The Generalized Multiprocessor Periodic Resource Interface Model for Hierarchical Multiprocessor Scheduling

Burmyakov, Artem; Bini, Enrico; Tovar, Eduardo

Published in:
[Host publication title missing]

2012

The Generalized Multiprocessor Periodic Resource Interface Model for Hierarchical Multiprocessor Scheduling

Artem Burmyakov*, Enrico Bini†, Eduardo Tovar*
*CISTER-ISEP Research Center, Polytechnic Institute of Porto, Portugal
†Lund University, Sweden

***Abstract*—Composition is a practice of key importance in software engineering. When real-time applications are composed it is necessary that their timing properties (such as meeting the deadlines) are guaranteed. The composition is performed by establishing an interface between the application and the physical platform. Such an interface does typically contain information about the amount of computing capacity needed by the application. In multiprocessor platforms, the interface should also present information about the degree of parallelism. Recently there have been quite a few interface proposals. However, they are either too complex to be handled or too pessimistic.**

In this paper we propose the Generalized Multiprocessor Periodic Resource model (GMPR) that is strictly superior to the MPR model without requiring a too detailed description. We describe a method to generate the interface from the application specification. All these methods have been implemented in Matlab routines that are publicly available.

I. Introduction

Reusing application code is a key design principle to shorten the overall design time. According to this design methodology, software components are designed in isolation, possibly by different developers. Then, during the integration phase, all components are bound to the same execution platform. Clearly, the integration must be performed in such a way that the properties of components are preserved even after the composition is made.

In real-time systems, the key property that has to be preserved during the integration phase is time predictability: a real-time application that meets all its deadlines when designed in isolation, should also meet all deadlines when it is integrated with other applications on the same system. This property is often guaranteed by introducing an *interface* between the application and the physical platform. Then the application is guaranteed over the interface, and the physical platform must provide a *virtual platform* that conforms with the interface. The scheduling problem over a virtual platform is often called *hierarchical scheduling* problem. In fact, the application tasks may contain an entire application in a hierarchical fashion. The benefit of using an interface-based approach is significant: during the design phase the interface of a virtual platform is designed such that the timing requirements of the application are met; during the integration phase the interfaces of all applications are combined over the same physical platform.

Typically, interfaces that allow composition of real-time components provide details about the amount of computation that can be provided by the virtual platform. This information can be provided with a varying degree of detail. For example, a very simple interface of a virtual processor can be just the fraction of provided time.

With the broad diffusion of multiprocessors, hierarchical scheduling problems have recently been considered over execution platforms that provide parallelism. The formulation of interface models for multiprocessor, however, requires the introduction of a new dimension: the *degree of parallelism*. This extra characteristic of the interface makes the problem certainly more challenging to be addressed.

The problem in selecting the appropriate interface is to find the most opportune balance between accuracy and simplicity of the interface. In this paper we propose a simple interface that is a generalization of a previously proposed one [20]. To better describe the context of our contribution, next we describe the most relevant related works.

A. Related works

The problem of composing real-time applications is certainly not new. There actually have been numerous contributions in this area. Being fully aware of the impossibility to provide a full coverage of the topic, we describe in this section the works that, to our best knowledge, are more related to ours.

One of the first papers to address the isolation of applications using *resource reservations* was published in 1993 by Parekh and Gallager [19], who introduced the Generalized Processor Sharing (GPS) algorithm to share a fluid resource according to a set of weights. Mercer *et al.* [17] proposed a more realistic approach where a resource can be allocated based on a required budget and period. Stoica *et al.* [22] introduced the Earliest Eligible Virtual Deadline First (EEVDF) for sharing the computing resource. Deng and Liu [6] achieved the same goal by introducing a

The research leading to these results was supported by the Marie Curie Intra European Fellowship within the 7th European Community Framework Programme.

two-level scheduler (using EDF as a global scheduler) in the context of multi-application systems. Kuo and Li [12] extended the approach to a Fixed Priority global scheduler. Kuo et al. [13] extended their previous work [12] to multiprocessors. However, they made very stringent assumptions (such as no task migration and period harmonicity) that restricted the applicability of the proposed solution.

Moir and Ramamurthy [18] proposed a hierarchical approach, where a set of P-fair tasks can be scheduled within a time partition provided by another P-fair task (called "supertask") acting as a server. However, the solution often requires the weight of the supertask to be higher than the sum of the weights of the served tasks [11].

Many independent works proposed to model the service provided by a uniprocessor through a supply function. Feng and Mok introduced the bounded-delay resource partition model [8]. Almeida *et al.* [1] provided timing guarantees for both synchronous and asynchronous traffic over the FTT-CAN protocol by using hierarchical scheduling. Lipari and Bini [15] derived the set of virtual processors that can feasibly schedule a given application. Shin and Lee [21] introduced the periodic resource model also deriving a utilization bound. Easwaran et al. [7] extended this model allowing the server deadline to be different than the period. Fisher and Dewan [9] proposed an approximation algorithm to test the schedulability of a task set over a periodic resource.

Recently, some authors have addressed the problem of how to specify the application interface for an application to be executed on multiprocessor systems, and provide appropriate schedulability analysis to check if the application is schedulable on the interface.

Leontyev and Anderson [14] proposed to use only the overall bandwidth requirement w as interface for soft real-time applications. The authors propose to allocate a bandwidth requirement of w onto $\lfloor w \rfloor$ dedicated processors, plus an amount of $w - \lfloor w \rfloor$ provided by a periodic server globally scheduled onto the remaining processors. An upper bound of the tardiness of tasks scheduled on such interface was provided.

