Optimal Dimensioning of a Constant Bandwidth Server

Giorgio Buttazzo
Scuola Superiore Sant'Anna
Pisa, Italy
buttazzo@sssup.it

Enrico Bini
Scuola Superiore Sant'Anna
Pisa, Italy
e.bini@sssup.it

Abstract

The Constant Bandwidth Server (CBS) is an effective scheduling technique frequently used to handle overruns and implement resource reservation in real-time systems where tasks have variable execution requirements. The behavior of the server is tuned by two parameters: the server bandwidth, which defines the fraction of the processor allocated to the task, and the server period, which defines the time granularity of the allocation. The effect of the granularity on task executions has never been studied before, so it is typically assigned using ad-hoc considerations. This paper presents a statistical study to evaluate the effects of the server parameters on task response times, and proposes a technique to compute the best parameters that minimize the average response time of the served tasks.

1. Introduction

Resource reservation is a recent technique proposed to reduce intertask interference in concurrent real-time applications characterized by highly dynamic behavior. Multimedia activities managing audio and video streams represent typical examples of tasks with timing requirements generating a highly variable computational load. In fact, if a task manages compressed frames, the time for coding/decoding each frame can vary significantly, making the worst-case execution time much higher than the average one. Also, when data are received from a communication network, the interarrival time of data packets is usually non deterministic. In these cases, a hard real-time approach would be very inefficient, because performing a real-time guarantee based on the worst-case behavior would clearly waste computational resources and significantly degrade the system's performance. On the other hand, designing the system using average-case values would increase resource efficiency, but would unavoidably cause transient overloads that, if not properly handled, would degrade system performance in an unpredictable fashion [9, 8].

To address this problem, the idea behind resource reservation is to partition the computational resources among the tasks according to their estimated (e.g., average) requirements, and then ensure that each task does not exceed its allocated fraction. As a consequence, a time accounting mechanism is required in the kernel to monitor the actual resource consumption, and an enforcement mechanism is also needed to make sure that each task will never exceed the percentage of the resource allocated to it. A task trying to exceed this amount is delayed to protect the other tasks from suffering extra interference.

Resource reservation is important since it provides a form of *temporal protection*, in the sense that the temporal behavior of a task is not affected by the temporal behavior of the other tasks running in the system, but only by its computational requirements and by the fraction of the resource allocated to it. For example, if a task is assigned a fraction $\alpha < 1$ of the total processor bandwidth, it behaves as if it were executing *alone* on a slower processor with speed α, independently of the behavior of the other tasks. This allows guarantees to be made for each task in "isolation", only based on the fraction of processor allocated to it.

1.1. Related work

Several methods have been proposed in the literature to implement resource reservation. The simplest approach to equally partition the processor among n tasks is to use a Round Robin scheduling scheme, according to which each task behaves as it were executing on a virtual processor n times slower.

The Generalized Processor Sharing (GPS) approach [14, 15] is an ideal method that treats a resource as a fluid that can be partitioned among the tasks. Each task instanta-

Proceedings of the 27th IEEE International
Real-Time Systems Symposium (RTSS'06)
0-7695-2761-2/06 $20.00 © 2006 IEEE

neously receives at time t a fraction $f_i(t)$ of the resource, defined as the task *share*. To compute the share of a resource, each task τ_i is assigned a weight w_i, and its share is computed as

$$f_i(t) = \frac{w_i}{\sum_{\tau_j \in \Gamma(t)} w_j}$$

where $\Gamma(t)$ is the set of tasks that are active at time t. If an appropriate admission control is performed, it is possible to find an assignment of weights to tasks to guarantee real-time performance to all the time sensitive activities. In fact, based on the task rate, the maximum response time for each task can be computed as C_i/F_i, where

$$F_i = \frac{w_i}{\sum_{\tau_j \in \Gamma} w_j},$$

and Γ is the set of all tasks in the system.

Since the ideal GPS schedule cannot be realized on a real system, other practical algorithms have been proposed to approximate its behavior. Proportional Share (PS) scheduling emulates the GPS by allocating resources in discrete time quanta having maximum size Q. Clearly, quantum-based allocation introduces an allocation error with respect to the fluid flow model, measured as the *lag*. For each task, the lag is defined as the difference between the execution time actually assigned to a task by the realistic algorithm and the amount of time assigned by the ideal fluid algorithm. Hence, the objective of a fair scheduler is to limit the lag to an interval as close as possible to 0.

The first known Proportional Share scheduling algorithm is Weighted Fair Queuing (WFQ), which emulates the behavior of a GPS system by using the concept of Virtual Time. Start Fair Queuing (SFQ) [11] is a proportional share scheduler that reduces the computational complexity of WFQ and increases the fairness by using a simpler definition of virtual time. Earliest Eligible Virtual Deadline First (EEVDF) is another algorithm [17] that provides an optimal bound on the lag experienced by each task.

