Supporting Component-based Development in Partitioned Multiprocessor Real-Time Systems

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

***Abstract*—The fast evolution of multicore systems, combined with the need of sharing the same platform for independently developed software, demands for new methodologies and algorithms that allow resource partitioning, while guaranteeing the isolation of concurrent applications. Unfortunately, a major problem that can break the isolation property of concurrent partitions is resource sharing. Although a number of resource access protocols exist for hierarchical uniprocessor systems, no protocols are available today for managing hierarchical partitions implemented on top a multiporcessor platform under partitioned scheduling.**

This paper presents a framework to support component-based design on a multiprocessor platform and proposes a novel reservation server mechanism, called M-BROE, to handle shared resources in multiprocessor systems in the presence of resource reservation scheduling mechanisms.

I. Introduction

The fast evolution of embedded computing systems is demanding for novel methodologies for managing software complexity and simplifying the analysis of software components that run on the same platform and share common resources.

In several application domains, from multimedia to automotive systems, software typically exceeds 1 million lines of source code and includes hundreds of concurrent activities that interact with each other in a complex manner.

For instance, in the automotive domain, the continuous demand of new functions increased the number of electronic control units (ECUs) up to a limit that is hard to manage, for space, weight, and energy constraints. As a result, the current industrial trend is to integrate multiple functions into the same ECU, thus saving space weight, and power consumption. For certification and safety reasons, proper operating system mechanisms have to be adopted to isolate the behavior of the different software components that coexist on the same platform.

An efficient and flexible scheduling technique to achieve temporal isolation among different software modules is resource reservation [1], [2]. According to this method, the available processor bandwidth is partitioned among the different modules through a budget enforcement mechanism which ensures that tasks executing within different partitions do not interfere with each others, so that a module receiving a fraction α_i of the total processor bandwidth behaves as if it were executing alone on a slower processor with a speed equal to α_i times the full speed. The advantage of this approach is that each module can be guaranteed independently of the behavior of the others.

This concept has been extended to support the development of hierarchical systems, where each reservation can host an entire application (or subsystem), whose tasks are handled by a local scheduler that can be selected independently of the other reservations. In addition, a subsystem can be further partitioned into different subsystems, and so on. The reservations defined at the first level of the hierarchy are managed by a global scheduler.

A major problem that can break the isolation property of resource reservations is resource sharing. In fact, suppose that a resource R is shared by two tasks τ_1 and τ_2 running within two subsystems S_1 and S_2, respectively, and consider the case in which τ_1 blocks on R while τ_2 is holding it. If the reservation budget of S_2 is exhausted while τ_2 is still inside the critical section, τ_1 will experience an extra blocking delay to wait for the budget of S_2 to be replenished.

To address such a problem, a number of resource access protocols have been proposed in the literature [3]–[6].

Today, the concept or reservation, and more generally the idea of temporal isolation is also accepted by the industry to limit the interference among independently developed software components and ensure higher predictability of the system. The temporal isolation is also specified in different standards, like AUTOSAR [7], adopted in automotive systems, and ARINC [8], adopted in avionics.

Another problem is that, with the evolution of processors, several industries are facing a transition from single core to multicore systems, hence it is crucial to provide support for resource reservation in multicore platforms. This kind of platforms, in fact, allow application providers to continue exploiting Moore's law greedy demand for computing power without incurring thermal and power problems. Although a number of approaches have been proposed to support resource reservations on multicore systems [9]–[11], none of them considered the possibility for the applications to share global resources. On the other hand, there are a few papers considering resource sharing in the scheduling of independently developed real-time applications on a partitioned multiprocessor system [12], [13], but they assume that each application is statically allocated on a dedicated core, thus no reservation mechanism is assumed for platform virtualization.

In this context, depending on whether resources are shared by tasks running within the same reservation or belonging to different reservations, the access must be regulated by different methods. In particular, three different types of resources can be distinguished: (i) *Local resources*, shared only by tasks be-

longing to the same reservation; (ii) *Processor-local resources*, shared by tasks belonging to different reservations allocated on the same core. (iii) *Global resources*, shared by tasks belonging to different reservations allocated to different cores.

Such a distinction is important because the resource type affects the protocol for accessing and executing the critical section and the protocol for managing the reservation budget. More specifically, local resources can be safely accessed by the Stack Resource Policy (SRP) [14] using any budget management mechanism; Processor-local resources can be accessed by H-SRP [3] combined with the BROE [6] algorithm for managing the reservation budget. At the moment, however, no protocol is available for safely accessing global resources in a multiprocessor platform supporting reservations under a partitioned approach. The situation is summarized in Table I:

	no reservation	same reservation	different reservations
same processor	PIP, PCP SRP, ...	*Local* PIP, PCP SRP, ...	*Processor-local* H-SRP + BROE H-SRP + SIRAP
different processors	MSRP ...	N.A.	*Global* Contribution of this paper

Table I: Types of shared resources and related protocols.

Contributions. This paper has four main contributions: first, a framework is presented to support component-based design for independently developed real-time applications on a multiprocessor platform; second, a novel reservation server, called M-BROE, is proposed to support resource sharing in multiprocessor systems in the presence of resource reservations; third, two protocol design alternatives (BCBS and BCAS) are presented and compared, highlighting their pros and cons; forth, two components interfaces are analyzed in terms of their limitations and benefits.

Paper structure. Section II presents the system model and our framework to support component-based design, also specifying the component interface needed for the integration. Section III describes the M-BROE reservation algorithm and the corresponding scheduling rules. Section IV presents the schedulability analysis for the proposed framework. Section V discusses design choices for M-BROE and presents experimental results comparing alternative behavior for M-BROE. Section VI describes extensions for the component interface and the corresponding schedulability analysis. Section VII discusses further research issues related to the optimal design of the virtual processors. Section VIII concludes the paper.

II. Framework and Modeling

This section describes the proposed framework to support component-based design under partitioned multiprocessor scheduling. We first introduce the system model, then we define the *component interfaces* for the integration, and finally, the *scheduling infrastructure* for managing reservations.

A. System model

This paper considers a system consisting of N software components $\Gamma_1, \Gamma_2, \ldots, \Gamma_N$. Each *software component* Γ_k (also referred to as component or application) consists of a set $\mathcal{T}(\Gamma_k)$ of n_k real-time periodic or sporadic tasks. Each task $\tau_i \in \mathcal{T}(\Gamma_k)$ 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.

The system has to be executed on a multiprocessor platform composed of M identical processors, each denoted as $\mathcal{P}_m$, with $m = 1, \ldots, M$. To enable a modular design and analysis of the system, each component Γ_k is statically partitioned over $\mathcal{M}$ virtual processors $S_j^k, j = 1, \ldots, \mathcal{M}$, each implemented by a reservation server characterized by a maximum budget Q_j^k and a period P_j^k. The ratio $\alpha_j^k = Q_j^k / P_j^k$ is referred to as the reservation bandwidth. The virtual processor at which a task $\tau_i \in \mathcal{T}(\Gamma_k)$ is assigned is denoted as $\mathcal{S}(\tau_i)$.

Each component Γ_k is abstracted through a *component interface* consisting of $\mathcal{M}$ couples (Q_j^k, P_j^k), with $j = 1, ..., \mathcal{M}$, representing the budget and the period of the reservation server associated to S_j^k. Hence, each component does not export any detailed information about the tasks composing it. Also note that, to keep the interface as simple as possible, *no information related to the resources accessed by each component is exported in the interface*. More complex interfaces are discussed in Section VI.

The allocation of virtual processors to physical processors is performed by a *component integrator* module, which is responsible for statically assigning each reservation server $S_j^k, j = 1, ..., \mathcal{M}$ to a specific processor $\mathcal{P}_m$. Note that such a mapping can be arbitrary, that is the j^{th} virtual processor is not forced to be allocated to the j^{th} processor. The processor on which the virtual processor S_j^k is assigned is denoted as $\mathcal{P}(S_j^k)$. No a priori assumptions are made on this mapping when component interfaces are designed. The *component integrator* is also responsible for admitting or rejecting each application Γ_k. In order to contain the complexity of the formulation, this paper assumes $\mathcal{M} = M$. The generic case in which $\mathcal{M} \geq M$ will be addressed in a future work. The framework described above is illustrated in Figure 1 for a platform composed of $M = 4$ processors.

