Hard Constant Bandwidth Server: Comprehensive Formulation and Critical Scenarios

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

Abstract

*The Constant Bandwidth Server (*CBS*) is one of the most used algorithms for implementing resource reservation upon deadline-based schedulers. Although many* CBS *variants are available in the literature, no proper formalization has been proposed for the* CBS *in the context of hard reservations, where it is essential to guarantee a bounded-delay service across applications. Existing formulations are affected by a problem that can expose the system to dangerous deadline misses in the presence of blocking. This paper analyzes such a problem and presents a comprehensive and consistent formulation of the* CBS *for hard reservation scenarios. An overview of the contexts in which a hard* CBS *can be applied is also provided, focusing on the impact that previous formulations can have on schedulability, when used in conjunction with specific resource sharing protocols or other scheduling mechanisms that may cause a server to block.*

1. Introduction

In real-time computing systems running multiple concurrent tasks, a fundamental property that has to be ensured to support a component-based development is *temporal protection*, which prevents unexpected overruns occurring in a task from affecting the execution of other tasks. Resource Reservation [1] represents the most powerful scheduling mechanism specifically conceived to achieve such a property.

The idea behind the notion of Resource Reservation is that each task (or set of tasks) is assigned a fraction of the CPU, and is scheduled in such a way that it will never demand more than its reserved bandwidth. With this abstraction, processor capacity is viewed as a quantifiable resource that can be reserved, like physical memory or disk blocks.

The need for temporal isolation arises in many contexts. In the real-time community, its primary motivation was to integrate hard, soft, and non-real-time tasks. Indeed, many real-time systems are not characterized by hard timing constraints, as is the case of multimedia applications, audio/video streaming, etc. For these applications, missing a deadline has no catastrophic consequences, but it only leads to performance degradation. When dealing with hybrid task-sets, composed of hard and soft tasks, temporal isolation allows protecting hard tasks from overruns generated by soft tasks.

More in general, achieving temporal isolation is necessary whenever a timely service has to be ensured in a system with heterogenous timing requirements and potential overload conditions. In case of dynamic or unpredictable computational workload, the system must be able to reconfigure or adapt itself, without affecting other functionalities. In such circumstances, each application can be protected from the timing interferences of other components by using a proper enforcement mechanism that preserves the temporal isolation.

The resource reservation framework is also effectively employed for hierarchical systems composed of a set of modular components, each handling its own application, where a different scheduling algorithm may be used within each component [2], [3]. Component-based design is increasingly used as a de facto approach to design complex embedded systems. In particular, it gives the possibility to handle the growing complexity of current industrial software systems and to support the design of *open environments* [4], [5], where independently developed real-time applications need to be validated and executed in isolation. Resource reservation can be efficiently used in such situations, by allocating different applications on different virtual processors, so that each application can execute in isolation, without being affected by the behavior of the other components.

Resource Reservation is typically implemented by assigning to each application a dedicated real-time server, called *reservation server*. Each server is characterized by a budget Q and period P, so that it provides to the corresponding application Q units of service every P time-units. The ratio $\alpha = Q/P$ is called *server bandwidth*. If an application A is assigned a reservation bandwidth α, it behaves as it were executing on a dedicated slower processor, with speed α times the original speed. However, the reserved budget may be granted with some delay with respect to a dedicated virtual processor, depending on the particular implementation of the server.

Definition 1 (Bounded-delay): A server is said to implement a *bounded delay* partition if in *any* time interval of length L the server provides the corresponding share of budget αL with a delay of at most Δ.

The bounded-delay property of a server is influenced by the budget replenishment policy, i.e., the rule(s) used to recharge the server budget upon depletion. Depending on the replenishment rule, it is possible to distinguish between *hard* and *soft* reservations.

Definition 2 (Hard reservation): A server is said to implement a *hard reservation* if, when the server budget is depleted, the server is suspended until the next replenishment time.

Definition 3 (Soft reservation): A server is said to implement

a *soft reservation* if, when the server budget is depleted, it is immediately replenished, so that the server remains always active.

In this paper, we will discuss the relation between the bounded-delay property and the budget replenishment policy for the most popular dynamic server: the Constant Bandwidth Server (CBS) [6], [7]. The basic idea behind the classic CBS is that, when the budget is exhausted, it is immediately recharged to $q_i = Q_i$, postponing the server deadline to $d_i = d_i + P_i$. Since a backlogged server[1] remains always active, the classic CBS implements a soft reservation.

However, as shown in [8], such a formulation presents a *deadline aging* problem. To understand this issue, consider a system consisting of two tasks τ_1 and τ_2 served by servers S_1 and S_2, respectively. As illustrated in Figure 1, at a certain instant, τ_1 is the only active task in the system and executes without being preempted. The associated server S_1 consumes all its budget, postponing its deadline several times. When task τ_2 is activated, server S_2 is assigned a short deadline and, according to EDF scheduling policy, it is allowed to execute. When the budget of S_2 is exhausted, its deadline is postponed. However, since the deadline of server S_1 is far away, S_2 has still the earliest deadline and can continue executing. As shown in the figure, S_1 will need to wait for a long time, without being able to guarantee Q_1 units of execution within P_1 time units, for multiple server periods. This problem is called *deadline aging*, and causes the amount of execution effectively granted by a server to depend on the activations and periods of the other servers.