A simple and effective approach for implementing resource reservation is to reserve for each task τ_i a specified amount of CPU time Q_i in every interval P_i. Some authors [16] tend to distinguish between *hard* and *soft* reservations. A hard reservation guarantees the reserved amount of time to the served task, but allows such a task to execute *at most* for Q_i units of time every P_i, whereas a soft reservation guarantees that the task executes *at least* for Q_i time units every P_i, allowing it to execute more if there is some idle time available.

A resource reservation technique for fixed priority scheduling was first presented in [13]. According to this method, a task τ_i is first assigned a pair (Q_i, P_i) (denoted as a CPU *capacity reserve*) and then it is enabled to execute as a real-time task for Q_i units of time every P_i. When the task consumes its reserved quantum Q_i, it is blocked until the next period, if the reservation is hard, or it is scheduled in background as a non real-time task, if the reservation is soft. At the beginning of the next period, the task is assigned another time quantum Q_i and it is scheduled as a real-time task until the budget expires. In this way, a task is *reshaped* so that it behaves like a periodic real-time task with known parameters (Q_i, P_i) and can be properly scheduled by a classical real-time scheduler.

Under EDF, resource reservation can be efficiently implemented using the Constant Bandwidth Server (CBS) [1, 2], which will be briefly recalled in Section 2. A comparison between CBS and proportional share approaches has been done by Abeni et. al. [3].

In general, for a given bandwidth $U_i = Q_i/P_i$, the (Q_i, P_i) parameters of a reservation clearly affect the response time of the served task, however when the jobs have highly variable computation times the effect of the parameters on the worst-case response time can only be analyzed through a probabilistic approach. Recently, the discipline of probabilistic timing analysis has significantly advanced [7, 10], and different approaches have been proposed to analyze the statistical behavior of specific real-time scheduling algorithms [4, 18, 5, 12]. Today, there are tools that can provide the probability distribution function of task execution times [6]. In this paper, a probabilistic approach is used to evaluate the effects of reservation parameters on job responsiveness and provide a criterion for choosing the best CBS parameters that minimize the average response time.

1.2. Contributions and summary

The Constant Bandwidth Server (CBS) is an effective scheduling technique to implement resource reservation. The behavior of the server is tuned by two parameters: the server bandwidth, which defines the fraction of the processor allocated to the task, and the server period, which defines the time granularity of the allocation. The effect of the granularity on task executions has never been studied before, so it is typically assigned using ad-hoc considerations. In this paper, we characterize the problem and presents a statistical study to design the server as a function of the served task. In particular, the paper provides the following contributions:

Proceedings of the 27th IEEE International
Real-Time Systems Symposium (RTSS'06)
0-7695-2761-2/06 $20.00 © 2006 IEEE

- It characterizes the response time of a served task as a function of the server parameters, taking overhead into account.
- It proposes a statistical study to analyze the distribution of response times as a function of the execution requirements.
- It provides a design methodology for optimizing server parameters to minimize the average response time of the served task.

The rest of the paper is organized as follows. Section 2 briefly recalls the CBS mechanism. Section 3 characterizes the response time of a served task as a function of the server parameters. Section 4 derives the probability distribution function of the response time as a function of that of the statistical execution requirements. Section 5 shows how to select the server period for some sample distributions. Section 6 describes how to select the best server period to minimize the average response time. Finally, Section 7 states our conclusions and future work.

2. Brief summary of the CBS

2.1. Terminology and assumptions

The Constant Bandwidth Server (CBS) [1, 2] is a service mechanism that implements soft resource reservations under the EDF scheduling algorithm. To achieve temporal protection among tasks, we assume that each task is handled by a dedicated CBS S_i, with two parameters: (Q_i, P_i), where Q_i is the *server maximum budget* and P_i is the *server period*. The ratio $U_i = Q_i/P_i$ is denoted as the *server bandwidth*. While the bandwidth U_i can be easily decided based on the fraction of processor that has to be reserved to the served activity, selecting the server period P_i it is not so clear and this is the subject of the present work.

The task τ_i handled by the server can be periodic or aperiodic, with hard or soft criticality, and consists of an infinite sequence of jobs $\tau_{i,j}$ ($j = 1, 2, \ldots$), each characterized by a release time $r_{i,j}$ and a computation time $C_{i,j}$, treated as a random variable. Periodic jobs are regularly activated with period T_i, so that $r_{i,j} = (j-1)T_i$. Aperiodic jobs are activated not earlier than they finish. Notice that, since we are interested in computing the task response times and deadlines do not affect the CBS schedule, no assumption is made on tasks relative deadlines. Hence, we focus on determining the best CBS parameters that minimize the average response time of the server jobs.

