QoS Guarantee Using Probabilistic Deadlines

Luca Abeni and Giorgio Buttazzo
Scuola Superiore S. Anna, Pisa
{luca,giorgio}@sssup.it

Abstract

This paper presents a probabilistic approach to guarantee the performance of a real-time system. While traditional real-time system analysis tends to guarantee that each task instance will complete its execution before its absolute deadline (hard guarantee), our approach permits to estimate the probability that it will happen.

Such a statistical guarantee is performed based on interarrival and execution times probability distributions, rather than their worst case values. The advantage of a probabilistic approach is a more efficient usage of system resources, allowing to give a certain level of deadline guarantee to task sets that the classical schedulability analysis would reject.

1. Introduction

Real-time activities are traditionally characterized by temporal constraints, called deadlines [17]. Each task τ_i is a stream of jobs $J_{i,j}$, where $J_{i,j}$ is released at time $r_{i,j}$, is characterized by a computation time $c_{i,j}$, and must finish before its absolute deadline $d_{i,j}$. Depending on the consequence of a deadline miss, a real-time task is said to be hard, soft, or firm.

A hard task cannot tolerate any deadline miss, since a single job that finishes after its deadline could jeopardize the entire system. A soft task can tolerate jobs that finish after their deadlines, whereas a firm task can tolerate only some job failure. Typically, a firm job should either finish before its deadline or not execute at all. In other words, a soft job that misses its deadline can still do some useful work, while a firm job that misses its deadline is useless, though it does not jeopardize the system.

Due to their criticality, hard real-time tasks must be guaranteed off-line based on their Worst Case Execution Time (WCET) and Minimum Interarrival Time (MIT). If a hard task is accepted, each of its jobs is guaranteed to finish before its deadline. Firm tasks are generally subject to on-line admission test, so a job is released only if it is guaranteed to complete before its deadline, otherwise it is rejected.

Soft tasks are often accepted in the system without any form of guarantee. In order not to jeopardize hard tasks' schedulability, soft tasks can be served in background with respect to hard and firm tasks or, to improve responsiveness, by an *aperiodic server*, which can be used both under static priority assignment [14, 21, 13], and dynamic priority assignment [23, 6, 22]. When an aperiodic sever is used to handle soft tasks, the mean response time can be reduced, but in general it is difficult to guarantee the performance of each single task. In fact, typically, th objective of an aperiodic server is to reduce the mean response time, rather than meeting individual timing constraints. A possible (but inefficient) solution for handling soft tasks characterized by timing constraints is to treat them as hard or firm real-time tasks.

A firm task model can be a good trade-off between hard and soft models, since it permits to guarantee that each accepted job will finish before its deadline. In this way it is possible to achieve a higher CPU utilization, while providing a certain level of guarantee on non-hard tasks. In general, it is possible to ensure a Quality of Service (QoS) based on the maximum number of consecutive skips, the minimum distance between two skips [12, 18], or the completion probability [2]. In [4, 3], the bandwidth saved by skipped jobs is exploited for enhancing aperiodic responsiveness.

Summarizing, classical schedulability tests require the exact knowledge of the WCET, $C_i = max_j\{c_{i,j}\}$, and the MIT, $T_i = min_j\{r_{i,j+1} - r_{i,j}\}$, so each hard task must be characterized by two parameters:

$$\tau_i = (C_i, T_i).$$

This model can cause an underutilization of the CPU if the MIT is much smaller than the mean interarrival time, or the WCET is much greater than the mean execution time.

In many practical real-time applications, the requirements for a task are less stringent than real-time theory usually permits. A more flexible approach can be to specify the deadline completion probability: for example, a task can still be acceptable if the probability of respecting a deadline is greater than a given value.

In [24], a probabilistic analysis of the Rate Monotonic (RM) scheduling algorithm is performed, and some solutions to guarantee a completion probability are presented.

In [2], the RM algorithm is modified to perform on-line acceptance of single jobs for ensuring (off line) a minimum probability that a job is accepted; these two works are both limited to a static priority (RM) assignment and to a semi-periodic (fixed interarrival times) task model.

In [10], a Time Division Multiplexing (TDM) model is used to share resources between tasks in a distributed system. In this paper and in two successive works ([11, 9]) a statistical approach is used at design phase to compute the probability distribution of tasks' finishing times and to check whether a distributed system verifies some given constraints on latency and minimum output rate.