Since it is not possible to provide an upper bound Δ on the service delay with which the server provides the reserved processor share in *any* time interval, the soft CBS is *not* a bounded-delay server[2].

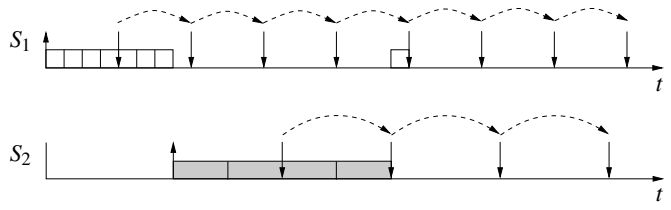

Figure 1: The *deadline aging* problem of CBS.

This issue is particularly negative for multimedia or interactive systems, because the potentially long service delay might determine a significant loss of quality of service and responsiveness. Even more importantly, this problem prevents the soft CBS to be used for hierarchical systems, where a set of real-time tasks needs to be guaranteed on each server with a given scheduling algorithm. If the server does not implement a bounded-delay partition, it would be difficult to analyze the schedulability of each task on the given server, because no lower-bound can be given on the supply provided to each task by the server in *any* time interval. For example, if a high priority task arrives at the beginning of a long black-out period, it would inevitably miss its deadline. Even if the supply curve of the server *since its initial activation* is always within $\Delta = 2(P-Q)$ time-units from a dedicated virtual processor of speed α, there could be long sub-intervals (longer than Δ) in which no service is granted. This is due to the over-provisioning of budget accorded to lower priority tasks previously scheduled onto the same server. If a high priority task happens to arrive during such intervals, a deadline is likely to be missed.

The *deadline aging* problem has been addressed by introducing a bounded-delay variant of CBS implementing hard reservations, denoted as Hard Constant Bandwidth Server (H-CBS). Many works in the literature refer to the H-CBS algorithm, but none of them provides a reference with a proper formalization. In this paper, we intend to close this gap, giving once and for all a consistent formulation for the H-CBS, and showing that different existing H-CBS formulations are affected by an algorithmic issue that can jeopardize the server behavior in critical scenarios.

System model. This paper considers a uniprocessor system, composed of N subsystems $S_k \in \mathscr{S}$, $k = 1,\ldots,N$, each implemented by a reservation server (also denoted as S_k) characterized by a budget Q_k and a period P_k. Each server is also characterized by a *server bandwidth* $\alpha_k = Q_k/P_k$, and a *worst-case service delay* $\Delta_k = 2(P_k - Q_k)$.

We assume each subsystem S_k runs an application Γ_k consisting of n_k periodic or sporadic preemptive tasks. A *local scheduler* is in charge of selecting the running task on each subsystem. For the sake of simplicity, in this work we consider a two-level hierarchical system, although our contributions can be extended to a generic multi-level hierarchical system, using the compositional real-time scheduling framework proposed by Shin and Lee [3].

All the results presented in this paper also hold in the particular case in which each server handles one single task, that is when $\forall k = 1,\ldots,N,\ n_k = 1$. This model can be useful to achieve timing protection among tasks, e.g., as specified by the AUTomotive Open System ARchitecture (AUTOSAR) [9].

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.

Paper structure. The remainder of the paper is organized as follows. Section 2 discusses the related work. Section 3 summarizes the rules of the H-CBS, giving a consistent formulation for such a scheduling mechanism. Section 4 discusses some problems with existing formulations under resource sharing and their impact on the state of the art. Finally, Section 5 states our conclusions.

2. Related work

The classical CBS algorithm [6] has been introduced by Abeni and Buttazzo with the purpose of providing efficient run-time support to multimedia applications in a real-time system. They proposed it as a scheduling methodology based on reserving a fraction of the processor bandwidth to each task, under the EDF scheduling algorithm.

Rajkumar et al. in [10] introduced the notion of *hard reservation*. They presented an algorithm that suspends the server upon budget depletion until the next replenishment time. The downside

1. A server is said to be *backlogged* whenever it has some pending workload to execute.

2. Note that the server is instead able to guarantee a reserved budget of αL in an interval of length L *starting with a server activation*, with a delay of at most $\Delta = 2(P-Q)$.

of this approach is that the algorithm is not work-conserving, so that the system may remain idle even when there are pending jobs to execute, significantly reducing the throughput.

To solve the problem of *deadline aging* present in the original formulation of CBS, Marzario et al. [8] presented the Idle-time Reclaiming Improved Server (IRIS), which implements *hard reservations* guaranteeing a minimum budget in any time interval, and ensuring a work-conserving behavior. It presents two main differences with respect to the original CBS algorithm. The first difference is that IRIS explicitly sets a recharging time for each server, to implement hard reservations. Secondly, it introduces a rule, denoted as *time-warping*, which allows the idle time to be *reclaimed* and distributed among the needing servers. According to this rule, when an idle time occurs, it is possible to advance the recharging times of all the servers that are ready to execute but are waiting for replenishment. As we will show in the following sections, the hard CBS implementation proposed for IRIS presents some problem when a server is reactivated after being blocked.