2.2. CBS rules

At each instant, two state variables are maintained for each server: the server deadline d_i and the actual server budget q_i. Each job handled by a server is scheduled using the current server deadline and whenever the server executes a job, the budget q_i is decreased by the same amount. At the beginning $d_i = q_i = 0$. Since a job is not activated while the previous one is active, the CBS algorithm can be formally defined as follows:

1. When a job $\tau_{i,j}$ arrives, if $q_i \geq (d_i - r_{i,j})U_i$, it is assigned a new server deadline $d_i = r_{i,j} + P_i$ and q_i is recharged to the maximum value Q_i, otherwise the job is served with the current deadline using the current budget.

2. When $q_i = 0$, the server budget is recharged at the maximum value Q_i and the server deadline is postponed at $d_i = d_i + P_i$. Notice that there are no finite intervals of time in which the budget is equal to zero.

As we can see, the behavior of the server is tuned by two parameters: the server bandwidth, which defines the fraction of the processor allocated to the task, and the server period, which defines the time granularity of the allocation. The effect of the granularity on task executions has never been studied before, so it is typically assigned using ad-hoc considerations.

For example, if the server task is periodic, it is natural to assign a server period equal to the task period. Then, the server bandwidth is given by Q_i/P_i, where the server budget Q_i can be assigned depending on task execution requirements. If Q_i is set equal to the task worst-case execution time or higher, the processing capacity is wasted but the task response becomes very predictable and the task can be guaranteed to meet all its deadlines. If Q_i is set equal to the average execution time, then the resource utilization is optimized, but the task will experience frequent execution overruns that will delay its completion.

When the server task is aperiodic, however, it is not intuitive to set the server parameters. For a given server bandwidth, a short period allows approximating the Generalized Processor Scheduling (GPS), in which the task executes at a uniform speed proportional to the bandwidth allocated to it. In practice, however, a short period causes the server to execute the task in many small chunks, so increasing the runtime overhead. On the other hand, a large period allows allocating a large budget (remember that $Q_i = U_i P_i$) so

Proceedings of the 27th IEEE International
Real-Time Systems Symposium (RTSS'06)
0-7695-2761-2/06 $20.00 © 2006 IEEE

preventing interruptions due to budget exhaustion, but increases the response time since the task is scheduled with a longer deadline.

To evaluate the effects of the server parameters on task response times, we first characterize the response time of the served task as a function of the server parameters, and then present a statistical study to compute the best parameters that minimize the average response time of the served jobs.

3. Response time characterization

In this section we evaluate the worst-case response time R_i of a job with computation time C_i served by a CBS with bandwidth U_i, as a function of the server budget Q_i. For the sake of clarity, we first derive R_i by neglecting the overhead, and then modify the expression to take the overhead into account.

From the CBS analysis [2], we know that, if the task set is feasible, that is, if the total processor utilization is less than 1, then the served job can never miss the current server deadline. Hence, the maximum response time R_i occurs when the other tasks in the system create the maximum interference on the server. If the computation time C_i of the served job is exactly a multiple of the server budget Q_i, then the job finishes at the server deadline, that is

$$R_i = \frac{C_i}{Q_i} P_i = \frac{C_i}{U_i}. \quad (1)$$

More generally, if the computation time C_i of the job is not multiple of the budget Q_i, the last portion of the job will not finish at the server deadline, but it will finish at most Δ_i units before the deadline, as shown in Figure 1, where

$$\Delta_i = \left\lceil \frac{C_i}{Q_i} \right\rceil Q_i - C_i. \quad (2)$$

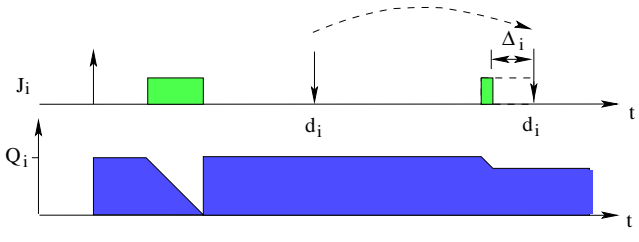

Figure 1. Worst-case finishing time of a job served by a CBS.

Hence, the response time of the job becomes

$$\begin{aligned} R_i &= \left\lceil \frac{C_i}{Q_i} \right\rceil P_i - \Delta_i \\ &= \left\lceil \frac{C_i}{Q_i} \right\rceil P_i - \left(\left\lceil \frac{C_i}{Q_i} \right\rceil Q_i - C_i \right) \\ &= C_i + \left\lceil \frac{C_i}{Q_i} \right\rceil (P_i - Q_i). \end{aligned} \quad (3)$$

Figure 2 illustrates the worst-case response time of a CBS as a function of the budget.

