

# Resource Partitioning among Real-Time Applications

Giuseppe Lipari, Enrico Bini

Scuola Superiore S. Anna

Pisa, Italy

E-mail: lipari@sssup.it, e.bini@sssup.it

## Abstract

When executing different real-time applications on a single processor system, one problem is how to compose these applications and guarantee at the same time that their timing requirements are not violated. A possible way of composing applications is through the resource reservation approach. Each application is handled by a dedicated server that is assigned a fraction of the processor. Using this approach, the system can be seen as a two-level hierarchical scheduler. A considerable amount of work has been recently addressed to the analysis of this kind of hierarchical systems. However, a question is still unanswered: given a set of real-time tasks to be handled by a server, how to assign the server parameters so that the task set is feasible? In this paper, we answer to the previous question for the case of fixed priority local scheduler by presenting a methodology for computing the class of server parameters that make the task set feasible.

## 1. Introduction

Thanks to the advances in the field of computer architectures, computers are getting faster and faster and it is then desirable to concurrently execute different real-time applications in the same system. The motivation is in cost reduction and in the re-use of legacy applications. This trend can be seen both in the general purpose computer area and in the embedded system area.

When executing different real-time applications in the same system, the question is how to schedule these applications and guarantee at the same time that their timing requirements are not violated. One simple way to do composition is to use a unique scheduling paradigm for the whole system and design all applications according to the chosen paradigm: then, it is possible to check the schedulability of whole the system by using already existing schedulability analysis tools.

However, sometime it is necessary to use an already implemented application “as it is”, without going back to the design phase. If an application is already working well on the old slower processor, it is less expensive to move it on the new faster processor without changing the code. However, we must guarantee that the old application will still meet its timing requirements even when other applications are scheduled in the same system.

An interesting problem is how to compose applications that come with their own scheduling strategy. In reality, different schedulers may be used in different context, and there is not a “catch-all” scheduler that is best for all kind of application domains. For example, applications that are event-triggered are best served by on-line scheduling algorithms like fixed priority or earliest deadline first; time triggered applications are best handled by off-line schedulers like the TTA [10].

One way of composing existing applications with different timing characteristics is to use a two-level scheduling paradigm (see Figure 1): at the *global level*, a scheduler selects which application will be executed next and for how long. Each application then possesses a *local scheduler* that selects which task will be scheduled next.

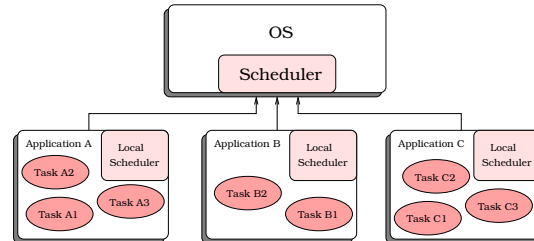


Figure 1. Hierarchical scheduler structure.

The global scheduler allocates the processor bandwidth and each application is served with a fraction of the total processor time distributed over the time line according to a certain law. Moreover, the global scheduler must “protect” one application from all others, by ensuring that if an application is requiring more than expected, it does not compromise the others.

If we could provide a “fluid allocation”, for example by using the Generalized Processor Sharing (GPS) [18, 19], then the bandwidth allocation would be easy. Unfortunately, the GPS algorithm is a theoretical abstraction that cannot be implemented in practice, but only approximated [9, 2, 1].

In this paper, we consider the class of algorithms that can be described by the *periodic server abstraction*. Each application is assigned a server that is characterized by the pair  $(Q, P)$ , with the meaning that the server gets  $Q$  units of execution every  $P$  units of time. The global schedul-

ing mechanism decides when to schedule the servers; the selected server, by using the local scheduling mechanism, decides which application task will be executed next.