In [11], Abeni et al. present the HGRUB server, which combines a hard reservation CBS with a reclaiming mechanism that modifies the CBS accounting and enforcement rules to take advantage of the bandwidth reserved to inactive servers. Also HGRUB presents the same problem of IRIS when a server reactivates after a blocking time.

A CBS extension to support hierarchical scheduling is presented in [12]. Being based on the original formulation of the CBS algorithm, it suffers from *deadline aging*, and does not implement hard reservations. Moreover, it makes the restrictive assumption of First-Come-First-Served (FCFS) job scheduling within each server.

In [5], Bertogna et al. propose the Bounded-Delay Resource Open Environment (BROE) server, which extends the CBS to handle resource sharing in a Hierarchical Scheduling Framework. To address the budget exhaustion problem when a global lock is held, BROE performs a budget check before granting access to each global resource. If the budget is sufficient to complete the global critical section, the lock is granted. Otherwise, it is denied, suspending the server until a replenishment time. Beside this budget-check mechanism, the bounded-delay version of the CBS implemented by BROE differs from the one adopted for IRIS and HGRUB in the rules to execute after a server reactivation. Even if not explicitly mentioned in [5], these rules allow avoiding a subtle problem upon server reactivation that may lead to a server deadline miss. As this problem is present in different works in the literature, we believe it is important to highlight it, bringing it to the attention of the real-time community.

3. H-CBS rules

In this section, the rules of the Hard Constant Bandwidth Server (H-CBS) are described in detail. Since the rules of CBS-based servers are sometimes expressed using the notion of virtual time, some other times using explicit budget relations, we hereafter provide both formulations. To simplify the notation, the server index is omitted in the server parameters.

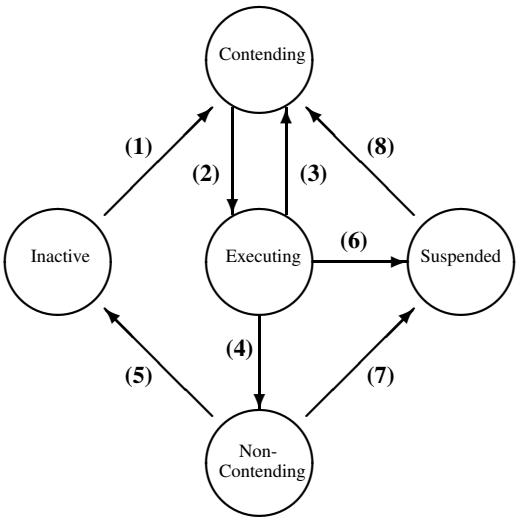

Figure 2: State transition diagram.

3.1. Virtual time based rules

The server is characterized by three dynamic variables, which are updated at runtime:

- a *deadline* d;
- a *virtual time* v;
- a *reactivation time* z.

Moreover, a server S is defined to be *backlogged* if it has any active jobs awaiting execution at that instant, and *non-backlogged* otherwise.

At each time instant t, a server can be in one of five possible states:

1) *Inactive*, when it is non-backlogged and $v \leq t$;
2) *Non-Contending*, when it is non-backlogged and $v > t$;
3) *Contending*, when it is backlogged and eligible to execute;
4) *Executing*, when it is backlogged and currently running;
5) *Suspended*, when it is backlogged and its virtual time has reached the server deadline ($v = d$), but the current time instant is before the reactivation time z of the server ($t < z$).

Note that in the definition of the *Inactive* and *Non-Contending* states, the current time t is compared with the virtual time v. The relation between t and v gives an indication on the possibility for the server to execute without violating its bandwidth α. Intuitively, when $v > t$, the server has executed for all its "fair share"; the opposite holds when $v \leq t$.

The *Suspended* state has been introduced to ensure a hard reservation behavior. Indeed, no analog of the *Suspended* state is present in the original definition of CBS [6].

Figure 2 illustrates the state transition diagram that describes the behavior of the server.

Being t the current time, the server variables are updated according to the following rules.

(i) The server is initially in the *Inactive* state. It transitions to *Contending* state when it wishes to contend for execution. This transition (label **(1)** in Figure 2) is accompanied by the

following actions:

$$\begin{aligned} d &\leftarrow t+P \\ v &\leftarrow t \end{aligned}$$

(ii) Only *Contending* servers are eligible to execute. When the earliest deadline *Contending* server is selected for execution, it undergoes transition **(2)** to the *Executing* state. While executing, its virtual time is incremented at a rate $1/\alpha$:

$$\frac{dv}{dt} = \frac{1}{\alpha}$$

(iii) When an *Executing* server is preempted by a higher priority one, it undergoes transition **(3)** back to the *Contending* state.

(iv) When an *Executing* server has no more pending jobs to execute, it transits to the *Non-Contending* state (transition **(4)**), and remains there as long as $v > t$.

(v) When $v \leq t$, a *Non-Contending* server transitions to the *Inactive* state (transition **(5)**).

(vi) If the virtual time v of an *Executing* server reaches the server deadline d, it undergoes transition **(6)** to the *Suspended* state. This transition is accompanied by the following actions:

$$\begin{aligned} z &\leftarrow v && (1) \\ d &\leftarrow v+P && (2) \end{aligned}$$

(vii) A *Non-Contending* server which desires to contend once again for execution (note $t < v$, otherwise it would be in the *Inactive* state) transits to the *Suspended* state (transition **(7)**). This transition is accompanied by the same actions (Eq. (1) and (2)) of transition **(6)**.