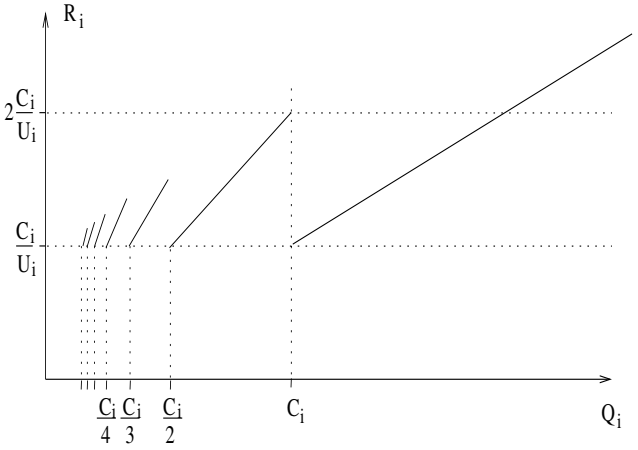

Figure 2. Worst-case response time of a CBS as a function of the budget.

From the graph shown in Figure 2 it is clear that, for a given job with constant execution time C_i, the minimum worst-case response time is C_i/U_i and can be achieved when C_i is a perfect multiple of Q_i. In practice, however, task execution time varies, inducing response time fluctuations due to the bandwidth enforcement mechanism achieved through deadline postponements. From Figure 2 it is also clear that such fluctuations would be reduced by making the budget very small compared to the average execution time, so that the server would approximate the ideal GPS. Unfortunately, a small budget (which means a short server period) causes the job to be split in many small chunks, increasing the runtime overhead. As a consequence, to properly set the server granularity P_i, the runtime overhead must be taken into account in the analysis.

3.1. Taking overheads into account

Whenever the budget is exhausted, the server deadline is postponed, so the served job can be preempted by other

Proceedings of the 27th IEEE International
Real-Time Systems Symposium (RTSS'06)
0-7695-2761-2/06 $20.00 © 2006 IEEE

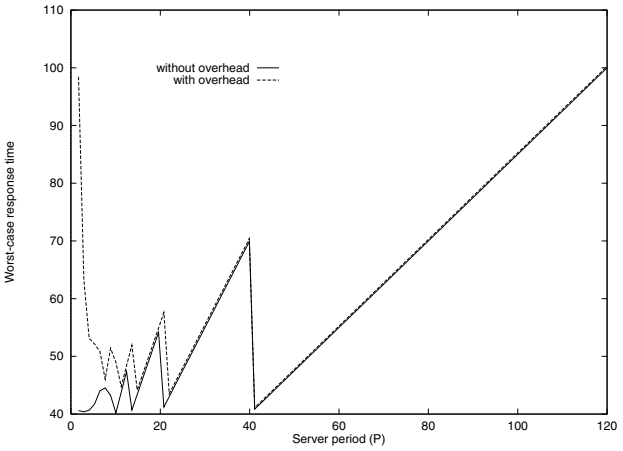

Figure 3. Worst-case response time of a CBS as a function of the period.

tasks with earliest deadline. If ϵ denotes the time needed for a context switch, then the overhead introduced by the CBS can be taken into account by subtracting such a time from the server budget. Hence, Equation (3) can be modified as follows:

$$R_i = C_i + \left\lceil \frac{C_i}{Q_i - \epsilon} \right\rceil (P_i - Q_i + \epsilon)$$
$$= C_i + \left\lceil \frac{C_i}{P_i U_i - \epsilon} \right\rceil (P_i - P_i U_i + \epsilon). \quad (4)$$

Figure 3 illustrates the worst-case response time of a CBS as a function of the period, with and without overhead. Equation (4) has been plotted for $C_i = 10$, $U_i = 0.25$, and $\epsilon = 0.2$. As is clear from the plot, the overhead prevents using small values of the period, hence it is interesting to find the value of the server period that minimizes the response time. Note that, for computing the probability distribution function of the response time, we actually need to express the response time as a function of the job execution time C_i.

Figure 4 illustrates the worst-case response time of a CBS as a function of the job execution time. As it can be noticed from Figure 4, the response time R_i can be upper bounded by the following linear function

$$R_i^{\mathsf{ub}} = P_i - Q_i + \epsilon + \frac{P_i}{Q_i - \epsilon} C_i \quad (5)$$

and lower bounded by

$$R_i^{\mathsf{lb}} = \frac{P_i}{Q_i - \epsilon} C_i. \quad (6)$$

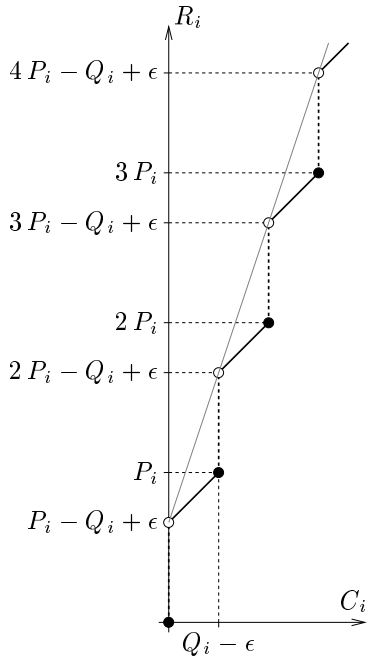

Figure 4. Worst-case response time of a CBS as a function of the job execution time.

