Exact and Approximate Supply Bound Function for Multiprocessor Periodic Resource Model: Unsynchronized Servers[*] [†]

Nima Moghaddami Khalilzad, Moris Behnam and Thomas Nolte
MRTC/Mälardalen University
P.O. Box 883, SE-721 23 Västerås, Sweden
nima.m.khalilzad@mdh.se

ABSTRACT

The Multiprocessor Periodic Resource (MPR) model has been proposed for modeling compositional real-time guarantees of real-time systems which run on a shared multiprocessor hardware. In this paper we extend the MPR model such that the execution of virtual processors (servers) is not assumed to be synchronized i.e., the servers can have different phases. We believe that relaxing the server synchronization requirement provides greater deal of compatibility for implementing such a compositional method on various hardware platforms. We derive the resource supply bound function of the extended MPR model using an algorithm. Furthermore, we suggest an approach to calculate an approximate supply bound function with lower computational complexity for systems where calculating their supply bound function is computationally expensive.

1. INTRODUCTION

In order to deal with increasing complexity of real-time systems, hierarchical scheduling techniques have been proposed and investigated for scheduling complex real-time systems consisting of multiple real-time components (applications) on a shared underlying hardware platform. Using such techniques components are developed independently and their timing behaviors are studied in isolation, while the correctness of the system is inferred from the correctness of its components. In the mean time, following the trend of servers and PCs, embedded real-time systems are subjected to the paradigm shift from single processor to multiprocessor hardware platforms. Therefore, there is a need for new techniques that can enable hierarchial scheduling on multiprocessor platforms which allow us to compose real-time systems and run them on a multiprocessor hardware. Recently, many studies have been conducted on this subject and a variety of models have been proposed.

In modeling hierarchical real-time systems, single processors alike multiprocessor, the system model often consists of two parts: resource supply model and task demand model. The resource supply model abstracts the underlying hardware resource such that each application has the illusion of running solo on an independent hardware, this virtual hardware is often called a *server*. The resource supply model represents the minimum amount of resource that a server provides in a given time interval. The amount of provided resource is often represented using a Supply Bound Function ($\texttt{sbf}(t)$). The resource demand model, however, represents the resource demand of real-time tasks. Similarly, the maximum demand is often represented using a Demand Bound Function ($\texttt{dbf}(t)$) [1]. Consequently, the schedulability test is performed using the $\texttt{sbf}(t)$, which is dependent on the resource model, and the $\texttt{dbf}(t)$ which is dependent on the scheduling policy.

When it comes to multiprocessor platforms, the resource supply model can either be flexible and represent the collectively provided resource of a set of processors [6], or it can be more detailed and represent the exact amount of provided resource by each processor. In the former case, as Lipari and Bini state in [9], the $\texttt{sbf}(t)$ depends on the fact that whether different servers (on different processors) are synchronized together or not. A number of works on multiprocessor hierarchical scheduling assume that the servers are synchronized [6, 13], while in this paper alike [9], we assume that the servers are not synchronized. Indeed synchronization on some hardware platforms can be expensive, therefore, we simplify the implementation phase of the composition for the system developers by relaxing this assumption. Figure 1 illustrates the supply bound function of a multiprocessor periodic resource model for two cases: synchronized servers and unsynchronized servers (the specifications are explained later in Example 1). The figure shows that when the servers are not synchronized the supply bound function at some points in time is lower than the synchronized servers case. The figure indicates that the schedulability analysis that is based on the assumption of having synchronized servers is not valid when this assumption is relaxed.

In this paper, we focus on the supply bound function of the multiprocessor periodic resource model, and we present an approach to calculate the $\texttt{sbf}(t)$ of the flexible resource model that Easwaran *et al.* presented in [6] with no assumptions on the server synchronization. Our approach is based on mapping the flexible model to a model that represents the exact amount of the contributed budget by each processor to the total budget, and then we derive the $\texttt{sbf}(t)$ for the new model. Furthermore, we present an approach for approximating the $\texttt{sbf}(t)$ which has lower computational

[†]This work is supported by the Swedish Research Council.
[*]Copyright is held by the authors.

complexity than calculating the actual $\mathtt{sbf}(t)$.

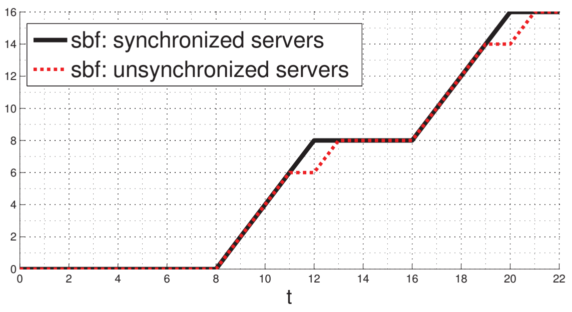

Figure 1: The $\mathtt{sbf}(t)$ of synchronized and unsynchronized servers

The rest of the paper is organized as follows. We first review the related work in Section 2, then we present the resource model in Section 3. The algorithm for calculating the exact supply bound function is presented in Section 4. Thereafter, we present the approximate supply bound function in Section 5. Finally, we conclude the paper in Section 6.

2. RELATED WORK

Hierarchial scheduling was first proposed as a method for composing real-time systems on single processor hardware platforms. Enabling independent development of real-time systems, Deng and Liu proposed hierarchical scheduling in [4]. Schedulability analysis under global fixed priority scheduling is presented in [7]. Mok *et al.* presented the bounded-delay model for single processor hardware platforms in [10]. Shin and Lee presented the periodic resource model for single processors in [11].