Some research has already been done on hierarchical composition of periodic servers. In particular, it is possible to perform a schedulability analysis of a group of tasks on a server, given the pair  $(Q, P)$  assigned to the server, the worst execution times, periods and deadlines of the tasks, and the internal scheduling algorithm for the task group. For example, Saewong et al. [22] presented a response time analysis for the schedulability of such a hierarchical system.

However, an open issue remains to be answered: *given a group of tasks, what is the “best” pair  $(Q, P)$  that can be assigned to the server so that the task group is schedulable?*

In this paper we present a technique that answers the previous question for applications composed by periodic (or sporadic) tasks, scheduled by a fixed priority local scheduler. Given an application (i.e. a set of tasks), we are able to find a class of possible parameters  $(Q, P)$  that make the application schedulable. The designer can then choose the best pair that meets his needs.

The paper is organized as follows: after an overview of the research in the field of hierarchical schedulers (Section 1.1), the system model and most of the concept and definition are presented in Section 2. In Section 3, we present our analysis and propose our method; finally in Section 4, we draw our conclusion and propose some future development.

### 1.1. Related work

The research on two-level scheduling algorithms can be considered a hot topic in the real-time system research. A general methodology for temporal protection in real-time system is the resource reservation framework [15, 21]. The basic idea, formalized by Rajkumar [20], is that each task is assigned a *server* that is reserved a fraction of the processor available bandwidth. If the task tries to use more than it has been assigned, it is *slowed down*. This framework allows a task to execute in a system as it were executing on a dedicated virtual processor, whose speed is a fraction of the speed of the processor. Thus, by using a resource reservation mechanism, the problem of schedulability analysis is reduced to the problem of estimating the computation time of the task without considering the rest of the system. Many server algorithm have been presented in the literature, both for fixed priority and dynamic priority schedulers [12, 23, 3, 24, 1]. In these models, the server executes together with hard real-time periodic tasks, and it is mainly used to handle aperiodic tasks.

Recently, many techniques have been proposed for extending the resource reservation framework to hierarchical scheduling.

Saewong et al. [22] proposed to use the Deferrable Server in a hierarchical way. They present a schedulability analysis that is based on the worst-case response time for a local fixed priority scheduler. Liu and Deng in [6, 5]

proposed a two-level hierarchical architecture, which uses the EDF as global scheduler and uses a dedicated TBS for each application. It is then possible to select the most appropriate scheduling algorithm for each application. The paper presents also a sufficient condition for schedulability. This work has been later extended by Kuo et al. [11] for using RM as global scheduling algorithm, but the authors assume that all tasks are periodic with harmonic periods.

Lipari and Baruah in [14, 13] presented the BSS scheduling algorithm that uses EDF as global scheduling algorithm, and permits to select any scheduling algorithm as application level scheduler. The paper presents schedulability conditions for applications that use EDF and RM as second level schedulers. However, the algorithm is complex to implement and assumes the knowledge of all task deadlines; in fixed priority scheduling, the absolute deadline may not be specified in the implementation of the task. Therefore, it is not possible to schedule legacy applications.

Feng and Mok [16] presented a general methodology for hierarchical partitioning of a computational resource. It is possible compose schedulers at arbitrary levels of the hierarchy. They also propose simple schedulability test for any scheduler at any level of the hierarchy, but these tests are only sufficient. In this paper, we will follow their same initial approach. However, while Feng and Mok concentrate their research on how to analyse and guarantee the schedulability of an application on a partition, we will address the reverse problem: given an application, scheduled by a local fixed priority scheduler, how to select the “best” server.

## 2. System model and definitions

An *application* is a set  $\Gamma_n$  of  $n$  periodic or sporadic tasks, ordered by decreasing priority. Every task  $\tau_i$  is characterized by a period  $T_i$  (or minimum interarrival time), a worst-case execution time  $C_i$  and a relative deadline  $D_i$  smaller than the period. In this paper we will consider independent tasks. The extension for interacting tasks is currently under research.

An application is further characterized by a *local scheduler*  $\sigma(\Gamma_n)$ . When the application is selected to execute by the global scheduler (see Figure 1), the local scheduler selects which application task will execute. In this paper we focus our attention on fixed priority schedulers.

