Proceedings of the 26th Euromicro Conference on Real-Time Systems (ECRTS 2014), Madrid, Spain, July 8-11, 2014.

Optimal Design for Reservation Servers under Shared Resources

Alessandro Biondi*, Alessandra Melani*, Marko Bertogna†, Giorgio C. Buttazzo*
*Scuola Superiore Sant'Anna, Pisa, Italy
†University of Modena and Reggio Emilia, Modena, Italy
Email: {alessandro.biondi, alessandra.melani, g.buttazzo}@sssup.it, m.bertogna@unimore.it

Abstract

Modularity and hierarchical-based design are crucial features that need to be supported in complex embedded systems characterized by multiple applications with timing requirements. Resource reservation is a powerful scheduling mechanism for achieving such goals and providing temporal isolation among different real-time applications. When different applications share mutually exclusive resources, a precise feasibility analysis can still be performed in isolation, using specific resource access protocols, taking into account only the application features and the reservation parameters. This paper presents a methodology for selecting the parameters of each reservation in order to guarantee the feasibility of the served applications and minimize the required bandwidth.

1. Introduction

Resource reservation is a general scheduling mechanism proposed to partition a computational resource among multiple applications to avoiding reciprocal interference. If an application A is assigned a fraction α (also referred to as reservation bandwidth) of the entire processor, it behaves as it were executing alone on a slower processor (with speed α times the original speed) entirely dedicated to it, independently of the execution behavior of the other applications. To achieve this goal, the kernel must not only guarantee that each application receives the required amount of bandwidth, but it must also prevent that it may consume more than its allocated fraction, to protect the other applications running in the system (temporal protection). The advantage of this method is that each task can be guaranteed in isolation, independently of the behavior of the other tasks. Resource reservation servers have been proposed both under fixed-priority [1] and deadline-based scheduling [2], [3].

The basic idea behind a reservation server is to provide a periodic service, executing the application for Q units of time every P units, where Q is denoted as the *server budget*, P as the *server period*, and $\alpha = Q/P$ as the *server bandwidth*. A reservation server is said to be *hard* if, when the budget is exhausted, the application is blocked until the beginning of the next period, at which the budget is replenished. A reservation server is said to be *soft* if, when the budget is exhausted, the application remains active and can be scheduled at a lower priority. Reservations have also been used in the context of hierarchical scheduling, to design real-time systems as a set of modular components, each handling its own application, which can be scheduled by a desired scheduling algorithm [4], [5].

When different applications share mutually exclusive resources, temporal isolation can be broken and proper resource access protocols need to be used to prevent that a budget exhaustion inside a critical section introduces extra interference on other applications.

To address this problem, different approaches have been presented in the literature. Abeni and Buttazzo proposed a simple solution under the Constant Bandwidth Server (CBS) [3], known as *budget overrun*: when the server budget is exhausted inside a resource, the server is allowed to consume extra budget until the critical section is completed. Davis and Burns [6] analyzed this approach under fixed priorities and proposed two versions of this mechanism:

- *Overrun with payback*, where the server pays back in the next execution instant, in that the next budget replenishment is decreased by the overrun value;
- *Overrun without payback*, where no further action is taken after the overrun.

Note that the budget overrun technique does not increase the response time of the served tasks, but implies a greater bandwidth requirement for the reservation. To avoid breaking the isolation property, such an extra bandwidth need to be subtracted from the server taking into account the largest possible overrun.

Another protocol, known as SIRAP, was proposed by Behnam et al. [7] and consists of introducing a budget check before granting the access to a resource: if the budget is sufficient to complete the critical section, the task can access the resource, otherwise the access is postponed until the next budget replenishment. This approach does not affect the execution of tasks in other reservations, but penalizes the response time of the served tasks.

A third protocol, named BROE (Bounded-Delay Resource Open Environment), was proposed by Bertogna et al. [8]. According to this method, when a task wants to enter a critical section and the budget is not sufficient for its completion, a full budget replenishment is planned at the earliest possible time that preserves both the server bandwidth and the maximum service delay. The server is blocked until the budget replenishment.

A simple test for verifying the feasibility of a real-time application scheduled by Earliest Deadline First (EDF) under a BROE reservation server, was first presented by Bertogna et al. [8] using an upper bound of the server supply function. Later, Biondi et al. [9] derived the exact supply function for BROE

and proposed a more precise schedulability test for EDF, using which BROE has been shown to outperform SIRAP under many configuration parameters.

Although feasibility tests are essential to guarantee the timing behavior of real-time applications handled inside reservations, no method is available today for determining the server parameters (Q,P) as a function of the application characteristics, exploiting the exact supply function derived for BROE. This paper therefore presents a methodology for designing the optimal parameters of a BROE server, showing how to use the improved supply function at server design time with *pseudo-polynomial complexity*. A complete design algorithm has been developed according to the proposed methodology, maintaining a *modular approach* that analyzes each application *in isolation*, without considering the specific parameters of the other components.

Paper structure. The remainder of this paper is organized as follows. Section 2 presents the system model, the global framework and the assumptions used throughout the paper. Section 3 briefly recalls the local schedulability test for real-time task sets running within a BROE reservation server, using the improved supply bound function presented in [9]. Section 4 analyzes the design problem, deriving the exact feasibility region of a BROE server for a given application. Section 5 shows how to explore the feasibility region to find the point that minimizes the required server bandwidth. Section 6 discusses the complexity of the proposed approach, proving that it is still pseudo-polynomial in the number of tasks. Section 7 presents the related works, explaining how the presented work improves over previously proposed suboptimal approaches. Finally, Section 8 states our conclusions and future work.

2. System model and framework

This paper considers a uniprocessor hierarchical system $\mathcal{S}$ consisting of a number N of subsystems $S_k \in \mathcal{S}$, $k = 1, \ldots, N$, each implemented by a BROE [8] reservation server (also denoted as S_k), characterized by a budget Q_k and a period P_k. The rules of a BROE server are summarized in Appendix A. For the sake of simplicity, we consider a two-level hierarchical system, although our contributions can be extended to a generic n-level hierarchical system, using the compositional real-time scheduling framework proposed by Shin and Lee [5]. The *global scheduler* is based on a Hard Constant Bandwidth Server [3], [10], a bandwidth preserving mechanism that determines which subsystem can access the CPU at any given time. Each subsystem uses a *local scheduler* to select the running task on the subsystem. In this paper, EDF is considered as local scheduling policy for each subsystem.

2.1. Task model

Each subsystem S_k runs an application Γ_k consisting of n_k periodic or sporadic preemptive tasks. Each task τ_i is characterized by a worst-case execution time (WCET) C_i, a period (or minimum interarrival time) T_i, and a relative deadline $D_i \leq T_i$. Within each subsystem, tasks are indexed by increasing relative deadlines.

2.2. Resource model

Two types of resources can be defined:

- *Local resource*: a resource shared among tasks within the same subsystem;
- *Global resource*: a resource shared among tasks belonging to different subsystems.

In the following, $Z_{i,j}$ denotes the longest critical section of τ_i related to resource R_j and $\delta_{i,j}$ denotes the WCET of $Z_{i,j}$.

Definition 1: The *Resource Holding Time* $H_{k,j}(i)$ of a global resource R_j accessed by a task $\tau_i \in \Gamma_k$ is the maximum amount of budget consumed by S_k between the lock and the corresponding release of R_j performed by τ_i.

Note that, if global resources are accessed by disabling local preemption, $H_{k,j}(i)$ is equal to $\delta_{i,j}$ of task $\tau_i \in \Gamma_k$. If local preemption is not disabled, $H_{k,j}(i)$ takes into account the worst-case local interference experienced by τ_i during the lock of R_j. Details on how to compute $H_{k,j}(i)$ are reported in Appendix C.

In addition, the maximum Resource Holding Time on a resource R_j for an application Γ_k is defined as

$$H_{k,j} = \max_{\tau_i \in \Gamma_k} \{H_{k,j}(i)\}. \quad (1)$$

Finally, the maximum Resource Holding Time for an application Γ_k is defined as

$$H_k = \max_j \{H_{k,j}\}. \quad (2)$$

To access shared resources in such a hierarchical framework, the *Stack Resource Policy* (SRP) can be used as it is for local resources, while it has to be extended [8], [11] for global resources. In the following, the global version of SRP presented in [8], [11] is adopted, referred to as SRP-G.

We use the notation $\{x\}_0$ to denote $\{0\} \cup \{x\}$.

2.3. Framework

Each subsystem S_k, running an application Γ_k, is implemented through a reservation server, and is characterized by an *interface* consisting of three parameters: a budget Q_k, a period P_k and a maximum resource holding time H_k.