In [15] a statistical analysis of the Earliest Deadline First (EDF) scheduling algorithm is presented. This analysis is based on the Real-Time Queuing theory, which adapts the classical queuing theory to handle queues containing clients characterized by a deadline. Other works study EDF from a probabilistic point of view: for example, [19, 7] prove that EDF possesses certain optimality properties minimizing lateness probabilities.

The use of a firm task model can be useful for solving this underutilization problem in the presence of relaxed (non-hard) tasks, but it still requires the a-priori knowledge of jobs' execution times. This problem makes firm tasks useless each time that the execution times cannot be exactly estimated. Another problem with the firm model is that jobs that cannot be guaranteed are rejected, though they could give some value also finishing after their deadline.

This problem can be addressed by adopting a soft task model, while using the *completion probability* as a metric for expressing and guaranteeing a desired QoS. While the aim of the classical hard schedulability analysis is to guarantee that each job will finish within its deadline, the aim of a QoS guarantee is to ensure that a job will finish within its soft deadline with a given probability. To perform a QoS guarantee we need to describe the tasks' parameters in a probabilistic way and introduce the concept of probabilistic deadline.

According to this approach, each task is described by a pair of Probability Distribution Functions (PDF):

$$\tau_i = (U_i(c), V_i(t))$$

where $U_i(c)$ is the probability that job $J_{i,j}$ has execution time c, and $V(t)$ is the probability that jobs' interarrival time is t. Hence,

$$\begin{aligned} U_i(c) &= P\{c_{i,j} = c\} \\ V_i(t) &= P\{r_{i,j+1} - r_{i,j} = t\}. \end{aligned}$$

Given a task set $\Gamma = \{\tau_1, ..., \tau_n\}$, the QoS guarantee is based on the notion of *probabilistic deadline*, δ. Our QoS guarantee does not require that the probabilistic deadline is always respected, but that it is respected with a given probability

$$X_i(\delta) = P\{f_{i,j} \le r_{i,j} + \delta\}$$

where $f_{i,j}$ is the job's finishing time, and $X_i(\delta)$ is the probability that job $J_{i,j}$ finishes before a deadline $d_{i,j} = r_{i,j} + \delta$.

Definition 1 *A task τ_i is guaranteed to respect a constraint (δ, p), if $X_i(\delta) \ge p$.*

The rest of the paper is organized as follows. Section 2 presents our scheduling model. Section 3 describes the analysis for performing a probabilistic guarantee. Section 4 illustrates some simulation results which validate the analysis. Section 5 shows how the proposed method can be used at design stage. Section 6 presents our conclusions.

2. Scheduling Model

Our work can be considered an extension of classical real-time scheduling. Conventional real-time systems schedule tasks by period (Rate Monotonic) [17], by relative deadlines (Deadline Monotonic) [16], or by absolute deadlines (Earliest Deadline First) [17]. This scheduling framework is acceptable for hard real-time activities, but it is too inefficient for soft real-time systems. For example, in a multimedia application the WCET and the MIT of the task cannot always be precisely estimated, and in many cases these parameters have a large variance.

On the other hand, performing the analysis using average values (rather than worst case ones) is not safe, because a misbehaved task would affect the execution of other tasks, causing them to miss their deadlines.

This problem can be solved by using a *bandwidth reservation strategy*: each task is assigned a fraction of the CPU bandwidth and the scheduling mechanism ensures that it will not require more than the reserved bandwidth. In this way, a task demanding too much time is just delayed, and cannot compromise the QoS guaranteed for the others; its worst-case finishing time only depends on its load.

Although any scheduling algorithm could be used in order to implement a reservation strategy (for example, in [20] a static priority assignment, derived from DM, is used to implement Processor Capacity Reserve), we decided to use a dynamic priority scheduler, based on EDF, to achieve better CPU utilization. Each soft task is handled by a dedicated Constant Bandwidth Server (CBS) [1]. The server mechanism assigns each job an initial deadline, which guarantees the reserved bandwidth U_s. The bandwidth isolation property is achieved by postponing the assigned deadline each time the task is demanding more than the reserved bandwidth. The server is described by two parameters Q_s and T_s, where Q_s is the *server budget* and T_s is the *server period* ($\frac{Q_s}{T_s}$ is the server bandwidth).