Virtual clustered-based multiprocessor scheduling [6], which is the extension of the periodic resource model for multiprocessor platforms, provides a flexible mechanism for scheduling hierarchical systems. In this approach the resource supply is abstracted using a Multiprocessor Periodic Resource (MPR) interface. The MPR interface consists of P, Q and m parameters which denote the total budget Q is provided in each period P using m virtual processors. This model provides a great deal of run-time flexibility since the budget distribution among m processors is performed during run-time depending on the load of processors. This flexibility can be exploited by the scheduler to serve the real-time tasks in an efficient way. It is shown in [6] that the minimum supply bound happens in the case where the total required budget is evenly divided among all processors and each processor's budget is equal to $\frac{Q}{m}$. Therefore, the supply bound function is derived based on this worst case budget distribution setting (this setting is called the *worst case platform* in [9]).

The main problem with the MPR interface is that it has an implicit assumption of the synchronization among virtual processors. It has been shown in [9] that the worst case platform does not exist if the virtual processors are not synchronized. Therefore, Lipari and Bini suggested a new interface model, namely the Bounded-Delay Multipartition (BDM) model to overcome this problem. The BDM interface consists of m, Δ and $[\beta_1,...,\beta_m]$ parameters which represent the number of virtual processors, the length of the longest interval with no resource and the bandwidth at each parallelism level respectively. In fact, the DBM model replaces the notion of period P in the MPR with delay (the longest interval with no resource) Δ. The BDM model does not require the servers to be synchronized. Nevertheless, due to the nature of the delay based models, the BDM can be very pessimistic which can result in low system run-time utilization and consequently higher cost of the system production. Besides, from an implementation point of view, periodic servers are more straight forward to implement, and the BDM model perhaps should be mapped to the MPR or any other periodic server based model for the implementation.

Bini *et al.* presented the Multi Supply Function (MSF) model in [3] for modeling the resource supply in hierarchial scheduling on multiprocessor platforms. The MSF is indeed a set of supply functions one associated with each server. The Parallel Supply Function (PSF) model [2] is also proposed as an alternative for modeling the resource supply of hierarchial multiprocessor systems. This model indicates a set of supply functions where each of them represent the minimum available supply at a certain parallelism level (from 1 to m). Since the MPR model offers greater deal of abstraction than the MSF and the PSF model, from a system integrator perspective, the MPR can be more suitable when composing real-time systems.

Zhu *et al.* have extended deferable servers to the context of multiprocessor platforms [13] where m deferable servers with a common period and different budgets are running on m processors. Analogous to the MPR model, the servers are assumed to be synchronized in this work.

Targeting soft real-time tasks, Leontyev and Anderson have presented a multi-level scheme for scheduling real-time tasks and they showed that under their scheme, the deadline tardiness of the tasks is bounded [8]. In contrast with other hierarchical schemes, in this work there is no loss in overall utilization moving down through the levels of hierarchy.

3. RESOURCE MODEL

We present the resource model for a processor cluster in this section. On a multiprocessor platform consisting of a total of n processors, a processor cluster is a set of m processors where $1 \leq m \leq n$. The processor clusters can be either physically or virtually mapped to the physical processors [6].

We present two types of resource interface models for the processor clusters: flexible and rigid. While the flexible model is the main focus in this paper, the reason behind introducing the rigid interface is that we use it to derive the supply bound function of the flexible interfaces.

3.1 Flexible interface model

Our flexible resource interface model is equivalent to the one that Easwaran *et al.* introduced in [6]. In this model resources are specified by the following tuple: $\Gamma = \langle m, P, Q \rangle$, which denotes that the multiprocessor cluster consisting of m processors in total provides Q units of budget every P period to its corresponding consumers. From a run-time point of view, this model is very flexible in the sense that the scheduler can decide how the total budget should be distributed among the processors in the cluster, in other words, each processor is free to provide as much resource as it wants, as long as the collective provided budget is equal to Q every P time units.

3.2 Rigid interface model

In contrast to the flexible interface model, in the rigid interface model each processor in the cluster is required to provide a specific amount of resource to its corresponding consumers. The rigid interface model is represented as follows: $\psi = \langle m, P, [q_1, ..., q_m] \rangle$, where q_i represents the exact amount of the budget of processor i ($1 \leq i \leq m$). Without loss of generality we assume that all q_i are stored non-increasingly i.e. $\forall i \; q_i \geq q_{i+1}$. In this model, the total provided budget is calculated by accumulating the budget of all processors in the cluster: $\sum_{i=1}^{m} q_i$. We use the following notation to refer to the budget distribution of a known platform (ψ): q_i^{ψ} where $1 \leq i \leq m$. Similarly Q^{Γ} represents the total available budget of the flexible interface Γ. Note that in this model processor i, regardless of the budget of other processors in the cluster, is obliged to provide q_i budget each period and it does not need to be synchronized with the other processors in its cluster.