Shin *et al.* [20] proposed the multiprocessor periodic resource model (MPR) that specifies a period, a budget and maximum level of parallelism of the resource provisioning. Since our work is a generalization of the MPR, in Section II-B we describe it in greater detail.

Chang *et al.* [5] proposed to partition the resource available from a multiprocessor by a static periodic scheme. The amount of resource is then provided to the application through a contract specification.

Bini *et al.* [4] proposed the Parallel Supply Function (PSF) interface of a virtual multiprocessor. This interface can be seen as a generalization of any possible interface model and it is the most resource-efficient. However, it is not investigated the assignment of the interface parameters that guarantee a real-time application.

Lipari and Bini [16] described an entire framework for composing real-time applications running over a multiprocessor. However their proposed interface was extremely simple.

B. Contributions of the paper

The contributions of the paper are highlighted in **bold** in the paragraph below.

In Section II we recall some previous interface models such as the Parallel Supply Function (PSF) and the Multiprocessor Resource Model (MPR). In Section III we provide an example illustrating that the MPR interface may require some more resource than actually needed. Section IV introduces **the Generalized Multiprocessor Periodic Resource model (GMPR)**. We also show **how to compute the PSF interface of a GMPR interface**. In Section V **a schedulability condition over a GMPR interface** is presented. This condition, inspired by the one proposed by Bertogna, Cirinei and Lipari [3], can be applied to several different policies for scheduling the application tasks. In Section VI we show **how to design a GMPR interface that requires the minimal resource and can guarantee a real-time application** specified by a set of sporadic tasks with deadline. In Section VII we briefly describe the problem of **scheduling the GMPR interfaces**. Finally, in Section VIII we report some simulations.

II. Background

As our work is tightly tied to several previous works, in this section we briefly review concepts and notations we borrow.

A. The Parallel Supply Function resource model

The parallel supply function (PSF) was proposed by Bini *et al.* [4] to characterize the resource allocation in hierarchical systems executed upon a multiprocessor platform. This interface introduces the minimum possible pessimism in abstracting the amount of resource provided by a platform. As a drawback it is certainly quite complicated to handled. Without entering all the details of the definition (that can indeed be found in [4]), we recall here the basic concepts.

Definition 1: The Parallel Supply Function interface (PSF) of a multiprocessor resource is composed by the set of functions $\{Y_k\}_{k=1}^{m}$, where $Y_k(t)$ is the *minimum* amount of resource provided in *any* interval of length t with a parallelism of *at most* k. The function $Y_k(t)$ is called the *level-k parallel supply function*.

To clarify this definition we propose an example. Suppose that in the interval $[0, 11]$ the resource is provided by three

processors according to the schedule drawn in gray in Figure 1.

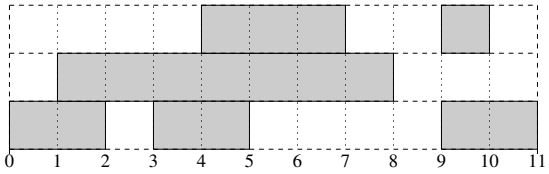

Figure 1. From a resource schedule to the PSF.

In this case $Y_1(11) = 10$ because there is always at least one processor available in $[0, 11]$ except in $[8, 9]$. Then $Y_2(11) = 16$; that is found by summing up all the resource except one with parallelism 3 (provided only in $[4, 5]$). Finally, $Y_3(11) = 17$; that is achieved by summing all the resources provided in $[0, 11]$. In general, the parallel supply functions are computed also by sliding the time window of length t and by searching for the most pessimistic scenario of resource allocation. This minimization is somehow equivalent to the one performed on uni-processor hierarchical scheduling [8], [15], [21].

Although the PSF interface is capable to tightly capture the amount of provided resource, its complexity prevents a straightforward application. It is unclear how a PSF interface should be designed so that an application is guaranteed. On the other extreme, next we report a very simple interface model.

B. The MPR interface model

The multiprocessor periodic resource model (MPR) [20] is one of the simplest resource abstractions. Its definition is as follows.

Definition 2: Let us set 0 as the time instant when the resource is firstly supplied. A Multiprocessor Periodic Resource model (MPR) is modeled by a triplet

$$\langle \Pi, \Theta, m \rangle,$$

where Π is the time period and Θ is the minimal amount of supply provided within each interval $[k\Pi, (k+1)\Pi)$, with $k \in \mathbb{N}$, by at most m processors. Often we also say that m is the *concurrency* (or the *degree of parallelism*) of the interface. The *utilization* of a MPR interface is the ratio $\frac{\Theta}{\Pi}$. In this work we assume that Π and Θ are positive integers.

Since a MPR interface fixes only the aggregated parameters Π, Θ and m of the supply pattern, any feasible allocation of Θ resource units per time period Π should preserve the schedulability of the underlying task set. In Figure 2, we show an example of the resource allocation of a MPR interface $\langle 7, 14, 3 \rangle$. It can be noted that in each period the allocation patterns may be different.

