 1998.

[18] C.-J. Hamann, L. Reuther, J. Wolter, H. Haertig, J. Loser, and S. Schonberg, "Quality-assuring scheduling-using stochastic be-

havior to improve resource utilization," in *Proceedings of the 22nd IEEE Real-Time Systems Symposium (RTSS'01)*, London, December 2001.

[19] C. L. Liu and J. Layland, "Scheduling alghorithms for multiprogramming in a hard real-time environment," *Journal of the ACM*, vol. 20, no. 1, 1973.

[20] M. Joseph and P. Pandya, "Finding response times in a real-time system," *The Computer Journal*, vol. 29, no. 5, p. 390, 1986.

[21] E. Bini, T. H. C. Nguyen, P. Richard, and S. K. Baruah, "A response-time bound in fixed-priority scheduling with arbitrary deadlines," *IEEE Trans. Computers*, vol. 58, no. 2, pp. 279–286, 2009.

[22] E. Bini and G. C. Buttazzo, "Schedulability analysis of periodic fixed priority systems," *IEEE Trans. Computers*, vol. 53, no. 11, pp. 1462–1473, 2004.

[23] A. F. Mills and J. H. Anderson, "A multiprocessor server-based scheduler for soft real-time tasks with stochastic execution demand," in *Proceedings of the 17th IEEE Real-Time and Embedded Technology and Applications Symposium*, vol. 1. IEEE, 2011, pp. 207–217.

[24] J. P. Lehoczky, "Real-time queueing theory," in *Proceedings of the IEEE Real-Time Systems Symposium*, December 1996.

[25] G. Buttazzo and E. Bini, "Optimal dimensioning of a constant bandwidth server," in *Real-Time Systems Symposium, 2006. RTSS'06. 27th IEEE International*. IEEE, 2006, pp. 169–177.

[26] A. Papoulis, "Probability, random variables, and stochastic processes." 2000.

[27] D. P. Heyman and J. M. Sobel, *Stochastic Models in Operations Research, Vol. I: Stochastic Processes and Operating Characteristics*. McGraw-Hill, 1982.

[28] D. Lindley, "The theory of queues with a single server," *Proc. Camb. Phil. SOc.*, vol. 48, pp. 277–289, 1952.

[29] K. S. Refaat and P.-E. Hladik, "Efficient stochastic analysis of real-time systems via random sampling," in *Proceedings of the 22nd*

Euromicro Conference on Real-Time Systems, Brussels, Belgium, July 2010, pp. 175–183.

[30] L. Palopoli, N. Manica, and L. Abeni, "Numerically efficient probabilistic guarantees for resource reservations," in *Proc. of the 17th IEEE International Conference of Emerging Technologies and Factory Automation (ETFA 2012)*, Krakow, Poland, September 2012.

[31] G. Latouche and V. Ramaswami, *Introduction to matrix analytic methods in stochastic modeling*. Society for Industrial Mathematics, 1999.

[32] G. Latouche and Ramaswami, "A Logarithmic Reduction Algorithm for Qouasi-Birth-Death Processes," *Journal of Applied Probability*, pp. 650–674, 1993.

[33] L. Palopoli, D. Fontanelli, N. Manica, and L. Abeni, "An analytical bound for probabilistic deadline," in *Proceedings of the 24th Euromicro Conference on Real-Time Systems*. Pisa, Italy: IEEE, September 2012, pp. 179–188.

[34] C. G. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006.

[35] G. Latouche and V. Ramaswami, *Introduction to matrix analytic methods in stochastic modeling*. Society for Industrial Mathematics, 1987, vol. 5.

[36] L. Abeni, N. Manica, and L. Palopoli, "Efficient and robust probabilistic guarantees for real-time tasks," *Journal of Systems and Software*, vol. 85, no. 5, p. 1147–1156, May 2012.

[37] L. Kleinrock, *Queuing Systems*. Wiley-Interscience, 1975.