We overload the word "platform" in the rest of the paper to refer to a rigid processor cluster interface ψ. Since the total budget can be distributed among processors in many ways, a single flexible interface Γ can be mapped to many platforms. We call the set of all possible platforms derived from a flexible interface Γ the *possible platforms of* Γ and we represent this set as follows

$$\Psi^{\Gamma} = \left\{ \forall \psi : \sum_{i=1}^{m} q_i^{\psi} = Q^{\Gamma} \right\}.$$

Note that when Q is not integer, we solve the mapping problem for $\lfloor Q \rfloor$ and then we add $Q - \lfloor Q \rfloor$ to q_1. Therefore, we assume that in the rigid model there exist at most one real budget (q_1), while the rest of the processors have integer budgets. The restriction that only a single processor will have real budget, limits the set of possible rigid interfaces that can be derived from a flexible interface.

3.3 Flexible interface versus rigid interface

So far we have introduced two interface models which can be used for composing real-time components on a multiprocessor hardware. When using rigid models, each component has its own q_i^{ψ} and the system integrator has to find a way to allocate q_i to the physical processors. This problem is a bin packing like problem which is known to be difficult to be solved. In addition, when adding or removing components, the allocation should be repeated.

On the other hand, when using a flexible interface we do not face this allocation problem and the scheduler is free to decide the allocations in any fashion at run-time. This property makes the flexible interfaces more suitable for compositional analysis in the sense that the integration phase is done without the need to consider the physical allocation of the components. However, calculating the supply bound function when using flexible interfaces, as we discuss in this paper, requires more computations than using rigid interfaces.

Therefore, there is a downside to both of the models and choosing either of them is a design decision which should be made by the system designers.

3.4 Packed platform of a flexible interface

The packed platform (ψ_p) of a flexible interface Γ is a member of Ψ^{Γ} in which the total budget Q is packed onto the minimum number of processors. A packed platform consists of $h = \lfloor \frac{Q}{P} \rfloor$ full budgets ($q_i = P$), one budget equal to $\mod(Q, P)$, and $m - h$ empty budgets ($q_i = 0$). For example the corresponding packed platform of the flexible interface $\Gamma = \langle 4, 8, 18 \rangle$ is $\psi_p = \langle 3, 8, [8, 8, 2, 0] \rangle$.

3.5 Balanced platform of a flexible interface

The balanced platform (ψ_b) of a flexible interface Γ is a member of Ψ^{Γ} in which the total budget is evenly divided among all processors in the cluster. Therefore, the balanced platform consists of $k = \mod(Q, m)$ budgets equal to $\lfloor \frac{Q}{m} \rfloor + 1$ and $m - k$ budgets equal to $\lfloor \frac{Q}{m} \rfloor$. For instance the balanced platform of $\Gamma = \langle 4, 8, 18 \rangle$ is $\psi_b = \langle 4, 8, [5, 5, 4, 4] \rangle$.

3.6 Deriving the possible platforms of a flexible interface

This problem of deriving the possible platforms of a flexible interface is analogous to the well know integer partitioning problem in number theory [12], where the problem is to find all possible ways that an integer number x can be written as a sum of some integer numbers which are called the partitions of x. However, in our problem we have two additional constraints which are the maximum number of partitions (m) and the maximum value of each partition (P). Hence, we can not directly use the algorithms presented for deriving the partitions of integer numbers. Therefore, the problem is to find all possible ways of writing Q as sum of ℓ integer numbers (q_i) where $\ell \leq m$, and each partition value is less than or equal to P. Recall that for avoiding redundant platforms we enforce the following requirement $\forall i \; q_i \geq q_{i+1}$, which is due to the fact that redundant platforms have equivalent `sbf`(t) and therefore are not of our interest.

In the rest of this section we present an algorithm for deriving the possible platforms of a given flexible interface. We start from the balanced platform and construct a tree where the root is ψ_b and new nodes are created by transferring a unit of the budget from one processor to another one. We present some definitions before presenting the algorithm.

Budget donator candidate is a processor that if its budget is reduced by one the remaining budget set is still ordered (non-increasingly). Any given platform has a budget donator candidate set (D^{ψ}) that is found using Algorithm 1. The algorithm loops through all budgets and selects the ones that are compatible with donating a unit of budget. `insert` is a function that inserts a new entry (i) to its input set (here D^{ψ}).

Algorithm 1 Deriving the budget donator candidate set

```
 1: function donators($\psi$)
 2: for $i = 2; i < m; i++$ do
 3:     if $q_i > 0$ & $q_i > q_{i+1}$ then
 4:         insert($D^{\psi}, i$);
 5:     end if
 6: end for
 7: if $q_m > 0$ then
 8:     insert($D^{\psi}, m$);
 9: end if
10: end function
```

Budget receiver candidate is a processor that if its budget is increased by one the remaining budget set is still ordered (non-increasingly). There is a budget receiver set associated with each budget donator of platforms (R_d^{ψ}) which is derived using Algorithm 2. The algorithm only loops through the budgets that are at the left hand side of the budget donator d, and finds the processors that are compatible with receiving a unit of budget.

Algorithm 2 Deriving the budget receiver candidate set of a given budget donator (d)

```
 1: function receivers($\psi, d$)
 2: for $i = 2; i < d; i++$ do
 3:     if $q_i < P$ & $q_i < q_{i-1}$ then
 4:         insert($R_d^{\psi}, i$);
 5:     end if
 6: end for
 7: if $q_1 < P$ then
 8:     insert($R_d^{\psi}, 1$);
 9: end if
10: end function
```

Budget donation operation is an operation in which one unit of a budget donator's budget q_d is transferred to a budget receiver budget q_r.