We assume the existence of a global integration level which is in charge of admitting or rejecting each S_k in the system. Each subsystem under analysis can be integrated according to a global schedulability test which only requires the knowledge of its interface. Hence, the parameters of the interface constitute the only information exported to the integration level, while the specific parameters of the tasks running inside the application are completely hidden to the other subsystems and not visible at the integration level.

Since the applications may share a certain number of global resources, we assume that the integration level specifies a System Maximum Resource Holding Time $\mathcal{H}$. This poses a constraint on the resource holding time H_k of each application that shares global resources and wants to be admitted in the system. As a result, the maximum Resource Holding Time of each application will never exceed $\mathcal{H}$. In formulas:

$$\forall k = 1, \ldots, N \;\; H_k \leq \mathcal{H}. \quad (3)$$

Moreover, for the BROE server to work correctly, the maximum resource holding time of any server must be smaller than the corresponding server budget:

$$\forall k = 1, \ldots, N \;\; H_k \leq Q_k. \tag{4}$$

2.4. Global integration

The admission control at the global level is performed through a schedulability check based on the interface parameters specified by each application in the system.

According to SRP-G, a server S_k can be blocked for a time B_k^G by a server S_ℓ with period $P_k < P_\ell$. This happens when S_ℓ locks a resource R, which is used by S_ℓ and by a server S_h with period $P_h \leq P_k$. Formally, the global blocking factor B_k^G can be expressed as follows:

$$B_k^G = \max_{P_\ell > P_k} \{H_{\ell,j} \mid R_j \text{ used by } S_h \wedge P_h \leq P_k\}_0. \tag{5}$$

The following Theorem has been proved in [8].

Theorem 1: A set of BROE servers $S_1, \ldots, S_N$ may be composed upon a unit-capacity processor without missing any deadline if

$$\forall k = 1, \ldots, N \quad \sum_{i:P_i \leq P_k} \alpha_i + \frac{B_k^G}{P_k} \leq 1. \tag{6}$$

Focusing on the definition of B_k^G expressed by Equation (5), we observe that B_k^G depends on the resources accessed by the other servers, the relative resource holding times of each server for each resource, and the relative order of server periods with respect to P_k. The modularity of the design approach does not allow the server interfaces to contain information on which resource is accessed by each server, and for how long. The only information that is "propagated" to the other subsystems is the maximum resource holding time H_k. Then, assuming all servers access all resources, Equation (5) becomes

$$B_k^G = \max_{P_\ell > P_k} \{H_\ell\}. \tag{7}$$

3. Local schedulability analysis for BROE

For the sake of simplicity, from now on, the index k associated to the parameters of a server S_k may be removed in those formulas that refer to a single generic server.

The local schedulability analysis of a reservation server can be performed using the test proposed by Shin and Lee [5], later extended by Baruah [12] to account for shared resources. According to this test, a task set Γ is schedulable by EDF on a reservation server S if

$$\forall t > 0 \quad B^L(t) + \mathsf{dbf}(\Gamma, t) \leq \mathsf{sbf}(S, t) \tag{8}$$

where

$$\mathsf{dbf}(\Gamma, t) \stackrel{\text{def}}{=} \sum_{i=0}^{n} \left\lfloor \frac{t - D_i + T_i}{T_i} \right\rfloor C_i$$

is the *demand bound function* of the task set Γ (i.e., the maximum computational demand of Γ in any interval of length $t > 0$), $\mathsf{sbf}(S, t)$ is the *supply bound function* of the server S (i.e., the minimum amount of service time provided by the server in any interval of length $t > 0$), and $B^L(t)$ is the maximum blocking time in the interval $(0, t]$ due to local resources, computed as the maximum critical section of tasks having deadline $> t$, accessing local resources common to at least one task with deadline $\leq t$, that is,

$$B^L(t) = \max\{\delta_{i,j} \mid D_i > t \;\wedge \; \exists \tau_\ell \text{ accessing } R_j \text{ with } D_\ell \leq t\}. \tag{9}$$

In the original analysis of the BROE protocol proposed by Bertogna et al. [8], the supply bound function in (8) is a linear lower bound $\mathsf{sbf}^L(S, t)$ of a bounded-delay partition (α, Δ), where α is the server bandwidth and Δ is the maximum service delay:

$$\mathsf{sbf}^L(S, t) = \alpha(t - \Delta), \tag{10}$$

with $\alpha = Q/P$ and $\Delta = 2(P - Q)$. The supply bound function adopted in the original BROE analysis does not include the worst-case resource holding time H of the server, because the linear lower bound $\mathsf{sbf}^L(S, t)$ already accounts for the worst-case delay introduced by the global resource access policy [8]. However, exploiting the information on H it is possible to improve the supply bound function, allowing a tighter schedulability analysis, as shown by Biondi et al. in [9].

In this work, the exact supply bound function of BROE presented in [9] is adopted, which can be formally expressed as follows:

$$\mathsf{sbf}^B(S, t) = \begin{cases} t - \Delta - (k-1)(P-Q) & t_A < t \leq t_B \\ kQ - kH & t_B < t \leq t_C \\ \alpha(t - \Delta) & t_C < t \leq t_D \end{cases} \tag{11}$$

where

$$k = \left\lceil \frac{t - \Delta}{P} \right\rceil \tag{12}$$

and t_A, t_B, t_C, and t_D are reported in Table 2. The function is equal to 0 in the interval $[0, \Delta]$. More details on the derivation of Equation (11) can be found in Appendix B.

t_A	$\Delta + (k-1)P$
t_B	$\Delta + (k-1)P + (Q - kH)$
t_C	$\Delta + kP - kH/\alpha$
t_D	$\Delta + kP$

Table 1: Values for sbf^B.

Figure 1 shows a graphical comparison between the periodic supply function, its linear bound and the exact supply bound function derived in [9]. As clear from the figure, the new supply bound function introduces some additional areas over the linear bound $\mathsf{sbf}^L(t)$. Such areas allow the test in Equation (8) to derive a tight schedulability condition, maintaining the same pseudo-polynomial computational complexity of the original test based on the linear $\mathsf{sbf}^L(t)$. The computation of the improved supply bound function for a given time t requires indeed just an additional modulo operation to identify the k^{th} period using Equation (12), which is done in constant time.

Note that after $\lceil Q/H \rceil$ periods, the supply bound function collapses to the linear $\mathsf{sbf}^L(t)$. Therefore, the smaller H with respect to Q, the larger the improvement with respect to $\mathsf{sbf}^L(t)$. In the extreme case in which no global resource is shared ($H = 0$), the $\mathsf{sbf}^B(t)$ coincides with the supply bound function

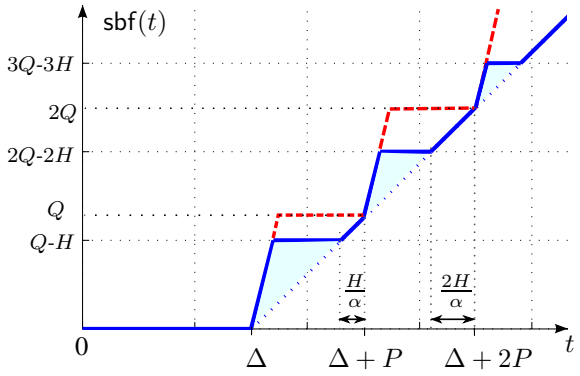

Figure 1: Supply bound functions: periodic (dashed line), linear $\alpha - \Delta$ (dotted line), and improved sbf^B proposed for BROE (continuous line).

$\mathsf{sbf}^P(t)$ of a periodic server. Conversely, when $H = Q$, $\mathsf{sbf}^B(t)$ is always equal to $\mathsf{sbf}^L(t)$.

Different methods have been proposed in the literature to reduce resource holding times by limiting (or disabling) local preemption when accessing global resources [13]. We will show that in almost all cases of interest it is possible to have a minimal resource holding time equal to the critical section length, so magnifying the improvement allowed by the supply bound function presented in this section.

In the following, we will also show that the improved supply bound function can be also used to optimally solve the problem of selecting the server parameters for a given application, with *pseudo-polynomial complexity*. The modularity of BROE's original approach is preserved, as the local schedulability of each application is still *validated in isolation*, without requiring the knowledge of the parameters of the other applications.

