Noname manuscript No.
(will be inserted by the editor)

A Design Flow for Supporting Component-based Software Development in Multiprocessor Real-Time Systems

Alessandro Biondi · Giorgio Buttazzo · Marko Bertogna

the date of receipt and acceptance should be inserted later

Abstract Component-based software development established as an effective technique to cope with the increasing complexity of modern computing systems. In the context of real-time systems, the M-BROE framework has been recently proposed to efficiently support component-based development of real-time applications on multiprocessor platforms in the presence of shared resources. The framework relies on a two-stage approach where software components are first partitioned upon a virtual multiprocessor platform and are later integrated upon the physical platform by means of component interfaces that abstract from the internal details of the applications.

This work presents a complete design flow for the M-BROE framework. Starting from a model of software components, a first method is proposed to partition applications to virtual processors and perform a synthesis of multiple component interfaces. Then, a second method is proposed to support the integration of the components by allocating virtual processors to physical processors. Both methods take resource sharing into account. Experimental results are also presented to evaluate the proposed methodology.

1 Introduction

The increasing need of adding new software functions in customer products is pushing software companies towards a hierarchical design approach, where multiple applications, initially executed on dedicated hardware units, are being integrated on the same hardware platform. For instance, in automotive

Alessandro Biondi
Scuola Superiore Sant'Anna, Pisa, Italy
E-mail: alessandro.biondi@santannapisa.it Giorgio Buttazzo
Scuola Superiore Sant'Anna, Pisa, Italy
E-mail: giorgio.buttazzo@santannapisa.it Marko Bertogna
University of Modena and Reggio Emilia, Modena, Italy
E-mail: marko.bertogna@unimore.it

systems such a hierarchical approach is motivated to contain the total number of electronic control units (ECUs) installed in a car, which implies a significant reduction of used space, weight, energy, and cost [26].

On the other hand, when multiple applications share the same hardware platform, new problems must be solved to make the hierarchical approach effective and predictable. One of them is caused by the reciprocal interference among concurrent software activities, which may introduce unbounded delays and cause unpredictable performance degradation [27].

An effective approach for containing the interference among concurrently running applications is the resource reservation mechanism [1, 25]. According to this approach, each application is executed within a dedicated processor partition, implemented by a *reservation server*. A reservation server S_k is a time provisioning mechanism that allocates a budget Q_k for the application every period P_k. In this case, the bandwidth reserved to an application results to be $\alpha_k = Q_k/P_k$. The reservation mechanism must guarantee that the served application receives a given fraction of the processor bandwidth, but at the same time it cannot consume more than the allocated amount, thus protecting the other applications from possible overruns (*temporal isolation*).

In the case in which application tasks make use of mutually exclusive resources shared between reservations (such as I/O devices or global memory buffers) the isolation property could be broken when the server budget expires within a critical section. In this case, in fact, an extra delay would be added to the tasks blocked on the same resource to wait not only for the release of the lock, but also for the next budget replenishment.

To solve this problem, various approaches have been proposed in the literature [8, 9, 11, 20]. Among them, thanks to an improved schedulability analysis [16], the BROE protocol [11] proposed by Bertogna, Fisher and Baruah has been found to perform best.

The BROE protocol, originally developed for uniprocessor systems, has been recently extended by Biondi et al. [15] into M-BROE to support the development of component-based hierarchial systems on multiprocessor platforms in the presence of shared resources. In the M-BROE framework (reviewed in Section 2), the tasks of a software component are statically allocated to virtual processors (implemented via reservation servers), which are in turn allocated to the physical processors at component-integration time. The resulting infrastructure relies on partitioned hierarchical scheduling and non-preemptive FIFO spin locks to regulate the access to shared resources.

Although the authors fully characterized the M-BROE protocol and provided the schedulability analysis of components on given reservation servers, the problems of partitioning applications on virtual processors and defining reservation parameters were not addressed in [15].

Other authors solved task partitioning in multicore systems using *Integer Linear Program* (ILP) formulations [5,7], but without considering reservations, nor resource sharing. Buttazzo et al. [19] addressed the problem of partitioning parallel applications upon reservation servers for platform virtualization and Khalilzad et al. [23] studied the problem allocating component interfaces

in multiprocessor systems under partitioned EDF scheduling, but in both the works no resource sharing has been considered. Wieder and Brandenburg [29] presented an optimal ILP-based partitioning strategy for fixed priority scheduling with shared resources and Al-Bayati et al. [2] addressed the same problem using heuristic approaches.

However, none of these works addressed partitioning methodologies under reservation servers while taking resource sharing into account. Resource sharing further complicates task partitioning problems (an example will be shown in Section 5), determining the need for ad-hoc approaches that explicitly take into account blocking times.

Contributions. This paper fills this gap by proposing the following contributions.

- A methodology based on a *Mixed-Integer Linear Program* (MILP) formulation is proposed for partitioning applications on virtual processors taking shared resources into account. Once an allocation is found, the synthesis of component interfaces is performed to find the optimal reservation parameters that guarantee the schedulability of the task set.
- Another MILP formulation is presented for allocating virtual processors to physical processors depending on the component interfaces. This formulation is able to find the optimal allocation with respect to the adopted schedulability analysis.

The combination of these two contributions provides a complete design flow for supporting component-based software development within the M-BROE framework.

Paper structure. The rest of the paper is organized as follows. Section 2 presents the system model and the M-BROE framework. Section 3 summarizes the schedulability results derived for the M-BROE framework. Section 4 defines the problem of partitioning the tasks and designing the virtual processors for the considered framework, while Section 5 presents a solution for such a problem based on MILP. Section 6 reports an MILP formulation for integrating the virtual processors of all components upon the physical platform. Section 7 presents some experimental results for evaluating the proposed methodology. Finally, Section 8 concludes the paper.

This paper extends a preliminary version of this work [14] published in RTNS 2016 by **(i)** considering an extended component interface that includes bounds on holding times of shared resources, **(ii)** presenting a new approach for performing component integration, which copes with the extended interface and allows handling two interfaces for each component (Section 6), **(iii)** reporting new experimental results based on a strengthen experimentation (Section 7), and **(iv)** presenting a simplified MILP formulation for task partitioning (Section 5).

2 Framework and Modeling

2.1 System model

We consider a system composed of N software components $\Gamma_1, \Gamma_2, \ldots, \Gamma_N$, also referred to as applications. Each *software component* Γ_k consists of a set $\mathcal{T}_k$ of n_k real-time periodic or sporadic tasks. Each task $\tau_i \in \mathcal{T}_k$ 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 runs on a multiprocessor platform consisting of M identical processors, each denoted as $\mathcal{P}_m$, with $m = 1, \ldots, M$. 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 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 $\tau_i \in \mathcal{T}_k$ is assigned is denoted as $\mathcal{S}(\tau_i)$. The set of tasks allocated to S_j^k is denoted as $\Gamma(S_j^k)$.

Each component exports a *component interface* (defined below). Basing on component interfaces, a *component integrator* is responsible for admitting or rejecting applications and statically assigning each reservation server to a specific physical processor. Whenever we need to discuss a specific component allocation, the processor on which server S_j^k is assigned is denoted as $\mathcal{P}(S_j^k)$. For the sake of simplicity, this paper assumes that $\mathcal{M} = M$. The framework described above is illustrated in Figure 1 for a platform of 4 processors.

Tasks can share resources through mutually exclusive critical sections. At a component level, we distinguish between *component resources*, accessed only by tasks belonging to the same component and *system resources*, accessed by tasks belonging to different components. When needed, we denote with $\mathcal{R}^C$ and $\mathcal{R}^S$ the sets of component and system resources, respectively.

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. We assume to have $\eta_{i,\ell} = 0$ if a task τ_i does not access resource R_ℓ.

For a resource R_ℓ accessed by a task τ_i, the *resource holding time* $H_{i,\ell}$ 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 non preemptively, its resource holding time is equal to the critical section length (i.e., $H_{i,\ell} = \delta_{i,\ell}$), otherwise it must include all possible preemption delays occurring within the critical section [10]. A component Γ_k can only be admitted in the system if all its tasks have a resource holding time bounded by $\mathcal{H}$, that is, if

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

For *component resources* R_q accessed by tasks allocated to different virtual processors, the sum of the maximum resource holding times of R_q from each

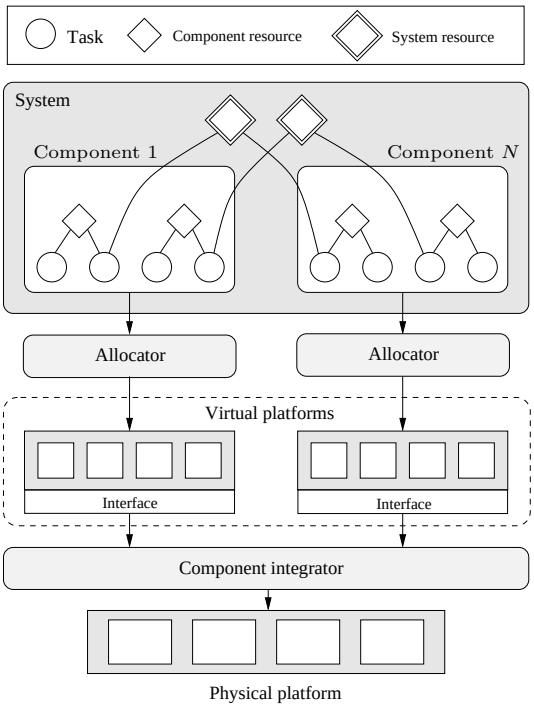

Fig. 1 Overview of the proposed hierarchical framework. Tasks can access component resources (shared within the same component) and system resources (accessible by all the components). The tasks of the components are allocated to virtual platforms, which are in turn allocated to the physical platform at component integration time depending on their corresponding interfaces.

virtual processor must be 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 executing on different virtual processors, formally defined as

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

2.2 Component Interface

Each component Γ_k is abstracted through a *component interface* that consists of $\mathcal{M}$ triples $(Q_j^k, P_j^k, \mathbf{H}_j^k)$, with $j = 1, \ldots, \mathcal{M}$, where

- Q_j^k is the *budget* of the j^{th} reservation server of component Γ_k;
- P_j^k is the *period* of the j^{th} reservation server of component Γ_k;

– $\mathbf{H}_j^k = \{H_{j,\ell_1}^k, \ldots, H_{j,\ell_{N_R}}^k, H_{j,V_k}^k\}$ is a vector of resource holding times, 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.

