Constant Bandwidth Servers with Constrained Deadlines*

Daniel Casini
Scuola Superiore S. Anna
Pisa, Italy
daniel.casini@santannapisa.it

Luca Abeni
Scuola Superiore S. Anna
Pisa, Italy
luca.abeni@santannapisa.it

Alessandro Biondi
Scuola Superiore S. Anna
Pisa, Italy
alessandro.biondi@santannapisa.it

Tommaso Cucinotta
Scuola Superiore S. Anna
Pisa, Italy
tommaso.cucinotta@santannapisa.it

Giorgio Buttazzo
Scuola Superiore S. Anna
Pisa, Italy
giorgio.buttazzo@santannapisa.it

ABSTRACT

The *Hard Constant Bandwidth Server* (H-CBS) is a reservation-based scheduling algorithm often used to mix hard and soft real-time tasks on the same system. A number of variants of the H-CBS algorithm have been proposed in the last years, but all of them have been conceived for implicit server deadlines (i.e., equal to the server period). However, recent promising results on semi-partitioned scheduling together with the demand for new functionality claimed by the Linux community, urge the need for a reservation algorithm that is able to work with constrained deadlines. This paper presents three novel H-CBS algorithms that support constrained deadlines. The three algorithms are formally analyzed, and their performance are compared through an extensive set of simulations.

CCS CONCEPTS

• **Computer systems organization** → **Real-time operating systems**; **Embedded software**;

KEYWORDS

Real-time scheduling, resource reservations, temporal isolation

ACM Reference format:
Daniel Casini, Luca Abeni, Alessandro Biondi, Tommaso Cucinotta, and Giorgio Buttazzo. 2017. Constant Bandwidth Servers with Constrained Deadlines. In *Proceedings of International Conference on Real-Time Networks and Systems, Grenoble, France, October 2017 (RTNS'17),* 10 pages.
https://doi.org/10.475/123_4

1 INTRODUCTION

Several real-time systems are characterized by multiple computational activities (tasks) that require to be *temporally isolated* among themselves, meaning that the temporal behavior of a task is not affected by the misbehavior of some other task, for instance due to execution overruns. This is the case of open environments [24], where independently developed software components need to be executed in isolation. An effective approach for achieving temporal isolation is the *resource reservation* mechanism [21], according to which each task is assigned a fraction of the total processor capacity (called *reservation bandwidth*). In this way, each task running in a reservation with bandwidth $\alpha \leq 1$ behaves (from a temporal point of view) as if it were executing alone on a virtual processor with speed α times the one of the physical processor, independently of the behavior of the tasks running in other reservations.

Resource reservation is especially useful to execute hard real-time tasks together with soft real-time tasks characterized by execution times subject to high variations (and for which it is difficult to find an upper bound) [9]. Resource reservation is also adopted to implement *hierarchical scheduling* [6, 24], where each reservation can handle one or more tasks and schedule them according to a local scheduling policy.

A reservation is typically implemented through a **reservation server**, which is a kernel mechanism that manages the reservation bandwidth by allocating a time budget Q every period P for the execution of the served tasks. Several reservation servers have been proposed in the real-time literature using different policies for managing the budget, both under fixed-priority schedulers and the Earliest Deadline First (EDF) algorithm [11]. Existing deadline-based servers have not been designed to manage constrained deadlines. However, recent results on semi-partitioned scheduling [8, 13] highlighted the need for ad-hoc solutions considering reservation servers having deadlines lower than their reservation periods. In fact, in these works, whenever a reservation cannot be allocated in any processor, it is split into multiple chunks using the C=D splitting scheme [10]. According to this scheme the resulting chunks have always a constrained deadline, thus implicitly requiring reservation servers with a relative deadline shorter than their period. The need for a theoretically sound framework for deadline-constrained reservations has also recently come out[1] in the context of on-going developments of the `SCHED_DEADLINE` scheduling class of the Linux kernel [19].

Contributions. This paper proposes three novel extensions to the popular **Hard Constant Bandwidth Server** (H-CBS) [7], in order to cope with constrained deadlines: the first one allows for ensuring the same worst-case guarantees as a sporadic real-time task that is configured with the same parameters of the server; the second and third ones build upon a sufficient schedulability test to improve the server performance in terms of average-case and

*This work has been partially supported by the RETINA Eurostars Project E10171.

RTNS'17, October 2017, Grenoble, France

ACM ISBN 123-4567-24-567/08/06…$15.00
https://doi.org/10.475/123_4

[1] https://lkml.org/lkml/2017/2/10/611

probabilistic metrics, thus being more suitable for soft real-time systems. The presented approaches are compared both in terms of analytical properties and empirical performance by simulation.

Paper Structure. The paper is organized as follows. Section 2 introduces the system model, the adopted notation, and reviews the needed background. Section 3 presents a general formulation to describe different variants of the H-CBS algorithm. Section 4 highlights shortcomings of existing state-of-the-art reservation algorithms when dealing with constrained deadlines. Section 5 presents the novel reservation algorithms extending the H-CBS to work with constrained deadlines. Section 6 presents some simulation results for validating the proposed approach. Finally, Section 7 draws the conclusions and illustrates possible future work on the topic.

2 SYSTEM MODEL AND BACKGROUND

This paper considers an arbitrary number of virtual processors running into a partitioned multiprocessor system. Each virtual processor is implemented by a reservation server $r_i = (Q_i, D_i, P_i)$ characterized by a budget Q_i, a relative deadline D_i, and a period P_i. Each reservation server has a bandwidth $\alpha_i = \frac{Q_i}{P_i}$ and a constrained deadline less than or equal to its period ($D_i \leq P_i$). If $\mathcal{R}_k$ is the set of reservations running on the k-th physical processor, its total utilization is $U_k = \sum_{r_i \in \mathcal{R}_k} \alpha_i$.

The workload running into a reservation r_i consists of a set Γ_i of sporadic tasks scheduled according to a *local* scheduling policy, where each task τ_j^i is characterized by a worst-case execution time C_j^i, a relative deadline D_j^i, and a period T_j^i. For the sake of simplicity, the superscript in task parameters denoting the reservation will be omitted when referring to a generic reservation or it is clear from the context. The server index is also removed whenever a single reservation server is considered. Reservation servers are assumed to be scheduled according to preemptive EDF and tasks are assumed to exchange data through asynchronous (i.e., non blocking) communication mechanisms.

In this paper, the proposed formulations for handling servers with constrained deadline are based on the H-CBS algorithm.

2.1 Background

A reservation server, although guaranteeing a desired bandwidth α_i, may introduce variable execution delays on the served tasks due to the fact that, once the budget is exhausted, the server's tasks will not be scheduled until the next replenishment time. The *worst-case service delay* Δ_i depends on the server parameters and the used budget management policy. Having a *bounded* worst-case service delay is a fundamental property for guaranteeing the schedulability of the workload executing inside the server [22]. In this work, a server is said to implement a *hard* reservation if it is able to guarantee *both* a bandwidth α_i and a worst-case (bounded) service delay Δ_i. [2]

2.1.1 Server Schedulability. The schedulability of a constrained-deadline reservation server can be verified through the Processor Demand Criterion (PDC) [3]. The PDC builds on the concept of *demand bound function* (dbf), which in this case represents the worst-case computational demand of a reservation in any interval $[0, t]$. When the demand of a server does not exceed the one of a sporadic task with the same parameters, its dbf becomes:

$$dbf_i(t) = \left\lfloor \frac{t + P_i - D_i}{P_i} \right\rfloor Q_i, \tag{1}$$

The processor demand criterion is recalled in Theorem 2.1.