The system consists of a set of applications, each one with a (possibly different) local scheduler. The aim of the global scheduler is to assign execution time to the applications according to a given rule. This rule can be static (i.e. the allocation is pre-computed off-line), or dynamic, according to some on-line algorithm.

In this paper we will concentrate our efforts in analysing the behavior of those on-line algorithms referred as *periodic servers*. Examples of these algorithms are the Polling Server, the Deferrable server, the Sporadic Server [23, 8, 24], the Constant Bandwidth Server [1], etc. All these al-

gorithms have different peculiarities. Since we do not want to concentrate on one particular mechanism, in Section 2.2 we will present an abstract model of a server and how this model is related to some existing mechanism.

However, it is important to point out that our methodology is very general and can be applied, with some simple customization, to many other partitioning mechanism like, for example, p-fair scheduling, static allocation (TTA), etc. In the following subsection, we give an overview of the different mechanisms for providing partitions.

### 2.1. Partitions

A partition is a function  $\Pi(t)$  that has values in  $\{0, 1\}$ . If  $\Pi(t) = 1$ , then the resource is allocated to the application at time  $t$ . A partition is periodic if it exists  $P > 0$  such that  $\Pi(t) \equiv \Pi(t + P)$  (see [7, 17]).

The global scheduler provides partitions among applications. A static algorithm pre-computes the partitions off-line, and at run-time a dispatch mechanism will make use of a simple table to allocate the resource. Examples are cyclic executives, the TTA [10], etc. Conversely, an on-line algorithm uses some rule for dynamically allocating the resource. Therefore, an on-line algorithm may produce different partitions every time it is executed, depending on the arrival times and execution times of the application tasks. Moreover, these partitions are not necessarily periodic. Examples are the Deferrable server, the Constant Bandwidth server, etc.

For a given partition, we define the minimum amount of time that is available to the application in every interval of length  $t$ .

**Definition 1** Given a partition  $\Pi(t)$ , we define the characteristic function  $Z_{\Pi}(t)$  as the minimum amount of time provided by the partition in every time interval  $t \geq 0$  units of time long:

$$Z_{\Pi}(t) = \min_{t_0 \geq 0} \int_{t_0}^{t_0+t} \Pi(x) dx$$

As an example, consider an off-line algorithm that produces a periodic partition  $\Pi(t)$  with period 8, which allocates slots 2,3,4 and slot 7. The corresponding characteristic function  $Z_{\Pi}(t)$  is plotted in Figure 2. Note that the worst-case interval starts at time 4.

### 2.2. The server abstraction

The concept of server has been originally devised for minimizing the response time of aperiodic tasks when scheduled together with hard real-time tasks [23, 8]. Recently, some server has been used for providing resource reservations mechanisms [20, 1]. Many server algorithms have been proposed in the literature, both in fixed priority and in dynamic priority systems. Since many of these mechanisms provide similar guarantees, in this paper we analyse a general abstraction of a server that subsumes (with some

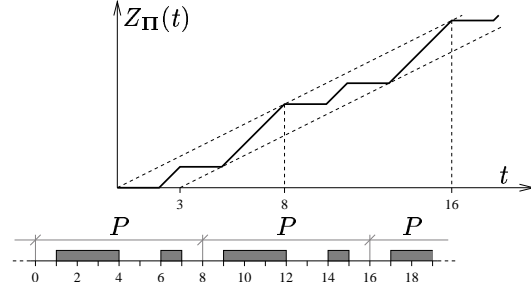


Figure 2. An example of  $Z_{\Pi}(t)$ .

important differences) all the algorithms cited so far.

A server is an abstraction that provides execution time to one or more tasks, according to a certain local scheduling algorithm. A server is characterized by two parameters  $(Q, P)$ , where  $Q$  is the *maximum budget*, and  $P$  is the *server period*. The system consists of a set of servers scheduled by a global scheduling algorithm. The server maintains two internal variables  $q$  and  $d$  that are updated according to the following rules.