The values in the vector of resource holding times are defined as follows: $H_{j,\ell}^k$ is the maximum resource holding times for system resource R_ℓ across all the tasks allocated to the j^{th} reservation server of component Γ_k, that is

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

while H_{j,V_k}^k is the maximum resource holding times across all the component resources that are **(i)** accessed by more than one reservation server and **(ii)** accessed by tasks allocated to the j^{th} reservation server of component Γ_k, that is

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

The abstraction of virtual component resources is provided to not export details related to each component resource at the component integration stage, thus hiding the internal usage of shared resources performed by each component. Note that this abstraction does not consider component resources that are only accessed by a single reservation server, as the contention for such resources is confined within a server and hence is meaningless for component integration purposes.

2.3 Scheduling infrastructure

Tasks allocated to a virtual processor are handled by a *local scheduler*, which can be any *fixed-priority* (FP) algorithm or *earliest-deadline first* (EDF) [24]. Tasks may include non-preemptive regions in which preemption is disabled for the local scheduler. Each virtual processor is implemented by an M-BROE server [15] and the various M-BROE servers are scheduled under *partitioned EDF* (P-EDF) scheduling on the M processors.

Once reservation servers are mapped to physical processors, three types of shared resources can be distinguished: *Local resources*, shared only by tasks handled by the same server; *Processor-local resources*, shared only by tasks executing on the same processor, but on different servers; *Global resources*, shared by tasks executing on different processors.

Local resources are accessed through the SRP [3] protocol, while *processor-local resources* are accessed by the H-SRP [20] protocol in conjunction with M-BROE, in a *local non-preemptive manner*. Finally, *global resources* are accessed by the MSRP [22] protocol in conjunction with M-BROE.

3 Summary of schedulability results

To make this work self consistent, this section summarizes the schedulability results derived for the M-BROE framework. In particular, after recalling how

to derive blocking factors, we summarize the schedulability tests used for local (tasks guarantee upon virtual processors) and global (virtual processors guarantee upon the physical platform) analysis. Please refer to [15] for further details.

3.1 Resource sharing

Under the M-BROE framework, global resources are protected by non-preemptive FIFO spinlocks. If a task τ_i wants to use a global resource locked from a task on another processor, τ_i starts spinning non-preemptively until the resource is granted. Critical sections on global resources are also executed non-preemptively and simultaneous requests from different processors are served in FIFO order.

To analyze blocking times related to spinlocks we rely on the MSRP [22] analysis: please note that although an improved analysis for spinlocks has been proposed in [28], it cannot be directly used for the M-BROE framework for several reasons as explained in [15] (page 4). According to the MSRP analysis, a bound on the maximum spinning time, denoted as *remote blocking*, is computed for each global resource R_ℓ accessed from a given processor and used to inflate the WCET of the tasks using R_ℓ. Also, non-preemptive spinning and non-preemptive access to global resources introduce a *non-preemptive blocking* factor that must be accounted for each task. The access to *local* and *processor local* resources is regulated by the SRP [3] and the H-SRP [17] protocols, generating *local blocking* and additional *non-preemptive blocking*, respectively.

In the following, we first provide an upper-bound for the spinning time and then report the expressions for computing *remote blocking*, *non-preemptive blocking* and *local blocking*.

Upper-bound for the spinning time. According to Lemma 1 in [15], a safe upper bound on the spinning time $\xi_{\ell,j}$ related to *system* resources R_ℓ is given by $\xi_{\ell,j} \leq (M-1)\mathcal{H}$.

When analyzing a given component, critical section lengths of component resources 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 the physical processor on which they will be executed (since it depends on the allocation performed at the stage of component integration). For this reason, a safe bound on the spinning time can be computed by assuming that all virtual processors of a component will be assigned to different physical processors. In this case, an upper bound on the spinning time $\xi_{\ell,j}$ for a *component resource* R_ℓ in Γ_k can be computed as

$$\xi_{\ell,j} \leq \sum_{S_j^k \neq S_m^k} \max\{\delta_{i,\ell} \mid \mathcal{S}(\tau_i) = S_m^k\}. \tag{4}$$

This fact imposes also that *processor-local resources* have to be always accounted as *global resources* to capture the worst-case in the local analysis.

Remote blocking. An upper-bound ξ_i on the *remote blocking* for task τ_i is computed by accounting for the maximum spinning time on each critical section of τ_i, with $\mathcal{P}(\mathcal{S}(\tau_i)) = \mathcal{P}_j$, that is:

$$\xi_i = \sum_{R_\ell} \{\eta_{i,\ell} \cdot \xi_{\ell,j} \mid R_\ell \text{ used by } \tau_i\}. \tag{5}$$

Non-preemptive blocking. Such a blocking is bounded by the longest non-preemptive section that cause arrival blocking. A non-preemptive section occurs when a task accesses a global resource and comprises (i) a potential non-preemptive spinning phase, which is originated by remote blocking, and (ii) the non-preemptive execution of critical sections for global resources. Note that, during the non-preemptive spinning phase, remote blocking is *transitively* transformed into arrival blocking: therefore, such phenomenon is also referred to as *transitive arrival blocking*.

Under local EDF scheduling, assuming that tasks τ_k and τ_i execute on processor $\mathcal{P}_j$, it can be computed as

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

Local blocking. The SRP blocking factor for task τ_i due to local resources is denoted by B_i^L and is given by critical sections of resources that are locked by tasks τ_L with deadlines greater than D_i and shared with tasks τ_H with deadlines less than or equal to D_i. Formally,

$$B_i^L = \max \{\delta_{L,\ell} \mid D_H \leq D_i < D_L \wedge \tau_L \text{ and } \tau_H \text{ use } R_\ell\}. \tag{7}$$

Since *non-preemptive blocking* and *local blocking* occur at the task's release, the resulting blocking is called *arrival blocking* and can be computed as

$$B_i = \max\{B_i^{\text{NP}}, B_i^L\}. \tag{8}$$

3.2 Local analysis

Using the processor demand criterion extended to include resource sharing [4], a task set $\mathcal{T}_k$ is schedulable by EDF under the M-BROE server if:

$$\forall t > 0 \quad B(t) + \mathsf{dbf}(t) \leq \mathsf{sbf}(t) \tag{9}$$

where

$$\mathsf{dbf}(t) = \sum_{\tau_i \in \mathcal{T}_k} \left(\left\lfloor \frac{t - D_i}{T_i} \right\rfloor + 1 \right)_0 (C_i + \xi_i), \tag{10}$$

$$B(t) = \max\{B_i \mid D_i \leq t\}, \tag{11}$$

and $\mathsf{sbf}(t)$ is the *supply bound function* for the M-BROE server reported in [15] and $(x)_0$ denotes $\max(0, x)$. Note that the $\mathsf{dbf}(t)$ in Equation (10) takes into account the computation time inflation ξ_i due to *remote blocking*.

The M-BROE framework also supports local fixed-priority scheduling; however, to date an analysis under local fixed-priority is not yet available in the published literature, hence this paper focuses on local EDF scheduling only. Besides this limitation, the authors are confident that the approach proposed in the following sections can also be applied under fixed-priority scheduling.

3.3 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 EDF scheduling on the M physical processors.

Under the M-BROE framework, also the reservations exported by the components can incur in arrival blocking, e.g., a server S_j^k may be blocked whenever there is a task executing on another server S_r^l that is non-preemptively spinning for accessing a global resource. Such blocking times must hence be taken into account in the schedulability analysis performed at the component integration stage, and can be bounded in an analogous manner as reported in Section 3.1 by considering the servers in place of the tasks (i.e., replacing the terms $\delta_{i,\ell}$ with $H_{j,\ell}^k$). The blocking incurred by M-BROE servers is analogous to the one incurred by tasks under P-EDF scheduling in conjunction to the MSRP [22]: please refer to [15, 22] for the detailed blocking analysis.

The schedulability test for guaranteeing the execution of virtual processors upon physical processors is reported in the following equation [15]:

$$\forall S_j^k : \mathcal{P}(S_j^k) = \mathcal{P}_m, \sum_{\substack{r,l: P_r^l \leq P_j^k \\ \wedge \\ \mathcal{P}(S_r^l) = \mathcal{P}_m}} \frac{Q_r^l}{P_r^l} + \frac{B_j^k}{P_j^k} \leq 1, \tag{12}$$

where B_j^k is a bound on the arrival blocking incurred by server S_j^k.

3.4 Table of symbols

In order to improve the paper readability, the main notation introduced in the system model is summarized in Table 1.

4 Partitioning and server design: problem definition

This section addresses the problem of partitioning application tasks on a set of virtual processors implemented by M-BROE reservation servers, and determining their configuration parameters in terms of budgets and periods.

Table 1 Main notation introduced in the system model.