[38] G. Modena, L. Abeni, and L. Palopoli, "Providing qos by scheduling interrupt threads," in *Work in Progress of the 14th IEEE Real-Time and Embedded Technology and Applications Symposium, (RTAS 2008)*, St. Louis, MO, April 2008.

[39] C. W. Mercer, S. Savage, and H. Tokuda, "Processor capacity reserves: Operating systems support for multimedia applications," in *Proceedings of the IEEE International Conference on Multimedia Computing and Systems*, May 1994.

[40] P. Druschel and G. Banga, "Lazy receiver processing (LRP): A network subsystem architecture for server systems," in *Proceedings of the second USENIX symposium on Operating Systems Design and Implementation (OSDI)*. Seattle, Washington, United States: [New York, NY, ACM Special Interest Group on Operating Systems], 1996, pp. 261–276.

[41] J. C. Mogul and K. K. Ramakrishnan, "Eliminating receive livelock in an interrupt-driven kernel," *ACM Transactions on Computer Systems*, vol. 15, August 1997.

[42] J. Brustoloni, E. Gabber, A. Silberschatz, and A. Singh, "Signaled receiver processing," in *Proceedings of the USENIX Annual Technical Conference*, June 2000, pp. 211–224.

[43] R. Black, P. Barham, A. Donnelly, and N. Stratford, "Protocol implementation in a vertically structured operating system," in *Proceedings of the 22nd Annual Conference on Local Computer Networks*, 1997.

[44] Y.Zhang and R. West, "Process-aware interrupt scheduling and accounting," in *Proceedings of the 27th IEEE International Real-Time Systems Symposium*, Rio de Janeiro, Brazil, December 2006.

[45] H. Haertig, R. Baumgartl, M. Borriss, C. joachim Hamann, M. Hohmuth, F. Mehnert, L. Reuther, S. Schönberg, and J. Wolter, "DROPS - OS support for distributed multimedia applications," in *In Proceedings of the Eighth ACM SIGOPS European Workshop*, Sintra, Portugal, September 1998.

[46] H. Haertig and M. Roitzsch, "Ten years of research on L4-based real-time systems," in *Proceedings of the Eighth Real-Time Linux Workshop*, Lanzhou, China, 2006.

[47] K. Jeffay and G. Lamastra, "A comparative study of the realization of rate-based computing services in general purpose operating systems," in *Proceedings of the Seventh IEEE International Conference on Real-Time Computing Systems and Applications (RTCSA)*, Cheju Island, South Korea, 2000, pp. 81–90.

[48] S. Ghosh and R. Rajkumar, "Resource management of the OS network subsystem," in *Proceedings of the Fifth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing, 2002.(ISORC 2002)*, 2002, pp. 271–279.

[49] S. Rostedt, "Internals of the rt patch," in *Proceedings of the Linux Symposium*, Ottawa, Canada, June 2007.

[50] L. Abeni and G. Lipari, "Implementing resource reservations in linux," in *Proceedings of Fourth Real-Time Linux Workshop*, Boston, MA, December 2002.

[51] M. Lewandowski, M. Stanovich, T. Baker, K. Gopalan, and Wang, "Modeling device driver effects in real-time schedulability analysis: Study of a network driver," in *Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium*, Bellevue, WA, 2007.

[52] T. Baker, A. Wang, and M. J. Stanovich, "Fitting linux device drivers into an analyzable scheduling framework," in *Proceedings of the Workshop on Operating Systems Platforms for Embedded Real-time Applications*, Pisa, Italy, July 2007.

[53] L. Henriques, "Threaded IRQs on Linux PREEMPT-RT," in *Proceedings of Fifth International Workshop on Operating Systems Platforms for Embedded Real-Time Applications (OSPERT)*, Dublin, Ireland, June 2009.

[54] L. Abeni, N. Manica, and L. Palopoli, "Reservation-based scheduling for irq threads," in *Proceedings of the 11th Real-Time Linux Workshop*, Dresden, Germany, September 2009.