#### Server Rules.

1. Initially,  $q = 0$ ,  $d = 0$  and the server is *inactive*.
2. When a task is activated at time  $t$ , if the server is *inactive*, then  $q = Q$  and  $d = t + P$ , and the server becomes *active*. If the server is already active, then  $q$  and  $d$  remain unchanged.
3. At any time  $t$ , the global scheduling algorithm selects one active server. When the server is selected, it executes the first task in its ready queue (which is ordered by the local scheduling policy).
4. While some application task is executing, the current budget  $q$  is decremented accordingly.
5. The global scheduler can *preempt* the server for executing another server: in this case, the current budget  $q$  is no longer decremented.
6. If  $q = 0$  and some task has not yet finished, then the server is *suspended* until time  $d$ ; at time  $d$ ,  $q$  is recharged to  $Q$ ,  $d$  is set to  $d + P$  and the server can execute again.
7. When, at time  $t$ , the last task has finished executing and there is no other pending task in the server, the server yields to another server. Moreover, if  $t \geq d - q \frac{P}{Q}$ , the server becomes *inactive*; otherwise it remains *active*, and it will become *inactive* at time  $d - q \frac{P}{Q}$ , unless another task is activated before.

The previously cited server algorithms differ from one another in the underlying global scheduling policy and in rules 6 and 7. For example, the Deferrable Server and the

Sporadic Server have different rules for recharging the budget; the Constant Bandwidth Server does not suspend the server when the budget is 0, but simply decreases its priority by postponing its absolute deadline.

We do not assume any particular global scheduling policy for the servers. It is possible to show that the algorithm described by the previous rules is similar to the Constant Bandwidth Server [1]. Therefore, it is possible to use EDF as global scheduler. However, if the server periods are harmonic, it is also possible to use the same algorithm with the Rate Monotonic scheduler.

### 2.3. Characteristic function of a server

Given a task set that has to be scheduled by a local fixed priority scheduler, our goal is to find the class of server parameters  $(Q, P)$  that make the set feasible. In this way, the designer can choose the best tradeoff between a large  $P$  and a small  $\alpha = \frac{Q}{P}$ . In fact, as we will see more in detail in Section 3.2, a small  $P$  may cause a high number of context switches between servers, but a large  $P$  leads to a high utilisation  $\frac{Q}{P}$  and then to a waste of computational resources.

In order to find all the possible feasible pairs  $(Q, P)$ , we first need to characterize the temporal behaviour of a server. In particular, we need to know the minimum amount of execution time that a server can provide in every interval of time  $t$  to its application.

**Definition 2** Given a server  $S$ , we define  $\text{legal}(S)$  as the set of partitions  $\Pi$  that can be generated by the server algorithm.

**Definition 3** Given a server  $S$ , we define  $Z_S(t)$  the minimum amount of time provided by the server  $S$  in every time interval  $t \geq 0$  units of time long.

$$Z_S(t) = \min_{\Pi \in \text{legal}(S)} Z_{\Pi}(t)$$

To see the importance of the function  $Z_S(t)$ , consider the schedulability problem of a single task  $\tau_i$  on the server  $S$ . If  $Z_S(D_i)$ , which is the minimum amount of time provided in every time interval  $D_i$ -long, is greater than or equal to the maximum possible time requested by the task  $\tau_i$  and all its higher priority tasks in the same interval, then task  $\tau_i$  is feasible on the server  $S$ .

In Figure 3, we plot the characteristic function  $Z_S(t)$  of a server with parameters  $Q = 5$  and  $P = 8$ . Since we do not assume any particular global scheduling algorithm, and we do not know the global system load, we consider the worst-case situation, when the application tasks are activated just after the budget is exhausted, and the first instant of time at which they will receive execution is after  $2(P - Q)$ .