In Section 4 we will show that this linear upper bound will be very useful for the purpose of minimizing the average response time.

4. Statistical analysis

We now consider the problem of selecting the best CBS parameters, such that the average task response time R_i is minimized. For this purpose we suppose to have the probability density function (p.d.f.) $f_C(c)$ of the task execution time, and the respective cumulative distribution function (c.d.f.) $F_C(c)$, representing the probability that the execution time is smaller than or equal to c. That is:

$$F_C(c) = \int_0^c f_C(x) \mathrm{d}x. \quad (7)$$

Since the minimization of the average R_i can in general be too complex, we first consider the problem of minimizing its linear upper bound R_i^{ub}. In this case, the average response time R_i^{avg} is computed as follows:

$$R_i^{\mathsf{avg}} = \int_0^{+\infty} \left(P_i - Q_i + \epsilon + \frac{P_i}{Q_i - \epsilon} x \right) f_C(x) \mathrm{d}x$$
$$= P_i - Q_i + \epsilon + \frac{P_i}{Q_i - \epsilon} C^{\mathsf{avg}}$$
$$= P_i (1 - U_i) + \epsilon + \frac{P_i}{P_i U_i - \epsilon} C^{\mathsf{avg}} \quad (8)$$

Proceedings of the 27th IEEE International
Real-Time Systems Symposium (RTSS'06)
0-7695-2761-2/06 $20.00 © 2006 IEEE

Hence, the period P_i which minimizes the average response time R_i^{avg} can be computed by simple functional analysis. Thus, we have:

$$\frac{\mathrm{d}R_i^{\mathsf{avg}}}{\mathrm{d}P_i} = 1 - U_i - \frac{\epsilon}{(P_i\,U_i - \epsilon)^2} C^{\mathsf{avg}} \tag{9}$$

which is equal to zero when

$$P_i^{\mathsf{opt}} = \frac{1}{U_i}\left(\epsilon + \sqrt{\frac{\epsilon\, C^{\mathsf{avg}}}{1-U_i}}\right). \tag{10}$$

Taking into account the true response time from Eq. (4), the search for the best value of P_i is more complex. In fact, we have:

$$\begin{aligned} R^{\mathsf{avg}} &= \int_0^{+\infty} \left(x + \left\lceil \frac{x}{Q_i - \epsilon} \right\rceil (P_i - Q_i + \epsilon)\right) f_C(x)\mathrm{d}x \\ &= C^{\mathsf{avg}} + (P_i - Q_i + \epsilon) \int_0^{+\infty} \left\lceil \frac{x}{Q_i - \epsilon} \right\rceil f_C(x)\mathrm{d}x \\ &= C^{\mathsf{avg}} + (P_i - Q_i + \epsilon) \sum_{k=0}^{+\infty} \int_{k(Q_i-\epsilon)}^{+\infty} f_C(x)\mathrm{d}x \\ &= C^{\mathsf{avg}} + (P_i - Q_i + \epsilon) \sum_{k=0}^{+\infty} (1 - F_C(k(Q_i - \epsilon))). \end{aligned} \tag{11}$$

In the next section we show how to find the exact value of the optimal period P_i of the CBS for some typical distributions.

5. Examples of p.d.f.

We first analyze the case in which the served task is characterized by two computation times $C_{\min}$ and $C_{\max}$ with different probabilities, and then consider the case of a computation time uniformly distributed in an interval $[C_{\min}, C_{\max}]$.

5.1. Two values

Consider the case in which the computation times of the served task can only have two values, $C_{\min}$ and $C_{\max}$. We suppose that $C_{\min}$ can occur with a probability of p and, consequently $C_{\max}$ can occur with a probability $1-p$. The cumulative distribution function is:

$$F_C(x) = \begin{cases} 0 & \text{when } x < C_{\min} \\ p & \text{when } C_{\min} \le x < C_{\max} \\ 1 & \text{when } x \ge C_{\max} \end{cases} \tag{12}$$