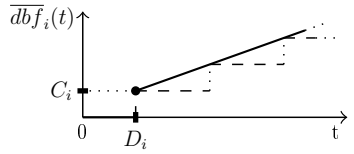

Figure 1: Approximate demand bound function of a sporadic task

Theorem 2.1 (Processor Demand Criterion (From [3])). *A set $\mathcal{R}$ of constrained-deadline reservations, whose demand does not exceed the one of a sporadic task with the same parameters, is EDF-schedulable iff*

$$\forall t \in \mathcal{D}, \sum_{\tau_i \in \mathcal{R}} dbf_i(t) \leq t$$

with $\mathcal{D} = \bigcup_{r_i \in \mathcal{R}} \{t = D_i + fT_i : t \leq L^* \wedge f \in \mathcal{N}_{\geq 0}\}$, where L^* is the maximum analysis interval (see [3, 25] for more details about L^*)).

The PDC has a pseudo-polynomial computational complexity if $U < 1$ (exponential if $U = 1$). Sufficient polynomial-time schedulability tests for constrained-deadline reservations can be derived by exploiting approximate demand bound functions as defined by Fisher et al. [18]. Such approximate demand bound functions with a single discontinuity (see Figure 1) are defined as follows:

$$\overline{dbf}_i(t) = \begin{cases} Q_i + \alpha_i(t - D_i) & \text{if } t \geq D_i, \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

A sufficient test exploiting the definition of $\overline{dbf}_i(t)$ is reported below.

Theorem 2.2. *A reservation set $\mathcal{R}_k$ of periodic constrained-deadline reservations with $U \leq 1$ is EDF-schedulable if*

$$\forall r_i \in \mathcal{R}_k, \ \alpha_i^* + \sum_{r_j \in \mathcal{R} : j \neq i \wedge D_j \leq D_i} \alpha_j \leq 1 \tag{3}$$

with $\alpha_i^* = \frac{Q_i^*}{D_i}$, and

$$Q_i^* = Q_i + \sum_{r_j \in \mathcal{R}_k : j \neq i \wedge D_j \leq D_i} \alpha_j (P_j - D_j)$$

Proof. From Theorem 2.1, if $\forall t \geq 0 \ \sum_{r_i \in \mathcal{R}_k} dbf_i(t) \leq t$ holds, then the system is schedulable. By definition, $\overline{dbf}_i(t) \geq dbf_i(t)$. Hence, if $\forall t \geq 0 \ \sum_{\tau_i \in \mathcal{R}_k} \overline{dbf}_i(t) \leq t$ holds, then the system is schedulable. The function $\overline{dbf}_i(t)$ is characterized by a discontinuity at $t = D_i$ and by a linear part with slope α_i for $t > D_i$. Then, the function $\overline{dbf}(t) = \sum_{r_i \in \mathcal{R}_k} \overline{dbf}_i(t)$ is constituted by a sequence of $n = |\mathcal{R}|$ discontinuities. Let $\delta_1, \ldots, \delta_n$ be the ordered sequence of discontinuities. Since in any interval $[\delta_i, \delta_{i+1}]$ the function $\overline{dbf}(t)$ is either constant or linear (with a slope less or equal

[2] Note that this definition differs to the one proposed by Rajkumar et al. [23].

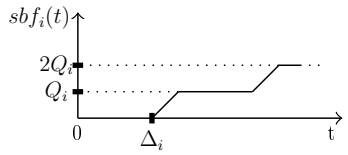

Figure 2: Supply bound function of a generic periodic reservation server

than U, with $U \leq 1$ by assumption) in both cases it is sufficient to check the condition $dbf(\delta_i) \leq \delta_i$. Hence, the following test can be considered:

$$\forall r_i \in \mathcal{R}_k, \sum_{r_j \in \mathcal{R}_k} \overline{dbf}_j(D_i) \leq D_i.$$

Then, substituting the first branch of Equation 2 the test becomes:

$$\forall r_i \in \mathcal{R}_k, \sum_{r_j \in \mathcal{R}_k : D_j \leq D_i} Q_j + (D_i - D_j)\frac{Q_j}{P_j} \leq D_i.$$

Finally, applying some algebraic transformations the condition can be rewritten as:

$$\forall r_i \in \mathcal{R}_k, \ \alpha_i^* + \sum_{r_j \in \mathcal{R}_k : j \neq i \wedge D_j \leq D_i} \alpha_j \leq 1$$

The theorem follows. □

2.1.2 Local schedulability. Guaranteeing the schedulability of the workload running inside a server requires the definition of the *supply bound function* (sbf). The sbf (shown in Figure 2) models the minimum amount of time available in a reservation r_i in every time interval of length t and can be derived by identifying the minimum time allocated to r_i in the worst-case scenario. The interested reader can refer to [24] for further details.

3 HARD CONSTANT BANDWIDTH SERVERS

Across the last two decades of literature on reservation algorithms, various incarnations of the H-CBS algorithm have been proposed by different authors. To shed the light on the existing proposals and provide a common basis for presenting the results of this work, this section presents a generalized formulation of the H-CBS. The formulation is based on a set of static rules and another set of *meta rules*. Such meta rules, which allow differentiating between the various versions, are then discussed to instantiate the existing variants of the H-CBS, as well as the novel ones proposed in Section 5 to cope with constrained deadlines.

A served workload is said to be *pending* from the time it is released (i.e., entering an empty service queue) to the time at which it completes its execution (i.e., leaving the service queue empty). Furthermore, the workload is said to be *ready* when it is pending and can make progress, that is, it is not suspended waiting for some event. The server is assumed to keep track of the state of the workload (pending or ready). As long as it has ready workload to execute, the server is agnostic with respect to the execution requests of the workload, which are managed according to an internal (server-independent) policy, such as *first-in-first-out* (FIFO) or EDF.

A generalized formulation for the H-CBS. A H-CBS server is described by three parameters (Q, D, P), where Q is the *maximum server budget*, D is the server relative deadline, and P is the *server period* (also named as the *reservation period*). The server works by tracking two dynamic state variables: the *current budget* q (also known as *runtime*) and the *scheduling deadline* d (also known as *server absolute deadline*). At any point in time, the server can be in one of the following states: `IDLE`, `READY`, and `THROTTLED`. The H-CBS algorithm relies on EDF scheduling and is subject to the following rules:

- **R1** Initially the server is in the `IDLE` state, where the budget and the scheduling deadline are initialized to $(q, d) = (0, 0)$.
- **R2** When the server starts having ready workload to execute at time t:
  - If the server is in the `IDLE` state, a new budget and scheduling deadline are generated as $(q, d) = \texttt{generate}(q, d, t)$. Then, the server will transit to the `READY` state at time $t_w = \texttt{wakeup}(q, d, t)$.
- **R3** At any point in time, the server in the `READY` state that has the earliest scheduling deadline d is selected for execution.
- **R4** Whenever the server executes the workload for δ time units, its budget is decreased as $q = q - \delta$. Furthermore, the server notifies the consumed budget to the system by means of the function $\texttt{account}(\delta, d)$.
- **R5** When the budget exhausts (i.e., $q = 0$), the server transits to the `THROTTLED` state.
- **R6** While a server is in the `THROTTLED` state, at time $p = d + P - D$ (end of the current reservation period) the server budget is recharged to $q = Q$ and a new scheduling deadline is assigned as $d = d + P$. Still at time p, if the server has ready workload to execute, then it transits to the `READY` state. Otherwise, the server transits to the `IDLE` state.
- **R7** Whenever the server is in the `READY` state and stops having ready workload to execute, it transits to the `IDLE` state.

3.1 The classic H-CBS

Among all the proposals, the most common variant of the H-CBS was first proposed by Marzario et al. [20] in 2004, and later reformulated by Abeni et al. [2]. Such an algorithm was designed for implicit deadlines only and can be expressed by defining $\texttt{generate}(q, d, t)$ as follows:

$$\texttt{generate}(q, d, t) = \begin{cases} (q, d) & \text{if } t < d - \frac{q}{\alpha}, \\ (Q, t + P) & \text{otherwise.} \end{cases} \quad (4)$$

Also, $\texttt{wakeup}(q, d, t) = t$ and $\texttt{account}(\delta, d)$ has no effect.