As the task set should be guaranteed under any possible resource allocation scenario, it is then necessary to find the

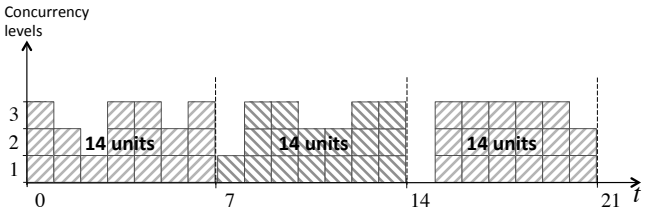

Figure 2. Graphical interpretation of a MPR model

worst-case supply allocation of the MPR. As shown by Shin *et al.* [20], the worst-case scenario is the one depicted in Figure 3. Since the PSF can be computed for any possible

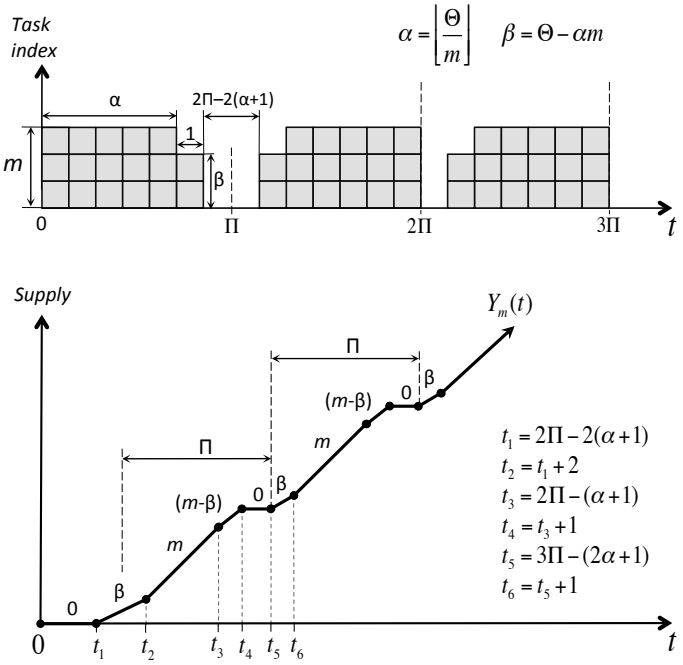

Figure 3. The worst-case supply allocation under the MPR model

resource allocation scheme, we can compute it also for the MPR interface. At the bottom of Figure 3 we show the level-m parallel supply function $Y_m(t)$ of a MPR interface. More details about this computation can be found in [20].

The computation of the PSF interface $\{Y_k\}_{k=1}^{m}$ of a MPR enables the adaptation of schedulability tests developed over a PSF interface to a MPR interface. More details about the schedulability test will be provided in Section V.

III. Motivation for extending the MPR interface

In this section we motivate the necessity for extending the MPR interface model. By proposing this extension we aim at minimizing the overall resource abstracted in the MPR interface required to guarantee the schedulability of the underlying task set.

i	C_i	T_i	D_i
1	6	40	40
2	13	50	50
3	29	60	60
4	27	70	70

Table I
AN EXAMPLE OF A TASK SET.

Assume that a MPR interface $\langle \Pi, \Theta, m \rangle$ abstracts the processing requirements of a real-time tasks set. By definition, a MPR interface specifies only the aggregated supply Θ. However, we show below that, preserving the schedulability, our approach allows to reduce the value of the required resource in the abstraction by further detailing its allocation in processors.

As an example, consider the tasks set with the parameters reported in Table I, to be scheduled by global EDF (GEDF) over the MPR interface. In this table, tasks are reported in rows and for each task we denote its execution time by C_i, its period by T_i, and its deadline by D_i.

After setting the period of the interface $\Pi = 15$, we compute a MPR interface $\langle \Pi, \Theta, m \rangle$ that can guarantee the task set. To check the schedulability, we reuse the PSF-based test proposed by Bini *et al.* [4] (see Section V for details). Based on this test, we determine that the minimal feasible value of resource to guarantee the schedulability is $\Theta = 39$. Notice that there is quite a significant gap between the utilization of the interface $\frac{\Theta}{\Pi} = 2.6$ and the utilization of the task set $\sum_i \frac{C_i}{T_i} = 1.28$.

As we will show in greater detail in the next sections, our proposed interface requires only 34 resource units per period, meaning that it has a utilization of $\frac{34}{15} = 2.267$.

IV. THE GENERALIZED MULTIPROCESSOR PERIODIC RESOURCE MODEL

As highlighted in Section III, the MPR resource model can lead to some waste of computational resources. In this section we describe a resource model that is better capable to tightly capture the resource requirement of the underlying task set.

Definition 3: Let us set 0 as the time instant when the resource is firstly supplied. We define the Generalized Multiprocessor Periodic Resource model interface (GMPR) as

$$\langle \Pi, \{\Theta_1, \ldots, \Theta_m\} \rangle,$$

where Π is the time period, Θ_k is the minimal supply provided by at most k processors. The period Π and all the values of Θ_k are positive integers. Also, the values of Θ_k must satisfy the following constraints for any $k = 1, \ldots, m$ (for notational convenience we denote $\Theta_0 = 0$):

$$\begin{aligned} 0 < \Theta_{k+1} - \Theta_k \leq \Pi \\ \Theta_{k+1} - \Theta_k \leq \Theta_k - \Theta_{k-1} \end{aligned} \quad (1)$$

By definition, a GMPR interface is a guarantee for the schedulability of a task set, meaning that any feasible supply allocation compliant to the GMPR model will result in meeting all the deadlines under the employed scheduling policy.

A. The Parallel Supply Functions of GMPR

To be able to borrow the schedulability tests developed over the PSF interface [4], we introduce the computation of the parallel supply functions $Y_k(t)$ for the GMPR specification.