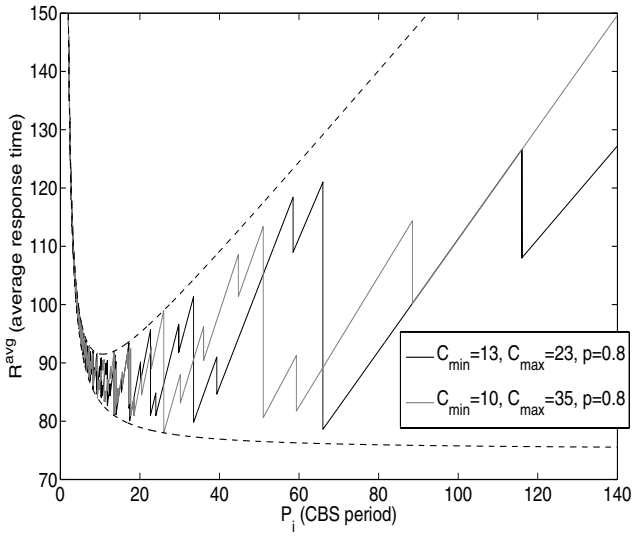

Figure 5. Average response time as a function of the period, for jobs with two computation times.

To simplify the equations, we define

$$\begin{cases} k_1 = \left\lceil \frac{C_{\min}}{Q_i - \epsilon} \right\rceil = \left\lceil \frac{C_{\min}}{P_i U_i - \epsilon} \right\rceil \\ k_2 = \left\lceil \frac{C_{\max}}{Q_i - \epsilon} \right\rceil = \left\lceil \frac{C_{\max}}{P_i U_i - \epsilon} \right\rceil \end{cases} \tag{13}$$

By taking into account the intervals where the c.d.f. is constant, Equation (11) becomes

$$\begin{aligned} R^{\mathsf{avg}} &= C^{\mathsf{avg}} + (P_i - Q_i + \epsilon)\left(\sum_{k=0}^{k_1-1} 1 + \sum_{k=k_1}^{k_2-1} (1-p)\right) \\ &= C^{\mathsf{avg}} + (P_i - Q_i + \epsilon)(k_1 + (1-p)(k_2 - k_1)) \\ &= C^{\mathsf{avg}} + (P_i(1-U_i) + \epsilon)(pk_1 + (1-p)k_2). \end{aligned} \tag{14}$$

Figure 5 illustrates the average response time as a function of the period, for jobs with two computation times. Two curves are reported, for different values of $C_{\min}$ and $C_{\max}$, having the same average value ($C^{\mathsf{avg}} = 15$). The figure illustrates also the upper bound and the lower bound obtained from equations (5) and (6), respectively.

5.2. Uniform density

We now consider the case in which the computation time is a random variable uniformly distributed between $C_{\min}$ and $C_{\max}$. Hence, the c.d.f. is given by:

$$F_C(x) = \begin{cases} 0 & \text{when } x < C_{\min} \\ \frac{x - C_{\min}}{C_{\max} - C_{\min}} & \text{when } C_{\min} \le x < C_{\max} \\ 1 & \text{when } x \ge C_{\max} \end{cases} \tag{15}$$

Proceedings of the 27th IEEE International
Real-Time Systems Symposium (RTSS'06)
0-7695-2761-2/06 $20.00 © 2006 IEEE

For this specific c.d.f., the average response time is

$$R^{\mathsf{avg}} = C^{\mathsf{avg}} + (P_i - Q_i + \epsilon) \left(\sum_{k=0}^{k_1-1} 1 + \sum_{k=k_1}^{k_2-1} (1 - \frac{k(Q_i-\epsilon) - C_{\min}}{C_{\max} - C_{\min}}) \right)$$

$$= C^{\mathsf{avg}} + (P_i - Q_i + \epsilon) \left(k_2 - \sum_{k=k_1}^{k_2-1} \frac{k(Q_i-\epsilon) - C_{\min}}{C_{\max} - C_{\min}} \right)$$

$$= C^{\mathsf{avg}} + (P_i - Q_i + \epsilon) \left(k_2 + \frac{C_{\min}}{C_{\max} - C_{\min}}(k_2 - k_1) - \sum_{k=k_1}^{k_2-1} \frac{k(Q_i-\epsilon)}{C_{\max} - C_{\min}} \right)$$

$$= C^{\mathsf{avg}} + (P_i - Q_i + \epsilon) \left(\frac{k_2 C_{\max} - k_1 C_{\min}}{C_{\max} - C_{\min}} - \frac{Q_i - \epsilon}{C_{\max} - C_{\min}} \sum_{k=k_1}^{k_2-1} k \right)$$

and finally

$$R^{\mathsf{avg}} = C^{\mathsf{avg}} + \frac{P_i - Q_i + \epsilon}{C_{\max} - C_{\min}} \left(k_2 C_{\max} - k_1 C_{\min} - (Q_i - \epsilon) \frac{k_2(k_2-1) - k_1(k_1-1)}{2} \right). \quad (16)$$

Figure 6 illustrates the average response time as a function of the period, for jobs with execution times uniformly distributed in a range, $[C_{\min}, C_{\max}]$. Four curves are reported, for different values of $C_{\min}$ and $C_{\max}$, having the same average value ($C^{\mathsf{avg}} = 15$). The figure illustrates also the upper bound and the lower bound obtained from equations (5) and (6), respectively.