In 2014, Biondi et al. [7] showed that this formulation is affected by a schedulability issue whenever the H-CBS is adopted in conjunction to other scheduling mechanisms that can introduce blocking times (e.g., non-preemptive sections, locking protocols, etc.), thus perturbing the standard EDF scheduling of the servers. The authors also proposed a solution to the identified issue, which can be expressed by defining $\texttt{wakeup}(q, d, t) = min(t, d - q/\alpha)$ and

$$\texttt{generate}(q, d, t) = \begin{cases} (Q, d - \frac{q}{\alpha} + P) & \text{if } t < d - \frac{q}{\alpha}, \\ (Q, t + P) & \text{otherwise.} \end{cases} \quad (5)$$

As it can be noted by looking at the differences between (4) and (5), the latter formulation always recharges the budget at the maximum value Q. The difference in the $\texttt{wakeup}(q, d, t)$ function was provided to guarantee a bounded service delay of $2(P - Q)$.

3.2 The revised H-CBS

While the traditional H-CBS algorithm tries to re-use the current budget (and generates a new scheduling deadline when the current budget is not usable), the "revised" H-CBS [1] (again, defined for $D = P$), instead, tries to re-use the current scheduling deadline as much as possible (at the cost of decreasing the budget) and is based on the following definition of the `generate()` function:

$$\texttt{generate}(q, d, t) = (\max\{q, \frac{Q}{P}(d - t)\}, d) \tag{6}$$

Moreover, $\texttt{wakeup}(q, d, t) = t$ and $\texttt{account}(\delta, d)$ has no effect.

3.3 The H-CBS-SO algorithm

The self-suspending task model [14] has been introduced to capture possible self-suspensions of the execution (waiting for some event) that are explicitly caused by the task behavior, i.e., not imposed by the scheduler or other scheduling mechanisms. Representative applications of such a model are in the context of semaphore-based locking protocols or in the use of hardware accelerators. Notably, self-suspending tasks received a lot of attention in the last years, being the corresponding schedulability analysis challenging and affected by several flaws in the literature [15]. A simple approach to analyze self-suspending tasks is the *suspension-oblivious* analysis, where suspension times are pessimistically accounted as execution times for the purpose of checking the system schedulability.

In 2015, Biondi et al. [4] showed that the suspension-oblivious analysis is not compatible with the standard H-CBS algorithm. To reconcile the H-CBS algorithm with a simple analysis for self-suspending tasks, the same authors proposed a variant of the H-CBS denoted as H-CBS-SO (H-CBS for suspension-oblivious analysis).

The H-CBS-SO maintains a logical queue, denoted as SS-QUEUE, that keeps track of the servers that have pending *but not* ready workload. The SS-QUEUE follows the EDF ordering. The key difference with respect to the standard H-CBS (both the formulations presented in Section 3.1 are compatible) lies in the account(δ, d) function, which in the H-CBS-SO is defined as:

$$\texttt{account}(\delta, d) = \begin{cases} q_{SS} = q_{SS} - \delta & \text{if } d_{SS} \leq d, \\ \text{no effect} & \text{otherwise;} \end{cases} \tag{7}$$

where q_{SS} and d_{SS} denote the budget and the deadline of the server at the head of the SS-QUEUE, respectively. The servers into the SS-QUEUE are subject to rules R5 and R6 described in Section 3, with the only difference that in rule R6 the server does not transits to the `READY` state (note that, according to rule R7, a server is in the `IDLE` state as long as it is into the SS-QUEUE). Furthermore, whenever a server leaves the SS-QUEUE, rule R1 is not triggered.

4 PROBLEM DEFINITION

The main objective of this work is the development of a reservation server with the following features: (i) providing the budget to the server load within a relative deadline less than or equal to the server period; (ii) guaranteeing at least a budget Q every period P; (iii) guaranteeing a bounded worst-case service delay Δ; and (iv) guaranteeing the schedulability of a set of such servers. Note that such features cannot be achieved by using the traditional formulations of the Hard Constant Bandwidth Server, because their *meta rules*

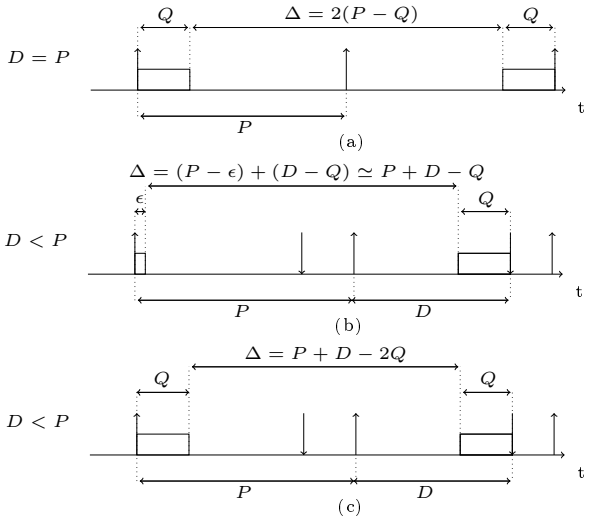

Figure 3: Worst-case delays considering: (a) the original H-CBS (D=P); (b) a solution which discards the budget whenever the server becomes `IDLE`. Inset (c) shows the desirable worst-case delay with constrained deadlines.

(e.g., see Equation 4) leverage utilization-based EDF schedulability theory, which is valid only for implicit deadlines.

The instantiation of such meta-rules is obtained manipulating the condition $U \leq 1$ which is necessary and sufficient to guarantee schedulability when dealing with implicit deadlines. For this reason, if any server simply executes according to its bandwidth the system is implicitly schedulable. A naive solution that avoids leveraging any specific schedulability test could consider to deplete the whole budget whenever a reservation server remains without ready workload. According to this approach, $\texttt{wakeup}(q, d, t) = t$ and $\texttt{account}(\delta, d)$ has no effect. Also, $\texttt{generate}(q, d, t)$ is defined as:

$$\texttt{generate}(q, d, t) = \begin{cases} (0, d) & \text{if } t < d - D + P, \\ (Q, t + D) & \text{otherwise.} \end{cases} \tag{8}$$

In doing so, however, the worst-case service delay would be much higher than the one that would be experienced with a traditional H-CBS (shown in Figure 3, inset (a)). Inset (b) of Figure 3 illustrates the worst-case service delay of such a naive solution. The desirable worst-case service delay of a constrained-deadline server is shown in Figure 3, inset (c). A second naive solution could consist in setting the period of a classical H-CBS equal to the relative deadline D thus leveraging the so called density-based schedulability test. For instance, Equation 4 could be reused considering D in place of P and the ratio $\frac{Q}{D}$ in place of α. Unfortunately, the condition $\sum_{r_i \in \mathcal{R}} \frac{Q_i}{D_i} \leq 1$ is only sufficient to guarantee servers schedulability and highly pessimistic. In particular, when using a constrained-deadline server in the context of C=D Semi-Partitioned scheduling, a single zero-laxity reservation would occupy a whole processor!

The next section proposes new solutions for reservation servers that can be used in this context.

5 PROPOSED SOLUTIONS

This section presents three solutions for realizing a H-CBS server with constrained deadlines. The first one is based on the H-CBS-SO algorithm presented in Section 3.3, and ensures the server schedulability with any safe schedulability test for EDF with constrained deadlines. The second one explicitly builds upon the sufficient EDF schedulability test of Theorem 2.2 and its objective is to improve the server performance with respect to the first solution, in terms of probabilistic performance metrics (such as deadline-miss ratio of soft real-time workload, response time percentiles, etc.) and average-case performance (such as average throughput). Finally, an improved variant of the second solution is also presented, which adopts the same rationale of the revised H-CBS presented in Section 3.2. Finally, differences among the proposed solutions are discussed.