Symbol	Description
Γ_k	k-th software component
$\mathcal{T}_k$	task set of the k-th software component
τ_i	i-th task
C_i	worst-case execution time of task τ_i
T_i	period (or minimum inter-arrival time) of task τ_i
D_i	relative deadline of task τ_i
$\mathcal{P}_m$	m-th physical processor
M	number of physical processors
S_j^k	j-th virtual processor of component Γ_k
Q_j^k	budget of S_j^k
P_j^k	reservation period of S_j^k
$\mathcal{S}(\tau_i)$	virtual processor to which τ_i is assigned
$\Gamma(S_j^k)$	set of tasks allocated to S_j^k
R_ℓ	ℓ-th resource
$\mathcal{R}^C$	set of component resources
$\mathcal{R}^S$	set of system resources
$\delta_{i,l}$	length of the longest critical section of τ_i for resource R_ℓ
$\eta_{i,\ell}$	number of critical sections of τ_i for resource R_ℓ
$\mathbf{H}_j^k$	vector of resource holding times for S_j^k
$\mathcal{H}$	upper bound for the resource holding times of all the resources

A partitioning methodology for the M-BROE framework must take into account three aspects simultaneously: (i) the computational demand of the application; (ii) the parameters (budget and period) of the reservation servers; and (iii) the blocking times related to resource sharing (also including the effect of spinlocks). In particular, the dependency of partitioning on resource sharing is better illustrated by the following example.

Example. Consider a component composed of 3 tasks with periods $T_1 = 10$, $T_2 = 20$ and $T_3 = 50$ ms and implicit deadlines, to be executed on a platform composed of two processors. Tasks τ_1 and τ_3 share a component resource R. If tasks τ_1 and τ_3 are assigned to different processors, then the access to R will be regulated by a spinlock. This solution penalizes the schedulability by generating non-preemptive blocking and increasing the execution times of τ_1 and τ_3 due to spinning time. Conversely, if the two tasks are allocated on the same processor, the blocking factor related to R is smaller (due to only uniprocessor SRP blocking). While it seems convenient allocating tasks on the same processor, it is easy to see that allocating τ_2 together with the other tasks may not be appropriate, due to the local blocking experienced by τ_2 (because $D_1 < D_2 < D_3$, from Equation (7)). If τ_2 is allocated to the other processor, it will experience no blocking. Note however, that τ_2 can experience a different blocking if it has a different period (e.g., $T_2 = 5$ ms).

The simple example presented above shows that resource sharing further complicates partitioning (which is intrinsically hard, being similar to a bin-packing problem), and must be taken into account to identify a "good" task allocation.

In this paper, task partitioning is formulated as an optimization problem. Unfortunately, however, given any performance objective (e.g., minimizing the overall bandwidth for the reservation servers), searching for an optimal solution is practically intractable for the following reasons:

- The exact EDF schedulability test requires the specification of a pseudo-polynomial number of check points [6] that depends on the parameters of the server upon which tasks execute (as explained in [11, 18]). Being the server parameters part of the output of the optimization problem (hence, unknown), upper bounds on them must be used, thus obtaining a potentially large set of constraints and variables.
- As shown in [18], it is possible to formulate an optimization problem to compute optimal BROE server parameters, minimizing the server bandwidth still ensuring the task schedulability. However, the exact problem formulation involves non-linear constraints, because the supply bound function of the server is non-linear.
- In the M-BROE framework, the access to global resources is protected by FIFO non-preemptive spinlocks, but an exact analysis for spinlocks is still missing, thus preventing the search for an optimal partitioning.

Given such limitations, this paper proposes a sub-optimal methodology for task partitioning and virtual processors design by splitting the problem in two phases: First, an MILP optimization is performed for partitioning tasks to virtual processors; then the optimal server parameters are computed through the approach presented in [18].

To overcome the problems highlighted above, the following approximations are proposed to express partitioning as an MILP optimization problem:

- The EDF schedulability is carried out by the Fully Polynomial Time Approximation Scheme (FPTAS), proposed by Fisher, Baker, and Baruah [21]. According to this approach, the workload of a task is described by the exact demand bound function for the first λ steps, and by a *linear* upper-bound for the remaining steps. Formal details about this approximation will be reported in Section 5.3.
- The reservation servers are approximated as ideal (fluid) virtual processors, running at a given speed α, which represents the server bandwidth. Note that using a classical bounded-delay (α-Δ) approximation, the optimization problem results non-linear.
- As done by Wieder and Brandenburg [29], blocking times in the presence of FIFO non-preemptive spinlocks are computed by the original MSRP (sufficient) analysis proposed by Gai et al. [22].

Please, note that approaching the problem through an MILP formulation guarantees that the achieved partitioning is optimal with regard to the assumed approximations. Once a task partitioning is obtained, the reservation servers of each component are designed with another optimization stage that makes use of the approach presented in [15] for computing the optimal server parameters that guarantee the schedulability of a given task set running upon the server. Such design steps are addressed in the following section.

It is worth clarifying that the approximation to fluid virtual processors is used only to *guide* the task partitioning and not at the stage of the server design when the task schedulability is enforced.

5 Partitioning and server design: optimization problem formulation

This section presents an MILP formulation to solve the problem of task partitioning upon virtual processors. The formulation presented in the following has been simplified with respect to the one reported in the conference version of this paper [14], where additional variables and constraints were adopted. It is also worth mentioning that the formulation is partially inspired by the one proposed by Wieder and Brandenburg [29] in the context of classical partitioned fixed-priority scheduling without reservation mechanisms.

As stated in Section 3, the worst-case blocking related to *component* resources is computed assuming that all the virtual processors of a component are assigned to different physical processors: without loss of generality, the index $k = 1, \ldots, M$ will be used for both physical processors and virtual processors. In the following we refer to a single component, hence (i) the component index is removed from all the terms used below, and (ii) all the tasks referred in the following are implicitly assumed to be part of a specific set $\mathcal{T}_k$, where Γ_k is the component of interest. All the real variables used in the optimization problem are implicitly constrained as greater than or equal to zero. For such variables, lower-bounds expressing the minimum (safe) value will be used to ensure schedulability. Note that any constraint that enforces such variables to be greater than a negative number has no effect. More specifically, let x be one of the optimization variables and let y be a *negative* term: any constraint of the form $x \geq y$ degenerates to $x \geq 0$, thus imposing no bound on x. This simple observation will result crucial in understanding the following constraints.

5.1 Decision variables for task allocation

The following decision (binary) variables are defined to decide on the task's allocation.

- $A_{i,k} \in \{0, 1\}$: binary variable that is set to 1 if and only if task τ_i is assigned to server S_k.

Since each task is assigned to exactly one server, the following constraint holds:

Constraint 1. $\forall \tau_i,\ \sum_{k=1}^{M} A_{i,k} = 1$.

5.2 Resource sharing - variables and constraints

Blocking times related to resource sharing are crucial to express the task schedulability as a constraint of the optimization problem. In the following two sections, constraints are derived to handle arrival blocking and spinning times.

5.2.1 Arrival blocking

To precisely encode blocking bounds in the MILP formulation, we decompose the arrival blocking incurred by a task into the contributions provided by each shared resource and each processor: this is accomplished with the definition of the following variables:

- $B_{i,\ell,k} \in \mathbb{R}_{\geq 0}$: real variable expressing a lower-bound on the arrival blocking imposed on τ_i by critical sections on resource R_ℓ, executed by tasks running on virtual processor S_k.

The key objective of this section consists in providing constraints for variables $B_{i,\ell,k}$. To this end, we have to distinguish between *component* and *system* resources. For both the types of resources, in the following we provide constraints that express blocking bounds *for all the types of blocking* identified in Section 3.1.

Considering component resources R_ℓ, the following constraint expresses a blocking bound in the case R_ℓ is implemented as a local resource:

Constraint 2. $\forall \tau_i, \forall R_\ell \in \mathcal{R}^C, \forall k = 1, \ldots, M,$
$\forall \tau_L \mid D_L > D_i,\ \forall \tau_H \mid D_H \leq D_i \wedge \eta_{H,\ell} > 0,$

$$B_{i,\ell,k} \geq \delta_{L,\ell} - \delta_{L,\ell} \cdot (3 - A_{i,k} - A_{L,k} - A_{H,k}).$$

Proof. Following Section 3.1, a task τ_i incurs in local blocking due to a local resource R_ℓ accessed by a task τ_L with $D_L > D_i$ when there exists another task τ_H with $D_H \leq D_i$ that also accesses R_ℓ. Both τ_L and τ_H must be allocated to the same virtual processor of τ_i and the amount of blocking is bounded by the largest critical section length $\delta_{L,\ell}$. If there not exists a virtual processor S_k to which τ_i, τ_L and τ_H are allocated, then the term $(3 - A_{i,k} - A_{L,k} - A_{H,k})$ is always positive and the constraint degenerates to zero, thus enforcing no bound. Otherwise, the constraint $B_{i,\ell,k} \geq \delta_{L,\ell}$ is enforced for each task τ_L that can generate local blocking, thus coping with the maximum local blocking as expressed by Equation (7). □

Conversely, when a component resource R_ℓ is implemented as a *global* resource, tasks allocated to the same virtual processor of τ_i that access R_ℓ may generate non-preemptive blocking to τ_i. This case is managed with the following constraint:

Constraint 3. $\forall \tau_i, \forall R_\ell \in \mathcal{R}^C, \forall k = 1, \ldots, M,$
$\forall \tau_L \mid D_L > D_i,\ \forall \tau_R \mid \eta_{R,\ell} > 0$

$$B_{i,\ell,k} \geq \delta_{L,\ell} - \delta_{L,\ell} \cdot (2 - A_{i,k} - A_{L,k}) - \delta_{L,\ell} \cdot A_{R,k}.$$