**Theorem 1** Given a server algorithm defined by the rules of Section 2.2, and with parameters  $(Q, P)$ , and defined

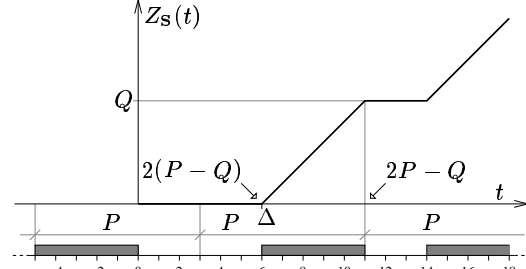


Figure 3. General case of periodic server.

$k = \left\lceil \frac{t-(P-Q)}{P} \right\rceil$ , its characteristic function  $Z_S(t)$  is:

$$Z_S(t) = \begin{cases} 0 & \text{if } t \in [0, P - Q] \\ (k-1)Q & \text{if } t \in (kP - Q, (k+1)P - 2Q] \\ t - (k+1)(P - Q) & \text{otherwise} \end{cases}$$

**Proof.** We have to compute the worst-case allocation to the server for every interval of time. Consider an interval starting at time  $t$ . There are 2 possibilities:

**case a :** The server is *inactive* at time  $t$ . In this case, according to rule 2, a new budget  $q = Q$  and a new deadline  $d = t + P$  are computed. Therefore, the worst-case allocation is depicted in Figure 4a.

**case b :** The server is *active* at time  $t$  and it has already consumed  $x$  units of budget. In this case, the worst possible situation is when the server is preempted by the global scheduler until time  $t = d - (Q - x)$ . The worst-case allocation is depicted in Figure 4b, and is minimum for  $x = Q$ .

By comparing the two cases, it is clear that case b, with  $x = Q$ , is the most pessimistic. The corresponding function is  $Z_S(t)$ <sup>1</sup>.

□

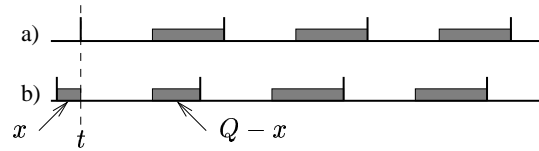


Figure 4. Worst-case allocation for the server.

Function  $Z_S(t)$  is complex to analyse. For this reason, we will instead consider a simple lower bound function  $\xi(t)$ :

$$\xi(t) = \max\{0, \alpha(t - \Delta)\}. \quad (1)$$

<sup>1</sup>Please note that Bernat et al. [3] and Saewong et al. [22] found the same kind of relationship for the Deferrable Server when scheduled by a fixed priority global scheduling algorithm.

where  $\alpha$  and  $\Delta$  are defined as follows:

- $\alpha$  is the share of the used processor, formally defined as  $\alpha = \lim_{t \rightarrow \infty} \frac{Z_S(t)}{t}$ ;
- $\Delta$  is the maximum delay in the time slots distribution, formally defined as  $\Delta = \max\{d \geq 0 : \exists t \geq 0 \ Z_S(t) \leq \alpha(t - d)\}$  or, equivalently,  $\min\{d \geq 0 : \forall t \geq 0 \ Z_S(t) \geq \alpha(t - d)\}$ .

The relationship between the characteristic function  $Z_S(t)$  and its lower bound function  $\xi(t)$  is shown in Figure 2. It is worth to note that the lower bound function  $\xi(t)$  is also used in [17, 7] by Feng and Mok to define a class of equivalent resource partitions.

### 3. Analysis of a fixed priority local scheduler

Now, we consider the problem of analysing the schedulability of a task set  $\Gamma_n$  on a server  $\mathbf{S}$ . This problem has been already approached in different ways. For example Saewong et al. [22] compute the worst-case response time of every task in the presence of the server. We propose a different approach because our ultimate goal is not only to check the feasibility on a given server, but also to find the “best” server that guarantees the schedulability of the task set. As usual in the real-time research, we must consider the worst-case scenario both for the server and for the task set.