5.1 A solution based on H-CBS-SO

As illustrated in Section 4, the key issue with the existing H-CBS formulations concerns the case in which a server is suspended due to the lack of ready workload to execute. A simple insight can be leveraged to overcome such an issue: since the H-CBS-SO was conceived to deal with self-suspending tasks running upon the server, a safe behavior can be achieved by treating any server suspension in the same manner.

The resulting algorithm is named as H-CBS^D-W and is defined as follows. First, a logical EDF-ordered queue, named S-QUEUE, is provided. Whenever a server transits to the `IDLE` state, then it is inserted into the S-QUEUE. The same definition of the account(δ, d) function reported in Equation (7) is adopted, where in this case q_{SS} and d_{SS} are respectively the budget and the deadline of the server at the head of the S-QUEUE. Analogously to the H-CBS-SO, the servers into the S-QUEUE are subject to rule R5. Also, when rule R5 applies to a server, the latter is removed from the S-QUEUE.

Finally, the following definitions are adopted to instantiate the meta-rules introduced in Section 3 for a server r_i:

$$\texttt{wakeup}(q, d, t) = t \tag{9}$$

and

$$\texttt{generate}(q, d, t) = \begin{cases} (q, d) & \text{if } r_i \text{ is into the S-QUEUE,} \\ (Q_i, t + D_i) & \text{otherwise.} \end{cases} \tag{10}$$

The following lemma shows that this approach makes the H-CBS^D-W compatible with any EDF schedulability test for constrained-deadline sporadic tasks.

Lemma 5.1. *If the H-CBS^D-W is adopted, the processor demand of a reservation $r_i \in \mathcal{R}_k$ never exceeds the one of a constrained-deadline sporadic task with worst-case execution time Q_i, minimum inter-arrival time P_i and relative deadline D_i.*

Proof. As long as a reservation has ready workload, rules R5 and R6 guarantee that a server r_i (i) does not execute for more than Q_i time units every period P_i, and (ii) will always have a relative deadline equal to D_i. Whenever a server stops having ready workload, according to the definition of the H-CBS^D-W, the server is inserted into the S-QUEUE, where it is still subject to rule R5, so preventing r_i to consume more than Q_i budget units within its current period. Such rules take effect because the budget of r_i is decremented by the account(δ, d) function whenever a sporadic task with the same parameters of r_i would have executed according to EDF (i.e., when r_i is at the head to the S-QUEUE and its deadline is shorter than all the ones of the servers that are in the `READY` state). By the definition of the generate(q, d, t) function, the server parameters do not change if r_i restarts having ready workload when it is into the S-QUEUE. Finally, consider the case in which r_i is not into the S-QUEUE and restarts having ready workload at time t. Let p be the end of the r_i's reservation period in which r_i stopped having ready workload (entering the S-QUEUE). If $t \geq p$, then according to Equation (10) the server restarts executing with budget Q_i and deadline $t + D_i$: note that this behavior is compatible with a sporadic arrival of a task configured with the same parameters of r_i. Otherwise, if $t < p$, then r_i must be in the `THROTTLED` state and hence cannot execute until time p, as a corresponding sporadic task would do. □

The following lemma bounds the maximum service delay of a `H-CBS`^{D}`-W` server.

Lemma 5.2. *Consider a set $\mathcal{R}_k$ of H-CBS^D-W reservations that are EDF-schedulable. The service delay of each server $r_i \in \mathcal{R}_k$ is no larger than $\Delta_i = P_i + D_i - 2Q_i$.*

Proof. Consider workload released at time t to be executed upon a server r_i. The service delay is maximized in the scenarios where (i) r_i has no available budget at time t; and (ii) the following periodic instance of the server, beginning at time p, starts providing service to the workload as late as possible without violating the server schedulability, i.e., at time $p + D_i - Q_i$. Among such scenarios, the worst-case service delay occurs when the distance between times t and $p + D_i - Q_i$ is maximal, which happens when (i) the first instance (beginning at time $p - P_i$) depletes all its budget as soon as possible and (ii) time t coincides with the earliest time in which such a budget is depleted. Following the definition of the H-CBS^D-W, the budget is consumed *only* when the server provides service or by means of the account(δ, d) function (see Equation (7)). In both the cases, the event discussed in (ii) cannot happen before time $t = p - P_i + Q_i$. The corresponding delay results equal to $(p + D_i - Q_i) - (p - P_i + Q_i) = D_i + P_i - 2Q_i$ and is illustrated in Figure 3-c. □

The above two lemmas imply that the H-CBS^D-W algorithm **(i)** guarantees the highest worst-case schedulability performance, as the server behaves no worse than a sporadic task, and **(ii)** provides the shortest possible worst-case service delay, as identified in Section 4. However, these features come at the cost of consuming the server budget even when the server is not actually executing (note that rule R4 may perform a double budget accounting by means of the account(δ, d) function). This drawback leaves room for improvement in terms of average-case performance and other metrics that are not related to the worst-case server behavior.

A pragmatic improvement. A possible way to get rid of the double budget accounting introduced by the H-CBS^D-W algorithm may consist in adopting a mechanism similar to the one proposed in the CASH [12] algorithm. CASH was designed for reclaiming unused budget at run-time and relies on an EDF-ordered queue of

servers with spare budget. For instance, the CASH mechanisms can be integrated with the H-CBS^D-W by considering the servers into the S-QUEUE as servers with spare budget, and contextually (and carefully) revising the budget accounting rules of the server without affecting the worst-case performance of the server. Due to lack of space, this alternative is left as a conjecture to be investigated in future work.

5.2 An analysis-based solution

The key idea of the algorithm proposed in this section is to leverage a schedulability test for constrained-deadlines servers to solve the problem identified in Section 4. This new algorithm is denoted as H-CBS^D. As discussed in Section 2.1.1, the exact schedulability test under EDF with constrained deadlines (Theorem 2.1) has a pseudo-polynomial complexity. Therefore, if such a test is used as a theoretical foundation to develop some algorithmic rules of the H-CBS^D, then the resulting algorithm would in turn require a pseudo-polynomial complexity, thus potentially leading to a high run-time overhead. To keep the algorithm overhead low, hence favoring its practical applicability, it is possible to leverage the (sufficient) linear-time schedulability test presented in Theorem 2.2. Note that this choice impacts on the admission test to be adopted for each reservation $r_i \in \mathcal{R}_k$, which must be based on Theorem 2.2: clearly, this reduces the schedulability performance of the system with respect to the case considered in the previous section. The H-CBS^D exploits Theorem 2.2 to design a new `generate` function for the generalized H-CBS formulation in Section 3. The wakeup$(q, d, t) = t$ and account(δ, d) functions have no effect, meaning that (i) the budget is always immediately available as soon as it is recharged and that, (ii) the budget is not decreased when the server is `IDLE`.

Deriving the generate function. Given a set of reservation servers $\mathcal{R}_k$, at any point in time t each reservation $r_i \in \mathcal{R}_k$ is characterized by a pair of dynamic state-variables: the absolute deadline d_i and the current budget q_i. When such parameters are used for a different time instant, say t', they are referred to as q'_i and d'_i. The goal of the following paragraphs is to derive a proper definition for the `generate` function by exploiting such dynamic parameters. First, Lemma 5.3 is provided to bound the demand generated by a H-CBS^D server within a time interval in which the `generate` function is not invoked. Then, such a bound is used to derive a schedulability test (Lemma 5.4) for the system within the same interval of interest. Finally, Lemma 5.4 is used to derive a definition for the `generate` function that guarantees to always assign a schedulable pair (q, d).

Lemma 5.3 (Run-time demand function). *Let t' be an arbitrary point in time after a call to* `generate()`*. Let $t'' > t'$ the first time after t' in which rule R2 triggers a call to the* `generate` *function for reservation r_j. Then, $\forall t' \leq t < t''$, the processor demand of r_j is bounded by its run-time demand function, defined as:*

$$rdf_j(t, q'_j, d'_j) = \begin{cases} q'_j + \alpha_j(t - d'_j) & \text{if } t \geq d'_j \\ 0 & \text{otherwise} \end{cases} \quad (11)$$

where q'_i and d'_i are the current budget and the current absolute deadline at time t', respectively.