Proof. Following Section 3.1, a task τ_i incurs in non-preemptive blocking due to a resource R_ℓ accessed by a task τ_L with $D_L > D_i$ when there exists another task τ_R allocated to *remote* virtual processor that also accesses R_ℓ (i.e., $\eta_{R,\ell} > 0$). Both τ_L and τ_i must be allocated to the same virtual processor and the amount of blocking is bounded by the largest critical section length $\delta_{L,\ell}$. If there not exists a virtual processor S_k to which both τ_i and τ_L are

allocated, then the term $(2 - A_{i,k} - A_{L,k})$ is always positive and the constraint degenerates to zero, thus enforcing no bound. If such a virtual processor S_k exists, then τ_R must be allocated to a virtual processor $\neq S_k$. Whenever this is not the case, $A_{R,k} = 1$ and the constraint degenerates to zero, thus enforcing no bound. Otherwise, the constraint $B_{i,\ell,k} \geq \delta_{L,\ell}$ is enforced for each task τ_L that can generate non-preemptive blocking, thus correctly enforcing a blocking bound as in the previous constraint. □

Similarly, still considering the case in which a component resource R_ℓ results in a *global* resource, it is possible that tasks allocated to the same virtual processor of τ_i experience remote blocking (i.e., originated by other virtual processors). As introduced in Section 3.1, we recall that remote blocking leads to non-preemptive spinning that in turn may prevent τ_i from executing. A lower-bound on such a transitive blocking is encoded by the following constraint:

Constraint 4. $\forall \tau_i, \forall R_\ell \in \mathcal{R}^C, \forall k = 1, \ldots, M,$
$\forall S_z \neq S_k, \forall \tau_L \mid D_L > D_i \wedge \eta_{L,\ell} > 0, \ \forall \tau_R$

$$B_{i,\ell,k} \geq \delta_{R,\ell} - \delta_{R,\ell} \cdot (2 - A_{i,z} - A_{L,z}) - \delta_{R,\ell} \cdot (1 - A_{R,k}).$$

Proof. A task τ_i allocated to virtual processor S_z incurs in transitive remote blocking due to a component resource R_ℓ originated by a virtual processor $S_k \neq S_z$ when (i) there exists a task τ_R allocated to S_k that accesses R_ℓ and (ii) there exists another task τ_L allocated to S_z with $D_L > D_i$ that also accesses R_ℓ. The amount of blocking is bounded by the largest critical section length $\delta_{R,\ell}$. Whenever such tasks do not exist, at least one of the terms $(2 - A_{i,z} - A_{L,z})$ and $(1 - A_{R,k})$ is positive and the constraint degenerates to zero, thus enforcing no bound. Otherwise, the constraint $B_{i,\ell,k} \geq \delta_{R,\ell}$ is enforced for each task τ_R, thus correctly encoding a bound on transitive blocking. □

Now, it remains to consider system resources. Following the M-BROE analysis [15], since system resources can be accessed by all the components, they must always be treated as global resources to cope with the worst-case. As a consequence, the accesses to such resources are accounted as non-preemptive blocking, so obtaining the following constraint:

Constraint 5. $\forall \tau_i, \forall R_\ell \in \mathcal{R}^S, \forall k = 1, \ldots, M,$
$\forall \tau_L \mid D_L > D_i$

$$B_{i,\ell,k} \geq \delta_{L,\ell} - \delta_{L,\ell} \cdot (2 - A_{i,k} - A_{L,k}).$$

Proof. The proof is analogous to the one of Constraint 3 assuming that task τ_R always exists ($A_{R,k} = 1$). □

Similarly, when remote blocking for a system resource R_ℓ is considered, it is not possible to infer on the critical section lengths on R_ℓ that are present in the other components. Hence, to be safe, we have to assume that a critical section of maximum length $\mathcal{H}$ is present on each virtual processor (please refer to Section 3), so obtaining the following constraint:

Constraint 6. $\forall \tau_i, \forall R_\ell \in \mathcal{R}^S, \forall k = 1, \ldots, M,$
$\forall S_z \neq S_k,\ \forall \tau_L \mid D_L > D_i \wedge \eta_{L,\ell} > 0$

$$B_{i,\ell,k} \geq \mathcal{H} - \mathcal{H} \cdot (2 - A_{i,z} - A_{L,z}).$$

Proof. The proof is analogous to the one of Constraint 4 assuming that task τ_R always exists ($A_{R,k} = 1$) and that $\delta_{R,\ell} = \mathcal{H}$. □

Finally, the following variables are introduced to model the blocking times corresponding to each resource:

- $B_{i,\ell} \in \mathbb{R}_{\geq 0}$: real variable expressing a lower-bound on the arrival blocking imposed on τ_i by critical sections on resource R_ℓ.

Such variables act as aliases to aggregate the blocking times generated by each processor: the following constraint is enforced to encode their definition:

Constraint 7. $\forall \tau_i, \forall R_\ell, \quad B_{i,\ell} = \sum_{k=1}^{M} B_{i,\ell,k}.$

5.2.2 Spinning time

As stated in Section 3, the use of non-preemptive FIFO spinlocks is accounted by inflating tasks' WCETs by means of the spinning time ξ_i generated by *remote blocking*. We decompose the spinning time ξ_i of a task τ_i by using the following variables:

- $\xi_{i,k} \in \mathbb{R}_{\geq 0}$: real variable expressing a lower-bound on the spinning time for task τ_i originated from virtual processor S_k.
- $\xi_{i,k,\ell} \in \mathbb{R}_{\geq 0}$: real variable expressing the contribution of the spinning time ξ_i in accessing the resource R_ℓ, originated from virtual processor S_k.

The per-processor spinning time $\xi_{i,k}$ can be expressed as

Constraint 8. $\forall \tau_i, \forall k = 1, \ldots, M, \quad \xi_{i,k} = \sum_{R_\ell} \xi_{i,k,\ell}.$

Similarly, the overall spinning time ξ_i of a task τ_i is formulated as

Constraint 9. $\forall \tau_i, \quad \xi_i = \sum_{k=1}^{M} \xi_{i,k}.$

Then, the key objective consists in identifying constraints that provide safe bounds on the spinning time $\xi_{i,k,\ell}$. Again, it is necessary to distinguish between *component* and *system* resources. For a *component* resource R_ℓ, the following constraint is provided:

Constraint 10. $\forall \tau_i, \forall k = 1, \ldots, M, \forall R_\ell,\ \forall \tau_x \mid \tau_i \neq \tau_x,$

$$\xi_{i,k,\ell} \geq \delta_{x,\ell} \cdot \eta_{i,\ell} \cdot A_{x,k} - \mathcal{B} \cdot A_{i,k}.$$

Proof. This constraint derives directly from the computation of the spinning time for component resources, as defined in Equations (4) and (5). The constraint collects the maximum critical section on R_ℓ of tasks τ_x allocated on virtual processor S_k. To account for the overall spinning time, the critical section length is multiplied for the number of critical sections on R_ℓ for τ_i (see Equation (5)). Thanks to the decision variable $A_{x,k}$, the first term becomes

zero if task τ_x is not allocated on server S_k. The remaining term $-\mathcal{B} \cdot A_{i,k}$ is provided to have zero spinning time contribution from the same processor at which τ_i is allocated (critical sections executed on the same processor of τ_i do not cause spinning). In this case, $\mathcal{B}$ represents a numerically large constant that dominates all possible values for the term $\delta_{x,\ell} \cdot \eta_{i,\ell}$, and can be formally defined as $\mathcal{B} = \max_{x,\ell}\{\delta_{x,\ell}\} \cdot \max_{i,\ell}\{\eta_{i,\ell}\}$. □

When a *system* resource R_ℓ is considered, the spinning time can be expressed as follows:

Constraint 11. $\forall \tau_i, \forall k = 1, \ldots, M, \forall R_\ell,\ \forall \tau_x \mid \tau_i \neq \tau_x,$

$$\xi_{i,k,\ell} \geq \mathcal{H} \cdot \eta_{i,\ell} \cdot A_{x,k} - \mathcal{B} \cdot A_{i,k}.$$

Proof. The proof follows from the one of Constraint 10, assuming critical sections of length $\mathcal{H}$. □

5.3 Schedulability - variables and constraints

This section presents the constraints for the optimization problem expressing the schedulability of a task set upon a reservation server. As stated in Section 3, the local schedulability upon an M-BROE server can be checked by Equation (9); however, as expressed at the beginning of Section 5, the exact test is not easily tractable in an optimization problem. To solve this problem, the workload of a task set is approximated using the FPTAS [21] approach. According to the FPTAS, the demand bound function $\mathsf{dbf}_i(t)$ of a task τ_i is exact for the first λ steps and then approximated with a linear upper-bound. Depending on the chosen value of λ, function $\mathsf{dbf}_i(t)$ can be approximated with any desired degree of accuracy. Formally, the FPTAS for the demand bound function is expressed as

$$\mathsf{dbf}_i^{(\lambda)}(t) = \begin{cases} \mathsf{dbf}_i(t), & \text{if } t \leq (\lambda - 1)T_i + D_i \\ C_i + \xi_i + (t - D_i)U_i, & \text{otherwise,} \end{cases} \tag{13}$$

where $U_i = (C_i + \xi_i)/T_i$, to account for the WCET inflation related to the use of spinlocks.

Using this approximation, the EDF schedulability has to be considered in $\lambda + 1$ time points for each task. The resulting sufficient EDF schedulability test for a set of tasks $\mathcal{T}$ is expressed as follows:

$$\begin{aligned} &\forall p = 0, 1, \ldots, \lambda,\ \forall t \in \mathrm{tSet}(p), \\ &B(t,p) + \sum_{\tau_i \in \mathcal{T}} \mathsf{dbf}_i^{(\lambda)}(t) \leq \mathsf{sbf}(t), \end{aligned} \tag{14}$$

where $\mathrm{tSet}(p)$ is the set of schedulability check-points [21] defined as

$$\mathrm{tSet}(p) = \bigcup_{\tau_i \in \mathcal{T}} \{pT_i + D_i\}, \tag{15}$$

and $B(t,p)$ is defined to approximate the blocking term of Equation (11) as

$$B(t,p) = \begin{cases} B(t), & \text{if } 0 \leq p < \lambda \\ \max_i\{B_i\}, & p = \lambda. \end{cases} \tag{16}$$

As stated by Baruah in [4], the blocking term $B(t)$ is zero for values of t larger than the maximum relative deadline of the tasks under analysis. As a consequence, the exact blocking function $B(t)$ can be used (i.e., $B(t,p) = B(t),\ \forall p \geq 0$) in the MILP formulation if a sufficiently large number λ of check-points is used for the FPTAS.