Following a similar reasoning as for the MPR in [20], the worst-case supply pattern for the GMPR model is as depicted in Figure 4. Let us introduce an auxiliary function

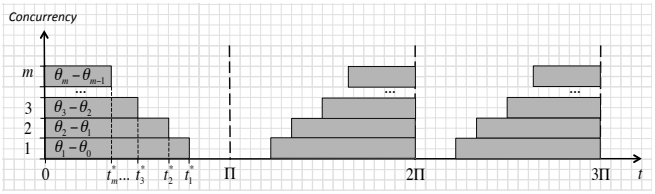

Figure 4. The worst-case supply allocation under the GMPR model

$\mathsf{supply}_k(t)$ to quantify the supply provided by the first k concurrency levels within the time interval $[0, t]$. According to the worst-case scenario of Figure 4, it follows that

$$\begin{aligned} \mathsf{supply}_k(t) = \\ \sum_{\ell=1}^{k} \min\{t, \Theta_\ell - \Theta_{\ell-1}\} + \left\lfloor \frac{(t-\Pi)_0}{\Pi} \right\rfloor \Theta_k + \\ + \sum_{\ell=1}^{k} (((t-\Pi)_0 \mod \Pi) - (\Pi - (\Theta_\ell - \Theta_{\ell-1})))_0 \end{aligned}$$

where $(x)_0$ denotes $\max(x, 0)$. Then, from the definition of the PSF function, it follows that

$$Y_k(\Delta t) = \min_{\forall t \geq 0} (\mathsf{supply}_k(t + \Delta t) - \mathsf{supply}_k(t))$$

Now we make the classic observation that a minimum of the previous expression must always occur at t equal to some instant of termination of a resource supply. These candidate time instants are denoted in Figure 4 by t_i^*. Hence the minimum can be computed over $T^* = \{t_1^*, t_2^*, ..., t_m^*\}$ without making any optimistic assumption. Therefore the PSF of a GMPR can be computed by

$$Y_k(\Delta t) = \min_{t \in T^*} (\mathsf{supply}_k(t + \Delta t) - \mathsf{supply}_k(t)). \quad (2)$$

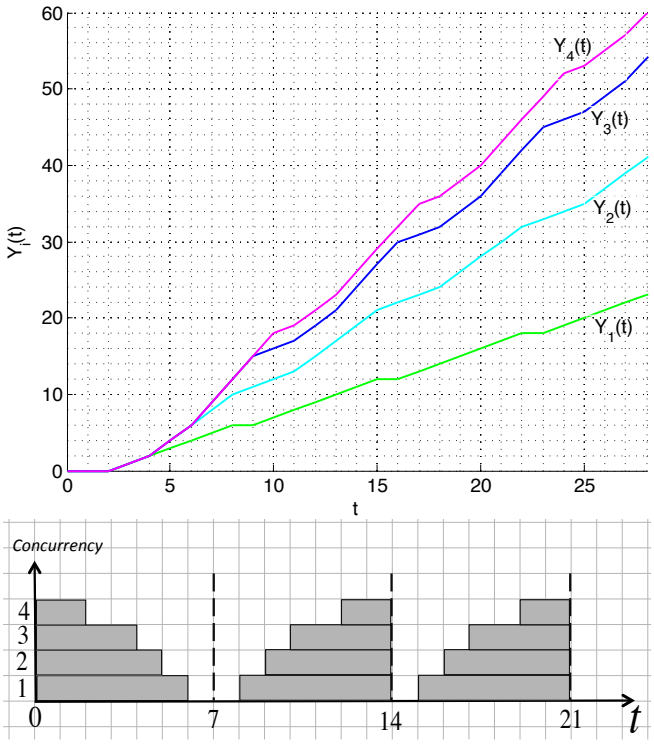

Figure 5. The PSF (top) and the worst-case supply pattern (bottom) of the GMPR interface $\langle 7, \{6, 11, 15, 17\}\rangle$

We also observe that the k-th function of the PSF can be upper bounded by the following simple linear function

$$Y_k(t) \leq \overline{Y}_k(t) = \frac{\Theta_k}{\Pi} t. \tag{3}$$

This upper bound will be exploited in Section VI to reduce the complexity of the algorithm to compute the GMPR interface of an application composed by a set of tasks.

As an example, in Figure 5 we illustrate the 4 parallel supply functions $\{Y_k(t)\}_{k=1}^4$ of the GMPR interface $\langle 7, \{6, 11, 15, 17\}\rangle$. At the bottom of the figure we also represent the worst-case resource supply that originates the parallel supply functions.

V. Schedulability over GMPR

The GMPR interface describes the amount of computing resources provided to an application. We can then formulate a schedulability test over the GMPR.

Let us consider a task set $\mathcal{T}$ composed by the tasks $\tau_1, \ldots, \tau_n$. Each task τ_i is modeled by its computation time C_i, period T_i, and deadline D_i.

As schedulability test for the application, we choose the extension of the test by Bertogna *et al.* [3] to the PSF interface developed in [4]. We choose this condition because it applies to several different application schedulers such as global EDF or global FP, although it assumes constrained deadline tasks, i.e. for all tasks τ_i, $D_i \leq T_i$. While choosing other tests is possible [2], the proposed formulation has the advantage of highlighting the constraint on the interface. Thanks to the lossless transformation of a GMPR interface into a PSF (see Section IV-A), we can apply directly the schedulability condition developed over PSF. Below we report, for completeness, the schedulability condition in the simpler expression proposed by Lipari and Bini [16].

Theorem 1 (Theorem 1 in [16]): A set of tasks $\{\tau_i\}_{i=1}^n$ is schedulable on a resource modeled by the PSF $\{Y_k\}_{k=1}^m$, if

$$\bigwedge_{i=1,\ldots,n} \bigvee_{k=1\ldots,m} k\, C_i + W_i \leq Y_k(D_i), \tag{4}$$

where W_i is the maximum *interfering workload* that can be experienced by task τ_i in the interval $[0, D_i]$, defined as

$$W_i = \sum_{j=1, j\neq i}^{n} \left\lfloor \frac{D_i}{T_j} \right\rfloor C_j + \min\left\{ C_j, D_i - \left\lfloor \frac{D_i}{T_j} \right\rfloor T_j \right\}, \tag{5}$$

if the application tasks are scheduled by global EDF. Instead if the application tasks are scheduled by global FP

$$W_i = \sum_{j \in \mathsf{hp}(i)} W_{ji}, \tag{6}$$

where hp denotes the set of indices of tasks with higher priority than i, and W_{ji} is the amount of interfering workload caused by τ_j on τ_i, that is

$$W_{ji} = N_{ji} C_j + \min\{C_j, D_i + D_j - C_j - N_{ji} T_j\} \tag{7}$$

with $N_{ji} = \left\lfloor \frac{D_i + D_j - C_j}{T_j} \right\rfloor$.