Proof. Similarly as argued in the proof of Lemma 5.1, as long as the server has ready workload, rules R5 and R6 guarantee that the server behaves as a sporadic task with the same parameters of the server. As a consequence, for $t \in [t', d'_j)$, the generated demand is zero, as the first instance of the server within $[t', t'']$ has deadline at time d'_j. Since the interval of interest started at time t', where the budget of r_j is q'_j, the demand generated in $[t', d'_j]$ is $q'_j \leq Q_i$. By exploiting the linear demand approximation introduced in Equation (2), from time d'_j on, the demand of r_j can be bounded as $q'_j + \alpha_i(t - d'_j)$. At any point in time $t \geq t'$, if the server stops having pending workload, it either transits to the `IDLE` state or it is in the `THROTTLED` state. Consider the first case. To continue to generate workload, the server must transit back to the `READY` state, which however involves a call to `generate()` (see rule R2). By hypothesis, this cannot happen in $[t', t'')$. Similarly, if the server is in the `THROTTLED` state, by rule R6 it will transit to the `IDLE` state as soon as its budget is replenished: therefore, the same argument used above applies. The lemma follows. □

To reduce clutter when presenting the following results, the function `state`(r_i, t') is defined to return the state of server r_i at time t'. Furthermore, the following reservation sub-sets are defined:

$$\mathcal{R}^r_k(t') = \{r_i \in \mathcal{R}_k : \texttt{state}(r_i, t') = \texttt{READY}\}, \text{ and}$$

$$\mathcal{R}^{tw}_k(t') = \{r_i \in \mathcal{R}_k : \texttt{state}(r_i, t') = \texttt{THROTTLED} \wedge r_i \text{ has ready workload}\}$$

Starting from Lemma 5.3, Lemma 5.4 is derived with the aim of formulating a novel definition for the `generate` function of H-CBS^D. In particular, given a reservation r_i which is applying rule R2, Lemma 5.4 allows verifying if the use of a given budget and a given absolute deadline guarantees the system schedulability.

Lemma 5.4. *Consider a reservation set $\mathcal{R}_k$ that is schedulable according to Theorem 2.2. Let t' be an arbitrary point in time after a call to* `generate`*. Let $t'' > t'$ the first time after t' in which any of the servers in $\mathcal{R}_k$ triggers a call to the* `generate` *function. The reservation set does not experience deadline misses $\forall t' \leq t < t''$ if:*

$$\forall t \in \mathcal{D}^*,\ rdf_j(t, q'_j, d'_j) + \sum_{r_i \in \{R^r_k(t') \setminus \{r_j\}\}} rdf_i(t, q'_i, d'_i) + \sum_{r_i \in \mathcal{R}^{tw}_k(t')} \overline{dbf}_i(t - t^{tw}_i) + \sum_{r_i \in \{\mathcal{R}_k \setminus \{\mathcal{R}^r_k(t') \cup \mathcal{R}^{tw}_k(t')\}\}} \overline{dbf}_i(t - t') \leq t - t' \quad (12)$$

where $t'' > t'$ is the next time in which a reservation $r_i \in \mathcal{R}_k$ triggers a call to generate(), $t^{tw}_i = d'_i - D_i + P_i$,

$$\mathcal{D}^* = \bigcup_{r_i \in \mathcal{R}^r_k(t')} \{d'_i\} \cup \bigcup_{r_i \in \mathcal{R}^{tw}_k(t')} \{d'_i + P_i\},$$

and $\forall r_i \in \mathcal{R}_k$, q'_i and d'_i are their current budget and absolute deadline at time t'.

Proof. The lemma follows from the processor demand criterion recalled in Theorem 2.1 provided that all the terms in Equation (12) are valid demand bounds. Under the hypothesis on times t' and t'', by Lemma 5.3 the first two terms are valid demand bounds for the reservation servers in the `READY` state (set $\mathcal{R}^r_k(t')$). A reservation server r_i in the `THROTTLED` state with absolute deadline d'_i will experience a full budget replenishment at $t^{tw}_i = d_i - D_i + P_i$, and cannot generate demand until the latter time. From time t^{tw}_i, since

the generate function is not called within $[t', t'']$, its maximum demand is the same of the one generated by a sporadic task with the same parameters of the server, hence is bounded by $\overline{dbf}_i(t - t_i^{tw})$. If a reservation server r_i is in the IDLE state, it must call the generate function before starting generating workload. Hence, its demand in the interval of interest is always zero, and consequently the fourth term is a valid bound. The set $\mathcal{D}^*$ includes all the discontinuities of the adopted demand bounds. The lemma follows. □

Lemma 5.4 guarantees the system schedulability until the next call to generate function occurs. The idea of the following results is to to use Lemma 5.4 within the definition of the generate function to ensure the same schedulability condition also when generate is called. Let q' and d' to be the vector of budgets and deadlines for reservations $r_i \in \{\mathcal{R}_k^r \setminus \{r_j\}\}$ at time t', respectively.

Based on Lemma 5.4, it is possible to define a function to determine whether a pair (q_j, d_j) can be used by a reservation server r_j without affecting the system schedulability:

$$\text{check}(q', d', q_j, d_j) = \begin{cases} \texttt{true} & \text{if Lemma 5.4 is satisfied} \\ \texttt{false} & \text{otherwise} \end{cases}$$

Then, the generate function is defined as follows (for a sake of simplicity and homogeneity with respect to previous formulations we omit the dependency from some parameters):

$$\text{generate}(q_j, d_j, t) = \begin{cases} (q_j, d_j) & \text{if check}(q', d', q_j, d_j) \wedge \\ & t < d_j \\ (Q_j, t + D_j) & \text{if } \neg\text{check}(q', d', q_j, d_j) \\ & \wedge\text{check}(q', d', Q_j, t_j^D, t) \wedge \\ & t < d_j \\ (max(0, q_j^*), d_j) & \text{otherwise} \end{cases} \tag{13}$$

where $t_j^D = t + D_j$, and $q_j^* = Q_j - (t - (d_j - D_j))$.

The rationale behind this function is that:

(1) The check function can be used to verify whether the current budget q_j can be safely re-used maintaining the current absolute deadline d_j.
(2) Otherwise, check is used to verify whether is safe to perform a full budget replenishment $q_j = Q_j$ at the current time t, with deadline $t + D_j$.
(3) Finally, if the first two items fail, the deadline remains unchanged and the current budget is set to the value that it would have reached if it always executed in $[d_j - D_j, min(d_j - D_j + Q_j, t)]$

With the algorithm definition in place, Lemma 5.5 guarantees the schedulability of H-CBS^D.

Lemma 5.5 (Safety in the worst case behavior). *The budget and absolute deadline pairs assigned by* generate *as defined for the H-CBS^D does not violate schedulability.*

Proof. Since Lemma 5.4 guarantees schedulability until the next call to generate occurs and the generate function uses Lemma 5.4, the safety of the first two cases of Equation (13) is automatically guaranteed. Considering the third branch of Equation (13), the reservation server r_i which is calling the generate function maintains the same deadline d_i and a budget q_i^* obtained considering that it executed while it was actually IDLE. It is worth noting that r_i is guaranteed with a budget $q_i \geq q_i^*$ (by means of a previous call to generate which exploited Lemma 5.4, if any, or by the admission test), regardless of the state of r_i. Then, the system remains schedulable if r_i requires less demand. The lemma follows. □

Lemma 5.6 shows that the H-CBS^D algorithm guarantees to each reservation r_i a full budget replenishment and an absolute deadline $t + D_i$ to each server r_i which wakes-up after at least P_i after the last replenishment.