4. Optimal design for the BROE server

Given a real-time application Γ to be scheduled within a reservation, this section describes how to derive the BROE server parameters P and Q that minimize the bandwidth required to guarantee the feasibility of Γ under EDF. Here, the bandwidth to minimize includes the runtime overhead σ required to perform a context switch. Hence, for a server that allocates a budget Q every period P, the effective bandwidth required by the reservation is considered to be:

$$\alpha' = \frac{Q+\sigma}{P} = \alpha + \frac{\sigma}{P}. \tag{13}$$

One may wonder why not including the resource holding time H in the objective function to minimize, as it also affects the system schedulability through the blocking term of Equation (6). The reason for ignoring H descends from a very nice property of the BROE server, which allows significantly decreasing the resource holding time of each task by executing each global critical section non-preemptively, without any schedulability loss. In particular, we will hereafter show that any application Γ that can be feasibly scheduled with EDF+SRP on a BROE server with $\alpha \leq 1/2$ can also be feasibly scheduled with EDF on the same BROE server using a simple non-preemptive locking protocol. In this way, the resource holding time of any task $\tau_i \in \Gamma$ is equal to the corresponding critical section length $\delta_{i,j}$, so that the maximum resource holding time of the application Γ is

$$H = \max_{i,j} \delta_{i,j}. \tag{14}$$

Besides the significant reduction in the resource holding time, the above formula shows that H is no more dependent on the other server parameters P and Q. As shown in Appendix C, this is not the case when using preemptive locking protocols, like SRP, for which the computation of H is influenced by P and Q through $\Delta/2 = P - Q$. Moreover, it is worth noting that the value given by Equation (14) cannot be further reduced by artificially increasing the local resource ceiling as proposed in [13], because the value of H is already the minimum possible one.

Therefore, there is no reason for including H in the objective function, as in most cases of interest this parameter is fixed, independent from P and Q, and already minimized, so that no further design choice can be taken to further reduce the impact of H on the global system schedulability.

Theorem 2: [Theorem 7 in [8]] Given an application Γ accessing a globally shared resource R_j that can be feasibly scheduled with EDF+SRP upon a BROE server with parameters (P, Q), if $\forall \tau_i \in \Gamma \ \delta_{i,j} \leq \frac{\Delta}{2}$, then Γ can be EDF-scheduled on the same BROE server executing each critical section of R_j *with local preemptions disabled*.

Using the above result, the following theorem can be proved.

Theorem 3: If an application Γ can be feasibly scheduled with EDF+SRP upon a BROE server with $\alpha \leq 1/2$, then Γ can be EDF-scheduled on the same BROE server executing each critical section *with local preemptions disabled*.

Proof: By definition, $\forall i, j \ \ \delta_{i,j} \leq H$. Then, by Equation (4),

$$\forall i, j \ \ \delta_{i,j} \leq Q.$$

Assuming $\alpha \leq 1/2$,

$$\Delta/2 = P - Q = Q\left(\frac{1}{\alpha} - 1\right) \geq Q.$$

Merging both inequalities, the precondition of Theorem 2 holds:

$$\forall i, j \ \ \delta_{i,j} \leq \Delta/2,$$

thus the theorem follows. □

The significance of the above theorem is manyfold: (i) it minimizes H improving global schedulability; (ii) it allows using a very simple non-preemptive locking protocol, reducing the scheduling overhead; (iii) it minimizes the number of preemptions experienced by an application while holding a lock, improving the predictability of such critical chunks of code; and (iv) it simplifies the computation of the resource holding times, without the need for iterative methods.

Moreover, the precondition $\alpha \leq 1/2$ is applicable to most cases of practical interest. Clearly, there could be at most one server in the system that does not meet this condition, as the sum of all server bandwidths cannot exceed the available processing power. For this reason, the design space exploration presented in the next sections will be limited to BROE servers with $\alpha \leq 1/2$. If a server is needed with $\alpha > 1/2$, it can be easily treated separately. Note that for such a server, Theorem 2 can still be used to execute

non-preemptive critical sections smaller than $\Delta/2$.

4.1. BROE design space

The objective of the proposed design method is to find, for each application Γ, the optimal server parameters (P_{opt}, Q_{opt}) that minimize the bandwidth α', under the following schedulability constraint:

$$\forall t \in dSet \quad B^L(t) + \mathsf{dbf}(\Gamma, t) \leq \mathsf{sbf}(S, t), \tag{15}$$

where $dSet$ is the set of all time instants in which the test has to be performed.

Exploiting the stepwise definition of the dbf, the cardinality of $dSet$ can be bounded by applying techniques from [14], considering only values of t satisfying $t \equiv (D_i + \beta T_i)$, for all $\tau_i \in \Gamma$, and some integer $\beta \geq 0$. By Equation (9), it is easy to see that the blocking $B^L(t)$ is null for $t \geq D_{max} \stackrel{\text{def}}{=} \max_{i=1}^{n}\{D_i\}$. Condition (15) then becomes

$$\mathsf{dbf}(\Gamma, t) \leq \mathsf{sbf}(S, t)$$

Let $U = \sum_{i=0}^{n}(C_i/T_i)$ and $U_i = (C_i/T_i)$. By removing the floor, $\mathsf{dbf}(\Gamma_k, t)$ can be upper bounded as follows:

$$\mathsf{dbf}(\Gamma, t) \leq \sum_{i=0}^{n}\left(\frac{t - D_i + T_i}{T_i}\right)C_i = Ut + \sum_{i=0}^{n}(T_i - D_i)U_i$$

which is a line with slope U. Moreover, $\mathsf{sbf}(S, t)$ can be lower bounded by $\mathsf{sbf}^L(S, t)$, which is a line with slope α. If $U < \alpha$, the two lines intersect at

$$t^* = \frac{\alpha\Delta + \sum_{i=0}^{n}(T_i - D_i)U_i}{\alpha - U}. \tag{16}$$

After t^*, $\mathsf{dbf}(\Gamma, t)$ will then always be lower than or equal to $\mathsf{sbf}(S, t)$, so that Condition (15) is satisfied. In order to check the feasibility of application Γ on a BROE server, it is then sufficient to check Condition (15) for all points $t_i \in dSet$ until $\max\{D_{max}, t^*\}$. This bound is pseudo-polynomial if the application utilization is a priori bounded from above by a constant less than α. Note that $U \leq \alpha$ is a necessary condition for feasibility. Alternatively, Condition (15) can be checked until the least common multiple of all tasks periods $T^{HP} \stackrel{\text{def}}{=} lcm\{T_1, T_2, \ldots, T_n\}$.

The computational demand of the task set Γ associated to the application Γ is expressed as a set of pairs (w_i, t_i), where the values t_i are all the test points in $dSet$, and $w_i = B^L(t_i) + \mathsf{dbf}(\Gamma, t_i)$. Equation (15) is then also equivalent to

$$\forall t_i \in dSet \quad w_i \leq \mathsf{sbf}(S, t_i). \tag{17}$$

In order to compute the optimal pair (P_{opt}, Q_{opt}), we proceed by defining the feasibility region for each demand point (w_i, t_i) as

$$\mathbb{D}(w_i, t_i) = \{(P, Q) \mid w_i \leq \mathsf{sbf}(S, t_i)\}. \tag{18}$$

Each $\mathbb{D}(w_i, t_i)$ represents the region in the P-Q plane of the feasible server parameters that guarantee the schedulability of the task set for the corresponding pair (w_i, t_i), according to Equation (17).

Since the EDF schedulability test (17) must be satisfied for each point in $dSet$, the exact feasibility region of the optimization problem is described as the intersection in the P-Q plane of all the regions found at each step i, that is,

$$\mathbb{D} = \bigcap_{d_i \in dSet} \mathbb{D}(w_i, t_i). \tag{19}$$

Hence, $\mathbb{D}$ represents all the possible pairs (P, Q) satisfying the local schedulability test given in Equation (17).

While Equation (18) is valid regardless of the specific supply function adopted, when using the exact BROE supply function expressed by Equation (11), the analytical expression of $\mathbb{D}(w, t)$ can be found by inverting Equation (17), assuming $\mathsf{sbf}(S, t) = \mathsf{sbf}^B(S, t)$.

Due to the piecewise definition of $\mathsf{sbf}^B(S, t)$, it is necessary to consider three separate cases (neglecting the trivial case for $t \leq \Delta$, where $\mathsf{sbf}^B(S, t) = 0$), determining three different feasibility regions:

- Region 1: $t_A < t \leq t_B$,

$$t - \Delta - (k-1)(P - Q) \geq w \;\Leftrightarrow\; Q \geq \frac{w - t}{k+1} + P \tag{20}$$

with

$$t_A < t \quad \Leftrightarrow \quad P < \frac{t + 2Q}{k+1} \tag{21}$$

$$t \leq t_B \quad \Leftrightarrow \quad P \geq \frac{t + Q + kH}{k+1}. \tag{22}$$

- Region 2: $t_B < t \leq t_C$:

$$kQ - kH \geq w \quad \Leftrightarrow \quad Q \geq \frac{w}{k} + H \tag{23}$$

with

$$t_B < t \quad \Leftrightarrow \quad P < \frac{t + Q + kH}{k+1} \tag{24}$$

$$t \leq t_C \quad \Leftrightarrow \quad P \geq \frac{tQ + 2Q^2}{2Q + kQ - kH}. \tag{25}$$

- Region 3: $t_C < t \leq t_D$:

$$\alpha(t - \Delta) \geq w \;\Leftrightarrow\; 2Q^2 + Q(t - 2P) - Pw \geq 0 \tag{26}$$

with

$$t_C < t \quad \Leftrightarrow \quad P < \frac{tQ + 2Q^2}{2Q + kQ - kH} \tag{27}$$

$$t \leq t_D \quad \Leftrightarrow \quad P \geq \frac{t + 2Q}{2 + k}. \tag{28}$$

Please note that k, defined according to Equation (12), depends on P, Q and t.