Below we exploit such a schedulability condition to compute the GMPR parameters $\Theta_1, \ldots, \Theta_m$ for a given task set.

VI. The GMPR Computation

When an application $\mathcal{T} = \{\tau_1, \ldots, \tau_n\}$ is given, it is of key importance to select the interface that can guarantee the timing constraints of the application and, at the same time, requires the minimal amount of resource. Hence, in this section we describe an algorithm to generate a GMPR interface $\langle \Pi, \{\Theta_k\}_{k=1}^m \rangle$ of a given sporadic task set $\{\tau_1, \ldots, \tau_n\}$. As schedulability condition, we choose the one of Theorem 1.

To compute a GMPR interface, we follow a similar approach as the one proposed by Shin *et al.* [20] to generate a MPR interface. First, the period Π of the GMPR interface is set by the system designer considering such aspects as preemption overheads and etc. Then for a fixed value of m (the parallelism of the interface) not smaller than $\left\lceil \sum_i \frac{C_i}{T_i} \right\rceil$, our algorithm finds the values of cumulative

Algorithm 1 Reduction of the search space.

```
1:  procedure REDUCESEARCHSPACE
2:      S_Θ ← ∅                                   ▷ initialize S_Θ
3:      for each τ_i ∈ T do
4:          compute v^i                           ▷ from Eq. (9)
5:          S_new ← {v_i}                         ▷ initialize S_new
6:          for v ∈ S_Θ do
7:              if ∀k, v^i_k ≤ v_k then
8:                  S_new ← ∅                     ▷ ignore v^i
9:                  break
10:             end if
11:             if ∀k, v^i_k ≥ v_k then
12:                 S_Θ ← S_Θ \ {v}               ▷ remove v
13:             end if
14:         end for
15:         S_Θ ← S_Θ ∪ S_new
16:     end for
17:     return S_Θ
18: end procedure
```

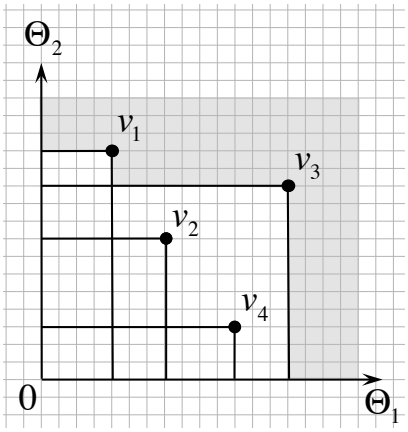

Figure 6. Illustration of the search space reduction.

i	C_i	T_i	D_i	W_i (GEDF)
1	12	40	40	38
2	23	50	50	37
3	15	60	60	57

Table II
AN EXAMPLE OF A TASK SET.

resource $\Theta_m, \ldots, \Theta_1$ such that the computing resource is minimized.

Rather than simply (but in a very time consuming way) enumerating all possible values of Θ_k as proposed by Shin *et al.* [20], we exploit the condition on Θ_k that follows from the linear upper bound of Eq. (3). In fact, from (4) and (3) it follows that any feasible values of $\Theta_1, \ldots, \Theta_m$ must also be such that

$$\bigwedge_{i=1,\ldots,n} \bigvee_{k=1\ldots,m} k\,C_i + W_i \leq \frac{\Theta_k}{\Pi} D_i,$$

from which we have the following condition on all Θ_k

$$\bigwedge_{i=1,\ldots,n} \bigvee_{k=1\ldots,m} \Theta_k \geq \left\lceil \frac{\Pi}{D_i}(k\,C_i + W_i) \right\rceil, \qquad (8)$$

by also accounting for the integrality of Θ_k.

The necessary condition of Eq. (8) can be exploited to reduce significantly the search space. For any task τ_i, let us define the vector $v_i \in \mathbb{N}^m$ as

$$v^i = \left[\left\lceil \frac{\Pi}{D_i}(C_i + W_i) \right\rceil, \ldots, \left\lceil \frac{\Pi}{D_i}(mC_i + W_i) \right\rceil \right]. \qquad (9)$$

The reduced search space is computed by Algorithm 1. We illustrate its execution by an example.

Let us assume to have 4 tasks and $m = 2$. Let us also assume that the values of v^1, v^2, v^3, v^4 are the ones depicted in Figure 6. In the first run of the outer loop (lines 3–16) the set S_Θ is empty. Then v^1 is simply added to S_Θ. When $i = 2$, none of the two conditions of lines 7, 11 are true, hence v_2 is also added to S_Θ. When $i = 3$, the condition at line 11 is true when $v = v^2$. Hence, v^2 can be removed from S_Θ because the schedulability condition (8) for $i = 3$ is stricter than the one for $i = 2$. Finally, when $i = 4$ the condition at line 7 is true when $v = v^3$ and then the vector v^4 can be ignored. It can be noted that Algorithm 1 for determining the reduced search space has complexity $o(n^2 m)$ that is polynomial. Moreover its result does not depend on the order in which the vectors v^i are visited.