In order to derive Ψ^{Γ}, we start off by running the budget donation operation on ψ_b, for all combinations of the donators and their corresponding receivers. Thereafter, we repeat this step for all children of ψ_b and create the next level of the tree. The procedure continues until we reach ψ_p, which is the packed platform, and since the budget donation operation can not be performed on the packed platform the algorithm stops branching. The pseudocode of this procedure is presented in Algorithm 3. `isNew` is a function that looks for its input platform (ψ') in its input set (Ψ^{Γ}) and returns true if it fails to find the platform. The budget donation operation takes place in line 7 and 8 of the algorithm, and in line 11 (when we find a new platform) we do a recursive call passing the recently found platform. Since the budget donation operation transfers only one unit of the budget at each step, and we run this operation on all combinations of donators and their corresponding receivers, we are guaranteed to i) reach ψ_p which is the termination condition of our recursive algorithm ii) find all possible platforms between ψ_b and ψ_p.

Algorithm 3 Deriving possible platforms of a flexible interface

```
 1: function platforms($\psi$, $\psi_p$)
 2: $D^{\psi}$ = donators($\psi$);
 3: for all $d \in D^{\psi}$ do
 4:     $R_d^{\psi}$ = receivers($\psi$, $d$);
 5:     for all $r \in R_d^{\psi}$ do
 6:         $\psi' = \psi$;
 7:         $\psi'.q_d = q_d - 1$;
 8:         $\psi'.q_r = q_r + 1$;
 9:         if $\psi' \neq \psi_p$ & isNew($\psi', \Psi^{\Gamma}$) then
10:             insert($\psi'$, $\Psi^{\Gamma}$);
11:             platforms($\psi'$, $\psi_p$);
12:             return $\psi'$;
13:         end if
14:     end for
15: end for
16: end function
```

Since we start from ψ_b where

$$\overbrace{q_1, \ldots, q_k}^{=\lfloor \frac{Q}{m} \rfloor + 1}, \overbrace{q_{k+1}, \ldots, q_m}^{=\lfloor \frac{Q}{m} \rfloor}$$

and we want to reach ψ_p where

$$\overbrace{q_1, \ldots, q_h}^{=P}, \quad \overset{=}{} \quad \overbrace{q_{h+1}}^{\bmod (Q,P)} \quad , \overbrace{q_{h+2}, \ldots, q_m}^{=0}$$

therefore q_{h+1} to q_m in ψ_b should be moved to a place between q_1 and q_h. In this process each q_i can at most move $i-1$ steps to the left, and the longest depth happens when the donator processor and the receiver processors at all steps are neighbors ($d = r + 1$). Therefore, the longest depth (κ) is calculated by the following equation

$$\kappa = \sum_{i=h+1}^{m} q_i \times (i-1), \qquad (1)$$

where m is the number of processors, $h = \lfloor \frac{Q}{P} \rfloor$, and q_i are the budgets of ψ_b. The total number of possible platforms is exponential in κ. Note that κ is derived without considering that i) all q_i are sorted non-increasingly ii) redundant nodes are not allowed to branch. Therefore, in practice the longest depth is less than or equal to κ. However, since the growth rate of the algorithm is exponential it is considered as a high complexity problem which might be intractable for some configurations of m, P and Q. A sample trace of the algorithm for the flexible interface $\Gamma = \langle 3, 8, 6 \rangle$ is presented in Figure 2. The redundant nodes are presented as gray nodes which are eliminated using the `isNew` function. The figure illustrates the necessity of eliminating redundant nodes since they are a considerable number of the total nodes.

4. SUPPLY BOUND FUNCTION

The Supply Bound Function ($\texttt{sbf}(t)$) represents the minimum amount of the resources that servers provide to their task set in a given time interval t. In this section we derive the supply bound function of both the flexible and the rigid interface models. For simplifying the presentation we drop

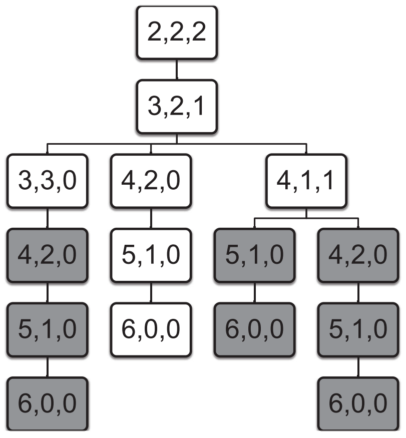

Figure 2: possible rigid platforms of $\Gamma = \langle 3, 8, 6\rangle$

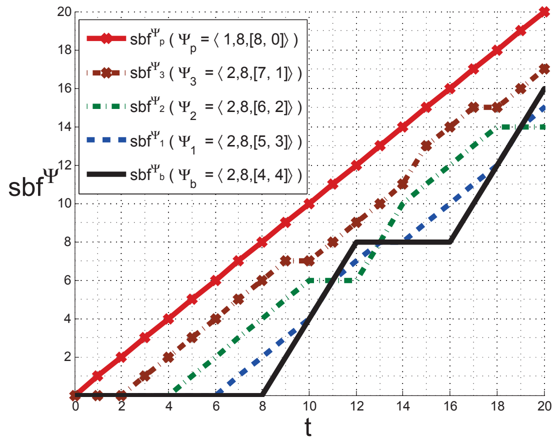

Figure 3: The sbf of all possible platforms of $\Gamma_1 = \langle 2, 8, 8\rangle$

t when referring to the supply bound function in the rest of the text.