When a new request arrives, the server checks whether it can be scheduled using the last assigned deadline, otherwise the request is assigned an initial deadline equal to $r_{i,j} + T_s$. Each time the job executes for Q_s time units, its scheduling deadline is postponed by T_s (in [1] a complete description of the algorithm can be found).

In this way, it is guaranteed that each active task can execute for Q_s time units each T_s, so a performance guarantee can be done. If C_i and T_i are the WCET and the MIT of task τ_i, using a CBS with parameters $(Q_s = C_i, T_s = T_i)$ a bandwidth $U_s = \frac{Q_s}{T_s} = \frac{C_i}{T_i}$ is reserved and it is guaranteed that the scheduling deadline will not be postponed (hard guarantee). Otherwise, a stochastic guarantee can be performed based on modeling the task execution time with a queue and the CBS as a deterministic time server.

3. Schedulability Analysis

To perform a statistical guarantee on soft tasks served by CBS the server is modeled as a queue, where each arriving job $J_{i,j}$ can be viewed as a request of $c_{i,j}$ time units. At any time, the request at the head of the queue is served using the current server deadline, so that it is guaranteed that Q_s units of time can be consumed within this deadline.

We analyze the following cases: a) variable computation time and constant interarrival time; and b) constant computation time and variable interarrival time.

3.1. Case a: c_i varies, T_i constant

This case treats the so called *semi-periodic* task model ([24, 2]), which can be used to model applications that manage Continuous Media (CM). For example, a video stream requires frames to be played periodically, but the decoding/playing time of each frame is not constant.

Let $U(c)$ be a generic PDF for the task's computation times and let the interarrival times be deterministic (in this case we can assign $T_s = T_i$), so that

$$V(t) = \begin{cases} 1 & if \quad t = T_i (= T_s) \\ 0 & \text{otherwise.} \end{cases}$$

Then, the system can be modeled with a queue in which :

1. a request of c_j units arrives every T_s units of time;
2. at most Q_s units can be served every T_s units of time.

Then, we can describe the system with a random process v_j defined as follows:

$$\begin{aligned} v_1 &= c_1 \\ v_j &= max\{0, v_{j-1} - Q_s\} + c_{i,j} \end{aligned}$$

where v_j indicates the length of the queue (in time units) at time jT_s, that is the units of times that are still to be server when the job $J_{i,j}$ arrives. It can be shown that the last absolute deadline assigned by the server to $J_{i,j}$ will be

$$d_{i,j} = r_{i,j} + \left\lceil \frac{v_j}{Q_s} \right\rceil T_s.$$

Since each job will finish before its last assigned deadline, the probability that a job finds a queue of length v_j when it arrives is a lower bound to the probability that the job finishes before a relative deadline

$$\delta = \left\lceil \frac{v_j}{Q_s} \right\rceil T_s.$$

If $\pi_k^{(j)} = P\{v_j = k\}$ is the state probability of process v_j and $U(h) = P\{c_j = h\}$ is the probability that an arriving job requires h units of computation time (since c_j is time invariant, $U(h)$ does not depend on j), $\pi_k^{(i)}$ can be computed by solving the equation

$$\Pi^{(j)} = M\Pi^{(j-1)} \tag{1}$$

where

$$\Pi^{(j)} = \begin{pmatrix} \pi_0^{(j)} \\ \pi_1^{(j)} \\ \pi_2^{(j)} \\ \pi_3^{(j)} \\ . \\ . \\ . \end{pmatrix}$$

and M is shown in Figure 1 (the complete proof can be found in [1]).

3.2. Case b: c_i constant, T_i varies

This case deals with a generalization of the *sporadic* task model that can be frequently encountered in CM applications. For example, this is the case of a task activated by external events, such a driver process activated by interrupts coming from a communication network.