Once S_Θ is determined by Algorithm 1, the GMPR generation process is then based on searching the assignment that requires the minimum amount of resource among all values $(\Theta_1, \ldots, \Theta_m)$ satisfying the following constraints

$$\forall v \in S_\Theta \;\; \exists k = 1, \ldots m, \quad \Theta_k \geq v_k \qquad (10)$$

$$\Theta_1 \leq \Pi \qquad (11)$$

$$\forall k = 1, \ldots, m-1 \quad \Theta_{k+1} - \Theta_k \leq \Theta_k - \Theta_{k-1} \qquad (12)$$

$$\Theta_m \geq \Theta_{m-1} \qquad (13)$$

where Condition (10) follows from (8), while Conditions (11)–(13) follow from Definition 3 of the GMPR interface.

A. Example of GMPR computation

We illustrate the algorithm by an example. Let us consider the task set $\mathcal{T}$ with the parameters reported in Table II. If the task set is scheduled by GEDF over the interface then, from Eq. (5), we can compute the quantities W_i that are reported in the last column of the table.

We set $\Pi = 15$ and $m = 2$. From (9), we have that $v^1 = (19, 24)$, $v^2 = (18, 25)$, and $v^3 = (18, 22)$. However,

by executing the REDUCESEARCHSPACE algorithm we find that the vector v^3 can be ignored, since the condition (8) with $i = 3$ is implied by the others. Hence $S_\Theta = \{v^1, v^2\}$.

The search space is depicted in Figure 7, in gray. Figure 7(a) shows the feasible values of (Θ_1, Θ_2) by only considering the constraints (11)–(13) that follow from Definition 3 of GMPR. In Figure 7(b) we show how much the search space is shrunk by enforcing the necessary condition of (8). Among the possible selections of (Θ_1, Θ_2), in Figure 7(b), we also show, which ones are capable to guarantee the deadline constraints of the task set (denoted by a black dot) and which ones are not (denoted by a red cross). Hence the GMPR interface that consumes the minimal amount of resource is $\langle 15, \{15, 26\}\rangle$. It is also interesting to observe that in this example the best MPR interface was $\langle 15, 27\rangle$ that consumes one unit of resource more than the best GMPR.

VII. SCHEDULABILITY ANALYSIS OF GMPR INTERFACES

Once the processing requirements of each component in a hierarchical system are abstracted using GMPR interfaces, they should be scheduled upon a hardware platform. For this purpose we introduce a notion of interface tasks. An interface task set for a GMPR interface $\langle \Pi, \{\Theta_k\}_{k=1}^m \rangle$ is defined as

$$\mathcal{T}' = \{\tau'_1 = (C'_1, \Pi), \ldots, \tau_m = (C'_m, \Pi)\},$$

where $C'_i = (\Theta_k - \Theta_{k-1})$. We recall that we set $\Theta_0 = 0$ for notational convenience. It is easy to see that the overall processing requirement of $\mathcal{T}'$ is Θ_m per period Π as

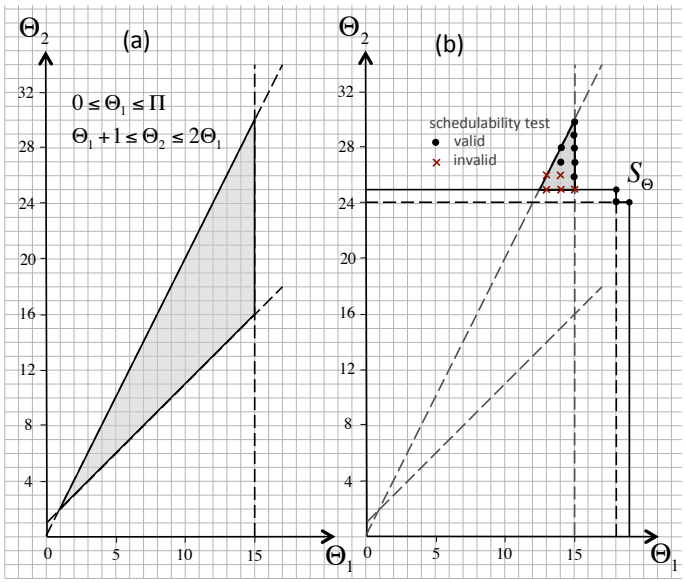

Figure 7. The example of a GMPR interface computation

$\sum_{k=1}^m C'_k = \Theta_m$. Therefore, we propose to schedule GMPR interfaces by transforming each one into interface tasks and to schedule the resulting union of these periodic tasks instead.

The notion of interface tasks supports another important property for hierarchical systems, which is *composability:* by the given GMPR interfaces of child components we can compute a GMPR interface of a parent component.

VIII. IMPLEMENTATION AND SIMULATIONS

The algorithm for generating GMPR interfaces is implemented in Matlab and it is available at http://retis.sssup.it/~bini/publications/2012GMPR.html.

In the performed experiments, we compared the utilization of the interface $\frac{\Theta_m}{\Pi}$ as the interface period Π varies. For all the three experiments reported below we plot the interface utilization of GMPR and MPR for both FP and EDF scheduling policies. The experiments were conducted by randomly generating task sets. All the experiments share the following characteristics:

- the minimum task period was random extracted between 20 and 40,
- the total utilization of tasks was set equal to $U = 1.5$, and
- the number of processors was set equal to $m = 4$.