4.1 The sbf of rigid interfaces

The sbf of a rigid interface $\texttt{sbf}^{\psi}$ can be seen as sum of m servers' sbfs with the period equal to P and given budgets q_i. Therefore, using the same equation that is presented in [11] for calculating the sbf we have:

$$\texttt{sbf}^{\psi}(t) = \sum_{i=1}^{m}\Big(\Big\lfloor\frac{t-(P-q_i^{\psi})}{P}\Big\rfloor \times q_i^{\psi} + \epsilon_i(t)\Big), \tag{2}$$

where

$$\epsilon_i(t) = \max\Big(t - 2(P-q_i^{\psi}) - p \times \Big\lfloor\frac{t-(P-q_i^{\psi})}{P}\Big\rfloor, 0\Big). \tag{3}$$

4.2 The sbf of flexible interfaces

In order to calculate the minimum supply provision of a flexible interface ($\texttt{sbf}^{\Gamma}$) we need to derive the worst case platform which provides the least amount of resources among all possible platforms. However, as Lipari and Bini state in [9], the worst case platform does not exist for the flexible interfaces. Although the balanced rigid platform is the worst case platform when we assume that the virtual processors are synchronized [6], when the assumption is relaxed it is not the worst case platform anymore. Therefore, a potential solution for calculating the sbf of flexible interfaces is to take the following steps:

1. Derive Ψ^{Γ} using Algorithm 3.
2. From the definition of the supply bound function, $\texttt{sbf}^{\Gamma}$ at each time point is the minimum of all $\texttt{sbf}^{\psi}$ at that time:
$$\texttt{sbf}^{\Gamma}(t) = \min\Big\{\texttt{sbf}^{\psi}(t)\Big\},\ \forall\psi \in \Psi^{\Gamma}. \tag{4}$$

As we discussed in the previous section, the complexity of step one is exponential, hence this solution might be intractable for some flexible interfaces. Therefore, in the rest of this section we take some actions in reducing the complexity of step one by removing the platforms that are not contributing in calculating the $\texttt{sbf}^{\Gamma}$. Indeed, we are looking for the platforms where $\texttt{sbf}^{\psi}$ crosses $\texttt{sbf}^{\psi_b}$ at least at one time point, or mathematically:

$$\forall\psi\,\exists t : \texttt{sbf}^{\psi}(t) < \texttt{sbf}^{\psi_b}(t).$$

Example 1. *Consider the following flexible interface $\Gamma_1 = \langle 2, 8, 8\rangle$, the* sbf *of all possible platforms is shown in Figure 3 (redrawn from [9]). Among all possible platforms of Γ_1, only $\texttt{sbf}^{\psi_1}$ and $\texttt{sbf}^{\psi_2}$ are crossing $\texttt{sbf}^{\psi_b}$. Therefore, in the proposed approach for calculating $\texttt{sbf}^{\Gamma}$, it is sufficient to only derive ψ_1 and ψ_2 in step 1, and proceed with the second step. Roughly speaking, we present an approach to exclude the platforms where the lower bound of their* sbf *is higher than the upper bound of $\texttt{sbf}^{\psi_b}$ (ψ_3 and ψ_p in this example).*

To this end, we first show how to calculate the lower bound (lsbf) and upper bound (usbf) of the supply bound function, afterwards we derive a subset of Ψ^{Γ} which is sufficient for calculating the $\texttt{sbf}^{\Gamma}$.

4.3 The lsbf of rigid interfaces

Analogous to the presented approach for calculating the $\texttt{sbf}^{\psi}$, linear lower bound for a rigid interface $\texttt{lsbf}^{\psi}$ can be calculated by accumulating the linear lower bound of m independent servers' lsbf with a given period (P) and budget (q_i). According to [11], the lsbf is calculated as follows

$$\texttt{lsbf}(t) = \frac{q_i}{P}\Big(t - 2(p-q_i)\Big), \tag{5}$$

therefore the lsbf of ψ is

$$\texttt{lsbf}^{\psi}(t) = \sum_{i=1}^{m}\frac{q_i^{\psi}}{P}\Big(t-2(p-q_i^{\psi})\Big) = \alpha(t-\Delta^{\psi}), \tag{6}$$

where

$$\alpha = \frac{Q}{P}, \tag{7}$$

and

$$\Delta^{\psi} = 2\Big(P - \frac{\sum_{i=1}^{m}(q_i^{\psi})^2}{Q}\Big). \tag{8}$$

Note that we overload Δ in the rest of the paper and it does not refer to the delay in the bounded delay model anymore. α of all platforms in Ψ^{Γ} are equal, however, their Δ can differ.

LEMMA 1. *The budget donation operation always outputs a platform in which its Δ is less than the Δ of its input platform ($\Delta^{\psi^{child}} < \Delta^{\psi^{parent}}$).*

Proof Assuming that $q_i^{\psi^{parent}}$ and $q_i^{\psi^{child}}$ represent the budget distributions of the input and output platform of the budget donation operation respectively, based on Equation 8 we should show:

$$\sum_{i=1}^{m}(q_i^{\psi^{child}})^2 - \sum_{i=1}^{m}(q_i^{\psi^{parent}})^2 > 0. \tag{9}$$

Since all budgets except q_d (q of the budget donator) and q_r (q of the budget receiver) are equal we can write:

$$(q_d - 1)^2 + (q_r + 1)^2 - q_d^2 - q_r^2 > 0, \tag{10}$$

$$q_r > q_d - 1, \tag{11}$$

which is true since $r < d$ and the budgets are sorted non-increasingly.