We now define a set of variables and constraints to express the EDF schedulability according to the FPTAS approach. All the variables contain the processor index k, since the schedulability has to be checked for each processor addressing a partitioned scheduling scheme. First of all, we introduce the following variables to account for the WCET inflation related to spinlocks:

- $J_{i,k} \in \mathbb{R}_{\geq 0}$: real variable expressing the inflated WCET of a task τ_i executing on virtual processor S_k.

Such a variable can be defined by using the following constraint:

Constraint 12. $\forall \tau_i, \forall k = 1, \ldots, M, \quad J_{i,k} = C_i + \xi_i - \mathcal{B} \cdot (1 - A_{i,k})$.

Proof. This constraint simply adds C_i to the overall spinning time ξ_i. Term $-\mathcal{B} \cdot (1 - A_{i,k})$ is provided to have a null inflated WCET if τ_i is not allocated to server S_k (i.e., when $A_{i,k} = 0$). $\mathcal{B}$ represents a numerically large constant that dominates all possible values of the term $C_i + \xi_i$, and can be defined as $\mathcal{B} = \max_i\{C_i\} \cdot (M-1)\mathcal{H} \cdot \max_{i,\ell}\{\eta_{i,\ell}\}$. □

Now note that Equations (14) involves the blocking time $B(t)$, defined in Equation (16). Having to express the schedulability in a limited number of time points, we introduce variables to quantify the blocking time at the p^{th} time point of a task:

- $PB_{k,p,j} \in \mathbb{R}_{\geq 0}$: real variable expressing the blocking time on virtual processor S_k at schedulability check-point $pT_j + D_j$ of task τ_j, with $p = 0, 1, \ldots, \lambda$.

Such a blocking time is expressed by the following constraint, making use of the blocking time $B_{i,\ell}$ expressed in Constraint 7:

Constraint 13. $\forall k = 1, \ldots, M, \forall p = 0, 1, \ldots, \lambda, \forall \tau_j, \forall R_\ell$

$$\forall \tau_i \mid D_i \leq pT_j + D_j \wedge p < \lambda$$
$$PB_{k,p,j} \geq B_{i,\ell} - \mathcal{B} \cdot (1 - A_{i,k}).$$

Proof. According Equations (11) and (16), the blocking at the schedulability check-point t involves blocking times of tasks having deadlines less than or equal to t for $p = 0, 1, \ldots, \lambda$. The p^{th} check-point originated from τ_j is $pT_j + D_j$ (see Equation (15)): this constraint accordingly considers all tasks τ_i having $D_i \leq pT_j + D_j$ (thus contributing to blocking) excluding the ones that are not allocated on S_k. Such tasks are excluded through the term $-\mathcal{B} \cdot (1 - A_{i,k})$, where $\mathcal{B}$ is a numerically large constant that dominates all possible values for

the term $B_{i,\ell}$ and can be formally defined as $\mathcal{B} = M\mathcal{H}$. Similarly, all tasks τ_i allocated to S_k are considered for $p = \lambda$, accounting for the maximum blocking on S_k. □

The computational supply provided by each virtual processor S_k is approximated by assuming ideal (fluid) virtual processors with bandwidth α_k. The following variables are introduced to support this choice:

– $\alpha_k \in \mathbb{R}_{\geq 0}$: real variable representing the bandwidth of virtual processor S_k, with $0 \leq \alpha_k \leq 1$.

At this point we have all the variables and the constraints to express the EDF schedulability on each virtual processor S_k:

Constraint 14. $\forall k = 1, \ldots, M,\ \forall p = 0, \ldots, \lambda,\ \forall \tau_j$

$$PB_{k,p,j} + \sum_{\tau_i \in \Gamma} \mathsf{dbf}_{i,k}^{(\lambda)}(pT_j + D_j) \leq \alpha_k \cdot (pT_j + D_j),$$

where

$$\mathsf{dbf}_{i,k}^{(\lambda)}(t) = \begin{cases} \left(\left\lfloor \frac{t - D_i}{T_i} \right\rfloor + 1 \right)_0 J_{i,k}, & \text{if } t \leq (\lambda - 1)T_i + D_i \\ J_{i,k} + (t - D_i) \left(\frac{J_{i,k}}{T_i} \right), & \text{otherwise.} \end{cases}$$

Proof. This constraint derives directly from the FPTAS schedulability test in Equation (14), for each virtual processor S_k. Thanks to the definition of variables $J_{i,k}$, the contribution in terms of demand bound function of tasks τ_i is allowed to be null if τ_i is not assigned to virtual processor S_k. □

5.4 **Objectives**

We now propose two alternative allocation strategies for the optimization problem, aiming at different objectives. The first one, denoted as **A**, aims at allocating the tasks of a component on a small set of virtual processors having high bandwidth; the second one, denoted as **B**, aims at distributing tasks among a larger set of virtual processors with lower bandwidth. Clearly, each allocation strategy leads to a different instance of the component interface. We now formalize the objectives for both strategies.

– Strategy **A**: minimize the overall bandwidth required by a software component, that is the sum of the bandwidths required by its virtual processors:

$$\textbf{minimize} \sum_{k=1}^{M} \alpha_k. \tag{17}$$

– Strategy **B**: minimize the maximum bandwidth required by the virtual processor of a component:

$$\textbf{minimize } \max\{\alpha_k\} = \textbf{minimize } \Lambda, \tag{18}$$

where Λ is an additional real variable of the optimization problem defined by the following constraint:

Constraint 15. $\forall k = 1, \ldots, M \quad \Lambda \geq \alpha_k$.

5.5 Interface synthesis

Given the allocation of the component tasks to the virtual platform, produced by the MILP solution, the design of the reservation server parameters is performed using the approach presented in [18]. Such an approach computes the *optimal* budget and period (under the assumed scheduling infrastructure) that guarantee the application schedulability while minimizing the bandwidth for each virtual. Then, for each reservation server and for each resource, the maximum resource holding times are computed according to the interface discussed in Section 2.1.

The resulting server parameters and vectors of resource holding times (for each virtual processor) constitute the component interface exported to the component integrator. Note that, different interfaces can be obtained depending on the allocation strategy (A or B) that is selected as objective of the proposed MILP formulation. Nevertheless, each component is guaranteed to be schedulable under both the interfaces by construction.

6 Virtual processor allocation

The goal of this section is to propose a methodology to partition the virtual processors of all the components to the M physical processors, that is the task performed by the component integrator. This is done by another MILP formulation that, with respect to the adopted analysis (see Section 3.3) and the interfaces exported by the components, is able to find an *exact solution* for the allocation problem. In this section, we assume that each component exports two interfaces denoted as type A and type B, which are obtained with strategies A and B presented in Section 5.4, respectively. The interfaces exported by the components are the input parameters for the formulation. The proposed MILP formulation is able to decide whether a component is integrated by using the interface of type A or B. This characteristic represents a key improvement with respect to the formulation presented in the conference version of this paper [14], where a single interface per component was considered.

To reduce clutter and improve readability, simplified notation is introduced to present the following results. The reservation servers exported by the components are denoted by S_i with $i = 1, \ldots, N \cdot M$, i.e., they are enumerated with a progressive index, thus getting rid of the double indexes that have been used in the previous sections. $\Gamma(S_i)$ is used to denote the corresponding component of S_i. The parameters of the interface of type A are denoted by Q_i^A, P_i^A and $\mathbf{H}_i^A$. Similarly, the ones of the interface of type B are denoted using a B superscript. Whenever an interface is not provided by a component (e.g., when a component provides only the one of type A), we assume that all its parameters are set to zero.

6.1 Decision variables for component allocation

As done for the task allocation problem addressed in Section 5, a set of binary variables are introduced to decide on the allocation of the virtual processors:

- $A_{i,k} \in \{0,1\}$: binary variable that is set to 1 if and only if the reservation S_i is allocated to physical processor $\mathcal{P}_k$.

The following variables are defined to decide which interface is selected:

- $I_j \in \{0,1\}$: binary variable that is set to 1 if and only if component Γ_j is allocated with interface A, and 0 if and only if component Γ_j is allocated with interface B.

Since a reservation must be assigned to only one processor, the following constraint holds:

Integration Constraint 1. $\forall S_i, \quad \sum_{k=1}^{M} A_{i,k} = 1.$

Finally, a constraint is enforced to exclude the interfaces that are not provided by the components. That is, if the interface of type A is not provided, then it forces the use of the one of type B. The dual constraint is enforced if the one of type B is not provided.

Integration Constraint 2.

$$\forall \Gamma_j \mid \nexists S_i \mid Q_i^A > 0 \wedge \Gamma(S_i) = \Gamma_k \quad I_j = 0.$$

$$\forall \Gamma_j \mid \nexists S_i \mid Q_i^B > 0 \wedge \Gamma(S_i) = \Gamma_k \quad I_j = 1.$$

6.2 Resource sharing: variables and constraints

As done in Section 5, the blocking originated by resource sharing is split for each resource R_ℓ and for each processor $\mathcal{P}_k$, and is managed with the following real variables:

- $B_{i,\ell,k} \in \mathbb{R}_{\geq 0}$: real variable expressing a lower-bound on the arrival blocking incurred by S_i due to critical sections on resource R_ℓ that are executed by tasks running on processor $\mathcal{P}_k$.