(viii) A *Suspended* server transitions back to the *Contending* state as soon as the current time t reaches the reactivation time z (transition **(8)**).

Note that a server may take two transitions instantaneously one after another. For example, when an *Executing* server becomes non-backlogged and $v \leq t$, transition **(5)** is taken instantaneously after transition **(4)**.

The above rules implement a non-work-conserving H-CBS server. A simple rule can be added to make the server work-conserving:

- When the processor is idle, every server is reset to the *Inactive* state.

This allows implementing a simple reclaiming mechanism that avoids idling the processor when there are backlogged servers.

3.2. Budget based rules

The rules of a H-CBS server can also be expressed in terms of period P and maximum budget Q. At any current time t, the server is characterized by an absolute deadline d and a remaining budget q. When a job executes, q is decreased accordingly.

The budget based rules are summarized below.

1) Initially, $q = 0$ and $d = 0$.
2) When H-CBS 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 server returns active replenishing its budget to Q and setting $d \leftarrow t_r + P$.
   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 $d \leftarrow d + P$.

According to the previous rules, a server running ahead of its guaranteed processor utilization may self-suspend when reactivating after an idle time 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 bandwidth than its allowed share, it will immediately replenish its budget, setting the deadline to $d \leftarrow t + P$.

The connection between this formulation and the one presented in Section 3.1 can be obtained considering the relation

$$v = t_a + \frac{Q-q}{\alpha},$$

which links the virtual time v with the current server budget q and the last server activation time t_a.

3.3. Considerations

The above rules implement a bounded-delay version of the CBS, which provides the corresponding processor share within a service delay of $\Delta = 2(P-Q)$ in any time-interval. The bounded-delay property is guaranteed by using an additional "Suspended" state, which allows the reservation to be "hard", avoiding the deadline aging problem.

It is important to note that there is no direct transition between the *Non-Contending* and the *Contending*/*Executing* states. A server that reactivates after being *Non-Contending* must first pass through a *Suspended* state before being executed again. This is one of the main differences with existing hard reservation formulations of the CBS, that instead allow a *Non-Contending* server to restart executing immediately after a new job request arrives, using the original deadline and the remaining budget. We believe this mechanism to be potentially dangerous in hard reservation scheduling scenarios, where a server might reactivate after being blocked by shared resource policies, as shown in the following section.

Note that a soft CBS that does not meet the bounded-delay property could still be used in hierarchical environments when a particular kind of scheduling algorithm is used. For example, when the scheduler replicates the same job execution order enforced on a dedicated virtual processor (VP) of speed α. In this case, it is possible to prove that the schedulability analysis is simplified, as it is sufficient to check whether each job completes at least Δ time-units before its deadline on the VP. However, note that the job execution order enforced by a given scheduler on a VP may differ from that enforced on a server, because of the different processor availability. Therefore, replicating the VP schedule on the server requires a significant amount of additional

runtime complexity[3].

A corollary of the above observation concerns the First-Come-First-Served (FCFS) policy. Note that, by definition, the job execution order using FCFS is always the same, on a server or a dedicated VP. As mentioned, this significantly simplifies the schedulability analysis, explaining why a soft CBS can be used in [12] for hierarchical environments adopting FCFS as a job scheduling policy.

If other policies are used instead, like EDF or FP, the schedulability on a soft reservation is much more difficult to check, due to the difficulties in finding a critical instant situation, i.e., the job release instance that leads to the worst-case response time of the tasks handled by the server. Indeed, while the critical instant on a dedicated VP is found when all tasks are synchronously released, this is not the case on a soft reservation, where a worse situation is found when a high priority task arrives after a lower priority one caused the deadline-aging problem. Besides imposing a larger schedulability penalty, the latter case makes the analysis much more complex.

4. H-CBS blocking problem

Beside the interference from higher priority instances, each server may experience some blocking due to globally shared resources concurrently accessed by other servers, or due to suspension mechanisms implemented in the system. In this section, we analyze more in detail how the blocking may jeopardize the behavior of classic H-CBS formulations, exposing the system to dangerous deadline misses. Note that the presented problem is different from (and orthogonal to) another blocking-related problem analyzed in many different papers, concerning the budget exhaustion problem of resource sharing servers [5], [13]–[15]. While different techniques are available to limit the blocking due to servers exhausting their budget while holding a global lock, most of these techniques do not solve the problem presented in this section.

As shown in [5], [15], a sufficient schedulability condition to safely compose multiple reservation servers in the presence of a generic blocking term can be derived as follows.

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

$$\forall k = 1,\ldots,N \;:\; \sum_{i:P_i \le P_k} \alpha_i + \frac{B_k}{P_k} \le 1, \tag{3}$$

where B_k represents the maximum blocking that can be imposed over a server S_k.

To clarify the blocking problem with existing H-CBS formulation, we hereafter show the case of blocking due to globally shared resources accessed through SRP-G [5], [13], one of the most popular protocols for arbitrating the access to resources shared by different servers. A similar problem arises also when the blocking is due to other scheduling mechanisms.