In the first experiment, reported in Figure 8, we set the maximum utilization of a single task equal to $U_{\max} = 0.4$ and the ratio between the maximum and minimum task periods $\frac{T_{\max}}{T_{\min}} = 1.5$. It can be observed that the gain in term

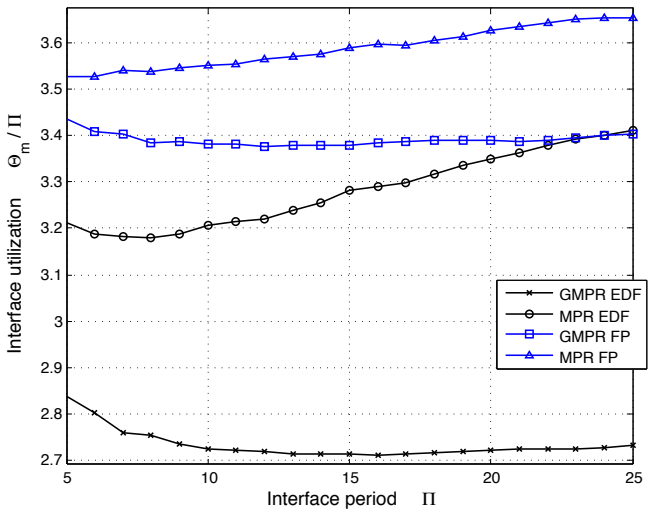

Figure 8. Case (a): $U_{\max} = 0.4$, $\frac{T_{\max}}{T_{\min}} = 1.5$.

of overall resource usage of GMPR w.r.t. MPR is in the order of 5%, when tasks are scheduled by FP (blue plots) and around 10% when tasks are scheduled by EDF (black

plots). Notice that the gain of GMPR increases with the period of the interface.

To explore the dependency on the weight of the individual tasks, in the second experiment we set $U_{\max} = 0.7$, keeping the ratio $\frac{T_{\max}}{T_{\min}} = 1.5$. Results are shown in Figure 9. With

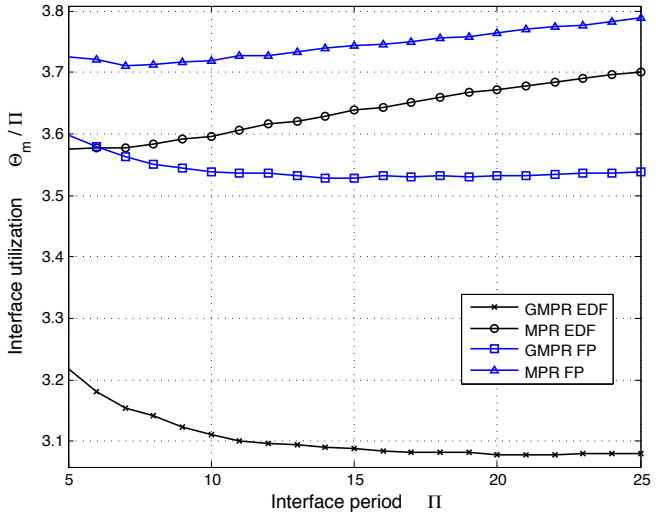

Figure 9. Case (b): $U_{\max} = 0.7$, $\frac{T_{\max}}{T_{\min}} = 1.5$.

these settings, the gain of GMPR compared to MPR is in the order of 10% for FP (blue plots) and 15% for EDF (black plots). The trend with an increasing gain as a function of Π is confirmed.

In the third and final experiment (depicted in Figure 10), we also investigate the dependency on the task periods by setting $\frac{T_{\max}}{T_{\min}} = 10$ and $U_{\max} = 0.4$. An interesting

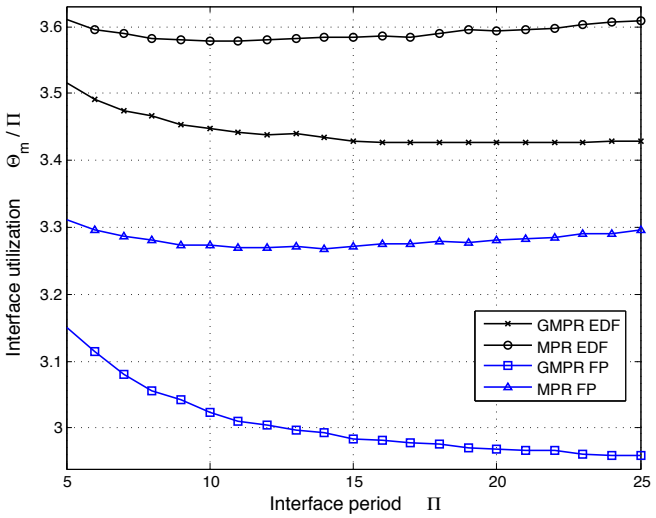

Figure 10. Case (c): $U_{\max} = 0.4$, $\frac{T_{\max}}{T_{\min}} = 10$.

phenomenon that we observe in this case is that FP requires a smaller amount of resource w.r.t. EDF. This has to be explained with the nature of the schedulability test. The gains of GMPR over MPR are in the same order of magnitude as in the previous experiments.

In all experiments we can observe a quite significant distance between the interface utilization, always around 3 and the task set utilization that is 1.5. This waste of resource, however, does not depend on the particular interface selected. It has instead to do with the pessimism introduced by the schedulability tests. We believe that if the schedulability tests can be tightened, for example by using more sophisticated tests that better account for the amount of task interference [10], then the loss due to the interface can certainly be reduced as well.

IX. Conclusions

Motivated by the need to save resource, we introduced the Generalized Multiprocessor Periodic Resource model. Since GMPR is a generalization of MPR, it can consume at most as much as MPR. We provided a schedulability algorithm for task sets scheduled over GMPR by FP or EDF. We also provided an algorithm that is capable to select the minimal interface parameters for a given set of tasks.

References

[1] Luís Almeida, Paulo Pedreiras, and José Alberto G. Fonseca. The FTT-CAN protocol: Why and how. *IEEE Transaction on Industrial Electronics*, 49(6):1189–1201, December 2002.