6. Selecting the period

From the results achieved in the previous section, we can make the following observations:

1. The lower bound and the upper bound do not depend on the specific distributions. This aspect is very useful because, deriving the precise distribution of tasks execution times is not easy and requires extensive testing, specific tools, and a profound knowledge of the processor architecture. On the other hand, the average execution time is much easier to estimate and can easily be used to quickly derive a suboptimal solution.

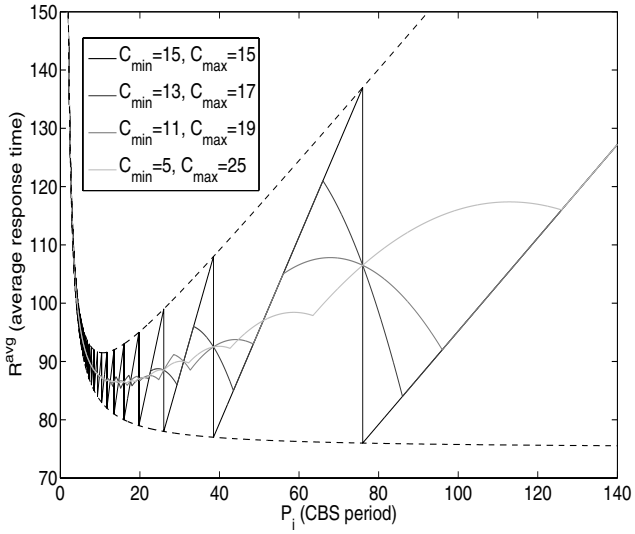

Figure 6. Average response time as a function of the period, for jobs with execution time uniformly distributed in a range.

2. The exact value of the optimal period that minimizes the average response time can be computed using numerical techniques, however small values of P_i are less sensitive to wrong estimations of the computation time distribution, hence safer. In fact, the fluctuations of the average response time increase with the server period (P_i) but decrease with the size of the range ($C_{\max} - C_{\min}$), and in general with the variance of the task computation time.

   As a consequence, for tasks with highly variable computation times, the response time is well approximated by the curve in the average between the upper bound R_i^{ub} and the lower bound R_i^{lb}, that is:

$$R_i^{\mathsf{ab}} = \frac{P_i(1-U_i) + \epsilon}{2} + \frac{P_i}{P_i U_i - \epsilon} C^{\mathsf{avg}} \quad (17)$$

   which has a minimum at

$$P_i^{\mathsf{ab}} = \frac{1}{U_i} \left(\epsilon + \sqrt{\frac{2\,\epsilon\, C^{\mathsf{avg}}}{1-U_i}} \right) \quad (18)$$

   Notice that, if this value is chosen when the variance is not high, we may incur in an appreciable error between the estimated response time and the real one. In fact, as is clear from Figure 6, a large fluctuation is experienced for narrow distributions and a small error in the computation time may lead to an unexpected increase in the response time.

Proceedings of the 27th IEEE International
Real-Time Systems Symposium (RTSS'06)
0-7695-2761-2/06 $20.00 © 2006 IEEE

In this cases, it may be safer to select a smaller period to reduce the response time fluctuations. This can be achieved by setting the CBS period as the minimum point of the upper bound curve, as given by Equation (10).

3. The amplitude of the fluctuations can be computed as the difference between the response time upper bound R_i^{ub} and the lower bound R_i^{lb} given by equations (5) and (6), respectively:

$$\Delta R_i = R_i^{\mathsf{ub}} - R_i^{\mathsf{lb}} = P_i(1 - U_i) + \epsilon \qquad (19)$$

which is linearly dependent on the value of P_i by the constant $(1 - U_i)$. This means that the action of reducing the period P_i for containing the fluctuations is more effective for servers with a small bandwidth.

7. Conclusions

In this paper we addressed the problem of dimensioning the parameters of a Constant Bandwidth Server (CBS) to minimize the average response time of the served jobs. To achieve temporal protection among the tasks in the system, we assumed that each task is handled by a dedicated CBS, whose parameters have to be tailored to the characteristics of the served task. We considered tasks with variable computation times and known distribution function, which can be estimated by code analysis or through an experimental evaluation.

In the ideal case, we observed that a CBS with a given bandwidth U_i could approximate the behavior of the fluid GPS model by using a very small time granularity (that is, a small value for the server period P_i) and then set the budget at $Q_i = P_i U_i$. However, a small period causes the job to be split in many small chunks, increasing the runtime overhead. As a consequence, to properly set the server granularity P_i, the runtime overhead must be taken into account in the analysis.