Unlike $\mathsf{sbf}^L(S, t)$, which can be described only in terms of P and Q, the improved supply function $\mathsf{sbf}^B(S, t)$ also depends on the maximum resource holding time H of the application. However, as explained in the previous subsection, this parameter can be considered constant and does not represent an output for the optimization procedure.

Let $Con^j(w, t)$ be the j-th feasibility region. Formally,

$$Con^1(w, t) := \{(P, Q) \mid (20) \wedge (21) \wedge (22)\}; \tag{29}$$

$$Con^2(w, t) := \{(P, Q) \mid (23) \wedge (24) \wedge (25)\}; \tag{30}$$

$$Con^3(w,t) := \{(P,Q) \mid (26) \wedge (27) \wedge (28)\}. \quad (31)$$

The union of such sets determines the region $\mathbb{D}(w,t)$ for any value of w and t, that is:

$$\mathbb{D}(w,t) = \bigcup_{1 \le j \le 3} Con^j(w,t). \quad (32)$$

Combining these regions according to (19), we obtain the final feasibility region composed of all pairs (P,Q) that guarantee the schedulability of the application:

$$\mathbb{D} = \bigcap_{d_i \in dSet} \mathbb{D}(w_i,t_i) = \bigcap_{d_i \in dSet} \bigcup_{1 \le j \le 3} Con^j(w_i,t_i). \quad (33)$$

4.2. Reducing the design space

In this section, the feasibility region $\mathbb{D}$ is refined by introducing some generic constraints that are valid for each test point, with the purpose of eliminating impossible scenarios. The feasibility region including such additional constraints will be denoted as $\mathbb{D}^+$.

A constraint on the server period can be derived considering the blocking time, given by Equation (7), introduced on the servers by global resources. The effect of the global blocking B^G is illustrated in Figure 2.

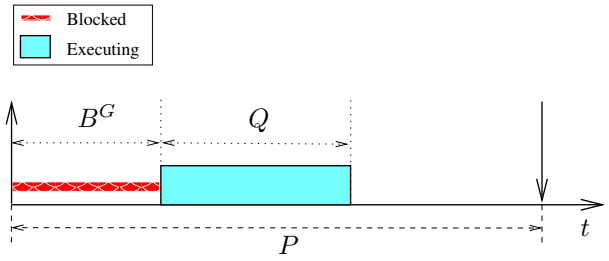

Figure 2: Global blocking of a server S.

In order to guarantee the schedulability of the server, it is easy to see that each server S must have at least a period

$$P^{min} = B^G + Q. \quad (34)$$

Note that this is only a necessary condition, since it does not take into account the interference of the other servers. Considering the interference coming from the other servers would be in contrast with our *modular design approach*, which does not allow selecting the parameters of a server *at design time* based on the parameters of other subsystems[1]. If this were not the case, every time a new server enters/leaves the system, or any server parameter is changed, it would be necessary to re-evaluate the design parameters of all the servers in the system, significantly increasing the overhead and the complexity of the design process. Moreover, selecting the parameters of a server based on the other system components would lead to a *circular dependency* in the server design process, because the server period selection has a clear impact on the interference to/from the other subsystems. This circular dependency could only be handled by considering a global design method that takes into account all parameters (not just the interfaces) of all applications composing the system at once. Besides being in contrast with our modular approach, such a global method has a huge computational complexity, limiting its applicability to static systems without the typical on-line variations of open environments.

However, using expression (34) in the selection of the parameters of a server would still give rise to a circular dependency, because the server period selection impacts the blocking to/from the other subsystems (see Equation (7)). To maintain the modularity of our approach, a more general upper bound for B^G can be derived considering that in our framework $H \le \mathcal{H}$. From Equation (7), it follows $B^G \le \mathcal{H}$. Hence, a lower bound on the minimum period that does not depend on the parameters of the other servers can be expressed as

$$P \ge \mathcal{H} + Q. \quad (35)$$

A simple constraint on the maximum service delay Δ can be obtained by

$$\Delta \le \Delta_{\max} \stackrel{\text{def}}{=} T_{\min} \stackrel{\text{def}}{=} \min\{T_1 - C_1, \ldots, T_n - C_n\}, \quad (36)$$

because no task can stand a service delay greater than its period. Being $\Delta = 2(P-Q)$, it follows

$$P \le \frac{T_{min}}{2} + Q. \quad (37)$$

Considering $\alpha \le \frac{1}{2}$,

$$\Delta = 2(P-Q) = 2P\left(1 - \frac{1}{\alpha}\right) \ge P.$$

An upper bound on the server period can then be derived from Equation (36):

$$P \le T_{\min}. \quad (38)$$

Finally, a lower bound on the server budget is derived from Equation (4):

$$Q \ge H. \quad (39)$$

As shown in Figure 3, Equations (35), (37), (38), and (39) determine a closed region with trapezoidal shape in the P-Q plane, denoted as $\mathbb{G}$:

$$\mathbb{G} \stackrel{\text{def}}{=} \{(P,Q) | \begin{cases} P \ge Q + \mathcal{H} \\ P \le Q + \frac{T_{\min}}{2} \\ P \le T_{\min} \\ Q \ge H \end{cases} \}$$

For uniformity, each of the four constraints will be denoted as $Con^j(w_i,t_i)$, with $j = 4,5,6,7$, respectively.

With the generic constraints introduced above, the definition of the feasibility region $\mathbb{D}^+$ can be formalized as follows:

$$\mathbb{D}^+ := \mathbb{D} \cap \mathbb{G}. \quad (40)$$

The overall optimization problem consists in finding the optimal point (P_{opt}, Q_{opt}) in the feasibility region $\mathbb{D}^+$ that minimizes the objective function α'.

1. Note that all subsystems are instead considered *at integration time*, when checking the global schedulability of all servers in the system. However, the global schedulability test is based on the reduced set of parameters specified in the interface of each server.

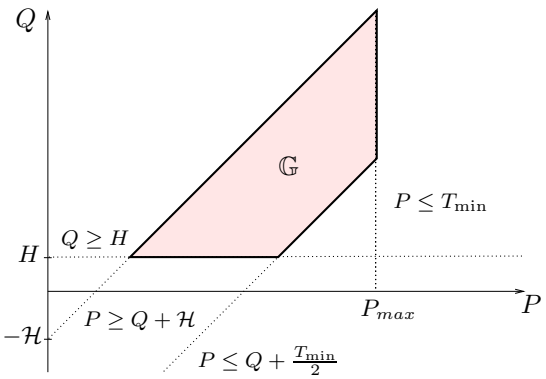

Figure 3: Region $\mathbb{G}$.

5. Design algorithm

Using the optimal design method presented in Section 4, this section illustrates how to derive a complete algorithm for computing the optimal server parameters. A fully-working implementation of the final algorithm is also available [15] for MATLAB®.

The idea underlying the algorithm is to build the feasibility region $\mathbb{D}^+$ iteratively and represent it only by means of vertices and edges connecting them. A computational representation of the contour is sufficient since the optimal point (P_{opt}, Q_{opt}) certainly belongs to the contour of the region.

Intuitively, the issue of defining the contour of the feasibility region is addressed by considering all the constraints as equalities, then intersecting them and filtering out the vertices and edges that do not satisfy the constraints as inequalities.