LEMMA 2. *The* `lsbf` *of the balanced platform (*$\texttt{lsbf}^{\psi_b}$*) is lower than any other possible platforms'* `lsbf`*.*

Proof According to Lemma 1, and given that ψ_b is the root of Algorithm 3

$$\Delta^{\psi_b} > \Delta^{\psi_b{}'} \tag{12}$$

where $\psi_b{}'$ represent any non-balanced platform derived from Algorithm 3, which according to 6 yields to:

$$\texttt{lsbf}^{\psi_b}(t) < \texttt{lsbf}^{\psi_b{}'}(t). \tag{13}$$

4.4 The `lsbf` of flexible interfaces

According to Lemma 2, the `lsbf` of a flexible interface ($\texttt{lsbf}^{\Gamma}$) is calculated by deriving the corresponding balanced platform and calculating $\texttt{lsbf}^{\psi_b}$,

$$\texttt{lsbf}^{\Gamma}(t) = \texttt{lsbf}^{\psi_b}(t) = \alpha(t - \Delta^{\psi_b}). \tag{14}$$

4.5 Upper bound of the `sbf`

In this section we present an upper bound for the `sbf` which is used for excluding the irrelevant platforms in calculating the $\texttt{sbf}^{\Gamma}$. The upper bound of the supply bound function (`usbf`) for independent servers on single processors with a common period P and given budget q_i according to [5] is as follows:

$$\texttt{usbf}(t) = \frac{q_i}{P}\Big(t - (P - q_i)\Big). \tag{15}$$

Therefore, $\texttt{usbf}^{\psi}(t)$ is:

$$\texttt{usbf}^{\psi}(t) = \sum_{i=1}^{m}\frac{q_i^{\psi}}{P}\Big(t - (P - q_i^{\psi})\Big) = \alpha(t - \theta^{\psi}), \tag{16}$$

where

$$\theta^{\psi} = P - \frac{\sum_{i=1}^{m}(q_i^{\psi})^2}{Q}. \tag{17}$$

LEMMA 3. *In calculating the* $\texttt{sbf}^{\Gamma}$ *for flexible interfaces it is sufficient to consider the following subset of* Ψ^{Γ}*:*

$$\Psi^{\theta} = \Big\{\forall\psi \in \Psi^{\Gamma}: \quad \Delta^{\psi} \geq \theta^{\psi_b}\Big\}.$$

Proof Recall step two in calculating $\texttt{sbf}^{\Gamma}$, since we are using the min function to calculate $\texttt{sbf}^{\Gamma}$ at each time point, the platforms where their `sbf` are absolutely more than $\texttt{sbf}^{\psi_b}$ do not affect the min function. Therefore, we want to exclude the platforms that fulfill the following condition:

$$\forall t: \texttt{sbf}^{\psi}(t) > \texttt{sbf}^{\psi_b}(t), \tag{18}$$

or

$$\forall t: \texttt{sbf}^{\psi}(t) \geq \texttt{usbf}^{\psi_b}(t), \tag{19}$$

which yields to excluding the following set:

$$\forall\psi: \forall t: \texttt{lsbf}^{\psi}(t) \geq \texttt{usbf}^{\psi_b}(t), \tag{20}$$

therefore Ψ^{θ} is of our interest in calculating $\texttt{sbf}^{\Gamma}$.

Based on Lemma 3, Algorithm 3 can be altered such that the stop condition ($\psi' = \psi_p$) is replaced by the following condition:

$$\Delta^{\psi} < \theta^{\psi_b}, \tag{21}$$

and using Equation 8 we have:

$$\sum_{i=1}^{m}(q_i^{\psi})^2 \geq Q(P - \frac{\theta^{\psi_b}}{2}), \tag{22}$$

therefore, the condition at line 9 in Algorithm 3 ($\psi' = \psi_p$) should be replaced by Inequality 22. Thereafter, we need to consider all output platforms of the altered algorithm for calculating the $\texttt{sbf}^{\Gamma}$:

$$\texttt{sbf}^{\Gamma}(t) = \min\Big\{\texttt{sbf}^{\psi}\Big\}\ \forall\psi \in \Psi^{\theta}. \tag{23}$$

For instance lets take the flexible interface $\Gamma_1 = \langle 2, 8, 8\rangle$ presented in Example 1. For this example we have: $\theta^{\psi_b} = 4$, hence the stop condition is: $\sum_{i=1}^{m}(q_i^{\psi})^2 \geq 48$. Therefore, ψ_3 $\Big(\sum_{i=1}^{m}(q_i^{\psi_3})^2 = 50\Big)$ and ψ_p $\Big(\sum_{i=1}^{m}(q_i^{\psi_p})^2 = 64\Big)$ do not need to be considered for calculating the $\texttt{sbf}^{\Gamma}$:

$$\texttt{sbf}^{\Gamma_1}(t) = \min\Big\{\texttt{sbf}^{\psi_b}(t), \texttt{sbf}^{\psi_1}(t), \texttt{sbf}^{\psi_2}(t)\Big\}.$$