Lemma 5.6. *Whenever H-CBS^D is adopted, at any point in time t such that $t \geq d_i - D_i + P_i$, the algorithm guarantees a pair $(q_i^*, d_i^*) = (Q_i, t + D_i)$ for each reservation r_i.*

Proof. At the system startup, a pair (q_i^*, d_i^*) is implicitly guaranteed by the admission test (Lemma 2.2). When the first two branches of the generate function are considered, the reservation-set is guaranteed exploiting Lemma 5.4. It can be observed that the proof of Lemma 5.4 would follow also without adding the term:

$$\sum_{r_i \notin \{\mathcal{R}_k^r \cup \mathcal{R}_k^{tw}\}} \overline{dbf}_i(t - t'),$$

because each reservation $r_i \notin \{\mathcal{R}_k^r \cup \mathcal{R}_k^{tw}\}$ does not consume budget at time t'. Then, in this case the presence of that term is sufficient to reserve a pair (q_i^*, d_i^*) to each server passing from IDLE to READY at least after P_i since the last replenishment. Conversely, when considering the third branch of generate the demand required by a reservation r_i is always lower or equal than the one that it would required if it would be READY (the budget is consumed even if the reservation is not actually executing). Hence, a pair (q_i^*, d_i^*) is implicitly guaranteed by the previous call(if any) to generate which did not use the third branch, or by the admission test. The lemma follows. □

As a consequence of Lemma 5.6, the third branch of the generate function is never taken whenever it is called after at least P_i after the last replenishment (at least the condition for choosing second branch always succeeds). Lemma 5.7 bounds the maximum service-delay of H-CBS^D.

Lemma 5.7 (Bounded-delay). *Consider a reservation set $\mathcal{R}_k$ of H-CBS^D reservations that are EDF-schedulable. The service delay of each server $r_i \in \mathcal{R}_k$ is no larger than $\Delta_i = P_i + D_i - 2Q_i$.*

Proof. According to the general H-CBS formulation presented in Section 3, whenever new workload arrives at time t^* while the server is IDLE, generate is called. Let us analyze separately the worst-case delay of the algorithm when the three different branches of the generate function are taken.

First branch. The budget q_i is available to be used within the absolute deadline d_i. The worst-case scenario occurs when the period instance of the server starts executing as late as possible, i.e., at time $t' = d_i - q_i$. Hence, the worst-case delay is equal to $t' - t^*$. Since $d_i - t^* \leq D_i$, then $t' - t^* \leq \Delta' = D_i$.

Second branch. A budget Q_i is available to be used within $t^* + D_i$. In the worst-case the server starts executing at $t'' = t^* + D_i - Q_i$, leading to a delay $\Delta'' = t'' - t^* = D_i - Q_i$.

Third branch. The budget q^* is available to be executed within the current absolute deadline d_i. By definition, q^* is the budget that the

server would has if it always executed from its last replenishment to time t^*. The scenario which maximizes the delay occurs when new workload arrives in the first time instant in which $q^* = 0$ and the following instance of the server executes as late as possible. This situation occurs when the workload arrives at at $t' = d_i - D_i + Q_i$ and the server starts executing at $t'' = d_i + P_i - Q_i$. The worst-case delay in this case is $\Delta''' = t'' - t' = d_i + P_i - Q_i - (d_i - D_i + Q_i) = P_i + D_i - 2Q_i$. Finally, the global worst-case delay allowed by the algorithm is $\Delta = max(\Delta', \Delta'', \Delta''') = P_i + D_i - 2Q_i$. The lemma follows. □

Improving the available budget. Alternatively, the first branch of `generate` could be defined to follow a principle similar to the one adopted in the revised CBS [1]: maintain the current scheduling deadline and use the maximum budget q_j allowed by Lemma 5.4. Such a budget can be derived as described in Lemma 5.8.

Lemma 5.8. *Consider a reservation set $\mathcal{R}_k$ that is schedulable according to Theorem 2.2. Given an absolute time t' in which a reservation r_j transited from* IDLE *to* READY *with an absolute deadline d_j, the maximum safe budget q_j which preserves the algorithm properties is:*

$$q_j^s = min_{t \in \mathcal{D}^* \wedge t \geq d_j} q_j(t),$$

where $\mathcal{D}^*$ is defined in Lemma 5.4, and

$$q_j(t) = -\alpha_j(t - d_j) - K + t - t' \tag{14}$$

with $K = \sum_{r_i \in \{R_k^r \setminus \{r_j\}\}} rdf_i(t, q_i, d_i) + \sum_{r_i \in \mathcal{R}_k^{tw}} \overline{dbf}_i(t - t_i^{tw}) + \sum_{r_i \notin \{\mathcal{R}_k^r \cup \mathcal{R}_k^{tw}\}} \overline{dbf}_i(t - t')$.

Proof. The lemma follows directly as a consequence of Lemma 5.4, writing $rdf_j(t, q_j, d_j)$ in its extended form (by means of Equation 12, substituting the first branch of its definition, and then considering only absolute-deadlines greater or equal than d_j) and solving with respect to q_j. □

In this case the `generate` function is defined as follows:

$$\text{generate}(q_j, d_j, t) = \begin{cases} min(Q_j, max(0, q_j^s)) & \text{if } t < d_j \wedge q_j^s > 0 \\ (Q_j, t + D_j) & \text{if } t < d_j \wedge q_j^s \leq 0 \wedge \\ & \text{check}(q', d', Q_j, t_j^D, t) \\ (max(0, q_j^*), d_j) & \text{otherwise} \end{cases}$$

where $t_j^D = t + D_j$. The algorithm obtained by leveraging Lemma 5.8 is named H-CBS^D-R. It is worth noting that using H-CBS^D-R each call to `generate` may cause the calling reservation to request a demand higher than the one of a sporadic task with parameters (Q_j, D_j, P_j) (but still guaranteeing schedulability), thus implementing a sort of budget-reclaiming mechanism.

5.3 Discussion

The proposed solutions provide the same worst-case guarantees, namely both of them are able to provide the same worst-case delay and to maintain schedulability among servers. However, they differ in many points:

Applicability. The H-CBS^D-W is independent of the schedulability test used for admitting reservations. This implies that it can be used both under global, partitioned and semi-partitioned scheduling. Conversely, the H-CBS^D and H-CBS^D-R algorithms leverage the approximated schedulability analysis presented in Lemma 5.4, which is a processor-demand based analysis suitable only for partitioned or semi-partitioned EDF-based scheduling. Moreover, the schedulability test used for admitting reservation is constrained to be the same as the one presented in Theorem 2.2.

Average-case performance. H-CBS^D and H-CBS^D-R leverage the schedulability analysis to improve the average-case performance and probabilistic metrics (typically used for soft real-time workload) with respect to H-CBS^D-W. In particular H-CBS^D-R potentially reclaims some spare budget whenever the `generate` function is called.

Implementation and Complexity. Algorithm H-CBS^D-W can implement all the meta-rules in constant-time. On the other hand, it may require the implementation of an additional server queue, the S-QUEUE. Differently, H-CBS^D and H-CBS^D-R do not require additional data structures. Also, `wakeup(q,d,t)` can still be implemented in constant-time and `account(q,d,t)` does not require any implementation. As a drawback, the `generate` function has linear complexity.

6 SIMULATION RESULTS

As previously shown (see Lemmas 5.2 and 5.7), all the algorithms presented in this paper have the same worst-case performance. Whenever each reservation server is used to schedule a single task with worst-case execution time smaller than or equal to than the maximum budget, minimum inter-arrival time larger than or equal to the reservation period, and the same relative deadline of the server, then all its deadlines are guaranteed to be respected (this is sometimes refferred as *hard schedulability property* of the CBS). Otherwise, some deadlines can be missed: in this case, the probability to respect or miss a deadline depends on the used algorithm. To test the performance of the three scheduling algorithms when serving this kind of real-time tasks, an extensive simulation study has been carried out and is presented in this section.