For each constraint Con^j, its contour $\mathbb{C}^j$ is defined as the set of points where the constraint is satisfied as equality. In general, $\mathbb{C}^j$ can be expressed as a set of vertices $\mathbb{V}^j$ and a set of edges $\mathbb{E}^j$. Similarly, $\mathbb{C}^{\mathbb{D}^+}$ denotes the contour corresponding to $\mathbb{D}^+$. The same notation holds for the sets of edges and vertices $E^{\mathbb{D}^+}$ and $\mathbb{V}^{\mathbb{D}^+}$. In addition, let $\mathcal{L}(e_i)$ be the locus of points in the P-Q plane belonging to $e_i \in \mathbb{E}^j$. Two fundamental properties are exploited to build $\mathbb{C}^{\mathbb{D}^+}$:

(i) $\forall e \in \mathbb{E}^{\mathbb{D}^+} \exists j \mid e \in \mathbb{C}^j$,
that is, the edges composing $\mathbb{C}^{\mathbb{D}^+}$ belong to the contour of at least one of the regions determined by the constraints listed in Section 4;

(ii) $\forall v \in \mathbb{V}^{\mathbb{D}^+} \exists j \mid v \in \mathbb{V}^j \vee \exists i, \ell \mid v \in \mathcal{L}(e_i) \cap \mathcal{L}(e_\ell)$,
that is, a vertex composing $\mathbb{C}^{\mathbb{D}^+}$ is either a proper vertex of a contour, or the intersection point between pairs of edges belonging to the contour of different regions.

5.1. How to find vertices and edges

The proposed algorithmic approach requires computing the intersections between the contours of regions to iteratively produce the vertices and edges defining $\mathbb{D}^+$. This can be obtained by solving systems of two equations representing pairs of edges. Such systems can be trivially solved in all cases except when dealing with edges of Con^1 and Con^2, which follow a staircase-like pattern, due to the presence of the *ceil* function. We addressed this issue by observing that the contour of such constraints can simply be expressed with a set of linear edges. Intuitively, it is sufficient to identify the points where the linearity of the constraints is broken and account such points as vertices. Analytically speaking, the ceil function $f(x) = \lceil x \rceil$ lies by definition on or above the function $g(x) = x$, and it is easy to see that the two functions are equal only at the integer points:

$$\lceil x \rceil = x \quad \Leftrightarrow \quad x \in \mathbb{Z}.$$

Looking at the definition of Con^1 and Con^2 (see Equations (20) and (23)), we can observe that the dependency on the *ceil* function is introduced by the definition of k (see Equation (12)). Hence, we need to find the values of P and Q where the argument of the *ceil* in (12) is an integer value, formally described by the following set:

$$\left\{ (P,Q) \mid k = \left\lceil \frac{t-\Delta}{P} \right\rceil = \frac{t-\Delta}{P} \right\}, \tag{41}$$

which gives (considering that $\Delta = 2(P-Q)$):

$$Q = \frac{(k+1)P - t}{2}. \tag{42}$$

The proper vertices of Con^1 and Con^2, i.e. the sets $\mathbb{V}^1$ and $\mathbb{V}^2$, can be found by imposing Equation (42) in Equations (20) and (23), respectively, expressed as equalities. This corresponds to solving the following linear systems:

$$\begin{cases} Q = \frac{w-t}{k+1} + P \\ Q = \frac{(k+1)P-t}{2} \end{cases} \qquad \begin{cases} Q = \frac{w}{k} + H \\ Q = \frac{(k+1)P-t}{2} \end{cases} \tag{43}$$

The vertices not satisfying the corresponding temporal constraints (Equations (21)-(22), and (24)-(25)) can be safely discarded.

These linear systems in two equations must be solved for $k \in [1, k_{max}]$, with $k \in \mathbb{Z}$. Theorem 4 states a possible upper-bound for k.

Theorem 4:

$$k \leq k_{max} = \left\lceil \frac{t}{H + \mathcal{H}} \right\rceil. \tag{44}$$

Proof: Replacing Δ in Equation (12), we get:

$$k = \left\lceil \frac{t - 2P + 2Q}{P} \right\rceil = \left\lceil \frac{t}{P} - 2 + 2\alpha \right\rceil.$$

From Equations (35) and (39), we get $P \geq H + \mathcal{H}$. Using this relation, together with $\alpha \leq 1$, in the previous expression of k, we obtain $k_{\max}$, proving the theorem. □

As we can see from Equation (26), also the contour of Con^3 is described by a non-linear function. However, intersecting $\mathbb{C}^3$ with other contours is in this case trivial, as it only requires solving a second degree equation, accepting only solutions with $P > 0$ and $Q > 0$. All the other constraints are even simpler to manage, since in the first quadrant (i.e., for positive values of P and Q) they do not involve any non-linear function. Thus, their set of vertices is empty, while their set of edges has exactly one element.

5.2. Algorithm definition

The resulting OPTBROE algorithm for computing the optimal BROE server parameters (P_{opt}, Q_{opt}) is reported in Figure 4.

```
1: procedure OPTBROE(t[], w[], H)
2:     finCont ← GENSHAPE0(t1, w1, H)          ▷ see Sec.4.2
3:     for i := 1 to size(t[]) do
4:         for j := 1 to 3 do
5:             cont[j] ← GETCONTOUR(j, ti, wi)  ▷ see Sec.5.1
6:             finCont ← INTERSECTCONTOURS(finCont, cont[j])
7:         end for
8:         finCont ← FILTERCONTOUR(i, finCont, t[], w[])
9:     end for
10:    (Popt, Qopt) ← COMPUTEOPTIMUM(finCont)
11:    return (Popt, Qopt)
12: end procedure
```

Figure 4: Algorithm for computing the optimal BROE server parameters.

It takes as inputs two vectors $(t[], w[])$, representing the values t_i and w_i of the dbf, and the maximum resource holding time H of the application. As explained in Section 4, these are the only parameters that are required to apply the proposed design methodology. The System Maximum Resource Holding Time $\mathcal{H}$ and the context switch overhead σ can be considered as global parameters of the optimization procedure.

At line 2, the algorithm first builds the initial feasibility region $\mathbb{G}$ by accounting for the generic constraints (Con^4 to Con^7) explained in Section 4.2. At line 3, the vector $t[]$ is scanned to incrementally refine the feasibility region at each demand point. To do this, the vertices and edges determined by each constraint Con^1, Con^2 and Con^3 are stored in a contour vector (line 5), and the final feasibility region is computed by intersecting such contours (line 6). The procedure INTERSECTCONTOURS splits intersecting edges into two edges, adding the intersection point as a new vertex. Once this is done for the three mentioned constraints, the region produced at each step must be filtered to remove vertices and edges that do not contribute to the feasibility region at each step (line 8). Finally, the algorithm computes the optimal server interface that satisfies all the constraints and returns the corresponding values.

The pseudocode of the filtering procedure FILTERCONTOUR is reported in Figure 5. As mentioned above, the feasibility region is built through an incremental refinement; hence, this procedure is in charge of removing edges that do not concur to the definition of the feasibility region at each step i. Each edge composing the current feasibility region (line 3) is discarded if at least one of the involved constraints is violated (line 4-6). At the end of the refinement process (after line 9), the contour $\mathbb{C}^{\mathbb{D}^+}$ of the feasibility region $\mathbb{D}^+$ is obtained in terms of vertices and edges.

Figure 6 reports procedure COMPUTEOPTIMUM for selecting the optimal server interface (P_{opt}, Q_{opt}) from the final contour. Such an optimal point must be either (i) a vertex of $\mathbb{C}^{\mathbb{D}^+}$ or (ii) a point of the objective function (Equation (13)) tangent to a non-linear edge. Procedure ADDTANGENTVERT adds to $\mathbb{C}^{\mathbb{D}^+}$ the candidate vertices falling in case (ii). They are found by considering the tangent point between the quadratic curve determining the non-linear edge and the objective function $Q = \alpha' P - \sigma$. Since there is just one non-linear constraint per checkpoint, ADDTANGENTVERT will add at most a pseudopolynomial number of points to finalCont. Once the complete set of candidate vertices for the optimum has been obtained, it is sufficient to scan such a set to select the vertex that minimizes the objective function (lines 4-9).

```
1: procedure FILTERCONTOUR(i, Cont, t[], w[])
2:     retCont ← Cont
3:     for each edge in Cont do
4:         inShapei ← edge ∈ ⋂_{j=1}^{i} 𝔻(wj, tj) ∩ 𝔾
5:         if ¬inShapei then
6:             retCont ← retCont \ edge
7:         end if
8:     end for
9:     return retCont
10: end procedure
```

Figure 5: Procedure for filtering the contour of a region at step i.

```
1: procedure COMPUTEOPTIMUM(finalCont)
2:     αopt ← 1
3:     finalCont ← ADDTANGENTVERT(finalCont)
4:     for each vertex in finalCont do
5:         if (vertex.Q + σ)/vertex.P < αopt then
6:             αopt ← (vertex.Q + σ)/vertex.P
7:             (Popt, Qopt) ← (vertex.P, vertex.Q)
8:         end if
9:     end for
10:    return (Popt, Qopt)
11: end procedure
```

Figure 6: Procedure for computing optimal server interface (P_{opt}, Q_{opt}).