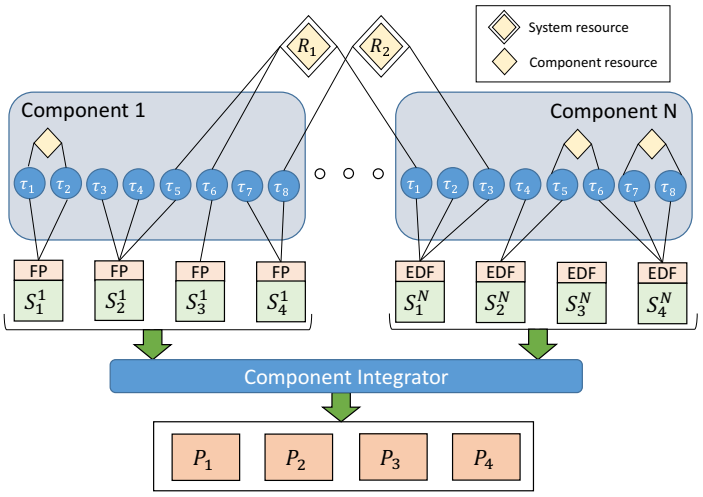

Figure 1: Proposed hierarchical framework.

Tasks can access shared resources through mutually exclusive critical sections. At a component level, we distinguish between two types of resources: (i) *component resources*, accessed only by tasks belonging to the same component, (ii) *system resources*, accessed by tasks belonging to different components.

For each resource R_ℓ, $\delta_{i,\ell}$ denotes the length of the longest critical section of task τ_i related to R_ℓ, while $\eta_{i,\ell}$ denotes the number of critical sections used by τ_i on R_ℓ.

Definition 1: **Maximum Resource Holding Time** The resource holding time $H_{i,\ell}$ for a resource R_ℓ accessed by a task τ_i is defined as the maximum budget consumed from the lock of R_ℓ until its unlock in the τ_i's code.

Note that if a resource is accessed in a non-preemptive manner, its resource holding time is equal to the critical section length (i.e., $H_{i,\ell} = \delta_{i,\ell}$). On the other hand, if a resource is accessed in a preemptive way (e.g., as under SRP), the resource holding time takes into account all possible preemption delays occurring within the critical section. Details on how to compute resources holding times can be found in [15].

To be admitted in the system, a component must have all its tasks with a bounded resource holding time. Formally, each component Γ_k must satisfy the following condition for each *system resource* R_ℓ:

$$\forall \tau_i \in \mathcal{T}(\Gamma_k), \forall R_\ell,\ H_{i,\ell} \leq \mathcal{H}, \tag{1}$$

Regarding *component resources* R_q used by tasks assigned to different virtual processors, we require that the sum of the maximum resource holding times of R_q from each virtual processor is bounded by $M\mathcal{H}$. Formally we require that

$$\forall \Gamma_k, \forall R_q \in \mathcal{R}_k,\ \sum_{j=1}^{M} \max\{H_{i,q} \mid \mathcal{S}(\tau_i) = S_j^k\} \leq M\mathcal{H}, \tag{2}$$

where $\mathcal{R}_k$ denotes the set of *component resources* accessed by tasks running on different virtual processors, formally defined as

$$\mathcal{R}_k = \{R_q \mid \exists \tau_1, \tau_2 \text{ using } R_q \wedge \mathcal{S}(\tau_1) \neq \mathcal{S}(\tau_2)\}. \tag{3}$$

B. Scheduling infrastructure

The tasks of a component allocated to a virtual processor are managed by a *local scheduler*, which can be selected between EDF or any fixed-priority (FP) algorithm. Under EDF, tasks are ordered by increasing relative deadlines, whilst under FP scheduling tasks are ordered by decreasing priorities, so that τ_1 is the highest priority task. We denote as local non-preemptive section a time interval in which the preemption is disabled for the local scheduler.

Each reservation associated with a virtual processor is implemented by an M-BROE server (i.e., the novel scheduling mechanism proposed in this paper). The M-BROE server is based on a *Hard Constant Bandwidth Server* (H-CBS) [16], [17] extended to support resource sharing in a multiprocessor environment providing resource reservations. The various M-BROE servers are scheduled under partitioned EDF scheduling on the M processors.

Given the mapping of each reservation server to processors, three types of shared resources can be identified:

- *Local resources*: shared only by tasks running upon the same reservation server;
- *Processor-local resources*: shared only by tasks executing on the same processor, but on different reservation servers;
- *Global resources*: shared by tasks executing on different processors.

The access to *local resources* is managed through the SRP [14] protocol, while the access to *processor-local resources* is performed by using the H-SRP [3] protocol in conjunction with the M-BROE server resource access policy (inherited from the original BROE server), such resources are accessed in a local non-preemptive manner. Contention for *global resources* is managed through the MSRP [18] protocol in conjunction with the M-BROE server resource access policy.

Please note that a *component resource* R_ℓ results to be a *local resource* only if all the tasks accessing R_ℓ are assigned to the same virtual processor. Similarly, a *system resource* will never results to be a *local resource* (except for the special case in which it is accessed by only one component).

III. The M-BROE Server

This section presents the M-BROE algorithm, which extends the BROE server [6] to support resource sharing in multiprocessor systems.

The original BROE server has been conceived as an extension of the H-CBS server to support resource sharing among tasks executing upon different reservations. As well known from the literature, the access to shared resources under resource reservations introduces the problem of budget depletion inside critical sections, which prolongs the blocking delay with the time needed to replenish the budget.

The BROE server solves this problem by introducing a budget check before entering a critical section, and managing the budget to preserve both the bandwidth and worst-case service delay of the server when resources are accessed. However, the BROE server has been designed for uniprocessor systems. Under multiprocessor partitioned scheduling, the BROE server can still be used in its original formulation, but there is no support for accessing global resources (i.e., resources shared by tasks running on different processors).

In the M-BROE server, the contention for global resources is managed through *FIFO non-preemptive spinlocks* by integrating the MSRP [18] protocol. The use of such a kind of spinlocks is also motivated by their performance in terms of schedulability, as identified by Wieder and Brandenburg [19] through an extensive set of experimental results. Spinlocks are widely used to address resource sharing in multiprocessor embedded systems, being an effective mechanism to ensure mutual exclusion requiring low runtime overheads and footprint for its implementation. For example, the AUTOSAR standard mandates the use of non-preemptive spinlocks.

For a task τ_i executing on processor $\mathcal{P}_j$ and accessing a global resource R_ℓ with FIFO non-preemptive spinlocks, the maximum spinning time $\xi_{\ell,j}$ (i.e., the maximum time for which a task wastes processor cycles waiting for R_ℓ) is bounded by the sum of the longest resource holding times of R_ℓ related to tasks running on other processors. Assuming that global resources are accessed in a non-preemptive fashion (as true in

the MSRP protocol), resource holding times reduce to critical section lengths, hence $\xi_{\ell,j}$ can be formally expressed as

$$\xi_{\ell,j} = \sum_{\mathcal{P}_j \neq \mathcal{P}_m} \max\{\delta_{i,\ell} \mid \mathcal{P}(\mathcal{S}(\tau_i)) = \mathcal{P}_m\}, \quad (4)$$

The next section introduces the rules that describe the M-BROE server: Rules 1,2,3 and 4 are from the original BROE server, while Rule 5 is the proposed extension to support resource sharing in multiprocessor systems.

A. M-BROE server rules

The rules of an M-BROE server with period P and maximum budget Q (bandwidth $\alpha = Q/P$) are summarized below.

At any time t, the server is characterized by an absolute deadline d and a remaining budget $q(t)$. When a job executes, $q(t)$ is decreased accordingly.

1) Initially, $q(t) = 0$ and $d = 0$.
2) When the M-BROE server is idle and a job arrives at time t, a replenishment time is computed as $t_r = d - q(t)/\alpha$:
   a) if $t < t_r$, the server is suspended until time t_r. At time t_r, the budget is replenished to Q and $d \leftarrow t_r + P$.
   b) otherwise the budget is immediately replenished to Q and $d \leftarrow t + P$;
3) When $q(t) = 0$, the server is suspended until time d. At time d, the server budget is replenished to Q and the deadline is postponed to $d \leftarrow d + P$.
4) When a task τ_i wishes to access a *processor-local resource* at a time t, a budget check is performed: if $q(t) \geq \delta_{i,\ell}$, there is enough budget to complete the critical section, hence the access is granted. Otherwise a replenishment time is computed as $t_r = d - q(t)/\alpha$:
   a) if $t < t_r$, the server is suspended until time t_r. At time t_r, the budget is replenished to Q and $d \leftarrow t_r + P$.
   b) otherwise the budget is immediately replenished to Q and $d \leftarrow t_r + P$.
5) When a task τ_i wishes to access a *global resource* R_ℓ at a time t, a budget check is performed: if $q(t) \geq (\xi_{\ell,j} + \delta_{i,\ell})$, there is enough budget to experience the maximum spinning time and complete the critical section, hence the access is granted. Otherwise a replenishment time is planned at time $t_r = d - q(t)/\alpha$ and the same operations of Rule 4-a or 4-b are performed.

IV. Schedulability Analysis

We now present the schedulability analysis for the proposed framework to support component-based design under multiprocessor partitioned scheduling. First, we briefly recall the MSRP protocol and its schedulability analysis. Then, we present the schedulability analysis for M-BROE used in conjunction with MSRP and H-SRP. Given an assignment of tasks to virtual processors, Section IV-C reports the schedulability test to guarantee a set of tasks executing within the M-BROE server implementing the virtual processor abstraction.