In this case,

$$U(c) = \begin{cases} 1 & if \quad t = C_i (= Q_s) \\ 0 & \text{otherwise} \end{cases}$$

and $V(t)$ is a generic PDF. In these conditions, the CBS behaves exactly like a TBS ([23, 22]) with a bandwidth $U_s = Q_s/T_s$. In fact, if $C_i = Q_s$ each job finishes exactly when the budget arrives to 0, and the server deadline is increased of T_s at the end of each job, which is assigned a deadline $d_{i,j} = \max\{r_{i,j}, d_{i,j-1}\} + T_s$, identical to that assigned by a TBS. It is also interesting to observe that, in this

$$M = \begin{array}{l} \quad\overbrace{\hphantom{U(0) \quad U(0) \quad \cdots \quad U(0)}}^{Q+1} \\ \begin{pmatrix} U(0) & U(0) & \cdots & U(0) & 0 & 0 & \cdot & \cdot & \cdot \\ U(1) & U(1) & \cdots & U(1) & U(0) & 0 & \cdot & \cdot & \cdot \\ U(2) & U(2) & \cdots & U(2) & U(1) & U(0) & 0 & \cdot & \cdot \\ U(3) & U(3) & & U(3) & U(2) & U(1) & U(0) & 0 & \cdot \\ \cdot & \cdot & & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & & \cdot & \cdot & \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot & \cdot & \cdot & \cdot & & \end{pmatrix} \end{array}$$

Figure 1. M **for semi-periodic tasks.**

situation, the CBS is equivalent to a Rate-Based Execution (RBE) model [8] with parameters $x = 1, y = T_i, D = T_i$.

In this situation, the system can be modeled by a $G/D/1$ queue: jobs arrive in the queue with randomly distributed interarrival times and the server processes a request each T_s time units.

We can define a random process w_j as $w_j = d_{i,j} - r_{i,j} - T_s$, so we have $d_{i,j} = r_{i,j} + T_s + w_j$. In this way, it is easy to find the distribution of the relative deadlines $d_{i,j} - r_{i,j}$ within which a job $J_{i,j}$ is served. In fact,

$$w_j = d_{i,j} - r_{i,j} - T_s \Rightarrow d_{i,j} - r_{i,j} = w_j + T_s.$$

Note that, since each job will finish before the deadline assigned by the server, $P\{d_{i,j} - r_{i,j} \le \delta\}$ is a lower bound for $P\{f_{i,j} - r_{i,j} \le \delta\}$.

Since $d_{i,j} = max\{r_{i,j}, d_{i,j-1}\} + T_s$, we have

$$\begin{aligned} w_{j+1} &= d_{i,j+1} - T_s - r_{i,j+1} = \\ &= max\{r_{i,j+1}, d_{i,j}\} + T_s - T_s - r_{i,j+1} = \\ &= max\{0, d_{i,j} - r_{i,j+1}\} = \\ &= max\{0, r_{i,j} + w_j + T_s - r_{i,j+1}\} = \\ &= max\{0, w_j - a_{j+1} + T_s\} \end{aligned}$$

having defined $a_{j+1} = r_{i,j+1} - r_{i,j}$. Since a_j is a stochastic stationary and time-invariant process and w_j is a Markov process, the matrix M describing the w_j Markov chain can be found defining $\pi_k^{(j)} = P\{w_j = k\}$ and $V(h) = P\{a_j = h\}$. In [1] the matrix M, shown in Figure 2, is derived.

3.3. Stability Considerations

For a generic queue, it is known that the queue is stable (i.e., the number of elements in the queue do not diverge to infinity) if

$$\rho = \frac{mean\ interarrival\ time}{mean\ service\ time} < 1.$$

Hence, the stability can be achieved under the following conditions:

$$\begin{array}{lcll} E[C] & < & Q_s & \text{in case a)} \\ E[T] & > & T_s & \text{in case b)} \end{array}$$

where $E[C]$ is the execution time expectation and $E[T]$ is the interarrival time expectation. In general, the stability is guaranteed if

$$\frac{E[C]}{E[T]} < \frac{Q_s}{T_s}.$$

If this condition is not satisfied the difference between the deadline $d_{i,j}$ assigned by the server to a job $J_{i,j}$ and the job release time $r_{i,j}$ will increase indefinitely. This means that, for preserving the schedulability of the other tasks, τ_i will slow down in an unpredictable manner.