5.3. Example

To clarify the method adopted for building the space of the feasible (P, Q) pairs and selecting the one which minimizes the bandwidth occupation, we propose a simple example. Consider an application Γ_k which is already expressed in terms of its dbf, i.e., as a set of pairs (w_i, t_i) (as mentioned in Section 4). The specific values of the test points of the dbf and its corresponding values are expressed as two vectors $\mathbf{t}$ and $\mathbf{w}$:

$$\mathbf{t} = \{200, 320, 400, 500, 600\},$$
$$\mathbf{w} = \{35, 70, 80, 120, 140\}.$$

The other parameters involved in the optimization procedure are $H_k = 15$, $\sigma = 10$, $\mathcal{H} = 20$.

Procedure OPTBROE is able to iteratively compute the feasibility region in the P-Q plane, and to identify the optimum point considering the objective function of Equation (13). Figure 7 represents the feasibility region obtained considering the entire dbf, and the tangent line identifying the optimum. In this specific case we get $(P_{opt}, Q_{opt}) = (133, 50)$, which leads to a minimum bandwidth

$$\alpha' = \frac{Q_{opt} + \sigma}{P_{opt}} = \frac{60}{133} = 0.45.$$

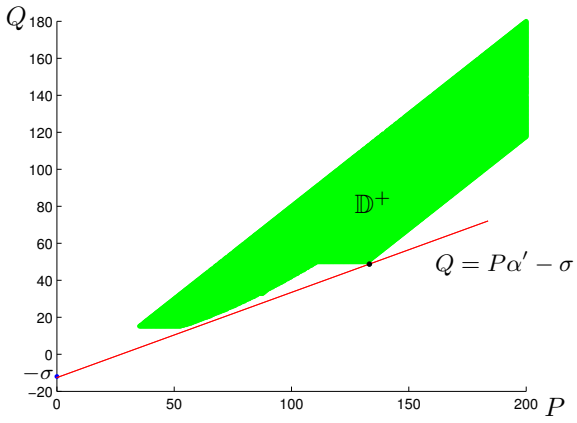

Figure 7: Feasibility region $\mathbb{D}^+$ and objective function in the optimum for the considered example.

Figure 8 illustrates the considered dbf function, together with the optimal sbf^B found through the optimization procedure. Please note how the improved supply bound function is able to better adhere to the shape of the dbf, allowing a smaller server bandwidth, as well as a reduced resource holding time.

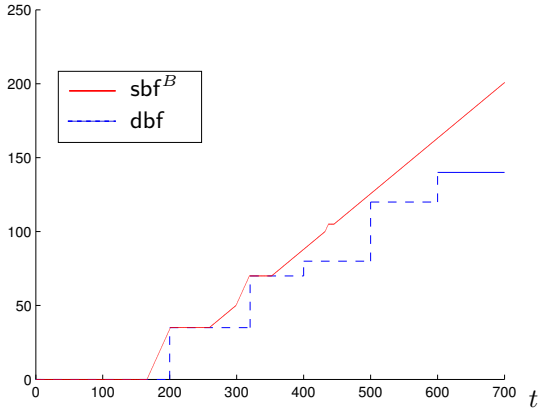

Figure 8: Optimal sbf^B and the dbf for the example.

6. Complexity

The number of steps of the iterative procedure is bounded by the pseudo-polynomial number of demand points to check given by Equation (16). At each point, the procedure OPTBROE is invoked. The complexity of the procedure is dominated by the invocation of INTERSECTCONTOURS, which increases the number of vertices to consider. This procedure is in charge of computing the intersection of two contours. At the generic step i, it computes the intersection between $\mathcal{C}^i$ and $\bigcap_{l=1}^{i-1} \mathcal{C}^l$, which respectively describe two non-convex regions in the P-Q plane. These regions are composed of linear edges and non-linear edges. Computing the intersection of non-convex regions with linear edges is a well-known problem in the literature: as shown by Bentley and Ottmann [16], this problem can be solved with $O(n\ log\ n)$ complexity, where n is the number of intersecting edges. The number of linear edges of each single region determined by each $\mathcal{C}^l$ is $O(k_{max})$ (see Theorem 4). On the other hand, the non-linear edges (given by Equation (26)) determine a convex region, and can be easily treated without affecting the worst-case complexity (as has been also shown by Bini et al. in [17]). Overall, procedure INTERSECTCONTOURS does not jeopardize the pseudo-polynomial complexity of the design algorithm.

There is still one remaining problem for bounding the complexity of the presented method. That is, the size of $dSet$ given by Equation (16) depends (through α and Δ) on the server parameters, which are not yet determined. To solve this problem, the method for deriving t^* presented in Section 4 can be modified as follows.

For every feasible BROE server S, the following inequality should hold:

$$\mathsf{sbf}(S,t) \leq \alpha_{\min}(t - \Delta_{\max}), \tag{45}$$

where the right-hand-side term represents a linear supply bound function with a service delay $\Delta_{\max}$ and supply rate $\alpha_{\min}$. The values $\Delta_{\max}$ and $\alpha_{\min}$ can be any upper (lower) bound on the maximum (minimum) possible service delay (supply rate) among all feasible BROE servers (i.e., BROE servers that can successfully schedule Γ).

If relation 45 is used to upper-bound the $\mathsf{sbf}(S_k,t)$, the derivation of Equation (16) becomes

$$t^* = \frac{\alpha_{\min}\Delta_{\max} + \sum_{i=0}^{n}(T_i - D_i)U_i}{\alpha_{\min} - U}. \tag{46}$$

While $\Delta_{\max}$ can be given by Equation (36), a lower bound on α can instead be iteratively derived as follows. Each checkpoint $t_i \in dSet$ is considered in increasing order, starting from the first point (w_1, t_1). For a given point (w_i, t_i), the smallest α_k that guarantees $\mathsf{sbf}(S_k, t_i) \geq w_i$ is computed applying procedure COMPUTEOPTIMUM to $\mathbb{D}(w_i, t_i)$. Note that the contour of a single region $\mathbb{D}(w_i, t_i)$ is rather simple, limiting the complexity of the procedure. A lower bound $\alpha_{\min} = Q/P$ is then easily derived from the values (P, Q) returned by the procedure. If $\alpha_{\min} \leq U$, it is necessary to continue to the next point in $dSet$. When a point is found for which $\alpha_{\min} > U$, a pseudo-polynomial upper bound t^* that does not depend on P_k and Q_k can be derived with Equation (46).

At this point, it is possible to either stop, or continue to the next point to find a smaller t^*. If a larger $\alpha_{\min}$ is found, a smaller t^* can be derived from Equation (16), further decreasing the number of checkpoints to evaluate. In any case, the method stops when the next point t_i to check is beyond t^*.

7. Related work

The first approaches proposed in the literature for designing servers parameters [4], [5], [18] did not consider resource sharing among different reservations. Bini et al. presented in [17] an optimal design for a bounded-delay partition without shared resources, considering a linear supply bound function for the server.

Shin et al. [19] and, later, Behnam et al. [20] proposed two methods for designing a reservation server considering resource

sharing with an overrun-based approach. In [21], [22], Behnam et al. proposed a design methodology for the BROE server. However, their approach considers a linear supply bound function, requires imposing a fixed period for each reservation server and assumes non-preemptive access to global resources.

A more complete approach for designing reservation servers was proposed by Van Den Heuvel et al. in [23]. The analysis for periodic servers considers SIRAP [7] and the overrun-based protocol for arbitrating the access to shared resources. However, like in the previous case, this work does not consider the improved supply function for BROE and still requires fixing a given period for each server at design time.

In [24], Bini et al. proposed an optimization methodology for an energy-aware scheduler, which inspired the approach presented in this paper.

8. Conclusions and future work

This paper presented a design framework for optimizing the server parameters in a hierarchical reservation system, where a number of real-time applications are concurrently executed in isolation within dedicated subsystems, sharing global resources. Reservations are implemented using BROE, which is currently the most effective server algorithm for handling reservations under resource sharing. The analysis is based on an improved supply bound function of BROE, and takes into account the overhead due to context switches between servers. Our framework is able to optimally select the server parameters with pseudo-polynomial complexity, assuming EDF is used for scheduling tasks within a reservation. A simple non-preemptive locking protocol is proposed to significantly simplify the whole design process in most cases of interest, reducing scheduling and preemption overheads, while improving system schedulability. As a future work, we plan to extend the BROE schedulability analysis under local fixed priority scheduling.

Acknowledgments

This work has been supported in part by the European Commission under the P-SOCRATES project (FP7-ICT-611016).

The authors like to thank Enrico Bini for the fruitful discussions that inspired the methodology presented in this work.