The algorithms have been tested in a widely variable set of situations, by randomly generating a workload composed by real-time tasks with random execution and inter-arrival times. The workload running into each reservation server consists of a single sporadic task τ_i. The workload (i.e., the computation time and the inter-arrival time of each job) is controlled by the following parameters:

(1) The ratio C_r between the average computation time $\overline{c_i}$ of a task τ_i and the maximum runtime Q_i of the server used to schedule it: $\overline{c_i} = C_r Q_i$.
(2) The ratio a between the average utilization of a task τ_i and the utilization of the server used to schedule it: $\frac{\overline{c_i}}{\overline{p_i}} = a\frac{Q_i}{P_i}$, where $\overline{p_i}$ is the average inter-arrival time of τ_i.
(3) The variance of the execution and inter-arrival times.

The sets of servers and tasks used in the simulations have been generated as follows. Given $n = 5$ reservations[3] and a target total utilization $U = \sum_i \frac{Q_i}{P_i}$, budgets and periods have been generated using the Randfixedsum algorithm [17]. The periods P_i of the reservations are distributed between 5000 and 500000. Then, the relative

[3] Simulations with a different number of reservation servers have been performed, showing that the number of reservations does not affect the results. Hence, only results for $n = 5$ are reported.

deadlines of each reservation server have been generated with uniform distribution in the interval $[Q_i + \beta(P_i - Q_i), P_i]$, with $\beta = 0.4$. Reservation sets that are not schedulable according to Theorem 2.2 have been discarded.

All the sets generated using this approach with $U \leq 0.7$ resulted to be schedulable according to Theorem 2.2. For $U = 0.75$, about 2% of the generated reservations sets are not schedulable according to Theorem 2.2, whereas they are all schedulable performing an exact schedulability test; the situation becomes more interesting for $U = 0.9$, when 49% of the generated server sets are not schedulable according to Theorem 2.2 but only 16% of the server sets are really not schedulable using an exact test. This fact has an impact on the usability of the proposed algorithms, since (as previously discussed) H-CBS^D-W can be used with any schedulability test while the other two algorithms require to use Theorem 2.2 to check the servers' schedulability[4].

After generating the reservation-sets, the task τ_i served by each server has been generated considering variables execution times, distributed according to a uniform random variable with average $\overline{c_i} = C_r Q_i$. Similarly, the inter-arrival times have been considered to be variable and distributed according to a uniform random variable with average value $\overline{p_i} = P_i \frac{\overline{c_i}}{aQ_i}$. The execution times are uniformly distributed in $[\overline{c_i} - \frac{sc_i}{2}, \overline{c_i} + \frac{sc_i}{2}]$ and the inter-arrival times are uniformly distributed in $[\overline{p_i} - \frac{sp_i}{2}, \overline{p_i} + \frac{sp_i}{2}]$, where $cs_i = Q_i sz$, $ps_i = P_i sz$, and sz is a parameter in the task set generation. The relative deadline of each task τ_i is equal to the relative deadline of the server used to schedule it.

The generated sets have been simulated by using event-based simulator. For each set of parameters, (U, a, C_r, sz), 100 different reservation-sets have been generated, and the number of missed deadlines for each task has been counted. The ratio of total missed deadlines has been computed for each run, averaging results over the 100 repetitions (also computing the 90% confidence interval).

The simulator has been validated by testing sets of tasks and servers generated as described above together with servers that have been correctly dimensioned to respect the hard schedulability property of the CBS[5]. For those, the simulations correctly reported 0 missed deadlines.

Impact of the Total Utilization U. In a first set of simulations, the impact of the total servers utilization $U = \sum_i \frac{Q_i}{P_i}$ has been evaluated by varying U in the interval $[0.5, 1]$ with steps of 0.1 while keeping C_r, a and sz constant. The simulations showed that U had no significant impact on the percentage of missed deadlines.

This is consistent with the temporal isolation property provided by the CBS algorithm: the real-time performance of a task depends on the parameters of the task and on the server used to schedule it, but does not depend on the other tasks and servers running in the system. Due to this result, in the following simulations the total utilization has been fixed to $U = 0.7$.

Impact of the Server Load. More interesting is the case in which the varied parameter is the parameter a (which is the ratio between the average utilization of the served task and the bandwidth of the reservation: this value can be considered an estimation of the "server load"). When a approaches 1, in the average case the served task tends to use all the execution time provided by the server, with a significant number of instances exceeding the provided budget and completing in subsequent periods, thus missing their deadlines. Decreasing a, the number of respected deadlines increases.

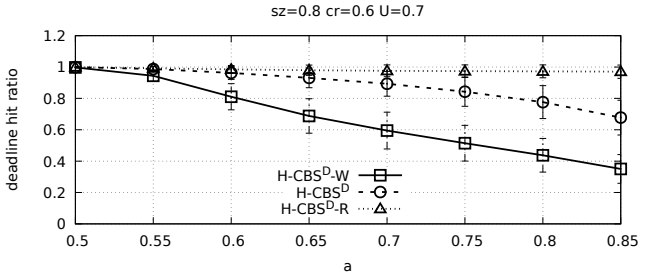

Figure 4: Deadline hit ratio vs a.

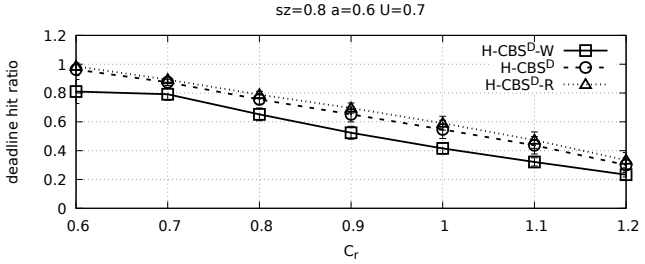

Figure 5: Deadline hit ratio vs C_r.

Figure 4 shows the fraction of deadlines that have been respected as a function of a, with $C_r = 0.6$ and $sz = 0.8$. For small values of a all the algorithms are able to respect almost all of the deadlines. When a increases, H-CBS^D-R continues to respect a very high percentage of deadlines, because it provides a sort of budget-reclaiming, as discussed in Section 5.3, while the performance of the other two algorithms degrade. Since H-CBS^R-W decreases the budget of a the server even if the task does not execute, from Figure 4 it results to perform worse.

Impact of the Average Execution Time. Figure 5 shows the fraction of respected deadlines as a function of C_r, with $a = 0.6$ and $sz = 0.8$. In this situation, the performance of H-CBS^D and H-CBS^D-R are similar, while H-CBS^R-W performs slightly worse. All the algorithms are strongly penalized by large C_r values.

Impact of the Execution and Inter-Arrival Times Variance. Figure 6 shows the fraction of respected deadlines as a function of sz, with $a = 0.6$ and $C_r = 0.6$. Again, H-CBS^R-W misses more deadlines than the other two algorithms. The plot also shows that the number of misses deadlines increases with sz, confirming that large variances in the execution and inter-arrival times have a negative impact on performance. The above observations are also confirmed in Figure 7, showing the deadline hit ratio vs both C_r and sz, in a summarizing 3D plot spanning across a different region of the parameters space, with C_r from 0.7 to 1.1, sz from 0.6 to 0.8 and $a = 0.6$.

[4]These results have been obtained by generating 1000 random task sets according to the explained algorithm, and checking their schedulability according to Theorems 2.2 and 2.1.

[5]Q_i larger or equal than the worst-case execution time of the served task and P_i smaller or equal than the minimum inter-arrival time, with the same relative deadline.