If a queue is stable, a stationary solution of the Markov chain describing the queue can be found; that is, there exists a solution Π such that $\Pi = \lim_{j \to \infty} \Pi^{(j)}$, so $\Pi = M\Pi$. This solution can be approximated by truncating the infinite dimension matrix M to an NxN matrix M' and solving the eigenvector problem $\Pi' = M'\Pi'$ with some numerical calculus technique.

3.4. An Example

Consider a generalized sporadic task with fixed computation time (equal to 1) and interarrival times distributed as shown in Figure 3. The classical hard real-time analysis permits to schedule this task as a $(1, 3)$ sporadic task giving a worst-case utilization factor of $1/3 = 0.33$. Using our approach, we can schedule the task with a $(1, T_s)$ CBS, with $T_s < E[T] = 7.6$. As an example, consider $T_s = 6$, giving an utilization factor of $1/6 = 0.16$.

The M matrix results as follows:

$$M = \begin{pmatrix} 0.9 & 0.9 & 0.7 & 0.3 & 0 & 0 & \cdot & \cdot \\ 0 & 0 & 0.2 & 0.4 & 0.3 & 0 & 0 & \cdot \\ 0 & 0 & 0 & 0.2 & 0.4 & 0.3 & 0 & \cdot \\ 0.1 & 0 & 0 & 0 & 0.2 & 0.4 & 0.3 & \cdot \\ 0 & 0.1 & 0 & 0 & 0 & 0.2 & 0.4 & \cdot \\ \cdot & 0 & 0.1 & 0 & 0 & 0 & 0.2 & \cdot \\ \cdot & & \cdot & \cdot & \cdot & & \cdot & \cdot \\ \cdot & & & \cdot & \cdot & \cdot & & \cdot \\ \cdot & & & & \cdot & \cdot & \cdot & \cdot \end{pmatrix}$$

Solving the system $\Pi = M\Pi$, we found the values

$$M = \begin{pmatrix}
\rho_0 & \rho_1 & \rho_2 & . & . \; . \; . \\
V(T-1) & V(T) & V(T+1) & . & . \; . \; . \\
V(T-2) & V(T-1) & V(T) & V(T+1) & . \; . \; . \\
V(T-3) & V(T-2) & V(T-1) & V(T) & . \; . \; . \\
. & . & . & . & \\
. & . & . & . & \\
. & . & . & . & \\
V(0) & V(1) & . & . & \\
0 & V(0) & . & . & \\
. & 0 & . & . & \\
. & & . & . & \\
. & & & . &
\end{pmatrix} \quad \text{with } \rho_i = \sum_{r=i+T}^{\infty} V(r)$$

Figure 2. M **for generalized sporadic tasks.**

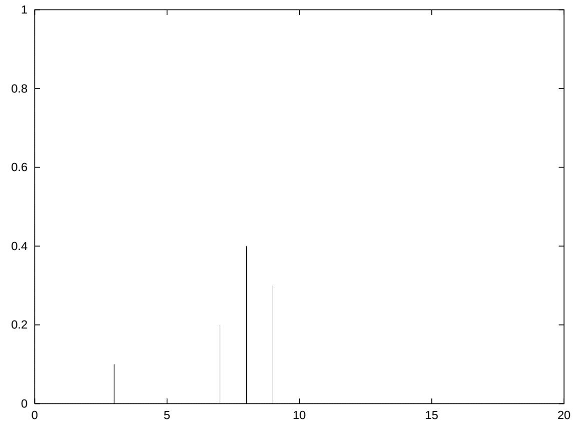

Figure 3. Interarrival time PDF $V(t)$**.**

$$\Pi = \begin{pmatrix}
0.815786 \\ 0.043039 \\ 0.023228 \\ 0.088615 \\ 0.009824 \\ 0.005252 \\ 0.009902 \\ 0.001696 \\ 0.000899 \\ 0.001138 \\ 0.000263 \\ 0.000138 \\ 0.000134 \\ 0.000038 \\ 0.000020 \\ 0.000016 \\ 0.000005 \\ 0.000003 \\ 0.000002 \\ 0.000001 \\ 0 \\ . \\ . \\ .
\end{pmatrix}$$

Figure 4. PDF of the last assigned deadline.

shown in Figure 4, that give the Cumulative Distribution Function (CDF) $X(\delta)$ shown in Figure 5.