Finally, the analysis for the *component integrator* is provided, that is, given an assignment of virtual processors to physical processors, we show how to guarantee the schedulability of the virtual processors according to their requirements specified in the component interface.

The SRP and H-SRP protocols and their integration with BROE are not recalled in this paper for space limitations, please refer to [6], [20] for additional details.

Schedulability analysis is presented for local EDF schedulers, but it can be easily extended for local FP scheduling; however such an analysis is not reported for space limits.

A. The MSRP protocol

The MSRP protocol has been proposed by Gai et al. [18] as an extension of the uniprocessor SRP protocol to deal with resources accessed from multiple processors. MSRP uses SRP as it is for local resources (i.e., the ones shared by tasks executing on the same processor), while it makes use of non-preemptive FIFO spinlocks to deal with global resources. When a task τ_i wants to access a global resource, and the resource is locked from a task on another processor, τ_i becomes non-preemptive and starts spinning (i.e., wasting processor cycles) until the access to the resource will be granted. The critical section on the global resource is also executed in a non-preemptive manner. Multiple requests from different processors are served in FIFO order, i.e., we can assume the existence of a global FIFO queue containing at most one task per processor.

The original analysis of MSRP relies on a *computation time inflation* for each task accessing a resource. That is, a bound on the maximum spinning time is computed for each global resource R_ℓ accessed from a given processor; then, such a bound is used to inflate the computation time of the tasks using R_ℓ. The amount of inflation imposed to each task is denoted as *remote blocking*. At the same time, a blocking factor related to non-preemptive spinning and non-preemptive access to global resources is accounted for each task. That is, when a task becomes non-preemptive it disallows higher-priority tasks to execute, imposing *non-preemptive blocking* on them. Finally, since MSRP relies on the classical SRP protocol for local resources, tasks are also affected by *local blocking*.

The next section integrates the original MSRP analysis [18] proposed by Gai et al. with M-BROE.

Please note that, as identified in [19], [21], the original MSRP analysis is inherently pessimistic for several reasons. Pessimism can be reduced by exploiting the full knowledge of task parameters and response-times (e.g., as shown by Wieder and Brandenburg in [19]). However, in a component-based environment, the parameters of the other applications are unknown, hence it is not possible to exploit tasks parameters (as periods and critical sections) to refine the analysis. [1]

On the other hand, the original MSRP analysis provides a simple as well as effective method to compute bounds on *remote blocking* and *non-preemptive blocking* by only using bounds on critical section lengths.

[1]The only exception to such a limitation is related to the case in which component resources are accessed by tasks assigned to different virtual processors: in this case, after component integration, these resources could result in global resources and it would be possible to reduce pessimism in the schedulability analysis for the tasks of a component (but not for tasks in other components).

B. Integrating MSRP in the M-BROE framework

We first present expressions for computing *remote blocking*, *non-preemptive blocking* and *local blocking*; then, in the next section we integrate them with the schedulability analysis of our proposed framework based on M-BROE.

Remote blocking. Given a global resource R_ℓ accessed from processor $\mathcal{P}_j$, an upper-bound on the maximum spinning time $\xi_{\ell,j}$ can be computed by using Equation (4). Then, to compute an upper-bound S_i on *remote blocking* on a task τ_i we have to account for the maximum spinning time for each critical section of τ_i, with $\mathcal{P}(\mathcal{S}(\tau_i)) = \mathcal{P}_j$, that is:

$$S_i = \sum_{R_\ell} \{\eta_{i,\ell}\xi_{\ell,j} \mid R_\ell \text{ used by } \tau_i\}. \quad (5)$$

The term S_i will be used to inflate the computation time of task τ_i. Please note that, since Rule 5 of the M-BROE server ensures that the budget will never exhaust during the spinning phase, we can account for a single spinning time $\xi_{\ell,j}$ for each critical section of R_ℓ used by τ_i.

Non-preemptive blocking. Due to non-preemptive spinning and non-preemptive access to global resources, tasks are affected by a blocking factor equal to the longest non-preemptive section causing arrival blocking. Under local EDF scheduling, such a blocking factor can be computed as:

$$B_i^{\text{NP}} = \max_{k,R_\ell}\{\xi_{\ell,j} + \delta_{k,\ell} \mid R_\ell \text{ used by } \tau_k \wedge D_k > D_i\}, \quad (6)$$

assuming that tasks τ_k and τ_i execute on processor $\mathcal{P}_j$ (i.e., $\mathcal{P}(\mathcal{S}(\tau_k)) = \mathcal{P}(\mathcal{S}(\tau_i)) = \mathcal{P}_j$).

Local blocking. Local resources are accessed using the classical (uniprocessor) SRP protocol. The SRP blocking factor for task τ_i due to local resources is denoted by B_i^L. For lack of space, we omit the expression for such blocking factor, please refer to [14] for details about SRP blocking.

The actual arrival blocking imposed on each task τ_i can be then computed as:

$$B_i = \max\{B_i^{\text{NP}}, B_i^L\}. \quad (7)$$

Upper-bound for the spinning time. Please note that a component does not know the parameters of the tasks in the other components. Moreover, when testing the schedulability of a component on its virtual processors, it is not possible to infer on the mapping that will be performed by the *component integrator*. Looking at the definition of the spinning time $\xi_{\ell,j}$ in Equation (4), we can observe that such an equation depends on (i) critical section lengths of the other tasks, (ii) the processor on which the tasks are assigned. Since *system resources* are potentially used by all the components, it is not possible to exactly compute the spinning time $\xi_{\ell,j}$. Hence, a safe upper-bound on that term has to be used in the schedulability analysis, that is expressed by the following lemma.

Lemma 1: For each *system resource* R_ℓ accessed by tasks executing on procesor $\mathcal{P}_j$ we have

$$\forall \ell, j \ \xi_{\ell,j} \leq (M-1)\mathcal{H}.$$

Proof: By the assumption stated in Equation (1), each task admitted in the system must have bounded critical sections such that $\delta_{i,\ell} \leq \mathcal{H}, \forall \tau_i, \forall R_\ell$. Since we are using non-preemptive FIFO spinlocks, we can conclude that the spinning time $\xi_{\ell,j}$ is bounded by at most a critical section length for each processor different from $\mathcal{P}_j$, leading to $\xi_{\ell,j} \leq (M-1)\mathcal{H}$. Hence the lemma follows. ■

Regarding *component resources*, critical section lengths are known, since they belong to tasks of the same component. However, when tasks are assigned to different virtual processors, it is not possible to infer on which physical processor such tasks will execute. For this reason, in order to compute a safe bound on the spinning time, we have to assume that all the virtual processors of a component will be assigned to different physical processors. Therefore, for each component Γ_k, the upper bound on the spinning time $\xi_{\ell,j}$ for a *component resource* R_ℓ can be computed as

$$\xi_{\ell,j} \leq \sum_{S_j^k \neq S_m^k} \max\{\delta_{i,\ell} \mid \mathcal{S}(\tau_i) = S_m^k\}.$$

C. Component local analysis

Now, it is possible to derive a schedulability test for a set of tasks executing upon an M-BROE server. In this section, we refer to a single M-BROE server used to implement one of the M virtual processors required from a component. The M-BROE is configured with parameters $\alpha = Q/P$ and $\Delta = 2(P-Q)$, where Q and P refer to budget and period of the server, as specified in the component interface.

Following the processor demand criterion extended to include resource sharing [22], a set of tasks $\mathcal{T}$ assigned to an M-BROE server is EDF-schedulable if:

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

where

$$\mathsf{dbf}(t) = \sum_{\tau_i \in \mathcal{T}} \left(\left\lfloor \frac{t - D_i}{T_i} \right\rfloor + 1 \right)_0 (C_i + S_i), \quad (9)$$

$$B(t) = \max\{B_i \mid D_i \leq t\}, \quad (10)$$

and $\mathsf{sbf}^B(S,t,X)$ is the *supply bound function* for the M-BROE server. In the equation above, the notation $(x)_0$ is used to denote $\max(0,x)$.

To compute a lower bound on the supply provided by an M-BROE server, we note that the only difference with respect to the original BROE server lies in the additional budget check performed whenever a global resource is accessed (see Rule 5 in Section III-A). Such a budget check is identical to the original one (Rule 4), using a larger threshold to include the spinning time, i.e., $q(t) \geq (\xi_{\ell,j} + \delta_{i,\ell})$. Therefore, we can safely re-use the same supply bound function provided by the original BROE server detailed in [20], [23], with a different budget threshold. In other words, the characterization of the processing bandwidth supplied by an M-BROE server does not introduced additional pessimism with respect to the case without global resources.