3. Note that the replicated schedule is non-work-conserving, as it might leave the server idle even when it has some pending job to execute, in order to enforce the same job execution order of the VP schedule.

4.1. Resource model

Two types of shared resource can be defined:

- *Local resource*[4]: 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 $Z_{i,j}$. From now on, the notation $\{x\}_0$ denotes $\{0\} \cup \{x\}$.

Definition 4: 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 $H_{k,j}(i)$ can be found in [5]).

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

$$H_{k,j} = \max_{\tau_i \in \Gamma_k} \{H_{k,j}(i)\}. \tag{4}$$

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

$$H_k = \max_j \{H_{k,j}\}. \tag{5}$$

In order to access shared resources, the *Stack Resource Policy* (SRP) [16] can be used as it is for local resources, while it has to be extended for global resources. The global version of SRP is summarized below [5], [13].

Global SRP (SRP-G). To handle global resources, each server S_k is assigned a preemption level π_k^S. Server preemption levels are ordered in inverse period order, such that $\pi_h^S > \pi_k^S \Leftrightarrow P_h < P_k$. Each global resource is assigned a global ceiling equal to

$$C_j^G = \max\{\pi_k^S \mid \exists \tau_i \in \Gamma_k \wedge \tau_i \text{ holds global } R_j\}_0.$$

A global system ceiling is defined as

$$\Pi^G = \max_j \{C_j^G\}.$$

A server S_k can preempt the currently scheduled server only if $\pi_k^S > \Pi^G$.

Note that, when a global resource is locked, the system ceiling Π^G is incremented, potentially causing a number of servers to be blocked.

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

$$B_k = \max_{P_\ell > P_k} \{H_{\ell,j} \mid R_j \text{ used by } S_h \wedge P_h \le P_k\}_0. \tag{6}$$

4. Please note that local resources are not defined when $n_k = 1$ (i.e., when the application Γ_k is composed by a single task).

4.2. Problem description

In the literature, different works [8], [11] proposed a formulation of a hard reservation CBS. Unfortunately, as we are going to show, all these formulations present a problem arising whenever a server is blocked upon re-activation. This is for instance the case when servers may share mutually exclusive global resources, and the earliest deadline server is waiting for a global lock to be released by another server. The same problem may however arise also for other kinds of blocking that a server may experience, as with suspension-based mechanisms or system sleep intervals.

To present the problem, we here focus on the case of hard reservation CBS servers that may access globally shared resources using the SRP-G protocol.

Please consider Rule (vii) of the H-CBS server in Section 3.2. In existing formulations of hard reservation CBS [8], this rule is not present, and a *Non-Contending* server wishing to once again contend for execution is allowed to directly transition to the *Contending* state. Equivalently, in the budget-based formulation, Rule (2a) in Section 3.2 is replaced by the following rule:

> Rule (2a)-OLD: *If $t < t_r$, the server maintains its current budget q and deadline d.*

Therefore, a server that reactivates when running ahead of its guaranteed processor share is not suspended, but may immediately contend for execution using the existing budget and deadline.

We hereafter show how such a different rule can affect the server schedulability in presence of blocking.

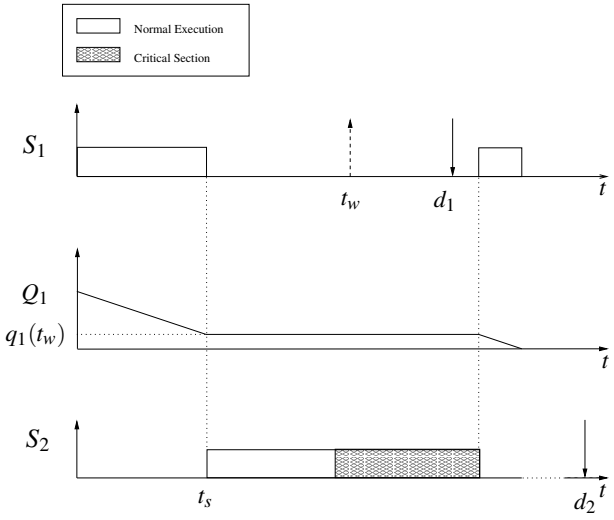

Figure 3: The *unbounded blocking bandwidth* problem.

Q_1	12	Q_2	20
P_1	24	P_2	80
$q_1(t_w)$	3	H_2	10

Table 1: Example values for the *unbounded blocking bandwidth* problem.

Consider two servers S_1 and S_2 sharing a global resource R, whose parameters are reported in Table 1. As shown in Figure 3, both servers are released at the same time instant $t = 0$, with deadlines $d_1 = P_1 = 24$ and $d_2 = P_2 = 80$, respectively. According to the global EDF policy, S_1 starts executing at $t = 0$. At time $t_s = 9$, S_1 has no more pending jobs to execute, so that S_2 may start executing. At some point, S_2 locks the resource R. Then, at time $t_w = 17$, the server S_1 becomes backlogged again. Using the budget-based rules, the replenishment time is computed as $t_r = d_1 - q(t_w)/\alpha_1 = 18$. Since $t_w < t_r$, in our formulation of the H-CBS, the server would be suspended until the reactivation time t_r, when the budget is refilled and the deadline updated. Instead, those approaches based on the original formulation (using Rule (2a)-OLD) do not suspend the server, but allow it to contend for execution with its old budget $q_1(t_w)$ and deadline d_1. However, according to SRP-G, S_1 is prevented from executing, since R is currently locked. As shown in Figure 3, the blocking imposed by S_2 causes the original deadline d_1 to be missed.