When analysing the schedulability of the task  $\tau_i$ , we analyse the situation at the *critical instant*, which corresponds to the time in which all higher priority tasks are released. Saewong et al. [22] proved that this is indeed the worst case for task  $\tau_i$  even in the presence of the server.

The worst-case allocation of resource provided by a server  $\mathbf{S}$  is given by its characteristic function  $Z_S(t)$ , which represents (see Definition 3) the minimum available time for the task set in any interval of length  $t$ . So, informally speaking, we can say that a server can schedule a task set if the time provided by the server is greater than or equal to the time requested by the tasks. In the following, we will characterize the worst-case workload requested by the task set.

#### 3.1. Characterisation of fixed priority scheduling

Let us first tackle the problem of finding the minimum processor speed that maintains the task set schedulable. Slowing down the processor speed by a factor  $\alpha \leq 1$ , is equivalent to scale up the computation times by  $1/\alpha$ :

$$\forall i = 1, \dots, n \quad \tilde{C}_i = C_i/\alpha. \quad (2)$$

The problem is to find the minimum speed  $\alpha_{\min}$ , keeping the system schedulable. In [4], Bini and Buttazzo found a new way to express the schedulability condition under a fixed priority scheduling algorithm as a set of linear inequalities in the computation times  $\tilde{C}_i$ .

**Theorem 2 (Theorem 3 in [4])** A task set  $\Gamma_n = \{\tau_1, \tau_2, \dots, \tau_n\}$  is schedulable **if and only if**:

$$\bigwedge_{i=1 \dots n} \bigvee_{t \in \mathcal{P}_{i-1}(T_i)} \sum_{j=1}^i \left\lceil \frac{t}{T_j} \right\rceil C_j \leq t \quad (3)$$

where  $\mathcal{P}_i(t)$  is defined by the following recurrent expression:

$$\begin{cases} \mathcal{P}_0(t) = \{t\} \\ \mathcal{P}_i(t) = \mathcal{P}_{i-1} \left( \left\lfloor \frac{t}{T_i} \right\rfloor T_i \right) \cup \mathcal{P}_{i-1}(t). \end{cases} \quad (4)$$

By introducing the speed factor  $\alpha$ , we can reformulate condition (3) taking into account the substitution given by Equation (2). The result is the following:

$$\begin{aligned} & \bigwedge_{i=1 \dots n} \bigvee_{t \in \mathcal{P}_{i-1}(T_i)} \sum_{j=1}^i \left\lceil \frac{t}{T_j} \right\rceil \frac{C_j}{\alpha} \leq t \\ & \bigwedge_{i=1 \dots n} \bigvee_{t \in \mathcal{P}_{i-1}(T_i)} \frac{1}{\alpha} \sum_{j=1}^i \left\lceil \frac{t}{T_j} \right\rceil C_j \leq t \\ & \bigwedge_{i=1 \dots n} \bigvee_{t \in \mathcal{P}_{i-1}(T_i)} \alpha \geq \frac{\sum_{j=1}^i \left\lceil \frac{t}{T_j} \right\rceil C_j}{t} \end{aligned}$$

and finally:

$$\alpha \geq \alpha_{\min} = \max_{i=1 \dots n} \min_{t \in \mathcal{P}_{i-1}(T_i)} \frac{\sum_{j=1}^i \left\lceil \frac{t}{T_j} \right\rceil C_j}{t} \quad (5)$$

where  $\alpha_{\min}$  is the minimum allowed speed rate of a processor still capable to schedule the task set.