Let X be the maximum budget threshold that drives the budget check, the supply function $\mathsf{sbf}^B(S,t,X)$ of the M-BROE server is reported in Equation (11).

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

where

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

and t_A, t_B, t_C, and t_D are reported in Table II. The function is equal to 0 in the interval $[0, \Delta]$. More details on the derivation of Equation (11) can be found in [20], [23].

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

Table II: Values for sbf^B.

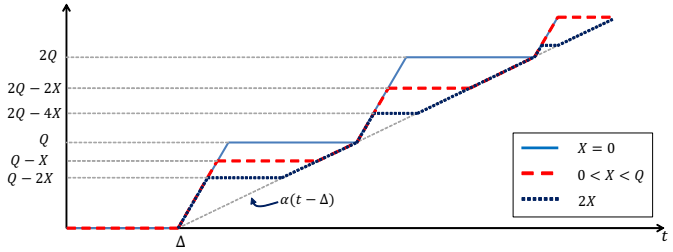

Figure 2: Examples of supply functions sbf^B for the M-BROE server for different values of parameter X.

The function $\mathsf{sbf}^B(S,t,X)$ is illustrated in Figure 2 for different values of X. As clear from the graph, the bigger the value of X, the lower the supply value for the same time interval $[0,t]$.

The value X (representing the maximum threshold used for the budget check of the M-BROE server) can be computed as:

$$X = \max_{R_\ell}\{\xi_{\ell,j} + \delta_{i,\ell} \mid R_\ell \text{ used by } \tau_i \in \mathcal{T}\}, \tag{13}$$

with R_ℓ non-local resource (since local resources do not involve a budget check) and assuming $\xi_{\ell,j} = 0$ for processor-local resources since processor-local resources do not involve a spinning phase (see Rule 4 of M-BROE).

Also note that the demand bound function reported in Equation (9) takes into account the computation time inflation related to *remote blocking* through the term S_i.

D. Component integration analysis

The *component integrator* has to ensure the schedulability of the reservation servers assigned to each processor. Each component provides a set of reservation servers to the *component integrator* according to the specified interface; such reservation servers will be scheduled under partitioned scheduling on the M physical processors. Hence, given an assignment of each reservation server to a processor, uniprocessor schedulability analysis from [6] can then be used. However, since the access to global resources results in non-preemptive sections, an additional blocking term has to be taken into account to ensure the schedulability of the servers. To better clarify this point, consider a task τ_i executing on a server S assigned to processor $\mathcal{P}(S)$. When τ_i locks a global resource R_ℓ, it can potentially experience a non-preemptive spinning phase, followed by a non-preemptive critical section on R_ℓ. In this case, *all* the other reservation servers S' assigned to processor $\mathcal{P}(S)$ (i.e., $\mathcal{P}(S) = \mathcal{P}(S')$) will be not able to preempt S. This phenomenon is denoted as *non-preemptive server blocking*. Such a blocking concurs with H-SRP blocking for *processor-local resources* to quantify the maximum blocking time imposed on each reservation server. Under the H-SRP protocol, servers can be blocked by at most one single critical section (like in classical SRP); further details about H-SRP can be found in [3], [6]. The overall blocking on each reservation server S depends on the resources used by the tasks executing upon S. Since our interface does not export information about shared resources, a bound on server blocking has to be used to verify the schedulability of the set of reservation servers assigned to each processor. Improved schedulability tests for the component integration can be derived by expanding the interface exported from each component, as discussed in Section VI.

The non-preemptive server blocking coming from a server S is originated from the largest non-preemptive section imposed by tasks executing upon S. By Lemma 1 is then possible to bound such a blocking factor for *system resources*. Similarly, Equation (2) allows to bound the server blocking related to *component resources*.

Now it is possible to derive a schedulability test for the *component integrator*. A set of components $\Gamma_0, \ldots, \Gamma_k, \ldots, \Gamma_n$ can be scheduled if, for each processor $\mathcal{P}_m$ with $m = 1, \ldots, M$, the following condition holds:

$$\forall S_j^k : \mathcal{P}(S_j^k) = \mathcal{P}_m, \sum_{\substack{r,l: P_r^l \leq P_j^k \\ \wedge \\ \mathcal{P}(S_r^l) = \mathcal{P}_m}} \frac{Q_r^l}{P_r^l} + \frac{M\mathcal{H}}{P_j^k} \leq 1. \tag{14}$$

V. M-BROE Design Choices

In the original BROE server, the problem of budget depletion inside a critical section is avoided by performing a budget check before locking a resource. When the budget is not enough to complete the critical section, BROE postpones the server deadline to perform a budget replenishment. This mechanism introduces an additional payback in terms of schedulability analysis, reflected in a reduced supply function of the BROE server with respect to a classical periodic server (as the H-CBS) where no resource sharing is taken into account (that is equivalent to the case for $X = 0$). With respect to Equation (11), the payback depends on the maximum threshold X used in the budget check: the greater the threshold, the lower the supply function provided by BROE (as illustrated in Figure 2).

When spinlocks are used, each locking of a global resource is composed of an initial phase of spinning followed by the actual critical section for the resource. Hence, if a budget exhaustion occurs in the spinning phase, the server is suspended

leaving the resource unlocked. One could think that it would be more convenient to perform the budget check after the spinning phase, i.e., just before entering the critical section for a global resource. However, we identified some disadvantages in following this approach, thus the M-BROE protocol has been defined to perform the budget check before the spinning phase, taking into account the maximum spinning time (see Rule 5 in Section III-A).

In the following we refer to a single M-BROE server and a generic task τ_i executing upon it. To distinguish between the two budget check schemes we denote them as:

- (i) *BCAS (Budget Check After Spinning)*, the scheme where the budget check is performed considering only the critical section length;
- (ii) *BCBS (Budget Check Before Spinning)*, the scheme where the budget check is performed considering both critical section length and maximum spinning time.

Formally, considering a task τ_i accessing a global resource R_ℓ at time t, we have the following budget checks: (i) *BCAS*: $q(t) \geq \delta_{i,\ell}$; (ii) *BCBS*: $q(t) \geq (\xi_{\ell,j} + \delta_{i,\ell})$.

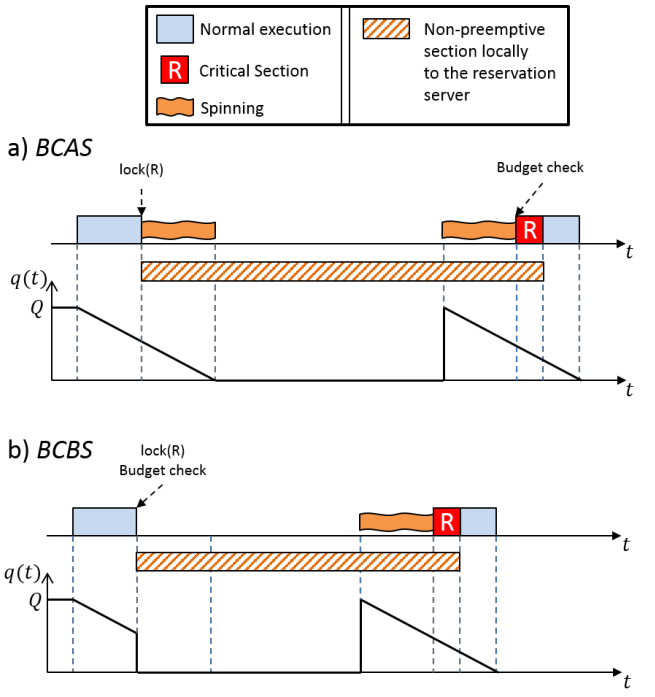

Figure 3: Schedule scenarios for the two budget check schemes.

A. The BCAS scheme

Suppose that a task τ_i executing on processor $\mathcal{P}_j$ (i.e., $\mathcal{P}(\mathcal{S}(\tau_i)) = \mathcal{P}_j$) is accessing a global resource R_ℓ under the *BCAS* scheme, as illustrated in Figure 3-a. First of all notice that, if the server budget exhausts during the spinning phase, τ_i cannot make progress in the resource contention, hence it has to be removed from the FIFO queue related to R_ℓ. In this way the task has to re-perform the lock on R_ℓ at the next budget replenishment of its server. This behavior results in three effects that have to be considered:

1) The maximum spinning time plus the critical section length of R_ℓ has to fit inside the full budget of the server, that is $Q \geq (\xi_{\ell,j} + \delta_{i,\ell})$, otherwise τ_i would incur in starvation in accessing R_ℓ also when the budget is replenished.
2) Still to avoid starvation in accessing R_ℓ, local preemption has to remain disabled until the critical section on R_ℓ will be completed. In fact, if the budget exhausts during the spinning phase and other tasks are allowed to execute before τ_i at the next budget replenishment, τ_i could never have sufficient budget to perform the access to R_ℓ, so incurring in starvation. Since in the worst-case the budget exhaustion can occur just before finishing the spinning phase, the *non-preemptive blocking* imposed by τ_i will result in $2\xi_{\ell,j} + \delta_{i,\ell}$. The access to resource R_ℓ will be then guaranteed at the next budget replenishment since $Q \geq (\xi_{\ell,j} + \delta_{i,\ell})$ and no preemption on τ_i is allowed until R_ℓ will be released.
3) For the same reason explained in point 2), the computation time of τ_i will be inflated by $2\xi_{\ell,j}$, that is, the task could wait (and then spin) in the FIFO queue associated to R_ℓ twice the worst-case waiting time before accessing the resource.