[2] Sanjoy Baruah, Vincenzo Bonifaci, Alberto Marchetti-Spaccamela, and Sebastian Stiller. Improved multiprocessor global schedulability analysis. *Real-Time Systems Journal*, 46:3–24, 2010.

[3] Marko Bertogna, Michele Cirinei, and Giuseppe Lipari. Schedulability analysis of global scheduling algorithms on multiprocessor platforms. *IEEE Transactions on Parallel and Distributed Systems*, 20(4):553–566, April 2009.

[4] Enrico Bini, Marko Bertogna, and Sanjoy Baruah. Virtual multiprocessor platforms: Specification and use. In *Proceedings of the* 30^{th} *IEEE Real-Time Systems Symposium*, pages 437–446, Washinghton, DC, USA, December 2009.

[5] Yang Chang, Robert Davis, and Andy Wellings. Schedulability analysis for a real-time multiprocessor system based on service contracts and resource partitioning. Technical Report YCS 432, University of York, 2008. available at http://www.cs.york.ac.uk/ftpdir/reports/2008/YCS/432/YCS-2008-432.pdf.

[6] Zhong Deng and Jane win-shih Liu. Scheduling real-time applications in Open environment. In *Proceedings of the* 18^{th} *IEEE Real-Time Systems Symposium*, pages 308–319, San Francisco, CA, U.S.A., December 1997.

[7] Arvind Easwaran, Madhukar Anand, and Insup Lee. Compositional analysis framework using EDP resource models. In *Proceedings of the* 28^{th} *IEEE International Real-Time Systems Symposium*, pages 129–138, Tucson, AZ, USA, 2007. IEEE Computer Society.

[8] Xiang Feng and Aloysius K. Mok. A model of hierarchical real-time virtual resources. In *Proceedings of the* 23^{rd} *IEEE Real-Time Systems Symposium*, pages 26–35, Austin, TX, U.S.A., December 2002.

[9] Nathan Fisher and Farhana Dewan. Approximate bandwidth allocation for compositional real-time systems. In *Proceedings of the* 21^{st} *Euromicro Conference on Real-Time Systems*, pages 87–96, Dublin, Ireland, July 2009.

[10] Nan Guan, Martin Stigge, Wang Yi, and Ge Yu. New response time bounds for fixed priority multiprocessor scheduling. In *Proceedings of the* 30^{th} *IEEE Real-Time Systems Symposium*, pages 387–397, Washington, DC, U.S.A., December 2009.

[11] Philip Holman and James H. Anderson. Group-based pfair scheduling. *Real-Time Systems*, 32(1–2):125–168, February 2006.

[12] Tei-Wei Kuo and Ching-Hui Li. Fixed-priority-driven open environment for real-time applications. In *Proceedings of the* 20^{th} *IEEE Real-Time Systems Symposium*, pages 256–267, Phoenix, AZ, U.S.A., December 1999.

[13] Tei-Wei Kuo, K. Lin, and Y. Wang. An open real-time environment for parallel and distributed systems. In *Proceedings of the* 20^{th} *International Conference on Distributed Computing Systems*, pages 206–213, Taipei, Taiwan, April 2000.

[14] Hennadiy Leontyev and James H. Anderson. A hierarchical multiprocessor bandwidth reservation scheme with timing guarantees. In *Proceedings of the* 20^{th} *Euromicro Conference on Real-Time Systems*, pages 191–200, Prague, Czech Republic, July 2008.

[15] Giuseppe Lipari and Enrico Bini. Resource partitioning among real-time applications. In *Proceedings of the* 15^{th} *Euromicro Conference on Real-Time Systems*, pages 151–158, Porto, Portugal, July 2003.

[16] Giuseppe Lipari and Enrico Bini. A framework for hierarchical scheduling on multiprocessors: from application requirements to run-time allocation. In *Proceedings of the* 31^{st} *Real-Time Systems Symposium*, pages 249–258, San Diego, CA, USA, December 2010.

[17] Clifford W. Mercer, Stefan Savage, and Hydeyuki Tokuda. Processor capacity reserves: Operating system support for multimedia applications. In *Proceedings of IEEE International Conference on Multimedia Computing and Systems*, pages 90–99, Boston, MA, U.S.A., May 1994.

[18] Mark Moir and Srikanth Ramamurthy. Pfair scheduling of fixed and migrating periodic tasks on multiple resources. In *Proceedings of the* 20^{th} *IEEE Real-Time Systems Symposium*, pages 294–303, Phoenix, AZ, U.S.A., December 1999.

[19] Abday K. Parekh and Robert G. Gallager. A generalized processor sharing approach to flow control in integrated services networks: the single-node case. *IEEE/ACM Transactions on Networking*, 1(3):344–357, June 1993.

[20] Insik Shin, Arvind Easwaran, and Insup Lee. Hierarchical scheduling framework for virtual clustering multiprocessors. In *Proceedings of the* 20^{th} *Euromicro Conference on Real-Time Systems*, pages 181–190, Prague, Czech Republic, July 2008.

[21] Insik Shin and Insup Lee. Periodic resource model for compositional real-time guarantees. In *Proceedings of the* 24^{th} *Real-Time Systems Symposium*, pages 2–13, Cancun, Mexico, December 2003.

[22] Ion Stoica, Hussein Abdel-Wahab, Kevin Jeffay, Sanjoy K. Baruah, Johannes E. Gehrke, and Charles Gregory Plaxton. A proportional share resource allocation algorithm for real-time, time-shared systems. In *Proceeding of the* 17^{th} *IEEE Real Time System Symposium*, pages 288–299, Washington, DC, U.S.A., December 1996.