The results achieved by the statistical analysis, including the overhead, show that

- For tasks with highly variable computation times, the response time is well approximated by the curve in the average between the upper bound R_i^{ub} and the lower bound R_i^{lb}, thus the period can be set at the value that minimizes this curve, that is

$$P_i = \frac{1}{U_i}\left(\epsilon + \sqrt{\frac{2\,\epsilon\,C^{\mathsf{avg}}}{1 - U_i}}\right).$$

- When the variance is not high, to prevent large fluctuations due to estimation errors, it may be safer to select the period corresponding to the minimum point of the upper bound curve, as given by Equation (10), that is

$$P_i = \frac{1}{U_i}\left(\epsilon + \sqrt{\frac{\epsilon\,C^{\mathsf{avg}}}{1 - U_i}}\right).$$

- Whenever the precise distribution of computation times is not known and only the average value is available, the best CBS period is the one that minimizes the upper bound of the response time, thus it is also given by Equation (10).

As a future work, we plan to address the analytical computation to derive the optimal CBS period and to extend the probabilistic analysis to the case of generic jobs activations, thus relaxing the assumption of having the interarrival time greater or equal to the response time of the previous job.

References

[1] L. Abeni and G. Buttazzo. Integrating multimedia applications in hard real-time systems. In *Proceedings of the* 19*th IEEE Real-Time Systems Symposium*, pages 4–13, Madrid, Spain, Dec. 1998.

[2] L. Abeni and G. Buttazzo. Resource reservation in dynamic real-time systems. *Real-Time Systems*, 27(2):123–167, July 2004.

[3] L. Abeni, G. Lipari, and G. Buttazzo. Constant bandwidth vs. proportional share resource allocation. In *Proceedings of the IEEE International Conference on Mutimedia Computing and Systems*, volume 2, pages 107–111, Firenze, Italy, June 1999.

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

[5] G. Bernat, A. Colin, and S. M. Petters. WCET analysis of probabilistic hard real-time systems. In *RTSS*, Austin (TX), U.S.A., Dec. 2002.

[6] G. Bernat, A. Colin, and S. M. Petters. pWCET: A tool for probabilistic worst-case execution time analysis of real-time systems. Ycs-2003-353, Department of Computer Science, University of York, Feb. 2003.

[7] A. Burns, G. Bernat, and I. Broster. A probabilistic framework for schedulability analysis. In *Proceedings of the* 3*rd ACM International Conference on Embedded Software*, pages 1–15, Philadelphia (PA), U.S.A., Oct. 2003.

[8] G. C. Buttazzo, G. Lipari, L. Abeni, and M. Caccamo. *Soft Real-Time Systems: Predictability vs. Efficiency*. Springer, 2005.

[9] M. Caccamo, G. Buttazzo, and L. Sha. Handling execution overruns in hard real-time control systems. *IEEE Transactions on Computers*, 51(7):835–849, July 2002.

[10] A. Ermedahl, F. Stappert, and J. Engblom. Clustered calculation of worst-case execution times. In *Proceedings of the* 6^{th} *International Conference on Compilers, Architectures, and Synthesis for Embedded Systems*, San José (CA), U.S.A., Oct. 2003.

[11] P. Goyal, X. Guo, and H. M. Vin. A hierarchical cpu scheduler for multimedia operating systems. In 2^{nd} *OSDI Symposium*, 1996.

[12] K. Kim, J. L. Diaz, J. M. Lopez, L. L. Bello, C.-G. Lee, and S. L. Min. An exact stochastic analysis of priority-driven periodic real-time systems and its approximations. *IEEE Transactions on Computer*, 54(11):1460–1466, 2005.

[13] C. W. Mercer, S. Savage, and H. Tokuda. Processor capacity reserves: Operating system support for multimedia applications. In *Proceedings of IEEE International Conference on Multimedia Computing and Systems*, pages 90–99, Boston (MA), U.S.A., May 1994.

[14] A. K. Parekh and R. G. Gallager. A generalized processor sharing approach to flow control in integrated services networks: the single-node case. *IEEE/ACM Transactions on Networking*, 1(3):344–357, June 1993.

[15] A. K. Parekh and R. G. Gallager. A generalized processor sharing approach to flow control in intergrated services networks: the multiple node case. *IEEE/ACM Transanctions on Networking*, 2(2):137–150, Apr. 1994.

[16] 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*, volume 3310, San José (CA), U.S.A., Jan. 1998.

[17] 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 *Proceeding of the* 17^{th} *IEEE Real Time System Symposium*, pages 288–299, Washington (DC), U.S.A., Dec. 1996.

[18] 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 (IL), U.S.A., Jan. 1995.

Proceedings of the 27th IEEE International
Real-Time Systems Symposium (RTSS'06)
0-7695-2761-2/06 $20.00 © 2006 IEEE