References

[1] C. W. Mercer, S. Savage, and H. Tokuda, "Temporal protection in real-time operating systems," in *Proc. of the 11th IEEE workshop on Real-Time Operating System and Software*, May 1994, pp. 79–83.

[2] 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.

[3] ——, "Resource reservations in dynamic real-time systems," *Real-Time Systems*, vol. 27, no. 2, pp. 123–165, 2004.

[4] G. Lipari and E. Bini, "A methodology for designing hierarchical scheduling systems," *Journal of Embedded Computing*, vol. 1, no. 2, pp. 257–269, April 2005.

[5] I. Shin and I. Lee, "Compositional real-time scheduling framework," in *Proceedings of the 25th IEEE Real-Time Systems Symposium*, Lisbon, Portugal, December 5-8, 2004, pp. 57–67.

[6] R. I. Davis and A. Burns, "Resource sharing in hierarchical fixed priority pre-emptive systems," in *Proceedings of the IEEE Real-time Systems Symposium (RTSS 2006)*, Rio de Janeiro, Brazil, December 5-8, 2006, pp. 257–268.

[7] M. Behnam, I. Shin, T. Nolte, and M. Nolin, "SIRAP: a synchronization protocol for hierarchical resource sharing in real-time open systems," in *Proceedings of the 7th ACM & IEEE International Conference on Embedded Software (EMSOFT 2007)*, Salzburg, Austria, October 1173, 2007.

[8] M. Bertogna, N. Fisher, and S. Baruah, "Resource-sharing servers for open environments," *IEEE Transactions on Industrial Informatics*, vol. 5, no. 3, pp. 202–219, August 2009.

[9] A. Biondi, G. Buttazzo, and M. Bertogna, "Schedulability analysis of hierarchical real-time systems under shared resources," RETIS Lab, Scuola Superiore Sant'Anna, Italy, Technical Report TR-13-01, July 2013.

[10] L. Marzario, G. Lipari, P. Balbastre, and A. Crespo, "IRIS: A new reclaiming algorithm for server-based real-time systems," in *Proc. of the IEEE Real-Time and Embedded Technology and Applications Symposium*, Toronto, Canada, May 2004.

[11] R. I. Davis and A. Burns, "Resource sharing in hierarchical fixed priority preemptive systems," in *Proc. of the 27th IEEE Real-Time Systems Symposium*, Rio de Janeiro, Brazil, December 5-8, 2006.

[12] S. Baruah, "Resource sharing in EDF-scheduled systems: a closer look," in *Proceedings of the 27th IEEE Real-Time Systems Symposium (RTSS'06)*, Rio de Janeiro, Brazil, December 5-8, 2006.

[13] M. Bertogna, N. Fisher, and S. Baruah, "Resource holding times: Computation and optimization," *Real-Time Systems*, vol. 41, no. 2, pp. 87–117, February 2009.

[14] S. Baruah, A. K. Mok, and L. E. Rosier, "Preemptively scheduling hard-real-time sporadic tasks on one processor," in *Proc. of the 11th IEEE Real-Time Systems Symposium (RTSS'90)*, Orlando, FL, 1990, pp. 182–190.

[15] *A MATLAB® optBROE algorithm implementation*, http://retis.sssup.it/∼a.biondi/optBROE.

[16] J. L. Bentley and T. A. Ottmann, "Algorithms for reporting and counting geometric intersections," *IEEE Computer Society*, 1979.

[17] E. Bini, G. Buttazzo, and Y. Wu, "Selecting the minimum consumed bandwidth of an EDF task set,," in *2nd Workshop on Compositional Real-Time Systems*, December 2009.

[18] L. Almeida and P. Pedreiras, "Scheduling within temporal partitions: Response-time analysis and server design," in *4th ACM Int. Conf. of Embedded Softw. (EMSOFT '04)*, Pisa, Italy, September 2004, pp. 95–103.

[19] I. Shin, M. Behnam, T. Nolte, and M. Nolin, "Synthesis of optimal interfaces for hierarchical scheduling with resources," in *Proc. of the 29th IEEE International Real-Time Systems Symposium (RTSS 2008)*, Barcelona, Spain, December 2008, pp. 209–220.

[20] M. Behnam, T. Nolte, M. Sjödin, and I. Shin, "Overrun methods and resource holding times for hierarchical scheduling of semi-independent real-time systems," *IEEE Transactions on Industrial Informatics*, vol. 6, no. 1, pp. 93–104, February 2010.

[21] M. Behnam, T. Nolte, and N. Fisher, "On optimal real-time subsystem-interface generation in the presence of shared resources," in *In proc. of ETFA 2010, IEEE Conference*, Bilbao, Spain, September 13-16, 2010.

[22] M. Behnam and N. Fisher, "Subsystem-interface generation in the presence of shared resources," in *Proceedings of the CRTS09 workshop in conjunction with the 30th IEEE International Real-Time Systems Symposium (RTSS '09)*, December 2009.

[23] M. van den Heuvel, M. Behnam, R. J. Bril, J. Lukkien, and T. Nolte, "Opaque analysis for resource sharing in compositional real-time systems," in *4th Workshop on Compositional Theory and Technology for Real-Time Embedded Systems (CRTS'11)*, November 2011.

[24] E. Bini, G. C. Buttazzo, and G. Lipari, "Minimizing cpu energy in real-time systems with discrete speed management," *ACM Transactions on Embedded Computing Systems*, vol. 8, no. 4, 2009.

[25] A. Easwaran, M. Anand, and I. Lee, "Compositional analysis framework using EDP resource models," in *Proceedings of the*

28th IEEE Real-Time Systems Symposium (RTSS 2007), Tucson, Arizona, USA, December 3-6, 2007.

[26] S. Baruah and A. Burns, "Sustainable scheduling analysis," in *Proceedings of the IEEE Real-time Systems Symposium (RTSS 2006)*, Rio de Janeiro, Brasil, December 5-8, 2006.

Appendix A.
BROE server rules

The rules of a BROE server with period P and maximum budget Q (bandwidth $\alpha = Q/P$) are summarized below. We assume that the server is serving an application having maximum Resource Holding Time H. At any time t, the server is characterized by an absolute deadline d and a remaining budget q. When a job executes, q is decreased accordingly.

1) Initially, $q = 0$ and $d = 0$.
2) When BROE is idle and a job arrives at time t, a replenishment time is computed as $t_r = d - q/\alpha$:
   a) if $t < t_r$, the server is suspended until time t_r. At time t_r, the budget is replenished to Q and $d \leftarrow t_r + P$.
   b) otherwise the budget is immediately replenished to Q and $d \leftarrow t + P$;
3) When $q = 0$, the server is suspended until time d. At time d, the server budget is replenished to Q and the deadline is postponed to $d \leftarrow d + P$.
4) When a pending task wishes to access a global resource at a time t, a budget check is performed: if $q \geq H$, there is enough budget to complete the critical section, hence the access is granted. Otherwise a replenishment time is computed as $t_r = d - q/\alpha$:
   a) if $t < t_r$, the server is suspended until time t_r. At time t_r, the budget is replenished to Q and $d \leftarrow t_r + P$.
   b) otherwise the budget is immediately replenished to Q and $d \leftarrow t_r + P$.

According to such rules, a server running ahead with respect to its guaranteed processor utilization will self-suspend in two cases: when reactivating after an idle time (Rule 2), and when trying to enter a global critical section with insufficient budget (Rule 4). In both cases, it will self-suspend until the guaranteed processor utilization is matched (time $t_r = d - q/\alpha$). At time t_r, the server budget is replenished to Q and the deadline is set to $d \leftarrow t_r + P$. When instead the server consumed less processor resources than its allowed share, it will immediately replenish its budget in the two mentioned cases. However, the deadline set when reactivating after an idle ($d \leftarrow t + P$, according to Rule 2) differs from the one set when trying to enter a global critical section with insufficient budget ($d \leftarrow t_r + P$, according to Rule 4).

Appendix B.
Derivation of the BROE supply function

This section presents the derivation of the improved supply bound function $\mathsf{sbf}^B(t)$ for the BROE server. The underlying motivation for this work is that the linear bound $\mathsf{sbf}^L(t)$, on which the original BROE schedulability test is based, can give a too pessimistic lower bound when an application Γ accesses global resources with small resource holding times. For this reason, the knowledge of the maximum resource holding time H of the global resources accessed by Γ is exploited to derive a tighter supply function for the BROE server.

Note that the use of H for the local analysis is compliant with the framework presented in Section 2. In fact, resource holding times are used by the global schedulability analysis and the resource access policy to avoid budget depletion within critical sections.