Now we introduce the delay  $\Delta$  in the analysis. In fact, when a task set is scheduled by a server, there can be a delay in the service because the server is not receiving any execution time from the global scheduler. To extend the previous result to the case when  $\Delta > 0$  we need to look at Equation (5) from a different point of view. In Figure 5, we show the worst-case workload for a task  $\tau_i$ , called  $W_i(t)$ , and the line  $\alpha_{\min}t$ . The line represents the amount of time that a processor with speed  $\alpha_{\min}$  provides to the task set. Task  $\tau_i$  is schedulable because:

$$\exists t^* \leq D_i : \alpha_{\min} t^* \geq W_i(t^*).$$

The presence of a delay  $\Delta$  forbids us to allocate time slots for an interval of length  $\Delta$ . This interval can start, in the worst case, at the critical instant for task  $\tau_i$ , i.e. when  $\tau_i$  and all higher priority tasks are released. It follows that the time provided by the server is bounded from below by the function  $\xi(t)$  previously defined in Equation (1). In Figure 5 we also show different functions  $\xi(t)$  for different values  $(\alpha, \Delta)$ . Therefore, when introducing  $\Delta$ , task  $\tau_i$  is schedu-

lable on server  $\mathbf{S}$ , characterized by function  $\xi(t)$ , if:

$$\exists t^* \leq D_i : \quad \xi(t^*) \geq W_i(t^*). \quad (6)$$

Notice that, as  $\Delta$  increases, the tangent point  $t^*$  may change. By using Equation (6), and increasing  $\Delta$  we can find all possible  $\alpha$  that make the task  $\tau_i$  schedulable.

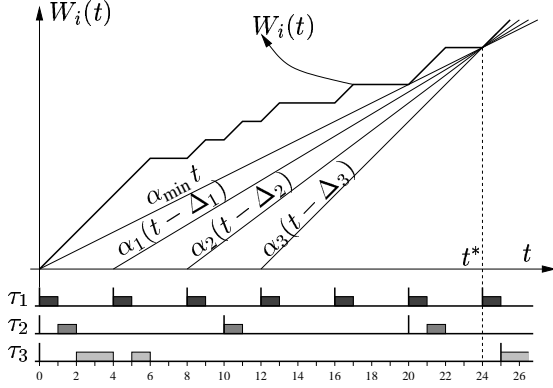


Figure 5. Workload and the  $\alpha_{\min}$  speed.

In order to find a closed formulation for the relation between  $\alpha$  and  $\Delta$  expressed by Equation (6), we need the following Lemma proved in [4].

**Lemma 1 (Lemma 4 in [4])** *Given a task subset  $\Gamma_i = \{\tau_1, \dots, \tau_i\}$  schedulable by fixed priorities and the set  $\mathcal{P}_i(b)$  as defined in Equation (4), the workload  $W_i(d)$  is*

$$W_i(d) = \min_{t \in \mathcal{P}_i(d)} \sum_{j=1}^i \left\lceil \frac{t}{T_j} \right\rceil C_j + (d - t).$$

By means of this lemma, the well known schedulability condition for the task set:

$$\forall i = 1 \dots n \quad C_i + W_{i-1}(D_i) \leq D_i$$

can be rewritten as follows:

$$\forall i = 1 \dots n$$

$$C_i + \min_{t \in \mathcal{P}_{i-1}(D_i)} \sum_{j=1}^{i-1} \left\lceil \frac{t}{T_j} \right\rceil C_j + (D_i - t) \leq D_i. \quad (7)$$

When the task set is served by a server with function  $\xi(t)$ , the schedulability condition expressed by Equation (7) becomes the following:

$$\forall i = 1 \dots n$$

$$\Delta + \frac{C_i}{\alpha} + \min_{t \in \mathcal{P}_{i-1}(D_i)} \sum_{j=1}^{i-1} \left\lceil \frac{t}{T_j} \right\rceil \frac{C_j}{\alpha} + (D_i - t) \leq D_i \quad (8)$$