4. Experimental Validation

To validate the correctness of our QoS guarantee, we implemented the guarantee mechanism in a tool that calculates $X_i(\delta)$ for semi-periodic tasks and generalized sporadic tasks, given $U(c)$ or $V(t)$, respectively. Then we used the scheduling simulator presented in [5] to simulate the CBS scheduler and measure the last server deadline assigned to a task and the finishing time of the task. We then compared the simulation results with the theoretical ones.

In the first experiment, we considered a semi-periodic task with computation times uniformly distributed in $[100, 400]$ and served by a CBS (Q_s, T_s) with $T_s = 1250$ and Q_s ranging from 260 to 400. Figure 6 shows the cumulative distribution function $X(\delta)$ computed by our tool (using the approximated Markov Chain solution), and the one measured using the scheduling simulator, when $Q_s = 270$. The continuous line represents the finishing time CDF obtained by simulation. As the reader can see, the results are very similar, and are a tight lower bound for the finishing time cumulative distribution function.

Table 1 shows the $X(\delta)$ values for three different Q_s values, computed using the proposed analysis and the simulator. It can be seen that the analysis results are close to the simulator's ones; it is worth noting that the analysis results are always less than or equal to the simulator's ones (they are a lower bound to the exact cumulative distribution func-

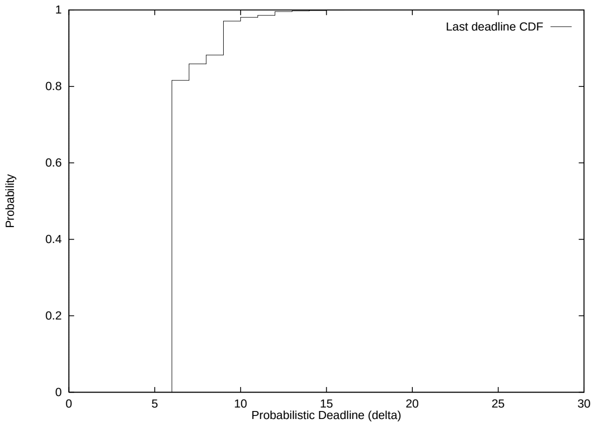

Figure 5. Probabilistic deadline CDF lower bound $X(\delta)$.

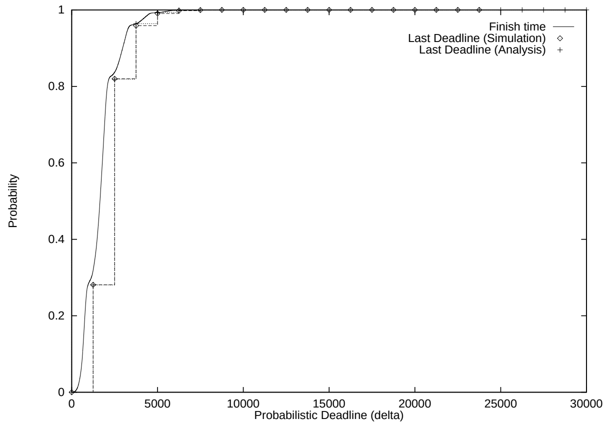

Figure 6. Simulation results vs analysis. The continuous curve represents the $X(\delta)$ CDF as a function of δ, whereas the step functions represent two lower bounds, computed by simulation and by analysis.

tion).

We also performed other similar experiments for semi-periodic and sporadic tasks and different input PDFs, verifying the consistency between the stochastic analysis and the simulation.

5. Design Issues

The presented method can be used at a design stage to assign the (Q_s, T_s) parameters to a CBS server. A correct definition of the server parameters should follow the steps outlined below:

1. Task model: the PDFs $U(c)$ and $V(t)$ are found. They can be histograms collected through experiments or closed form functions that approximate the real distributions.

2. Task constraints definition: the designer assigns the (δ, p) constraints to the tasks.

3. Statistical analysis: based on the task's parameters $(U(c), V(t))$, the tool calculates the *server characteristics* that is a set of curves showing how $X(\delta)$ changes as a function of the server parameters.

4. Parameter setting: the designer finds the (δ, p) point on the server characteristics and selects the corresponding server parameters.