In this example, using Theorem 1, it is easy to verify the schedulability of the system. In particular, for $k = 1$, $\alpha_1 + B_1/P_1 = \alpha_1 + H_2/P_1 = 22/24 < 1$, while for $k = 2$, $\alpha_1 + \alpha_2 = 12/24 + 20/80 = 3/4 < 1$. Therefore, despite the schedulability test of Theorem 1 is passed, the above example shows that, not extending the deadline of S_1 upon reactivation, a deadline miss occurs.

4.3. Problem analysis

To understand why the schedulability test of Theorem 1 does not hold for existing hard reservation CBS servers implementations, it is necessary to analyze how the blocking impacts the instances generated by servers re-using the old deadline and budget upon reactivation.

In Theorem 1, the server blocking introduced by SRP-G is considered as in the original analysis for SRP, where non-suspending tasks are addressed. A non-suspending task incurs only *arrival blocking*, i.e., it can only be blocked upon its arrival. Equation (3) considers only this kind of blocking, introducing a blocking bandwidth term of B/P.

In the case depicted in Figure 3, instead, the server behaves as a task that is suspended and reactivated, making the existing analysis unsuitable for such a scenario. When the server is resumed at time t_w, the blocking time B impacts on a time interval of $d_1 - t_w$ units. This leads to a blocking bandwidth of $B/(d_1 - t_w)$, which is greater than the one considered in Equation (3), being $d_1 - t_w < P_1$. More in general, as t_w may potentially be arbitrarily close to the server deadline d_1, the blocking bandwidth may tend to infinity, being

$$\lim_{t_w \to d_1} \frac{B}{d_1 - t_w} = \infty.$$

The original hard reservation CBS implementation are therefore said to suffer from an *unbounded blocking bandwidth* problem.

4.4. Problem solution

The *unbounded blocking bandwidth* problem of existing implementations can be solved by ensuring that when a server reactivates, its deadline will be at least P time-units away. While this is always the case when the server executed for less than its allocated budget (executing transition **(1)** from the *Inactive* state, or, equivalently, Rule 2-b), this is not true for

existing implementations when the server is running ahead of its proportional processor share.

Instead, our H-CBS implementation suspends the server, undergoing transition **(7)** (or, equivalently, applying Rule (2a) in Section 3.2). The transition is accompanied by a full budget replenishment when the reactivation time is reached, postponing the deadline P time units after the reactivation time. Therefore, a bounded blocking bandwidth factor of B/P is maintained. In the example of Figure 3, the server S_1 would be suspended at time t_w, and reactivated at time t_r with a deadline $d_1 = t_r + P_1$ greater than $t_w + P_1$.

Please also note that the server suspension imposed by Rule (2a) is crucial to guarantee a maximum service delay of $\Delta = 2(P-Q)$. Omitting such a rule would lead to a maximum service delay greater than Δ, as shown in the following example.

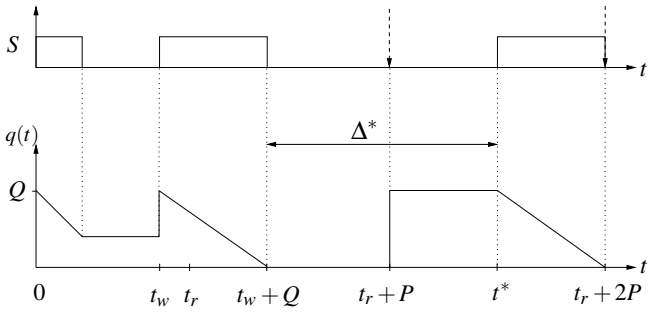

Figure 4: Scenario with a service delay greater than $\Delta = 2(P-Q)$.

Consider the scenario illustrated in Figure 4, where a server S starts executing at time $t = 0$, and then becomes *Non-Contending* when it has no more job to execute. At time t_w, the server becomes backlogged again. Assuming $t_w < t_r$, Rule (2a) is applied, enforcing a full budget replenishment $q = Q$ and shifting the deadline to $d = t_r + P$. Rule (2a) also requires the server be suspended until time t_r. If the server is not suspended, it may immediately start executing at time t_w, as shown in the figure. The server may execute until time $t_w + Q$, when the server exhausts its budget. Applying Rule 3, the server is suspended until its deadline $d = t_r + P$. When $t = d$, the budget is refilled, postponing the deadline to $d = t_r + 2P$. In the worst-case scenario, the server restarts executing at the latest possible time that guarantees its schedulability, that is $t^* = t_r + 2P - Q$. In this case, the maximum service delay Δ^* can be computed as

$$\Delta^* = t^* - (t_w + Q).$$

By replacing t^* in the above equation,

$$\Delta^* = t_r + 2P - Q - (t_w + Q) = (t_r - t_w) + 2(P-Q).$$

Since $\Delta = 2(P-Q)$,

$$\Delta^* = (t_r - t_w) + \Delta.$$

Since Rule (2a) is applied only when $t_r > t_w$, it follows $(t_r - t_w) > 0$, obtaining a service delay Δ^* strictly greater than $2(P-Q)$.