We recall that the vectors $\mathbf{H}_i^A$ and $\mathbf{H}_i^B$, which are exported by the component interfaces, provide bounds on the resource holding times of the shared resources. In the following, such bounds are used to derive a set of constraints that aim at bounding the arrival blocking incurred by the reservation servers.

To ease the presentation of the following three constraints, we define

$$e_i(\nu_i) = \begin{cases} (1 - I_j) & \text{if } \nu_i = A \\ I_j & \text{if } \nu_i = B \end{cases}$$

where $\Gamma(S_i) = \Gamma_j$. This term has the following meaning: $e(A) = 0$ if and only if S_i is allocated with interface A, while $e(B) = 0$ if and only if S_i is allocated with interface B.

We can now begin by bounding the blocking due to processor-local resources.

Integration Constraint 3. $\forall \nu_i \in \{A,B\}, \forall S_i, \forall R_\ell, \forall k = 1,\ldots,M,$
$\forall \nu_j \in \{A,B\}, \forall S_j \mid P_j > P_i, \quad \forall \nu_h \in \{A,B\}, \forall S_h \mid P_h \leq P_i$

$$B_{i,\ell,k} \geq H^{\nu_j}_{j,\ell} - \mathcal{B}(e_i(\nu_i) + e_j(\nu_j) + e_h(\nu_h)) - \mathcal{B}(3 - A_{i,k} - A_{j,k} - A_{h,k}).$$

Proof. Following the H-SRP blocking analysis, a server S_i allocated to processor $\mathcal{P}_k$ can be blocked by a processor-local resource R_ℓ only if there exist two servers S_h and S_j, both allocated to $\mathcal{P}_k$, that access R_ℓ and with $P_h \leq P_i > P_j$. If such servers exist, the time S_i can be blocked due to R_ℓ is bounded by the maximum resource holding time $H^A_{j,\ell}$, if S_j is allocated with interface A, or $H^B_{j,\ell}$, if S_j is allocated with interface B. Whenever the three servers are not allocated to the same processor $\mathcal{P}_k$, the term $(3 - A_{i,k} - A_{j,k} - A_{h,k})$ is positive and hence no bound is enforced. Furthermore, whenever the constraint considers at least one of such servers that is part of an interface that is not admitted, the term $(e_i(\nu_i) + e_j(\nu_j) + e_r(\nu_r))$ is positive and hence no bound is enforced. □

In a similar manner, the following constraint is provided to bound the non-preemptive blocking generated by the access to global resources.

Integration Constraint 4. $\forall \nu_i \in \{A,B\}, \forall S_i, \forall R_\ell, \forall k = 1,\ldots,M,$
$\forall \nu_j \in \{A,B\}, \forall S_j \mid P_j > P_i, \quad \forall \nu_r \in \{A,B\}, \forall S_r \mid H_{r,\ell} > 0$

$$B_{i,\ell,k} \geq H^{\nu_j}_{j,\ell} - \mathcal{B}(e_i(\nu_i) + e_j(\nu_j) + e_r(\nu_r)) - \mathcal{B}(2 - A_{i,k} - A_{j,k}) - \mathcal{B} \cdot A_{r,k}.$$

Proof. A server S_i allocated to processor $\mathcal{P}_k$ incurs in non-preemptive blocking when there exists a server S_j allocated to $\mathcal{P}_k$ that accesses a *global* resource R_ℓ. A resource R_ℓ is global if there exists at least one server S_r allocated to a processor $\neq \mathcal{P}_k$ that accesses R_ℓ (i.e., $H^A_{r,\ell} > 0$ or $H^B_{r,\ell} > 0$, depending on which interface is selected for S_r). If such servers exist, the time S_i can incur in non-preemptive blocking due to R_ℓ is bounded by the maximum resource holding time $H^A_{j,\ell}$, if S_j is allocated with interface A, or $H^B_{j,\ell}$, if S_j is allocated with interface B. Whenever the S_i and S_j are not allocated to the same processor $\mathcal{P}_k$, or S_r is not allocated to a processor $\neq \mathcal{P}_k$, at least one of the terms $(2 - A_{i,k} - A_{j,k})$ and $A_{r,k}$ is positive and hence no bound is enforced. Furthermore, whenever the constraint considers at least one of such servers that is part of an interface that is not admitted, the term $(e_i(\nu_i) + e_j(\nu_j) + e_r(\nu_r))$ is positive and hence no bound is enforced. □

The following constraint enforces a bound on transitive remote blocking.

Integration Constraint 5. $\forall \nu_i \in \{A,B\}, \forall S_i, \forall R_\ell, \forall k = 1,\ldots,M,$
$\forall \mathcal{P}_x \neq \mathcal{P}_k, \forall \nu_j \in \{A,B\}, \forall S_j \mid P_j > P_i \wedge H_{j,\ell} > 0, \quad \forall \nu_r \in \{A,B\}, \forall S_r$

$$B_{i,\ell,k} \geq H^{\nu_r}_{r,\ell} - \mathcal{B}(e_i(\nu_i) + e_j(\nu_j) + e_r(\nu_r)) - \mathcal{B}(2 - A_{i,x} + A_{j,x}) - \mathcal{B}(1 - A_{r,k}).$$

Proof. According to the MSRP blocking analysis, a server S_i that is allocated to a processor $\mathcal{P}_x \neq \mathcal{P}_k$ incurs in transitive remote blocking originated by processor $\mathcal{P}_k$ if there exists a server S_j allocated to $\mathcal{P}_x$ that accesses a *global*

resource R_ℓ that is used by a server S_r allocated to $\mathcal{P}_k$. If such servers exist, the time S_i can incur in transitive remote blocking due to R_ℓ is bounded by the maximum resource holding time $H^A_{r,\ell}$, if S_r is allocated with interface A, or $H^B_{r,\ell}$, if S_r is allocated with interface B. Whenever both S_i and S_j are not allocated to the same processor $\mathcal{P}_x$, or S_r is not allocated to a processor $\mathcal{P}_k$, at least one of the terms $(2 - A_{i,x} - A_{j,x})$ and $(1 - A_{r,k})$ is positive and hence no bound is enforced. Furthermore, whenever the constraint considers at least one of such servers that is part of an interface that is not admitted, the term $(e_i(\nu_i) + e_j(\nu_j) + e_r(\nu_r))$ is positive and hence no bound is enforced. □

Finally, we enforce that the maximum arrival blocking incurred by each server is the maximum of the arrival blocking times generated by each resource.

Integration Constraint 6. $\forall S_i, \forall R_\ell, \quad B_i \geq \sum_{k=1}^{M} B_{i,\ell,k}$.

Proof. The term $\sum_{k=1}^{M} B_{i,\ell,k}$ provides a bound on the arrival blocking incurred by S_i due to resource R_ℓ. Since under MSRP a server can incur in arrival blocking due to at most one resource, the maximum of all the terms $\sum_{k=1}^{M} B_{i,\ell,k}$ (for each resource R_ℓ) yields a safe bound. □

6.3 Server schedulability: variables and constraints

To express the schedulability at the integration level in the optimization problem formulation, we have to derive constraints from the test reported in Equation (12). The idea is to provide variables and constraints representing the contribution of bandwidth on each physical processor. To this end, two real variables are defined for each reservation server S_i:

- $\Phi^A_{i,k} \in \mathbb{R}_{\geq 0}$: a real variable expressing a lower-bound on the budget demanded by server S_i if allocated with interface A and to processor $\mathcal{P}_k$;
- $\Phi^B_{i,k} \in \mathbb{R}_{\geq 0}$: a real variable expressing a lower-bound on the budget demanded by server S_i if allocated with interface B and to processor $\mathcal{P}_k$;

The following constraint is provided to enforce correct values for such variables. The constraint makes use of a numerical constant $\mathcal{B}$ that is used to represent infinity, which can be formally defined as $\mathcal{B} = \max_i\{Q_i\}$.

Integration Constraint 7. $\forall \Gamma_j, \forall S_i \mid \Gamma(S_i) = \Gamma_j$,

$$\Phi^A_{i,k} \geq Q^A_i \cdot I_j - \mathcal{B}(1 - A_{i,k}),$$

$$\Phi^B_{i,k} \geq Q^B_i (1 - I_j) - \mathcal{B}(1 - A_{i,k}).$$

Proof. If component Γ_j is allocated with interface of type A, then $I_j = 1$ and no bound is enforced on Q^B_i; otherwise, if Γ_j is allocated with interface of type B, then no bound is enforced on Q^A_i. For both the interface types, if a server S_i is *not* allocated to processor $\mathcal{P}_k$, then no bound is enforced on both $\Phi^A_{i,k}$ and $\Phi^B_{i,k}$. As a consequence, $\Phi^A_{i,k} \geq Q^A_i$ is enforced if and only if (i) interface A is selected and (ii) S_i is allocated to $\mathcal{P}_k$. The same holds for $\Phi^B_{i,k} \geq Q^B_i$ when interface B is selected. □

Variables $\Phi^A_{i,k}$, $\Phi^B_{i,k}$ and B_i are finally used in the following constraint that enforces the schedulability of the servers under P-EDF and MSRP (see Equation (12)). The constraint exploits the fact that, whenever no bound is enforced on the above mentioned variables, then they are allowed to degenerate to zero and hence provide no contribution to the schedulability test.