As an example, consider a semi-periodic task, having the execution times PDF shown in Figure 7, and period $T_i = 100$. The server period T_s is fixed to $T_i = 100$; we must choose a server bandwidth. The server characteristics can be calculated using the presented method, and are shown in Figure 8. Choosing a probabilistic deadline δ

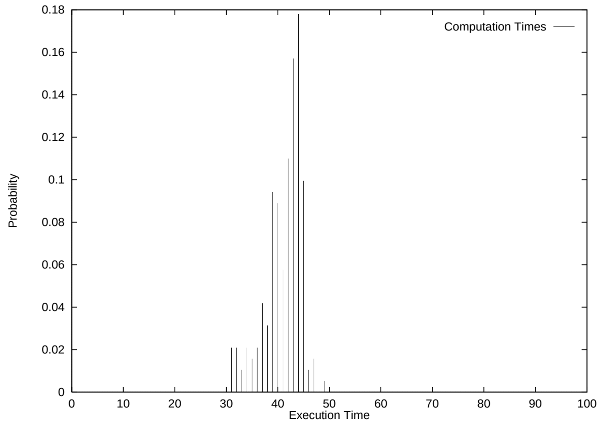

Figure 7. Execution times PDF.

and a completion probability p, the server maximum budget Q_s is univocally determined: in our example, we choose $(\delta = 100, p = 0.8)$, that results in $Q_s = 44$.

A full design methodology for distributed real-time systems can be found in [10, 11, 9]. In these papers, the authors show how to allocate resources in a distributed pipeline in order to meet some constraints like the input-output delay and the minimum output rate. The design process is based on partitioning the resources between tasks and using a probabilistic approach to check whether the constraints are respected. This step is repeated iteratively in order to find an adequate resource allocation.

δ	$Q_s = 280$		$Q_s = 320$		$Q_s = 400$	
	Simulation	Analysis	Simulation	Analysis	Simulation	Analysis
1250	0.388033	0.387972	0.677513	0.677459	1	1
2500	0.934535	0.934177	0.999879	0.999860	1	1
3750	0.994164	0.994103	1	1	1	1
5000	0.999548	0.999520	1	1	1	1
6250	0.999979	0.999979	1	1	1	1
7500	1	1	1	1	1	1
8750	1	1	1	1	1	1
10000	1	1	1	1	1	1

Table 1. $X(\delta)$ lower bounds obtained by simulation and analysis for different δ and Q_s values.

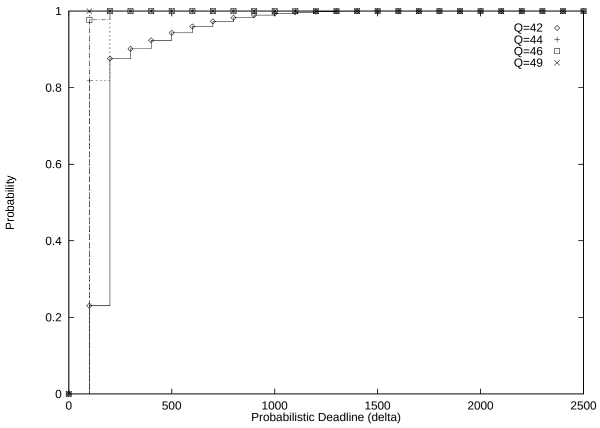

Figure 8. Server characteristics, representing the CDF $X(\delta)$ as a function of δ for different Q_s values.

6. Conclusions

In this paper, we showed how to perform QoS guarantee using *probabilistic deadlines*, when a Constant Bandwidth Server mechanism is used for soft tasks scheduling.

Our method has been formally analyzed and validated through intensive simulation, showing how it permits to achieve a desired QoS with better system utilization.

As a future work, we want to extend the proposed method to more general situations, in which both interarrival and execution times are randomly distributed.

Moreover, the proposed method has been tested, but not yet utilized in practical situations: as a future work, we plan to use it in a design phase (off-line guarantee) and check if it is usable for run time resource management (on-line QoS guarantee). In this case, we need a method for automatically deriving the interarrival and execution times PDFs at run time, using system measurements as a feedback.

7. Acknowledgements

This work has been carried out under the financial support of the Ministero dell'Università e della Ricerca Scientifica e Tecnologica (MURST) in the framework of the Project "Design Methodologies and Tools of High Performance Systems for Distributed Applications".