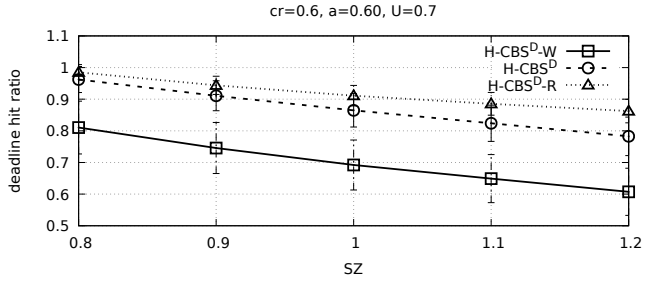

Figure 6: Deadline hit ratio as a function of sz.

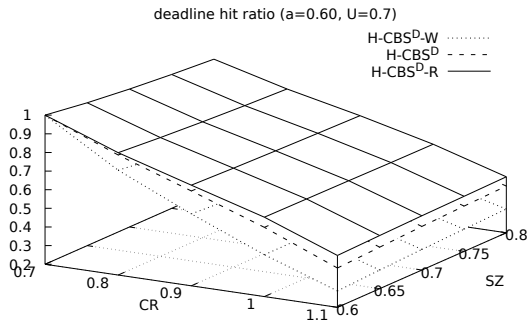

Figure 7: Deadline hit ratio as a function of sz and C_r.

7 CONCLUSIONS AND FUTURE WORK

This paper proposed three different algorithms which extend the Hard Constant Bandwidth Server to work with constrained deadlines. All the three algorithms showed having the same worst-case properties. However, the first one is a generic solution independent of the adopted schedulability test, whereas the others build upon a specific sufficient schedulability test and showed having better performance in terms of probabilistic metrics (i.e., deadline hit ratio). In the literature, many CBS-based reservation algorithms were presented, but none of them is able to deal with constrained deadlines. The proposed algorithms can be used as building blocks to efficiently (e.g., in terms of worst-case delay) implement mechanisms for Semi-Partitioned scheduling in real-time operating systems. Future research directions include the support for shared resources under constrained-deadline reservations both in uniprocessor and multiprocessor platforms (e.g., extending the BROE protocol [5]) and the evaluation of the overhead introduced by means of an implementation in a real operating system kernel (e.g., Linux). Moreover, both H-CBS^D and H-CBS^D-R could be extended to support global scheduling with bounded tardiness [16].

REFERENCES

[1] L. Abeni, G. Lipari, and J. Lelli. 2014. Constant bandwidth server revisited. In *Proceedings of the Embed With Linux 2014 Workshop (EWiLi 2014) (CEUR Workshop Proceedings)*, Vol. 1291. Lisboa, Portugal. http://ceur-ws.org/Vol-1291

[2] L. Abeni, L. Palopoli, C. Scordino, and G. Lipari. 2009. Resource Reservations for General Purpose Applications. *IEEE Transactions on Industrial Informatics* 5, 1 (Feb 2009), 12–21.

[3] S. K. Baruah, L. E. Rosier, and R. R. Howell. 1990. Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor. *Real-time systems* 2, 4 (1990), 301–324.

[4] A. Biondi, A. Balsini, and M. Marinoni. 2015. Resource Reservation for Real-time Self-suspending Tasks: Theory and Practice. In *Proc. of the 23rd International Conference on Real Time and Networks Systems (RTNS '15)*. ACM, 10.

[5] A. Biondi, G. C. Buttazzo, and M. Bertogna. 2015. Supporting component-based development in partitioned multiprocessor real-time systems. In *Proceedings of the 27th Euromicro Conference on Real-Time Systems (ECRTS 2015)*. Lund, Sweden.

[6] A. Biondi, G. C. Buttazzo, and M. Bertogna. 2016. Schedulability Analysis of Hierarchical Real-Time Systems under Shared Resources. *IEEE Trans. Comput.* 65, 5 (2016), 1593–1605.

[7] A. Biondi, A. Melani, and M. Bertogna. 2014. Hard Constant Bandwidth Server: Comprehensive Formulation and Critical Scenarios. In *Proceedings of the 9th IEEE International Symposium on Industrial Embedded Systems (SIES 2014)*. Pisa, Italy.

[8] B. Brandenburg and M. Gul. 2016. Global Scheduling Not Required: Simple, Near-Optimal Multiprocessor Real-Time Scheduling with Semi-Partitioned Reservations. In *Proceedings of the 37th IEEE Real-Time Systems Symposium (RTSS 2016)*. Porto, Portugal.

[9] B. B. Brandenburg and J. H. Anderson. 2007. Integrating Hard/Soft Real-Time Tasks and Best-Effort Jobs on Multiprocessors. In *19th Euromicro Conference on Real-Time Systems (ECRTS'07)*. 61–70.

[10] A. Burns, R. Davis, P. Wang, , and F. Zhang. 2012. Partitioned EDF scheduling for multiprocessors using a C=D task splitting scheme. *Real-Time Systems* 48 (2012), 3–33.

[11] G. C. Buttazzo. 2011. *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications, Third Edition*. Springer.

[12] M. Caccamo, G. Buttazzo, and L. Sha. 2000. Capacity sharing for overrun control. In *Proc. of the 21st IEEE conference on Real-time systems symposium*. Orlando, Florida, USA.

[13] D. Casini, A. Biondi, and G. Buttazzo. 2017. Semi-Partitioned Scheduling of Dynamic Real-Time Workload: A Practical Approach Based On Analysis-driven Load Balancing. In *Proceedings of the 29th Euromicro Conference on Real-Time Systems (ECRTS 2017)*. Dubrovnik, Croatia.

[14] J.J. Chen, G. Nelissen, and W.H. Huang. 2016. A unifying response time analysis framework for dynamic self-suspending tasks. In *28th Euromicro Conference on Real-Time Systems (ECRTS)*. IEEE.

[15] J.J. Chen, G. Nelissen, W.-H. Huang, M. Yang, K. Bletsas B. Brandenburg, C. Liu, P. Richard, F. Ridouard, N. Audsley, R. Rajkumar, and D. de Niz. 2016. *Many suspensions, many problems: A review of selfsuspending tasks in real-time systems*. Technical Report. Faculty of Informatik, TU Dortmund.

[16] U. M. C. Devi and J. H. Anderson. 2005. Tardiness bounds under global EDF scheduling on a multiprocessor. In *26th IEEE International Real-Time Systems Symposium (RTSS'05)*. 12 pp.–341.

[17] P. Emberson, R. Stafford, and R. Davis. 2010. Techniques for the synthesis of multiprocessor tasksets. In *Proc. of the 2nd International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS 2010)*. Brussels, Belgium.

[18] N. Fisher, T. P. Baker, and S. Baruah. 2006. Algorithms for Determining the Demand-Based Load of a Sporadic Task System. In *12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'06)*. Sydney, Australia.

[19] J. Lelli, C. Scordino, L. Abeni, and D. Faggioli. 2016. Deadline Scheduling in the Linux kernel. *Software: Practice and Experience* 46, 6 (Jun 2016), 821–839.

[20] L. Marzario, G. Lipari, P. Balbastre, and A. Crespo. 2004. IRIS: A New Reclaiming Algorithm for Server-Based Real-Time Systems. In *10th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS 2004)*. Toronto, Canada.

[21] C. W. Mercer, S. Savage, and H. Tokuda. 1994. Processor capacity reserves for multimedia operating systems. In *Proceedings of IEEE international conference on Multimedia Computing and System*. Boston, Massachusetts, USA.

[22] A. K. Mok, X. Feng, and D. Chen. 2001. Resource partition for real-time systems. In *Proceedings Seventh IEEE Real-Time Technology and Applications Symposium (RTAS 2001)*.

[23] R. Rajkumar, K. Juvva, A. Molano, and S. Oikawa. 1998. 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.

[24] I. Shin and I. Lee. 2004. Compositional real-time scheduling framework. In *Proceedings of the 25th IEEE Real-Time Systems Symposium(RTSS 2004)*. Lisbon, Portugal.

[25] F. Zhang and A. Burns. 2009. Schedulability Analysis for Real-Time Systems with EDF Scheduling. *IEEE Trans. Computers* 58, 9 (2009), 1250–1258.