When global resources are not accessed by tasks running upon a reservation server (i.e., $H = 0$), the BROE server behaves as a classical hard CBS server (like IRIS [10]), whose supply bound function is described as follows [23], [25]:

$$\mathsf{sbf}^P(t) = \max \left\{ \begin{matrix} 0, \\ (h(t)-1)Q, \\ t - (h(t)+1)(P-Q) \end{matrix} \right\}, \tag{47}$$

where $h(t) = \lceil \frac{t-P+Q}{P} \rceil$. The function $\mathsf{sbf}^P(t)$ is shown in Figure 1 as a dashed line.

When considering resource sharing, the worst-case supply can be found by considering Rule 4 of BROE in Section A, which reduces $\mathsf{sbf}^P(t)$ in some time intervals. Note that Rule 2 does not affect $\mathsf{sbf}^B(t)$, since it is applied only when the server is resumed from an idle state, and therefore will never be invoked in the busy period considered in the worst-case scenario.

The BROE supply function is derived by analyzing the worst-case behavior of the server considering the access to global resources. In the following, the $\mathsf{sbf}^B(t)$ is derived in the first two periods after the worst-case delay Δ, and then generalized for the generic k^{th} period.

First period. After its worst-case delay $t = \Delta$, the server will be able to execute for at least $Q-H$ units of time. Hence, after time $t = \Delta+Q-H$, a pending task wishing to access a global resource R can experience the condition $q < H$. Under this condition, Rule 4 causes a deadline shift to $t_r + P$, suspending the server H units earlier than in the more favorable $\mathsf{sbf}^P(t)$. Then, the latest time the server can resume execution is at time $t_1 = t_r + P - Q$. Since $t_r = \Delta + Q - H/\alpha$, then $t_1 = \Delta + P - H/\alpha$. Such a point lies at the intersection of the original $\mathsf{sbf}^L(t)$ (dotted line in Figure 1).

It is worth observing that the same condition imposed by Rule 4 ($q < H_j$) can occur at any time in $(\Delta + Q - H,\ \Delta + Q]$; this is because the access to a global resource R_j can occur with an arbitrary current budget q. Hence, to cope with all possible scenarios, it must be $\mathsf{sbf}^B(t) = \mathsf{sbf}^L(t) = \alpha t$ for each t in the interval $(\Delta + P - H/\alpha,\ \Delta + P]$.

Second period. In the second period $[\Delta + P,\ \Delta + 2P)$, the reduction of the $\mathsf{sbf}^B(t)$ with respect to $\mathsf{sbf}^P(t)$ is more significant. The reason is that the server, when resuming the execution at time $t_a = \Delta + P - H/\alpha$, can be suspended again after executing for $Q-H$ time units, thus the $\mathsf{sbf}^P(t)$ is "cropped" earlier than in the previous period. In this scenario the server has a deadline at $d = t_a + P$ and $t_r = d - q/\alpha = t_a + P - H/\alpha$. The latest time the server can resume execution can be computed as $t_2 = t_r + P - Q$ which, substituting the expression of t_r, gives $t_2 = \Delta + 2P - 2H/\alpha$. This point lies again at the intersection of the original $\mathsf{sbf}^L(t)$. For the same reason explained above, to cope with all possible scenarios, it must be $\mathsf{sbf}^B(t) = \mathsf{sbf}^L(t) = \alpha t$ for each t in the interval $(\Delta + 2P - 2H/\alpha,\ \Delta + 2P]$.

Generic $\mathbf{k^{th}}$ period. In general, the reduction of the $\mathsf{sbf}^B(t)$ with respect to $\mathsf{sbf}^P(t)$ increases period by period, until it reduces to $\mathsf{sbf}^L(t)$ from some t on.

Considering the k^{th} period after Δ, with k given by Equation (12), the interval in which $\mathsf{sbf}^B(t) = \mathsf{sbf}^L(t)$ can be expressed as

$$[\Delta + kP - \frac{kH}{\alpha},\ \Delta + kP]. \tag{48}$$

The larger k, the larger the interval. The first period in which, for all t, $\mathsf{sbf}^B(t) = \mathsf{sbf}^L(t)$ can be derived by finding the first k that satisfies the following inequality:

$$\Delta + kP - \frac{kH}{\alpha} \leq \Delta + (k-1)P,$$

which gives $k \geq Q/H$.

Figure 9 shows the $\mathsf{sbf}^B(t)$ in the k^{th} period after Δ, for $k < Q/H$. The values of the timing parameters used in the figure are reported in Table 2.

Hence, the BROE supply bound function can be formally expressed as in Equation (11).

Finally, it is worth observing that H is just an upper bound of the resource holding time, while the effective locking time can be significantly smaller, depending on the interference caused by higher priority jobs preempting the critical section. A robust and sustainable [26] analysis should therefore consider the case in which the global lock is released an infinitesimal time after the budget replenishment and another lock request is made $Q-H$ time-units after that, as correctly considered in the above discussion.

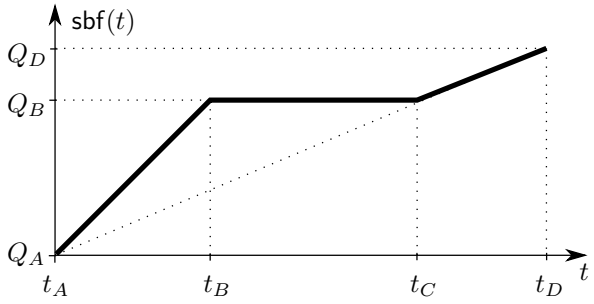

Figure 9: $\mathsf{sbf}^B(t)$ in the k^{th} period after the service delay.

t_A	$\Delta + (k-1)P$
t_B	$\Delta + (k-1)P + (Q - kH)$
t_C	$\Delta + kP - kH/\alpha$
t_D	$\Delta + kP$
Q_A	$(k-1)Q$
Q_B	$kQ - kH$
Q_D	kQ

Table 2: Values for Figure 9.

Appendix C.
Resource holding time computation

The budget check mechanism of BROE guarantees that when a server S locks a global resource, it will be able to execute for the duration of the whole resource holding time $H \leq Q$ without being suspended due to a budget depletion. This means that the server will be able to execute for at least H time units with a maximum service delay of $(P-Q) = \Delta/2$ after the lock is granted.

The supply bound function to consider for computing the resource holding time is therefore null for $\Delta/2$ time-units, and then grows with unitary slope until the resource is unlocked (see Figure 10).

Under EDF, the cumulative execution request of jobs that can preempt a task τ_i while it is holding a resource R_j for t units of time is given by (see [13])

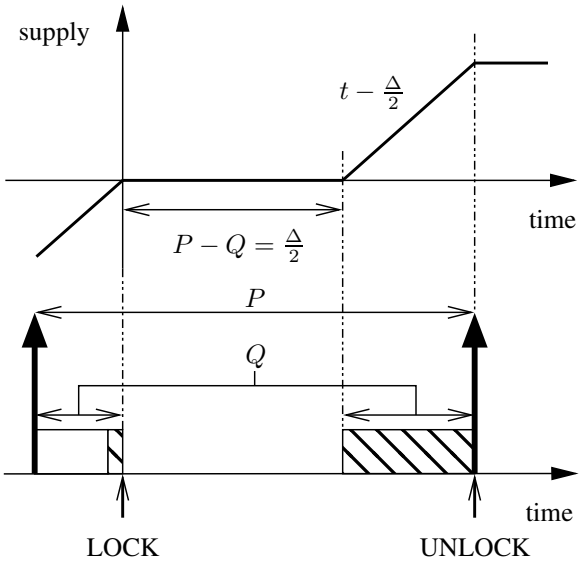

Figure 10: Worst case supply when a global resource is locked.

$$F_i(t) = \frac{\Delta}{2} + \delta_{i,j} + \sum_{k=1}^{\pi_j - 1} \left\lceil \frac{\min(t, D_i - D_k)}{T_k} \right\rceil C_k, \qquad (49)$$

where π_j represents the local ceiling of resource R_j, and $\delta_{i,j}$ represents the WCET of longest critical section of τ_i related to R_j. Let t_i^* be the smallest fixed point of function $F_i(t)$ (i.e., $F_i(t_i^*) = t_i^*$). If $F_i(t)$ exceeds $\min(Q, d_i)$, the iteration can be aborted rejecting the application.

Note that the term $\Delta/2$ is needed to take into account higher priority tasks arriving while the server is not executing. However, the resource holding time represents the maximum supply needed by an application to release a lock. It is therefore a measure of the *execution time* needed, rather than a measure of the *real time* elapsed between the lock and release of the resource. The resource holding time of task τ_i for resource R_j is then given by

$$H_j(i) = t_i^* - \frac{\Delta}{2}. \qquad (50)$$

More details on the above technique can be found in [13], where it is proved for a similar case that a fixed point is reached within a finite (pseudo-polynomial) number of steps.