B. The BCBS scheme

Consider now the *BCBS* scheme in the same scenario addressed for BCAS, as illustrated in Figure 3-b. In this case the server budget will never exhaust during the spinning phase. Hence, when the budget check has been passed, task τ_i will be able to spin for the maximum waiting time in the FIFO queue without being removed from the queue. In this way the WCET of τ_i will be inflated by $\xi_{\ell,j}$, as in the original MSRP analysis [18], that is half the inflation required by *BCAS*. Note that, as with the *BCAS* scheme, we must have $Q \geq (\xi_{\ell,j} + \delta_{i,\ell})$, otherwise the budget check will always fail. For the same reason, the *non-preemptive blocking* imposed by τ_i is $\xi_{\ell,j} + \delta_{i,\ell}$, which is less than the one under *BCAS* scheme.

However, while the *BCBS* scheme results in less WCET inflation and less *non-preemptive blocking*, it has a payback in terms of supply function of the server. As recalled at the beginning of this section, the greater the threshold used in the budget check, the lower the supply function provided by the M-BROE server. Hence, under *BCBS* we will have a lower supply with respect to *BCAS*.

C. Comparing BCBS and BCAS

Table III summarizes the differences between the BCAS and BCBS schemes identified above.

	BCAS	BCBS
Non-preemptive blocking	$2\xi_{\ell,j} + \delta_{i,\ell}$	$\xi_{\ell,j} + \delta_{i,\ell}$
WCET inflation	$2\xi_{\ell,j}$	$\xi_{\ell,j}$
Budget check threshold	$\delta_{i,\ell}$	$\xi_{\ell,j} + \delta_{i,\ell}$

Table III: Comparison between BCAS and BCBS for a task τ_i (executing on processor $\mathcal{P}_j$) accessing a global resource R_ℓ.

In spite of the presented advantages of BCBS over BCAS, we proved (see Appendix A) that it is not possible to derive an analytically dominance of a scheme over the other. Nevertheless, *BCBS* is more attractive in most practical cases, since it enforces a lower WCET inflation with respect to *BCAS*, which could result disruptive for tasks having a consistent number of critical sections.

To better evaluate the performance in schedulability of such two schemes, we carried out a set of experimental results based on synthetic workload.

Synthetic workload generation A random bandwidth α for an M-BROE server is generated in the range $[0.1, 0.95]$. Then, a random server budget Q is generated in the range $[M\mathcal{H}, 10M\mathcal{H}]$ and the server period P is then computed as $P = Q/\alpha$. A set $\mathcal{T}$ of n tasks is randomly generated using the UUniFast [24] algorithm, enforcing $2P \leq T_i \leq 10P, \forall \tau_i \in \mathcal{T}$. The overall utilization of such tasks is set to be $\psi \cdot \alpha$, where ψ denotes the server load normalized to the server bandwidth. We assume the existence of N_R global resources. For each global resource R_ℓ, a random number of tasks in the range $[1, \text{rsf} \cdot n]$ is selected to use R_ℓ. The rsf parameter (resource sharing factor) indicates "how many" tasks use global resources. For each task τ_i accessing R_ℓ, we generate $\eta_{i,\ell} \in [1, \eta^{\text{MAX}}]$ critical sections of length $\delta_{i,\ell} \in (0, \mathcal{H}]$. To obtain realistic task sets, we enforce $C_i \geq \sum_{R_\ell}(\eta_{i,\ell} \cdot \delta_{i,\ell}),\ \forall \tau_i \in \mathcal{T}$. All the random variables are generated with uniform distribution.The values used for the parameters M, N_R, η^{MAX}, ψ, rsf, n and $\mathcal{H}$ will be specified later for each specific experiment and will be reported in the caption of the graphs.

Experimental results In all the experiments presented below we performed the schedulability test presented in Section IV-C comparing the *BCAS* and *BCBS* schemes. Each value plotted in the graphs is computed as the average over a population of 5000 randomly generated task sets.

A first experiment has been carried out to measure the schedulability ratio as a function of the server load ψ. Parameter ψ has been varied from 0.25 to 1, with step 0.05. The result of this experiment is reported in Figure 4. Clearly, under both schemes the schedulability ratio decreases as ψ increases, but BCBS always outperforms BCAS.

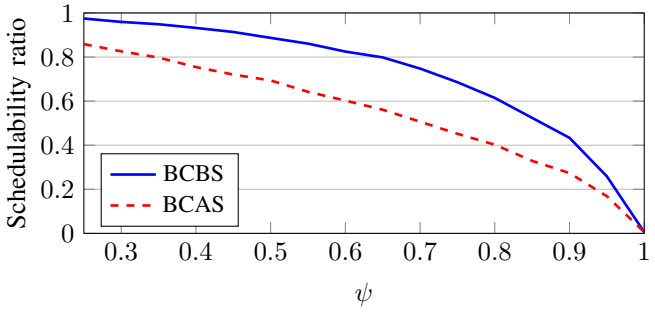

Figure 4: $M = 4, \eta^{\text{MAX}} = 4, \text{rsf} = 0.5, n \in [2, 10], N_R = 5$.

Figure 5 illustrates the result of a second experiments aimed at observing the schedulability ratio as a function of the maximum number η^{MAX} of critical sections per task. Parameter η^{MAX} has been varied from 1 to 10. As shown in the graph, the gap between BCBS and BCAS tends to increase as η^{MAX} increases, due to the greater WCET inflation required under the BCAS scheme.

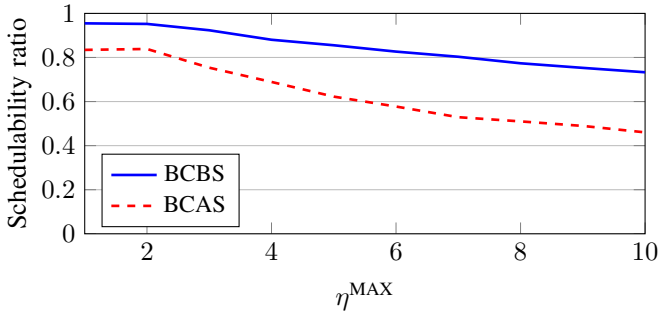

Figure 5: $M = 4, \psi = 0.5, \text{rsf} = 0.5, n \in [2, 10], N_R = 5$.

A third experiment has been carried out to measure the schedulability ratio as a function of the number of tasks executing upon the M-BROE server. The number of tasks n has been varied from 2 to 15. Results are reported in Figure 6. In this case, a significant increasing gap can be noticed between BCBS and BCAS, as n increases.

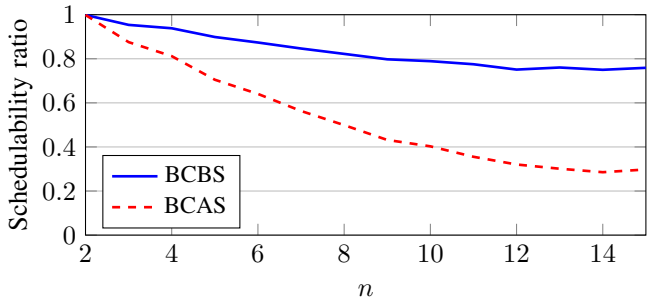

Figure 6: $M = 4, \psi = 0.5, \text{rsf} = 0.5, \eta^{\text{MAX}} = 4, N_R = 5$.

Figure 7 reports the results of a last experiment aimed at evaluating the schedulability ratio as a function of the resource sharing factor rsf, which specifies the ratio of tasks accessing global resources. This parameter has been varied from 0.1 to 1 with step 0.1. In this case, the performance improvements of BCBS over BCAS is marginal for low values of rsf, while tends to increase as rsf increases.

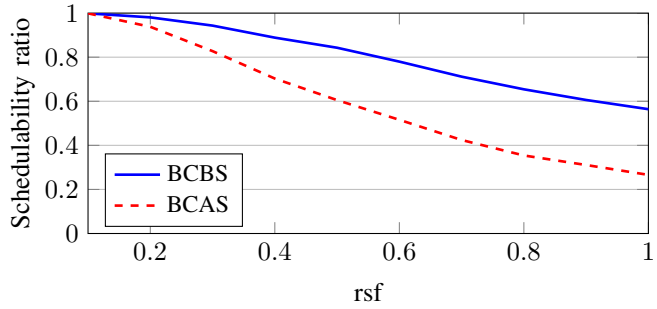

Figure 7: $M = 4, \psi = 0.5, \eta^{\text{MAX}} = 4, N_R = 5, n \in [2, 10]$.