[55] N. Manica, L. Abeni, and L. Palopoli, "Reservation-based interrupt scheduling," in *Proceedings of the 16th IEEE Real-Time and Embedded Technology and Applications Symposium, (RTAS 2010)*, Stockholm, Sweden, April 2010.

[56] I. Leslie, D. McAuley, R. Black, T. Roscoe, P. Barham, D. Evers, R. Fairbairns, and E. Hyden, "The design and implementation of an operating system to support distributed multimedia applications," *IEEE Journal on Selected Areas in Communications*, vol. 14, no. 7, pp. 1280–1297, 1996.

[57] D. Faggioli, F. Checconi, M. Trimarchi, and C. Scordino, "An EDF scheduling class for the Linux kernel," in *Proceedings of the Eleventh Real-Time Linux Workshop*, Dresden, Germany, September 2009.

[58] P. Rallo, N. Manica, and L. Abeni, "Inferring temporal behaviours through kernel tracing," DISI - University of Trento, Tech. Rep. DISI-10-021, April 2010.

[59] G. Buttazzo, G. Lipari, L. Abeni, and M. Caccamo, *Soft Real-Time Systems: Predictability vs. Efficiency*. Springer Verlag, 2005.

[60] I. Stoica, H. Abdel-Wahab, K. Jeffay, S. K. Baruah, J. E. Gehrke, and C. G. Plaxton, "A proportional share resource allocation algorithm for real-time, time-shared systems," in *Proceedings of the IEEE Real-Time Systems Symposium*, December 1996.

[61] C. W. Mercer, S. Savage, and H. Tokuda, "Processor capacity reserves for multimedia operating systems," Carnegie Mellon University, Pittsburg, Tech. Rep. CMU-CS-93-157, May 1993.

[62] D. Reed and R. F. (eds.), "Nemesis, the kernel – overview," May 1997.

[63] L. Sha, R. Rajkumar, and J. P. Lehoczky, "Priority inheritance protocols: An approach to real-time synchronization," *IEEE Transactions on Computers*, vol. 39, no. 9, September 1990.

[64] J. E. Sasinowski and J. K. Strosnider, "ARTIFACT: A platform for evaluating real-time window system designs," in *IEEE Real-Time Systems Symposium*, 1995, pp. 342–352. [Online]. Available: citeseer.ist.psu.edu/sasinowski95artifact.html

[65] N. Feske and H. Hartig, "Dope - a window server for real-time and embedded systems," in *Proc.of 24th IEEE Real-Time Systems Symposium (RTSS 2003)*, 2003, pp. 74–77.

[66] J. Shapiro, J. Vanderburgh, E. Northup, and D. Chizmadia, "Design of the eros trusted window system," in *Proceedings of the 13th USENIX Security Symposium*, 2004.

[67] H. Tokuda, T. Nakajima, and P. Rao, "Real-time mach: Toward a predictable real-time system," in *USENIX Mach Workshop*, October 1990, pp. 73–82.

[68] G. C. Buttazzo, "Hartik: A real-time kernel for robotics applications," in *Proceedings of the IEEE Real-Time Systems Symposium*, December 1993.

[69] K. Jeffay, D. Stone, and D. Poirier, "Yartos: kernel support for efficient, predictable real-time systems," 1992.

[70] I. Molnar *et al.*, "Real-time linux wiki," http://rt.wiki.kernel.org.

[71] S. Childs and D. Ingram, "The linux-srt integrated multimedia operating system: Bringing qos to the desktop," in *Proceedings of the IEEE Real-Time Technology and Applications Symposium (RTAS 2001)*, Taipei, Taiwan, 2001.

[72] W. Tarreau, "The ocbench scheduler benchmark," http://linux.1wt.eu/sched.

[73] S. Saewong, R. R. Rajkumar, J. P. Lehoczky, and M. H. Klein, "Analysis of hierarchical fixed-priority scheduling," *ecrts*, vol. 00, p. 173, 2002.

[74] G. Lipari and E. Bini, "A methodology for designing hierarchical scheduling systems," *Journal of Embedded Computing*, vol. 1, no. 2, pp. 257–269, 2004.