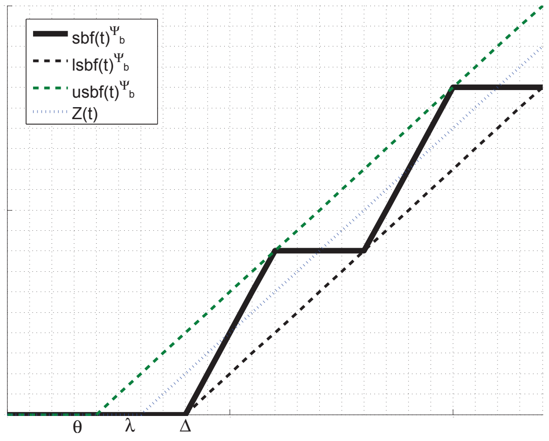

Figure 4: The sbf, the lsbf and the usbf of a balanced platform

5. APPROXIMATE sbf OF THE FLEXIBLE INTERFACES

According to 23, we can reduce the number of platforms that have to be investigated when calculating the $\texttt{sbf}^{\Gamma}$, however, this subset (Ψ^{θ}) may still include too many platforms that makes the computations intractable. In this section we propose an approach to derive an approximate supply bound function for the flexible interfaces (asbf). In this approach we consider λ such that $\theta^{\psi_b} \leq \lambda \leq \Delta^{\psi_b}$ and we replace θ^{ψ_b} with λ in Equation 22 to get a new termination condition for branching ($\psi' = \psi_p$) in Algorithm 3:

$$\sum_{i=1}^{m} (q_i^{\psi})^2 \geq Q(P - \frac{\lambda}{2}). \qquad (24)$$

Therefore in this approach we consider the following subset of Ψ^{Γ}:

$$\Psi^{\lambda} = \Big\{ \forall \psi \in \Psi^{\Gamma} : \quad \Delta^{\psi} \geq \lambda \Big\}.$$

This new condition confines the search space, therefore we can get an approximate sbf investigating a lower number of platforms. Using λ we cut the tree in earlier branches than the original algorithm, therefore, the time complexity of the algorithm is reduced. Figure 4 illustrates the relation between the actual upper bound and the approximate upper bound ($Z(t)$). For calculating the $\texttt{sbf}^{\Gamma}$ we **exclude** the platforms where the sbf is located at the left hand side of $\texttt{usbf}^{\psi_b}$, however, for calculating the $\texttt{asbf}^{\Gamma}$ we **exclude** all the platforms where the sbf is located at the left hand side of $Z(t)$.

The approximate sbf is:

$$\texttt{asbf}^{\Gamma}(t) = \min \Big\{ \texttt{sbf}^{\psi}(t), Z(t) \Big\} \quad \forall \psi \in \Psi^{\lambda}, \qquad (25)$$

where

$$Z(t) = \alpha(t - \lambda), \qquad (26)$$

Γ	$\langle 8,16,40 \rangle$	time (sec)	$\langle 4,64,80 \rangle$	time (sec)
Ψ^{Γ}	6360	192.64	4089	37.06
Ψ^{θ}	5650($\simeq$ 88%Ψ^{Γ})	159.22	3652($\simeq$ 89%Ψ^{Γ})	30.43
ψ^{λ_1}	2259($\simeq$ 35%Ψ^{Γ})	23.12	2245($\simeq$ 54%Ψ^{Γ})	11.30
ψ^{λ_2}	507($\simeq$ 7%Ψ^{Γ})	0.99	938($\simeq$ 22%Ψ^{Γ})	1.98

Table 1: Ψ^{Γ}, Ψ^{θ} and Ψ^{λ} for sample flexible interfaces ($\lambda_1 = 0.5(\theta + \Delta)$ and $\lambda_2 = 0.75(\theta + \Delta)$)

because according to Lemma 1, all other platforms that we are not considering in the min function have smaller Δ which means their lsbf (and consequently their sbf) is more than $Z(t)$ at all time points. Note that the min operation in Equitation 25 is critical in ensuring that the approximation is safe.

Recall Example 1, if we assign $\lambda = 6$, the termination condition is $\sum_{i=1}^{m} (q_i^{\psi})^2 \geq 40$, therefore, for calculating $\texttt{asbf}^{\Gamma_1}$ we only need to consider ψ_b and ψ_1 together with the following line $Z_1(t) = (t - 6)$. Hence, we have:

$$\texttt{asbf}^{\Gamma_1}(t) = \min \Big\{ \texttt{sbf}(t)^{\psi_b}, \texttt{sbf}(t)^{\psi_1}, Z_1(t) \Big\}.$$

Table 1 shows two flexible interfaces and corresponding number of rigid interfaces that should be investigated in order to calculate the $\texttt{sbf}^{\Gamma}$ and the $\texttt{asbf}^{\Gamma}$ in addition to the analysis time that it took for our MATLAB code to derive them. In these two examples, more than 10% of the possible platforms are irrelevant in calculating $\texttt{sbf}^{\Gamma}$, and when calculating $\texttt{asbf}^{\Gamma}$ the higher the number of the platforms included in the min function, the higher the accuracy of the approximation.