Integration Constraint 8. $\forall k = 1, \ldots, M, \forall \nu_i \in \{A, B\}, \forall S_i \mid P_i^{\nu_i} > 0$

$$\frac{B_i}{P_i} + \sum_{\nu_j \in \{A,B\}} \sum_{\substack{S_j \\ P_j^{\nu_j} \leq P_i^{\nu_i} \\ P_j^{\nu_j} > 0}} \frac{\Phi_{j,k}^{\nu_j}}{P_{j,k}^{\nu_j}} \leq 1 + \mathcal{B}(1 - A_{i,k}) + \mathcal{B} \cdot e(\nu_i).$$

Proof. Consider a processor $\mathcal{P}_k$ and a server S_i according to an interface ν_i. First note that if (i) S_i is *not* allocated to $\mathcal{P}_k$ ($A_{i,k} = 0$) or (ii) the component of S_i is *not* allocated with interface ν_i ($e_i(\nu_i) = 1$), then the RHS of the inequality becomes infinite and hence no constraint is enforced. Now, suppose that this is not the case, thus the LHS is equal to 1. Given that variables $\Phi^A_{i,k}$, $\Phi^B_{i,k}$ and B_i yield valid lower-bounds on the server budget and arrival blocking, respectively, the inequality reduces to Equation (12). Hence the constraint enforces the schedulability test for the servers. □

7 Experimental results

This section presents some experimental results aimed at evaluating the proposed methodology. Experiments have been conducted to evaluate the performance in terms of schedulability ratio for the whole methodology, and to measure the run time of the procedure for partitioning an application upon a virtual processor. The proposed MILP formulations have been implemented with the IBM CPLEX solver running on a machine equipped with 40 cores Intel Xeon E5-2640 @ 2.4 GHz and 128 GB of RAM. Note that the presented results are related to a different experimentation with respect to the one presented in the conference version of this paper [14], as it considers **(i)** a different approach for performing component integration, **(ii)** a different workload generation strategy, which in particular considers more shared resources, and **(iii)** a different MILP formulation for performing the task partitioning. Also, the experiments have been executed on a different machine and with a re-engineered implementation of the proposed algorithms.

7.1 Workload generation

Given an overall system utilization U, N software components were generated. The utilizations U_k of the components were generated using the UUnifast [12] algorithm, limiting their values in the range [0.15, 1.5]. The task set $\mathcal{T}_k$ in each component Γ_k was generated by fixing a number of tasks n, each with

utilization $u_i \leq 0.8$ generated by UUnifast. Tasks periods T_i were randomly chosen in the set $\{5, 10, 20, 30, 50, 80, 100, 120, 150, 200\}$ ms, and tasks computation times were computed as $C_i = u_i T_i$. We assumed the presence of NR^C component resources *for each component* and NR^S system resources. For each resource R_ℓ, a random number of tasks in the range $[1, \text{rsf} \cdot n_k]$ was selected with uniform distribution to access R_ℓ. The rsf parameter (resource sharing factor) indicates how many tasks use a resource. For each task τ_i accessing R_ℓ, we randomly generated $\eta_{i,\ell} \in [1, \eta^{\text{MAX}}]$ critical sections of length $\delta_{i,\ell} \in (0, \mathcal{H}]$, both with uniform distribution. To have realistic task sets, we enforced $C_i \geq \sum_{R_\ell}(\eta_{i,\ell} \cdot \delta_{i,\ell}),\ \forall \tau_i \in \mathcal{T}_k$, for each component Γ_k $(k = 1, \ldots, N)$.

7.2 Experiment 1

This experiment was carried out to evaluate the schedulability performance of the entire approach proposed in this paper. For each component generated according to the strategy described in the previous section, the partitioning and interface synthesis method presented in Section 5 has been applied. Both strategies A and B have been considered, thus generating two interfaces for each component. Then, the component integration has been performed with the approach presented in Section 6. We denote a system schedulable when the integration of the components that constitute the system succeeds, i.e., when the approach of Section 6 founds an allocation and a configuration of interfaces under which *all* the corresponding virtual processors are schedulable. It is worth repeating that, under the proposed design flow, the schedulability of the virtual processors of a component Γ transitively ensures the schedulability of the tasks in Γ.

For comparison purposes, three variants of the proposed approach have been tested:

1. **A**, which corresponds to adopting only interfaces obtained with strategy A described in Section 5;
2. **B**, which corresponds to adopting only interfaces obtained with strategy B described in Section 5;
3. **A** $\vee$ **B**, which denotes the case in which the full methodology is applied, thus allowing the selection of the best interface for each component to favor their integration.

Note that preliminary results related to the third variant were also presented in the conference version of this paper [14]: however, the corresponding MILP formulation has been fully detailed only in the present paper. The experimentation considered systems consisting of $N = 5$ components to be executed on a physical platform including $M = 4$ processors. The schedulability performance of the proposed methodology was evaluated as a function of the system utilization U, which has been varied from 1.5 to M with step 0.25. For each value of U, the schedulability ratio was computed over 500 systems, hence testing a total of 27500 components and 137500 tasks. The following param-

eters have been kept constant during the experimentation: $\lambda = 30$, rsf= 0.3, $\eta^{\text{MAX}} = 2$, $\mathcal{H} = 100\mu s$.

Figure 2 reports the results of this experiment obtained with $n = 5$ tasks per component, $NR^C = 2$ component resources per component (for a total of 10 component resources), and $NR^C = 2$ system resources. Note that, for this particular setting of parameters, A is more effective than B for almost all the tested system utilizations, with A being able to guarantee four times more system instances than B for $U = 3$. As expected, the mixed strategy (A $\vee$ B) outperforms the others, being able to admit almost 80% of the generated systems for utilizations up to 3.25 (corresponding to the 80% of the available utilization). Figure 3 reports the results of the same experiment repeated with $n = 8$ tasks per component. As it can be observed from the graph, there is a significant degradation for the B approach, while A and A $\vee$ B show a slight degradation with the exception of the cases with high utilization ($U \geq 3.25$). The main causes of the performance degradation can be attributed to (i) the schedulability penalties introduced by resource reservation, which generally increase with the number of tasks managed by a server, and (ii) the higher impact of resource sharing (note that the adopted method for generating the workload tends to generate more critical sections as the number of tasks increases). According to this interpretation, the degradation results much higher for the B approach because more servers tend to be generated (recall that strategy B tends to spread the tasks across the virtual processors) and the consequent higher likelihood of implementing component resources as global resources, which hence result in higher blocking times.

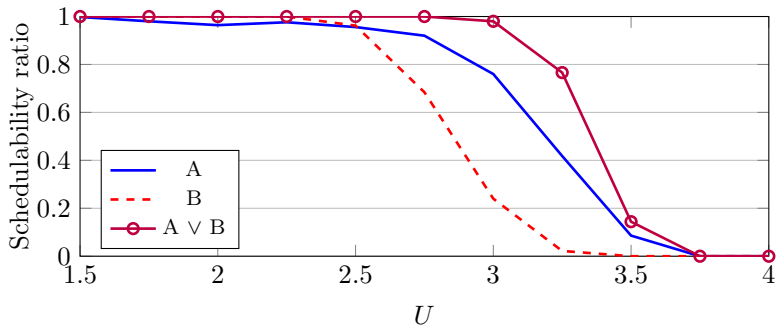

Fig. 2 Schedulability ratio as a function of the system utilization U for $n = 5$ (25 tasks in total), $NR^S = 2$, and $NR^C = 2$.

To better evaluate the impact of resource sharing on the schedulability ratio, another test was performed with an increased number of resources: $NR^C = 4$ component resource for each component (for a total of 20 component resources) and $NR^S = 4$ system resources. The result of this test is reported in Figure 4 for $n = 5$, and in Figure 5 for $n = 8$. As it can be noted from the graphs, all the three approaches show a performance degradation. For $n = 5$, the mixed strategy (A $\vee$ B) is able to guarantee more than 60% of the tested systems up to $U = 3$. For $n = 8$, the results show a clear degrada-

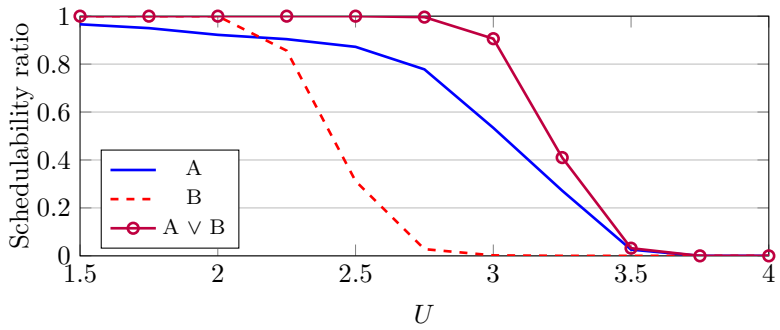

Fig. 3 Schedulability ratio as a function of the system utilization U for $n = 8$ (40 tasks in total), $NR^S = 2$, and $NR^C = 2$.

tion, and A $\vee$ B is able to guarantee about 60% of the tested systems up to $U = 2.75$.

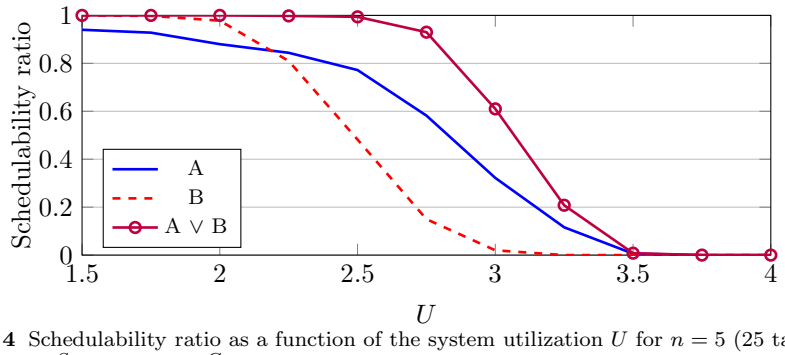

Fig. 4 Schedulability ratio as a function of the system utilization U for $n = 5$ (25 tasks in total), $NR^S = 4$, and $NR^C = 4$.