When instead the server is suspended until time t_r, it is easy to see that the worst-case service delay cannot exceed Δ, as illustrated in Figure 5.

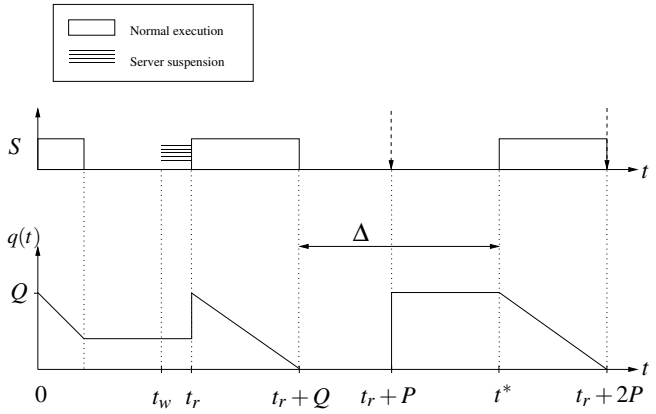

Figure 5: Scenario with a service delay equal to $\Delta = 2(P-Q)$.

The blocking bandwidth is therefore upper bounded by B/P, and the schedulability test of Theorem 1 can be efficiently used. This explains why Rule (2a) requires the server to suspend.

4.5. Impact of the problem

This section presents an overview of previously proposed CBS-based approaches that are affected by the described problem in presence of blocking.

The IRIS algorithm was originally presented in [8] for isolating single tasks, and is traditionally seen as a reference implementation of a hard reservation CBS. It was the first work to highlight the shortcomings of the original CBS, proposing a solution for the *deadline aging* problem. However, IRIS suffers from the *unbounded blocking bandwidth* problem presented in this paper when it is adopted for servers that may experience blocking conditions. In these scenarios, IRIS formulation needs to be revised introducing a suspension mechanism equivalent to the one of our H-CBS implementation. In particular, enhancing IRIS with Rule (2a) allows avoiding the unbounded blocking bandwidth problem.

A similar hard CBS formulation suffering from the same problem has been proposed by Mancina et al. in [17], implementing it in the context of Minix 3, a highly dependable microkernel-based OS.

In a hierarchical scheduling context, a hard reservation CBS server has been used by Lipari and Bini in [2], [18], as a generic extension of the classic CBS [6]. Although the purpose was to propose a generic server for a hierarchical scheduling system, the formulation of such a server reflects very closely the one of IRIS, and is thus affected by the same issue.

In [19], Kumar et al. proposed an algorithm called R-CBS, to provide an online reconfiguration mechanism for the CBS server, both in its *soft* and *hard* versions. The hard formulation proposed in the paper is also prone to the *unbounded blocking bandwidth* problem.

Various servers has been specifically conceived to enhance hard CBS implementations with different kinds of mechanisms for the reclaiming of unused bandwidth, like HGRUB [11] and SHRUB [20]. Since they allow a server that is running ahead of its reserved bandwidth to reactivate with the existing budget

and deadline, they are all affected by the *unbounded blocking bandwidth* problem.

As the presented problem arises in presence of blocking, it may affect many existing works dealing with resource sharing in server-based environments. When servers may share global resources, a well-known problem concerns the possible budget exhaustion inside a critical section, significantly increasing the blocking over higher priority servers. Different papers have been proposed to deal with this budget exhaustion problem [5], [13]–[15], [21], which, as mentioned, is orthogonal to the *unbounded blocking bandwidth* problem highlighted in this paper. We hereafter recall the principal ones.

One of the first solutions is based on budget overrun, i.e., when a server exhausts its budget while holding a global resource, it is allowed to consume extra budget until the end of the critical section. This solution was first proposed by Ghazalie and Baker in [22], and then used by Abeni and Buttazzo in [6] under the CBS algorithm. Later, Davis and Burns [13] proposed two versions of this mechanism: *overrun with payback*, where the next budget replenishment is decreased by the overrun value, and *overrun without payback*, where no action is taken after the overrun. Behnam et al. [14] extended this mechanism under EDF.

Another solution has been proposed with the SIRAP protocol in [15], introducing a budget check before each global lock, so that a server is granted access to a global resource only if its budget is sufficient to complete the critical section. Otherwise, the server must wait until the next budget replenishment.

A similar budget check mechanism has been proposed with the BROE protocol in [5], using a smarter budget replenishment mechanism. Whenever the budget is not sufficient to complete the critical section, a full budget replenishment is planned at the earliest possible time that preserves both the server bandwidth and the maximum service delay, suspending the server until the next budget replenishment. Notably, BROE is the only server we found in the literature that is not affected by the *unbounded blocking bandwidth* problem, as it suspends a server upon reactivation when it is running ahead of its proportional share.

In all other cases, the *unbounded blocking bandwidth* problem may appear every time a hard reservation CBS is used without the suspension enforced by Rule (2a). Since all mentioned server-level resource sharing protocols do not prevent a server to block, a situation similar to the one shown in Figure 3 will always be possible.