References

[1] L. Abeni and G. Buttazzo. Integrating multimedia applications in hard real-time systems. In *IEEE Real Time System Symposium*, Madrid, Spain, December 1998.

[2] A. K. Atlas and A. Bestavros. Statistical rate monotonic scheduling. In *IEEE Real Time System Symposium*, Madrid, Spain, December 1998.

[3] G. C. Buttazzo and M. Caccamo. Minimizing aperiodic response times in a firm real-time environment. *IEEE Transactions on Software Engineering*, 25, 1999.

[4] M. Caccamo and G. Buttazzo. Exploiting skips in periodic tasks for enhancing aperiodic responsiveness. In *IEEE Real-Time Systems Symposium*, pages 330–339, San Francisco, 1997.

[5] A. Casile, G. Buttazzo, G. Lamastra, and G. Lipari. Simulation and tracing of hybrid task sets on distributed systems. In *Real Time Computing Systems and Applications*, 1998.

[6] T. M. Ghazalie and T. Baker. Aperiodic servers in a deadline scheduling environment. *Real-Time Systems*, 9, 1995.

[7] J. Hong, X. Tan, and D. Towsley. A performance analysis of minimum laxity and earliest deadline scheduling in a real-time system. *IEEE TRANSACTIONS ON COMPUTERS*, 38(12):1736–1744, December 1989.

[8] K. Jeffay and D. Bennet. A rate-based execution abstraction for multimedia computing. In *Network and Operating System Support for Digital Audio and Video*, 1995.

[9] D.-I. Kang, R. Gerber, and L. Golubchik. Automated techniques for designing embedded signal processors on distributed platforms. Technical Report UMIACS-TR-98-57, University of Mariland, Oct 1998.

[10] D.-I. Kang, R. Gerber, and M. Sakena. Performance-based design of distributed real-time systems. In *IEEE Real-Time Technology and Applications Symposium*, pages 2–13, June 1997.

[11] D.-I. Kang, R. Gerber, and M. Sakena. Parametric design syntesis of distributed embedded systems. Technical Report UMIACS-TR-98-18, University of Mariland, Mar 1998.

[12] G. Koren and D. Shasha. Skip-over: Algorithms and complexity for overloaded systems that allow skips. In *IEEE Real Time System Symposium*, Pisa, 1995.

[13] J. Lehoczky and S. Ramos-Thuel. An optimal algorithm for scheduling soft-aperiodic tasks in fixed-priority preemptive systems. In *Proceedings of the IEEE Real-Time Systems Symposium*, December 1992.

[14] J. Lehoczky, L. Sha, and J. Strosnider. Enhanced aperiodic responsiveness in hard real-time environments. In *Proceedings of the IEEE Real-Time Systems Symposium*, December 1987.

[15] J. P. Lehoczky. Real-time queueing theory. In *IEEE Real Time System Symposium*, December 1996.

[16] J. Leung and J. W. Whitehead. On the complexity of fixed priority scheduling of periodic real-time tasks. *Performance Evaluation*, 2(4), 1982.

[17] C. L. Liu and J. Layland. Scheduling alghorithms for multiprogramming in a hard real-time environment. *Journal of the ACM*, 20(1), 1973.

[18] P. R. M. Hamdaoui. A dynamic priority assignment technique for streams with (m,k)-firm deadlines. *IEEE TRANSACTIONS ON COMPUTERS*, 44(12):1443–1451, december 1995.

[19] S. Panwar and D. Towsley. On the optimality of the STE rule for multiple server queues that serve customers with deadlines. Technical Report COINS TR 88-81, University of Massachusset, 1988.

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

[21] B. Sprunt, L. Sha, and J. P. Lehoczky. Aperiodic scheduling for hard real-time systems. *The Journal of Real-Time Systems*, 1989.

[22] M. Spuri and G. Buttazzo. Scheduling aperiodic tasks in dynamic priority systems. *Real-Time Systems*, 10(2), 1996.

[23] M. Spuri and G. C. Buttazzo. Efficient aperiodic service under the earliest deadline scheduling. In *IEEE Real-Time Systems Symposium*, December 1994.

[24] 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 *Real-Time Technology and Applications Symposium*, pages 164–173, Chicago,Illinois, January 1995.