Overall, in all the tested configurations (that are many more than the ones reported in this paper), the BCBS scheme always outperformed the BCAS scheme.

VI. Extending the component interface

As stated in Section II, each component Γ_k exports an interface to the component integrator consisting of M couples (Q_j^k, P_j^k), with $j = 1, ..., M$, representing the budget and the period of the reservation server associated to S_j^k. This interface is minimal and does not export any information related to resource sharing of a component. For this reason, the blocking bound $M\mathcal{H}$ has to be used in Equation (14) when component are integrated, assuming that *all* the reservation servers serve tasks accessing *all* resources for their *maximum* resource holding time $\mathcal{H}$. Moreover, still because we do not know which resources are used by each reservation server, *processor-local resources* will always be accounted as *global resources* through the blocking bound $M\mathcal{H}$. On the other hand, besides having evident disadvantages, the interface (Q_j^k, P_j^k) is very simple and potentially allows to change component details related to resource sharing without request a new admission procedure at the component integrator.

To reduce pessimism when components are integrated (then improving component acceptance), we analyze an extension of the component interface by including a set of parameters related to resource sharing. For each component Γ_k, the extended interface is composed of M triples $(Q_j^k, P_j^k, \vec{H}_j^k)$, with $j = 1, ..., M$. The new exported parameter $\vec{H}$ is a vector of resource holding times defined as follows:

$$\vec{H}_j^k = \{H_{j,\ell_1}^k, \ldots, H_{j,\ell_{N_R}}^k, H_{j,V_k}^k\},$$

where $R_\ell, \ldots, R_{\ell_{N_R}}$ are system resources and R_{V_k} is a *virtual* component resource used to abstract all the component resources used in Γ_k. This abstraction is provided since we do not want to export details related to each component resource.

The value of H_{V_k} is the maximum resource holding time among all the component resources used by tasks executing on virtual processor S_j^k, that are non-local resources. Component resources resulting in local resources do not have to be taken into account since they will be accessed using the classical SRP protocol; hence, such resources cannot generate blocking time for a reservation server. To exclude local resources in computing H_{V_k}, we can use the set $\mathcal{R}_k$ defined in Equation (3). Then, it is possible to compute H_{V_k} as

$$H_{j,V_k}^k = \max_{R_q, \tau_i} \{\delta_{i,q} \mid \mathcal{S}(\tau_i) = S_j^k \wedge R_q \in \mathcal{R}_k\}.$$

Similarly, for each system resource R_ℓ, we compute $H_{j,\ell}^k$ as

$$H_{j,\ell}^k = \max_{\tau_i} \{\delta_{i,\ell} \mid \mathcal{S}(\tau_i) = S_j^k\}.$$

Now, using the extended interface $(Q_j^k, P_j^k, \vec{H}_j^k)$, it is possible to improve the component integration analysis of Section IV-D by computing a tighter bound on server blocking for Equation (14).

As stated in Section IV-D, the blocking B_j^k imposed on each reservation server associated to S_j^k is the maximum between the *non-preemptive server blocking* $B_j^{k,\text{NP}}$ (caused by global resources) and the H-SRP blocking $B_j^{k,\text{H-SRP}}$ (caused by processor-local resources). Hence we have

$$B_j^k = \max\{B_j^{k,\text{NP}}, B_j^{k,\text{H-SRP}}\}.$$

Given an assignment of reservation servers to processor, thanks to the extended interface, it is possible to identify which resource results in a global resource and which resource results in a processor-local resource.[2]

The H-SRP blocking for a reservation server S_j^k (please refer to [3], [6] for further details) can be computed as:

$$B_j^{k,\text{H-SRP}} = \max_{l,r:P_r^l > P_k^j} \left\{ H_{r,\ell}^l \mid \exists S_v^u, H_{v,\ell}^u > 0 \wedge P_v^u \leq P_k^j \right\}_0. \tag{15}$$

where servers S_j^k, S_r^l and S_v^u are assigned to the same processor (i.e., $\mathcal{P}(S_j^k) = \mathcal{P}(S_r^l) = \mathcal{P}(S_v^u)$).

Non-preemptive blocking for a server S_j^k assigned to processor $\mathcal{P}_x$ can be computed by looking at the maximum non-preemptive section in accessing global resources, that is:

$$B_j^{k,\text{NP}} = \max_{l,r:P_r^l > P_k^j} \left\{ \xi_{\ell,x} + H_{r,\ell}^l \mid \mathcal{P}(S_r^l) = \mathcal{P}_m \right\}_0,$$

where the spinning time $\xi_{\ell,x}$ for R_ℓ on processor $\mathcal{P}_x$ can be bounded (exploiting the parameters of the extended interface) by

$$\xi_{\ell,x} \leq \sum_{\mathcal{P}_x \neq \mathcal{P}_m} \max\{H_{r,\ell}^l \mid \mathcal{P}(S_r^l) = \mathcal{P}_m\}.$$

Finally, it is possible to refine the integration analysis of Equation (14) with the following schedulability test:

$$\forall S_j^k : \mathcal{P}(S_j^k) = \mathcal{P}_m, \sum_{\substack{r,l:P_r^l \leq P_j^k \\ \wedge \\ \mathcal{P}(S_r^l) = \mathcal{P}_m}} \frac{Q_r^l}{P_r^l} + \frac{B_j^k}{P_j^k} \leq 1. \tag{16}$$

VII. Virtual processor design and allocation

Although there is no space in this paper to discuss how to partition a given application over a virtual multiprocessor platform and how to select the best reservation parameters for each component, we would like to point out some issues that are worth to be further investigated.

For our multi-processor reservation framework to work properly, any assignment of tasks to virtual processors is valid provided the two following conditions are met: (i) the schedulability test in Equation (8) is passed for each virtual processor; and (ii) the bound on resource holding times of *component resources* expressed by Equation (2) is satisfied.

Whether allocating the application tasks on a small set of "heavy" reservations, or distribute them among a larger set of "lighter" servers is a design choice that is strictly connected to the mapping policy adopted at the component integration level. In particular, these two alternative strategies may lead to different performance depending on the application scenario.

[2]Please note that a reservation server which is not using a resource R_ℓ will have $H_{j,\ell}^k = 0$, hence it is possible to identify if a resource will be only used by reservation server assigned to the same processor or not.

Partitioning tasks into a smaller number "heavy" reservations allows more tasks to be scheduled within each reservation server, potentially reducing the number of virtual processors accessing a component resource. This allows reducing the spinning time and the non-preemptive blocking due to component resources. However, having heavier servers may impose more constraints on the component integrator, which has to allocate servers to physical cores satisfying the schedulability test given in Equation (14) or (16). As in a bin packing problem, larger bins are more difficult to accommodate in the available space left by the previous allocations, leaving a considerable free utilization on each core.

Conversely, using a higher number of lighter servers makes the integration phase easier, but satisfying the local schedulability test on each virtual processor and the requirements on the resource holding time of component resources becomes more difficult. This happen because: (i) a fewer number of tasks may be assigned to each virtual processor; and (ii) tasks accessing the same resource are likely to be spread among different virtual processors, increasing the spinning and blocking terms, leading to a larger resource holding time that may exceed the allowed parameter $M\mathcal{H}$.

Similar considerations can be made at the component integration level. These problems can be formulated as an ILP optimization problem.

We plan to analyze these design decisions in a future work, where we will propose an integrated component interface design and component integration for a set of applications scheduled with the M-BROE framework.

VIII. Conclusions

This paper presented a framework for supporting component-based design in multiprocessor real-time systems under partitioned scheduling. Each component is statically partitioned and guaranteed over a set of virtual processors; then, an interface representing the computational requirements of such virtual processors is provided to a component integrator. Virtual processors are implemented through reservation servers. Components are integrated assigning virtual processors to the physical processors available in the computing platform. To support the proposed framework, a novel scheduling mechanism, called M-BROE, has been presented to enable resource sharing among independently developed real-time applications running on the multiprocessor system under hierarchical reservations. The resource access protocol makes use of FIFO non-preemptive spinlocks to ensure mutual exclusion of the resources shared among different processors. Blocking factors have been estimated and a schedulability test has been derived under partitioned EDF scheduling.

Considering that the presented protocol is based on a budget check performed when a task attempts to enter a critical section, two budget checking schemes have been analyzed: budget check before spinning (BCBS) and budget check after spinning (BCAS). Although it has been proved that none of the two schemes dominates the other, extensive simulations experiments on synthetic generated task sets showed that BCBS performs significantly better than BCAS in most practical situations.