[75] X. A. Feng and A. K. Mok, "A model of hierarchical real-time virtual resources," in *RTSS '02: Proceedings of the 23rd IEEE Real-Time Systems Symposium (RTSS'02)*. Washington, DC, USA: IEEE Computer Society, 2002, p. 26.

[76] Z. Deng and J. W. S. Liu, "Scheduling real-time applications in open envirovment," in *Proceedings of the IEEE Real-Time Systems Symposium*, December 1997.

[77] J. K. Strosnider, J. P. Lehoczky, and L. Sha, "The deferrable server algorithm for enhanced aperiodic responsiveness in hard-real-time environments," *IEEE Transactions on Computers*, vol. 4, no. 1, January 1995.

[78] T. P. Baker, "Stack-based scheduling of real-time processes," *Real-Time Systems*, no. 3, 1991.

[79] D. D. Niz, L. Abeni, S. Saewong, and R. R. Rajkumar, "Resource sharing in reservation-based systems," in *Proceedings of the 22nd IEEE Real-Time Systems Symposium (RTSS'01)*, December 2001.

[80] G. Lipari, G. Lamastra, and L. Abeni, "Task synchronization in reservation-based real-time systems," *IEEE Transactions on Computers*, vol. 53, no. 12, pp. 1591–1601, December 2004.

[81] N. Watkins, J. Straub, and D. Niehaus, "A flexible scheduling framework supporting multiple programming models with arbitrary semantics in linux," in *Proceedings of the Eleventh Real-Time Linux Workshop*, Dresden, Germany, September 2009.

[82] M. Danish, Y. Li, and R. West, "Virtual-CPU scheduling in the quest operating system," in *Proceedings of the 17th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS 2011)*. Chicago (IL): IEEE, April 2011, pp. 169–179.

[83] U. Steinberg, A. Böttcher, and B. Kaue, "Timeslice donation in component-based systems," in *Proceedings of the Workshop on Operating Systems Platforms for Embedded Real-Time applications (OSPERT 2010)*, Brussels, Belgium, 2010, p. 91.

[84] G. Parmer, "The case for thread migration: Predictable IPC in a customizable and reliable OS," in *Proceedings of the Workshop on Operating Systems Platforms for Embedded Real-Time applications (OSPERT 2010)*, Brussels, Belgium, 2010, p. 91.

[85] G. Lamastra, G. Lipari, and L. Abeni, "A bandwidth inheritance algorithm for real-time task synchronization in open systems," in *Proceedings of the 22nd IEEE Real-Time Systems Symposium (RTSS'01)*, London (UK), December 2001.

[86] U. Steinberg, J. Wolter, and H. Haertig, "Fast component interaction for real-time systems," in *Proceedings of the 17th Euromicro Conference on Real-Time Systems (ECRTS05)*, Palma de Mallorca (Spain), July 2005.

[87] T. P. Baker, "A stack-based allocation policy for realtime processes," in *Proceedings of the IEEE Real Time Systems Symposium*, december 1990.

[88] D. Faggioli, G. Lipari, and T. Cucinotta, "The multiprocessor bandwidth inheritance protocol," in *Real-Time Systems (ECRTS), 2010 22nd Euromicro Conference on*. IEEE, 2010, pp. 90–99.

[89] S. K. Baruah, A. K. Mok, and L. E. Rosier, "Preemptively scheduling hard-real-time sporadic tasks on one processor," 1993, pp. 182–190.

[90] T. Cucinotta, L. Abeni, L. Palopoli, and F. Checconi, "The wizard of os: a heartbeat for legacy multimedia applications," in *Proceedings of the 7^{th} IEEE Workshop on Embedded Systems for Real-Time Multimedia (ESTIMedia 2009)*, Grenoble, France, October 2009.

[91] S. Rostedt, "Finding origins of latencies using ftrace," in *Proceedings of the Eleventh Real-Time Linux Workshop*, Dresden, Germany, September 2009.