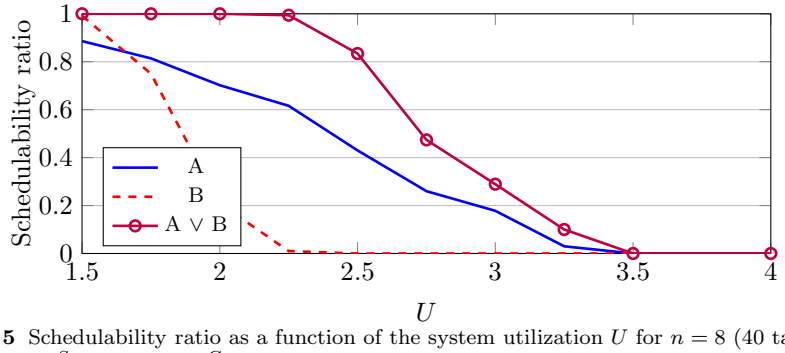

Fig. 5 Schedulability ratio as a function of the system utilization U for $n = 8$ (40 tasks in total), $NR^S = 4$, and $NR^C = 4$.

7.3 Experiment 2

In this experiment, the run time needed to solve the MILP formulation for task partitioning has been measured as a function of the number of tasks (n) in a component. For each value of n, the average and the maximum run time was measured over 500 randomly generated components. Figures 6 and 7 report run time taken by strategies (A) and (B), respectively. The results are collected under the same parameter configuration used in Figure 2. The maximum run time shows an exponential trend as the number of tasks increases. This result is expected since the number of the variables on the MILP formulation increases with the number of tasks. However, note that both the strategies have a maximum running time lower than one minute for components of $n = 10$ tasks. Therefore, such results show that the proposed task partitioning approach is perfectly compatible with the timeframe of offline (design-time) activities, e.g., those in the automotive domain, where the considered number of tasks is particularly representative.

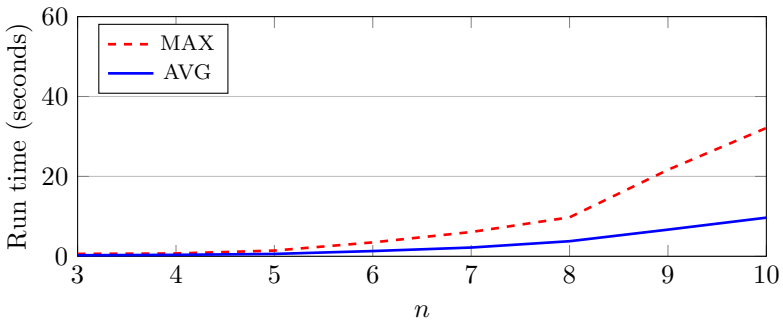

Fig. 6 Average and maximum run time for solving the MILP formulation for task partitioning with strategy (A).

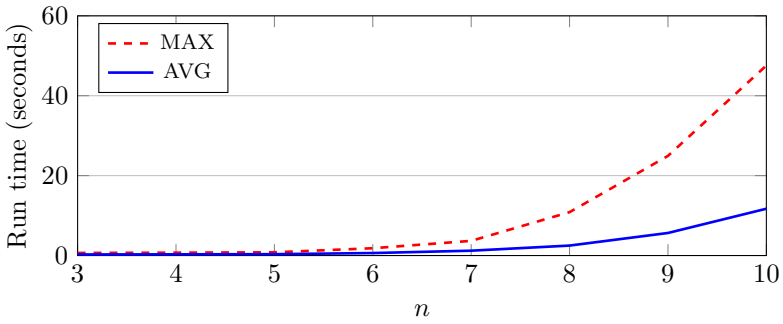

Fig. 7 Average and maximum run time for solving the MILP formulation for task partitioning with strategy (B).

8 Conclusions and future work

This paper presented a component-based design methodology for supporting the integration of independently developed real-time applications upon multiprocessor platforms in the presence of shared resources. Applications consists of a set of periodic and sporadic real-time tasks that can use both component and system-level resources. The physical platform is abstracted through a number of virtual platforms, one for each component, each consisting of a set of virtual processors implemented by reservation servers.

The proposed methodology uses an MILP formulation to partition each application upon the virtual multiprocessor platform taking shared resources into account. Two allocation strategies have been proposed as a objective of the formulation. Once an allocation is found for each component, the synthesis of the virtual processors is performed to find the optimal reservation parameters that can guarantee the schedulability of the applications. Then, a component integrator uses an MILP formulation for allocating all virtual processors to the physical processors to preserve the schedulability of the system.

Simulation experiments on synthetic applications have been carried out to validate the effectiveness of the approach. The achieved results showed that, under a representative setting, the proposed design methodology is able to admit 90 percent of the generated systems having utilization up to 3 on a quad-processor platform, in the presence of shared resources and reservations.

As a future work we plan to investigate non-linear optimization problem formulations for the task partitioning, as well as resource sharing driven heuristics, hence proposing a comparison study through extensive simulation experiments. Furthermore, it would be interesting to integrate the support for real-time lock-free algorithms to protect shared-memory objects [13].

Acknowledgements

This work has been partially supported by the RETINA Eurostars Project E10171 and received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement 688860. The authors like to thank Enrico Bini for the fruitful discussions that helped this work.

References

1. L. Abeni and G. Buttazzo. Resource reservations in dynamic real-time systems. *Real-Time Systems*, 27(2):123–165, 2004.
2. Z. Al-bayati, Y. Sun, H. Zeng, M. D. Natale, Q. Zhu, and B. Meyer. Task placement and selection of data consistency mechanisms for real-time multicore applications. In *Proc. of the 21st IEEE Real-Time and Embedded Technology and Application Symposium (RTAS 2015)*, Seattle, WA, USA, 2015.
3. T. P. Baker. Stack-based scheduling for realtime processes. *Real-Time Systems*, 3(1):67–99, April 1991.
4. 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.

5. S. Baruah and E. Bini. Partitioned scheduling of sporadic task systems: an ILP-based approach. In *Proc. of the Conference on Design and Architectures for Signal and Image Processing*, Bruxelles, Belgium, November 24-26, 2008.
6. S. Baruah, L. Rosier, and R. Howell. Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor. *Journal of Real-Time Systems*, 2, 1990.
7. S. K. Baruah. Partitioning real-time tasks among heterogeneous multiprocessors. In *Proceedings of the International Conference on Parallel Processing (ICPP 2004)*, Montreal, Quebec, Canada, August 15-18, 2004.
8. M. Behnam, T. Nolte, M. Sjödin, and I. Shin. Overrun methods and resource holding times for hierarchical scheduling of semi-independent real-time systems. *IEEE Transactions on Industrial Informatics*, 6(1):93–104, February 2010.
9. 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.
10. M. Bertogna, N. Fisher, and S. Baruah. Resource holding times: Computation and optimization. *Real-Time Systems*, 41(2):87–117, February 2009.
11. M. Bertogna, N. Fisher, and S. Baruah. Resource-sharing servers for open environments. *IEEE Transactions on Industrial Informatics*, 5(3):202–219, August 2009.
12. E. Bini and G. C. Buttazzo. Measuring the performance of schedulability tests. *Real-Time Systems*, 30(1-2):129–154, 2005.
13. A. Biondi and B. Brandenburg. Lightweight real-time synchronization under P-EDF on symmetric and asymmetric multiprocessors. In *Proceedings of the 28th Euromicro Conference on Real-Time Systems (ECRTS 16)*, July 2016.
14. A. Biondi, G. Buttazzo, and M. Bertogna. Partitioning and interface synthesis in hierarchical multiprocessor real-time systems. In *Proceedings of the 24th International Conference on Real-Time Networks and Systems (RTNS 2016)*, October 2016.
15. A. Biondi, G. Buttazzo, and M. Bertogna. 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, July 8-10, 2015.
16. A. Biondi, G. C. Buttazzo, and M. Bertogna. Schedulability analysis of hierarchical real-time systems under shared resources. *IEEE Transactions on Computers*, 65(5):1593 – 1605.
17. 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, June 18-20, 2014.
18. 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 2014)*, Madrid, Spain, July 9-11, 2014.
19. G. Buttazzo, E. Bini, and Y. Wu. Partitioning real-time applications over multicore reservations. *IEEE Transactions on Industrial Informatics*, 7(2):302–315, May 2011.
20. 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)*, pages 257–268, Rio de Janeiro, Brazil, Dec. 5-8, 2006.
21. N. Fisher, T. Baker, and S. Baruah. Algorithms for determining the demand-based load of a sporadic task system. In *Proceedings of the International Conference on Real-time Computing Systems and Applications (RTCSA)*, Sydney, Australia, August 2006.
22. 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*, 2001.
23. N. Khalilzad, M. Behnam, and T. Nolte. On component-based software development for multiprocessor real-time systems. In *Proc. 21st IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, August 2015.
24. C. Liu and J. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the Association for Computing Machinery*, 20(1):46–61, January 1973.

25. C. W. Mercer, S. Savage, and H. Tokuda. Processor capacity reserves for multimedia operating systems. In *Proceedings of IEEE international conference on Multimedia Computing and System*, May 1994.
26. M. D. Natale and A. S. Vincentelli. Moving from federated to integrated architectures in automotive: The role of standards, methods and tools. *Proc. of the IEEE*, 98(4):603–620, April 2010.
27. L. Thiele. Model-based design of real-time systems. In *Keynote speeach given at the 26th Euromicro Conference on Real-Time Systems (ECRTS 2014), Madrid, Spain*, July 10th, 2014.
28. 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)*, pages 45–56, December 2013.
29. A. Wieder and B. Brandenburg. Efficient partitioning of sporadic real-time tasks with shared resources and spin locks. In *Proc. of the 8th IEEE International Symposium on Industrial Embedded Systems (SIES 2013)*, June 2013.