Another contribution of the paper was to propose two component interfaces: a simple one that only requires the knowledge of the parameters (budget and period) of each reservation, and a more detailed one that also exports the knowledge of the used resources. The simpler interface introduces more pessimism in the analysis at component integration, whereas the second removes such a pessimism by taking into account the knowledge about resource usage and thus allowing computing a more precise bound on the blocking terms.

Acknowledgements

This work has been partially supported by the 7th Framework Programme JUNIPER (FP7-ICT-2011.4.4) project, founded by the European Community under grant agreement n. 318763, and the 7th Framework Programme P-SOCRATES (FP7/2007-2013), grant agreement n. 611016.

References

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

[2] L. Abeni and G. Buttazzo, "Resource reservations in dynamic real-time systems," *Real-Time Systems*, vol. 27, no. 2, pp. 123–165, 2004.

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

[4] 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.

[5] M. Behnam, I. Shin, T. Nolte, and M. Nolin, "SIRAP: a synchronization protocol for hierarchical resource sharing in real-time open systems," in *Proc. of the 7th ACM & IEEE International Conference on Embedded Software (EMSOFT 2007)*, Salzburg, Austria, October 1-3, 2007.

[6] 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.

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

[8] ARINC, *ARINC 651: Design Guidance for Integrated Modular Avionics*. Airlines Electronic Engineering Committee (AEEC), November 1991.

[9] H. Leontyev and J. Anderson, "A hierarchical multiprocessor bandwidth reservation scheme with timing guarantees," vol. 43, no. 1, September 2009, pp. 60–92.

[10] E. Bini, G. Buttazzo, J. Eker, S. Schorr, R. Guerra, G. Fohler, K.-E. Arzen, V. R. Segovia, and C. Scordino, "Resource management on multicore systems: The ACTORS approach," *IEEE Micro*, vol. 31, no. 3, pp. 72–81, May-June 2011.

[11] R. Inam, N. Mahmud, M. Behnam, T. Nolte, and M. Sjdin, "The multi-resource server for predictable execution on multi-core platforms," in *Proc. of the 20th IEEE Real-Time and Embedded Technology and Applications Symp. (RTAS 2014)*, Berlin, Germany, April 15-17, 2014.

[12] F. Nemati, M. Behnam, and T. Nolte, "Independently-developed real-time systems on multi-cores with shared resources," in *Proceedings of 23th Euromicro Conference on Real-time Systems (ECRTS11)*, 2011.

[13] S. Afshar, N. Khalilzad, F. Nemati, and T. Nolte, "Resource sharing among prioritized real-time applications on multiprocessors," in *6th International Workshop on Compositional Theory and Technology for Real-Time Embedded Systems*, 2013.

[14] T. P. Baker, "Stack-based scheduling for realtime processes," *Real-Time Systems*, vol. 3, no. 1, pp. 67–99, April 1991.

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

[16] A. Biondi, A. Melani, and M. Bertogna, "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, 18-20 June, 2014.

[17] 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.
[18] P. Gai, G. Lipari, and M. D. Natale, "Minimizing memory utilization of real-time task sets in single and multi-processor systems-on-a-chip," in *Proceedings of IEEE Real-Time Systems Symposium 2011*.
[19] A. Wieder and B. Brandenburg, "On spin locks in AUTOSAR: Blocking analysis of FIFO, unordered, and priority-ordered spin locks," in *Proceedings of the 34th IEEE Real-Time Systems Symposium (RTSS'2013)*, December 2013, pp. 45–56.
[20] A. Biondi, G. Buttazzo, and M. Bertogna, "Schedulability analysis of hierarchical real-time systems under shared resources," RETIS Lab, Scuola Superiore Sant'Anna, Italy, Tech. Report TR-13-01, July 2013.
[21] B. Brandenburg, "Scheduling and locking in multiprocessor real-time operating systems," in *Ph.D. dissertation, The University of North Carolina at Chapel Hill*, 2011.
[22] S. Baruah, "Resource sharing in EDF-scheduled systems: a closer look," in *Proceedings of the 27th IEEE Real-Time Systems Symposium (RTSS'06)*, Rio de Janeiro, Brazil, December 5-8, 2006.
[23] A. Biondi, A. Melani, M. Bertogna, and G. Buttazzo, "Optimal design for reservation servers under shared resources," in *Proceedings of the 26th Euromicro Conference on Real-Time Systems (ECRTS 14)*, Madrid, Spain, 9-11 July, 2014.
[24] E. Bini and G. C. Buttazzo, "Measuring the performance of schedulability tests," *Real-Time Systems*, vol. 30, no. 1-2, pp. 129–154, 2005.

APPENDIX A
ANALITYCAL COMPARISON BETWEEN BCBS AND BCAS

Lemma 2: The BCBS scheme does not analytically dominate the BCAS scheme.

Proof: Consider an M-BROE server S having budget Q and period P, leading to a bandwidth $\alpha = Q/P$ and a worst-case service delay $\Delta = 2(P-Q)$. Consider a simple task set composed of 2 tasks, τ_1 and τ_2, executing upon the M-BROE server under local EDF scheduling.

Task τ_2 has a single critical section of length $\delta_{2,\ell}$ on a global resource R_ℓ, while task τ_1 does not have critical sections. Let also ξ_ℓ be the maximum spinning time for accessing R_ℓ. Under the BCBS scheme, the budget check threshold will be $X^{\text{BCBS}} = \delta_{2,\ell} + \xi_\ell$, while under the BCAS scheme it will be $X^{\text{BCAS}} = \delta_{2,\ell}$. In addition, suppose that $Q > 2X^{\text{BCBS}}$. The non-preemptive blocking for BCBS is equal to $B^{\text{BCBS}} = \delta_{2,\ell} + \xi_\ell$, while for BCAS is equal to $B^{\text{BCAS}} = \delta_{2,\ell} + 2\xi_\ell$. Such a blocking factor is imposed from time $t \geq D_1$. For task τ_1, we set $C_1 = 2Q - 2X^{\text{BCBS}} + \epsilon - B^{\text{BCBS}}$, $D_1 = \Delta + P + 2Q - 2X^{\text{BCAS}} + \epsilon$, and $T_1 = 10P > D_1$. For task τ_2, we set $C_2 = \delta_{2,\ell} + \epsilon$ and $D_2 = T_2 = T_1$. Parameter ϵ is an arbitrary small number such that $0 < \epsilon < \xi_\ell$ (e.g., $\epsilon = 1$).

The schedulability of the task set has to be only checked in two time instants, that are $t_1 = D_1$ and $t_2 = T_1 = T_2$. Let us consider the BCBS scheme first: in this case, we have $\mathsf{dbf}(t_1) = C_1$ and $\mathsf{sbf}^B(S, t_1, X^{\text{BCBS}}) = 2Q - 2X^{\text{BCBS}}$. Therefore, verifying the schedulability condition of Equation (8) we obtain

$$B^{\text{BCBS}} + \mathsf{dbf}(t_1) \leq \mathsf{sbf}^B(S, t_1, X^{\text{BCBS}}),$$
$$2Q - 2X^{\text{BCBS}} + \epsilon \leq 2Q - 2X^{\text{BCBS}}$$

leading to $\epsilon \leq 0$, hence the test fails.

Now, consider the BCAS scheme. In this case, we have $\mathsf{dbf}(t_1) = C_1$ and $\mathsf{sbf}^B(S, t_1, X^{\text{BCAS}}) = 2Q - 2X^{\text{BCAS}}$. Replacing such terms in the schedulability test we obtain:

$$B^{\text{BCAS}} + \mathsf{dbf}(t_1) \leq \mathsf{sbf}^B(S, t_1, X^{\text{BCAS}}),$$
$$B^{\text{BCAS}} + 2Q - 2X^{\text{BCBS}} + \epsilon - B^{\text{BCBS}} \leq 2Q - 2X^{\text{BCAS}},$$
$$B^{\text{BCAS}} + 2X^{\text{BCAS}} - 2X^{\text{BCBS}} + \epsilon - B^{\text{BCBS}} \leq 0,$$
$$B^{\text{BCAS}} + 2X^{\text{BCAS}} - 3(\delta_{2,\ell} + \xi_\ell) + \epsilon \leq 0,$$
$$\delta_{2,\ell} + 2\xi_\ell + 2\delta_{2,\ell} - 3(\delta_{2,\ell} + \xi_\ell) + \epsilon \leq 0,$$
$$2\xi_\ell - 3\xi_\ell + \epsilon \leq 0.$$

Hence, the schedulability check at time t_1 succeeds. Following a similar reasoning it is easy to show that the check also succeeds at time t_2. ■

Lemma 3: The BCAS scheme does not analytically dominate the BCBS scheme.

Proof: Consider an M-BROE server S having budget Q and period P, leading to a bandwidth $\alpha = Q/P$ and a worst-case service delay $\Delta = 2(P-Q)$. Consider a simple task set composed of 2 tasks, τ_1 and τ_2, executing upon the M-BROE server under local EDF scheduling.