For the same reason, the problem also arises whenever global contention is arbitrated using other protocols, such as server-level versions of the Priority Ceiling Protocol (PCP) or Priority Inheritance Protocol (PIP) [23]. Instead, the Bandwidth Inheritance Protocol (BWI), proposed by Lamastra et al. in [24] as an extension of PIP for the CBS algorithm, is not affected by the same issue. In particular, BWI is able to guarantee the server schedulability without introducing any blocking bandwidth factor in the feasibility test. This interesting property however is achieved by significantly increasing the pessimism in the schedulability test of each reservation server.

5. Conclusions

This paper presented a comprehensive formalization for the Hard Constant Bandwidth Server (H-CBS), a variant of the CBS algorithm which guarantees a hard-reservation scheme for CPU allocation. In the literature, previous hard reservation formulations for the CBS have been proposed; however, they are affected by the *unbounded blocking bandwidth* effect, which can lead to dangerous deadline misses. Specifically, we highlighted how such inconsistent formulations can impact the system schedulability when resource sharing is considered. More in general, the identified issue is not strictly related to resource sharing, but it may emerge whenever a hard reservation CBS is integrated with other scheduling mechanisms, such as suspension-based protocols or system sleep intervals, which require the reservation server to block. This work gives, once and for all, a consistent formulation for the H-CBS, leaving it as a reference for the real-time research community.

Acknowledgments

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

References

[1] C. W. Mercer, S. Savage, and H. Tokuda, "Processor capacity reserves for multimedia operating systems," in *Proceedings of IEEE international conference on Multimedia Computing and System*, May 1994.

[2] 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.

[3] 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.

[4] Z. Deng and J. W.-S. Liu, "Scheduling real-time applications in an Open environment," in *Proceedings of the 18th IEEE Real-Time Systems Symposium (RTSS 1997)*, San Francisco, CA, USA, December 3-5, 1997, pp. 308–319.

[5] 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.

[6] 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 2-4 1998.

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

[8] 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 25-28 2004.

[9] AUTOSAR, *AUTOSAR Release 4.1, Specification of Operating System*, http://www.autosar.org, 2013.

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

[11] L. Abeni, L. Palopoli, C. Scordino, and G. Lipari, "Resource reservations for general purpose applications," *IEEE Transactions on Industrial Informatics*, vol. 5, no. 1, pp. 12–21, February 2009.

[12] G. Lipari and S. Baruah, "A hierarchical extension to the constant bandwidth server framework," in *Proceedings of the 7th Real-Time Technology and Application Symposium*, Taipei, Taiwan, 30 May - 1 June 2001, pp. 26–35.

[13] 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.
[14] M. Behnam, I. Shin, T. Nolte, and M. Nolin, "Scheduling of semi-independent real-time components: Overrun methods and resource holding times," in *Proceedings of 13th IEEE Int. Conference on Emerging Technologies and Factory Automation (ETFA'08)*, Hamburg, Germany, September 15-18, 2008.
[15] ——, "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 1-3, 2007.
[16] T. P. Baker, "Stack-based scheduling for realtime processes," *Real-Time Systems*, vol. 3, no. 1, pp. 67–99, April 1991.
[17] A. Mancina, D. Faggioli, G. Lipari, J. N. Herder, B. Gras, and A. S. Tanenbaum, "Enhancing a dependable multiserver operating system with temporal protection via resource reservations," *Real-Time Systems*, vol. 43, no. 2, pp. 177–210, 2009.
[18] G. Lipari and E. Bini, "Resource partitioning among real-time applications," in *Proceedings of the 15th Euromicro Conference on Real-Time Systems*. Porto, Portugal: IEEE, 2003, pp. 151–158.
[19] P. Kumar, N. Stoimenov, and L. Thiele, "An algorithm for online reconfiguration of resource reservations for hard real-time systems," in *Proc. of the 24th Euromicro Conference on Real-Time Systems*, Pisa, Italy, July 11-13 2012, pp. 245–254.
[20] T. Cucinotta, L. Abeni, L. Palopoli, and G. Lipari, "A robust mechanism for adaptive scheduling of multimedia applications," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 10, no. 46, pp. 1–24, November 2011.
[21] M. Åsberg, T. Nolte, and M. Behnam, "Resource sharing using the rollback mechanism in hierarchically scheduled real-time open systems," in *Proc. of the 19th Real-Time and Embedded Technology and Applications Symposium*, Philadelphia, PA, USA, April 9-11 2013, pp. 129–140.
[22] T. Ghazalie and T. Baker, "Aperiodic servers in a deadline scheduling environment," *Real-Time Systems*, vol. 9, pp. 31–67, 1995.
[23] L. Sha, R. Rajkumar, and J. Lehoczky, "Priority inheritance protocols: An approach to real-time synchronization," *IEEE Transactions on Computers*, vol. 39, no. 9, pp. 1175–1185, September 1990.
[24] G. Lamastra, G. Lipari, and L. Abeni, "A bandwidth inheritance algorithm for real-time task synchronization in open systems," in *Proc. of the 22nd IEEE Real-Time Systems Symposium*, London, UK, December 3-6 2001, pp. 151–160.