The number of possible platforms of a flexible interface is positively correlated with P and m because when increasing them, there are more possibilities for the total budget to be distributed on different processors. However, increasing Q does not necessarily increase the number of possible platforms. For instance when $Q = m \times P$, we have $\psi_b = \psi_p$ and the number of possible platforms is one. Figure 5 shows the relation between Q and the number of possible platforms for the flexible interface $\Gamma = \langle 5, 16, Q \rangle$ ($Q \in [1, m \times P]$). The figure indicates that the number of possible platforms increases until $Q = \frac{m \times P}{2}$, and decreases afterwards. The trend is analogous for all flexible interfaces, which can be explained by the longest depth (κ) presented in Equation 1, where increasing Q has two consequences: i) increases q_i in ψ_b which increases κ ii) increases h ($h = \lfloor \frac{Q}{P} \rfloor$) and therefore decreases κ. Hence, depending on the dominating factor, the number of possible platforms may either be positively or negatively correlated with Q. From the figure we observe that the dominant factor is (i) until $Q = \frac{m \times P}{2}$ and thereafter it is (ii), therefore the difference between the number of platforms in calculating asbf and sbf is more significant in platforms where Q is around $\frac{m \times P}{2}$.

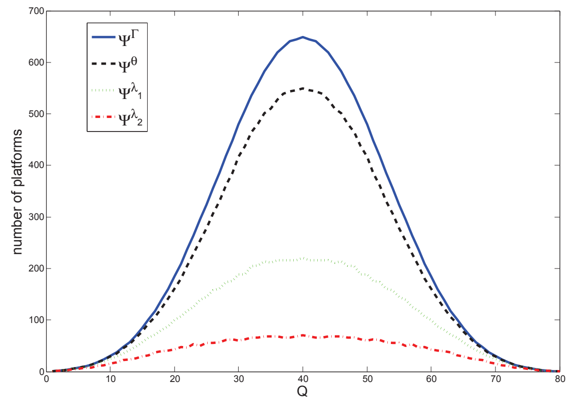

Figure 5: Number of platforms in Ψ^{Γ}, Ψ^{θ} and Ψ^{λ} for $\Gamma = \langle 5, 16, Q\rangle$ ($\lambda_1 = 0.5(\theta + \Delta)$ and $\lambda_2 = 0.75(\theta + \Delta)$)

6. CONCLUSION

In this paper we presented an approach for calculating the supply bound function of multiprocessor periodic resource interfaces when the servers are not synchronized. Being independent from server synchronization makes the model compatible with all types of hardware platforms (even with the ones where synchronization is expensive). Furthermore, due to the exponential complexity of calculating the actual supply bound function, we proposed an approach for calculating an approximate supply bound function with lower computational complexity.

The next step in our work is to evaluate the difference between using actual and approximate supply bound functions using an extensive number of randomly generated systems. We also intend to compare the periodic resource interface with the bounded delay interface. Finally we will look into the presented algorithm for mapping the flexible interface to the rigid interface(s) and try to further reduce its complexity using some heuristics.

7. REFERENCES

[1] S. Baruah, A. Mok, and L. Rosier. Preemptively scheduling hard-real-time sporadic tasks on one processor. In *Proceedings of the 11th Real-Time Systems Symposium (RTSS '90)*, pages 182 – 190, December 1990.

[2] E. Bini, M. Bertogna, and S. Baruah. Virtual multiprocessor platforms: Specification and use. In *Proceedings of the 30th IEEE Real-Time Systems Symposium, (RTSS '09)*, pages 437 –446, December 2009.

[3] E. Bini, G. Buttazzo, and M. Bertogna. The multi supply function abstraction for multiprocessors. In *Proceedings of the 15th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, (RTCSA '09)*, pages 294 –302, August 2009.

[4] Z. Deng and J. W.-S. Liu. Scheduling real-time applications in an open environment. In *Proceedings of the 18th IEEE Real-Time Systems Symposium (RTSS '97)*, pages 308 – 319, December 1997.

[5] A. Easwaran, M. Anand, and I. Lee. Compositional analysis framework using EDP resource models. In *Proceedings of the 28th IEEE Real-Time Systems Symposium, (RTSS '07)*, pages 129 – 138, December 2007.

[6] A. Easwaran, I. Shin, and I. Lee. Optimal virtual cluster-based multiprocessor scheduling. *Real-Time Systems*, 43(1):25 – 59, September 2009.

[7] T.-W. Kuo and C.-H. Li. A fixed-priority-driven open environment for real-time applications. In *Proceedings of the 20th IEEE Real-Time Systems Symposium (RTSS '99)*, pages 256 – 267, December 1999.

[8] H. Leontyev and J. Anderson. A hierarchical multiprocessor bandwidth reservation scheme with timing guarantees. In *Proceedings of the 20th Euromicro Conference on Real-Time Systems (ECRTS '08)*, pages 191 – 200, July 2008.

[9] G. Lipari and E. Bini. A framework for hierarchical scheduling on multiprocessors: From application requirements to run-time allocation. In *Proceedings of the 31st IEEE Real-Time Systems Symposium (RTSS '10)*, pages 249 –258, December 2010.

[10] A. Mok, X. Feng, and D. Chen. Resource partition for real-time systems. In *7th Real-Time Technology and Applications Symposium (RTAS '01)*, pages 75 – 84, May 2001.

[11] I. Shin and I. Lee. Periodic resource model for compositional real-time guarantees. In *Proceedings of the 24th IEEE Real-Time Systems Symposium, (RTSS '03)*, December 2003.

[12] H. S. Wilf. Lectures on integer partitions, `http://www.math.upenn.edu/~wilf/PIMS/PIMSLectures.pdf`.

[13] H. Zhu, S. Goddard, and M. Dwyer. Response time analysis of hierarchical scheduling: The synchronized deferrable servers approach. In *32nd IEEE Real-Time Systems Symposium (RTSS '11)*, pages 239 – 248, December 2011.