Task τ_1 has a single critical section of length $\delta_{1,\ell}$ on a global resource R_ℓ, while task τ_2 does not have critical sections. Let also ξ_ℓ be the maximum spinning time for accessing R_ℓ. Under the BCBS scheme, the budget check threshold will be $X^{\text{BCBS}} = \delta_{1,\ell} + \xi_\ell$, while under the BCAS scheme it will be $X^{\text{BCAS}} = \delta_{1,\ell}$. In addition, suppose that $Q > 2X^{\text{BCBS}}$. The non-preemptive blocking for BCBS is equal to $B^{\text{BCBS}} = \delta_{1,\ell} + \xi_\ell$, while for BCAS is equal to $B^{\text{BCAS}} = \delta_{2,\ell} + 2\xi_\ell$. Such a blocking factor is imposed from time $t \geq D_1$. For task τ_1, we set $C_1 = Q - X^{\text{BCBS}} - B^{\text{BCBS}} - \xi_\ell - \epsilon$, $D_1 = \Delta + Q - X^{\text{BCAS}} + \epsilon$, and $T_1 = 10P > D_1$. For task τ_2, we set $C_2 = \epsilon$ and $D_2 = T_2 = T_1$. Parameter ϵ is an arbitrary small number such that $0 < \epsilon < \xi_\ell$ (e.g., $\epsilon = 1$).

The schedulability of the task set has to be only checked in two time instants, that are $t_1 = D_1$ and $t_2 = T_1 = T_2$. Let us consider the BCBS scheme first: in this case, we have $\mathsf{dbf}(t_1) = C_1 + \xi_\ell$ and $\mathsf{sbf}^B(S, t_1, X^{\text{BCBS}}) = Q - X^{\text{BCBS}}$. Therefore, verifying the schedulability condition of Equation (8) we obtain

$$B^{\text{BCBS}} + \mathsf{dbf}(t_1) \leq \mathsf{sbf}^B(S, t_1, X^{\text{BCBS}}),$$
$$Q - X^{\text{BCBS}} - \epsilon \leq Q - X^{\text{BCBS}},$$

leading to $\epsilon \geq 0$, hence the test is passed. Following a similar reasoning, it is easy to show that the check also succeeds at time t_2.

Now, consider the BCAS scheme. In this case, we have $\mathsf{dbf}(t_1) = C_1 + 2\xi_\ell$ and $\mathsf{sbf}^B(S, t_1, X^{\text{BCAS}}) = Q - X^{\text{BCAS}}$. Replacing such terms in the schedulability test we obtain:

$$B^{\text{BCAS}} + Q - X^{\text{BCBS}} - \epsilon - B^{\text{BCBS}} + \xi_\ell \leq \mathsf{sbf}^B(S, t_1, X^{\text{BCAS}}),$$
$$B^{\text{BCAS}} + Q - X^{\text{BCBS}} - \epsilon - B^{\text{BCBS}} + \xi_\ell \leq Q - X^{\text{BCAS}},$$
$$B^{\text{BCAS}} - X^{\text{BCBS}} - \epsilon - B^{\text{BCBS}} + \xi_\ell \leq -X^{\text{BCAS}},$$
$$\delta_{1,\ell} + 3\xi_\ell - 2(\xi_\ell + \delta_{1,\ell}) - \epsilon \leq -\delta_{1,\ell},$$
$$\xi_\ell - \epsilon \leq 0,$$

Hence, the test fails for t_1. ■

APPENDIX B
ADDITIONAL EXPERIMENTAL RESULTS

Figures 8, 9, 10, 11, 12 and 13 report additional experimental results comparing the *BCBS* and *BCAS* schemes. The values for the configuration parameters used for such experiments are reported in the caption of the figures.

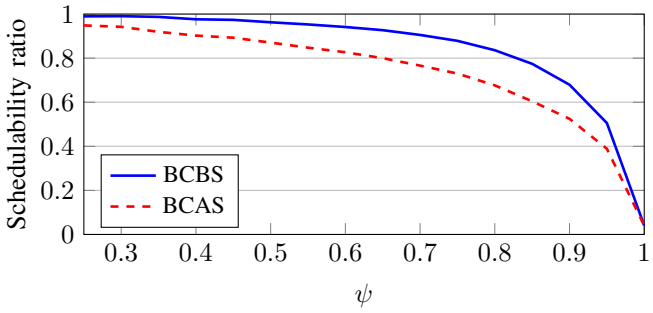

Figure 8: $M = 4, \eta^{\text{MAX}} = 4, \text{rsf} = 0.3, n \in [2, 10], N_R = 5$.

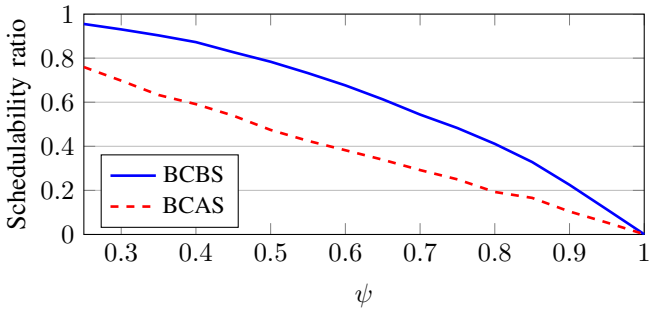

Figure 9: $M = 4, \eta^{\text{MAX}} = 4, \text{rsf} = 0.75, n \in [2, 10], N_R = 5$.

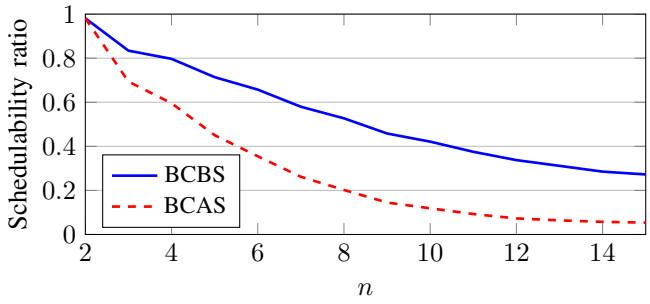

Figure 10: $M = 4, \eta^{\text{MAX}} = 4, \text{rsf} = 0.75, \psi = 0.75, N_R = 5$.

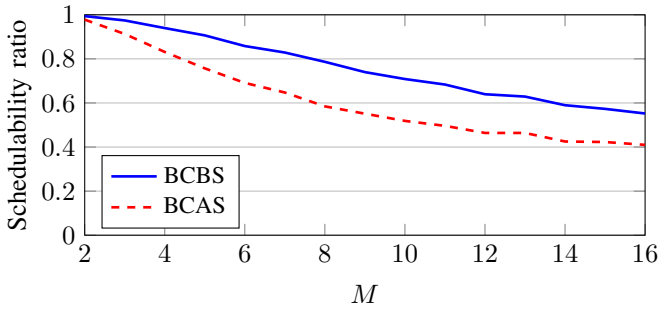

Figure 11: $\eta^{\text{MAX}} = 4, \text{rsf} = 0.3, \psi = 0.6, n \in [2, 10], N_R = 5$.

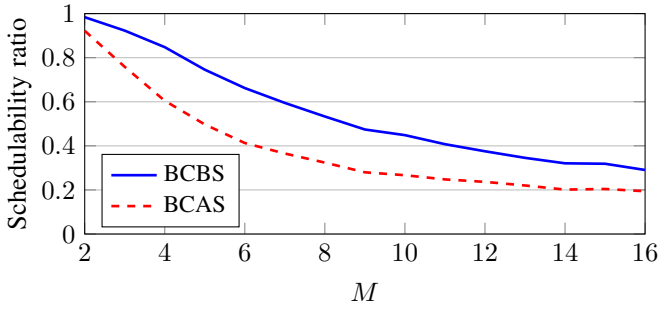

Figure 12: $\eta^{\text{MAX}} = 4, \text{rsf} = 0.5, \psi = 0.6, n \in [2, 10], N_R = 5$.

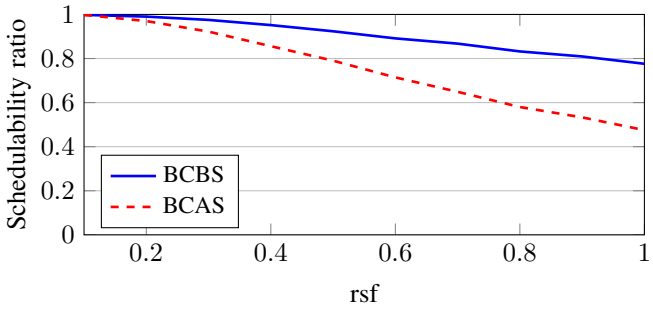

Figure 13: $M = 4, \eta^{\text{MAX}} = 2, \psi = 0.6, n \in [2, 10], N_R = 5$.