Since the link between  $(\alpha, \Delta)$  is now explicit, we can manipulate the previous expression to obtain a direct relationship between  $\alpha$  and  $\Delta$ . In fact, the schedulability con-

dition of the single task  $\tau_i$  can be written as:

$$\Delta \leq D_i - \left( \frac{C_i}{\alpha} + \min_{t \in \mathcal{P}_{i-1}(D_i)} \sum_{j=1}^{i-1} \left\lceil \frac{t}{T_j} \right\rceil \frac{C_j}{\alpha} + (D_i - t) \right)$$

and, simplifying the expression, we finally obtain:

$$\Delta \leq \max_{t \in \mathcal{P}_{i-1}(D_i)} t - \frac{1}{\alpha} \left( C_i + \sum_{j=1}^{i-1} \left\lceil \frac{t}{T_j} \right\rceil C_j \right). \quad (9)$$

To take into account the schedulability of all the tasks in the set (and not only  $\tau_i$  as done so far), this condition must be true for every task. Hence, we obtain the following theorem.

**Theorem 3** *A task set  $\Gamma_n = \{\tau_1, \tau_2, \dots, \tau_n\}$  is schedulable by a server characterized by the lower bound function  $\xi(t)$  if:*

$$\Delta \leq \min_{i=1 \dots n} \max_{t \in \mathcal{P}_{i-1}(D_i)} t - \frac{1}{\alpha} \left( C_i + \sum_{j=1}^{i-1} \left\lceil \frac{t}{T_j} \right\rceil C_j \right) \quad (10)$$

where:

$$\begin{cases} \mathcal{P}_0(t) = \{t\} \\ \mathcal{P}_i(t) = \mathcal{P}_{i-1} \left( \left\lceil \frac{t}{T_i} \right\rceil T_i \right) \cup \mathcal{P}_{i-1}(t). \end{cases}$$

**Proof.** If  $\Delta$  satisfies Equation (10), then it satisfies all the equations (9) for every task in the set. Then every task is schedulable on such a local scheduler and so the whole set is, which proves the theorem.  $\square$

The last theorem does not only provide the schedulability condition for the task set  $\Gamma_n$  on a particular server, but it also gives a rule to select the server parameters to make the task set schedulable.

### 3.2. How to design a server

In our process of designing a server for an application  $\Gamma_n$ , the first step is to characterise the application by specifying all the individual task parameters. Once this step is carried out, by applying Theorem 3 a class of  $(\alpha, \Delta)$  pairs is obtained. On this set, which guarantees by definition the schedulability of application  $\Gamma_n$ , we perform the server selection by optimizing a desired cost function. One possible cost function is the overhead of the scheduler. When choosing the server parameters, we must balance two opposite needs:

1. the required bandwidth should be small, to not waste the total processor capacity;
2. the server period should be large, otherwise the time wasted in context switches performed by the global scheduler will be too high.

Then, a typical cost function to be minimised may be:

$$c_1 \frac{T_{\text{Overhead}}}{P} + c_2 \alpha \quad (11)$$

where  $T_{\text{Overhead}}$  is the global scheduler context switch time,  $P$  is the server period,  $\alpha$  is the fraction of bandwidth, and  $c_1$  and  $c_2$  are two designer defined constants. Moreover some additional constraints in the  $(\alpha, \Delta)$  domain, other than those specified by Equation (10), may be required. For example, if we use a fixed priority global scheduler, to maximize the resource utilisation we could impose the server periods to be harmonic.

$i$	$T_i$	$C_i$	$D_i$
1	4	1	4
2	10	1	10
3	25	3	25

Table 1. An example:  $\Gamma_3$  data

To clarify the whole methodology of server parameters selection, we provide a practical example. Suppose we have a set of three tasks  $\Gamma_3$  with the data shown in table 1 (for simplicity, we choose  $D_i = T_i$ , but the approach is the same when  $D_i < T_i$ ). The utilisation is  $U = 1/4 + 1/10 + 3/25 = 47/100$ , hence  $\alpha$  cannot definitively be smaller than  $47/100$ . The schedule corresponding to the worst-case scenario (i.e. the critical instant) is shown in Figure 6.

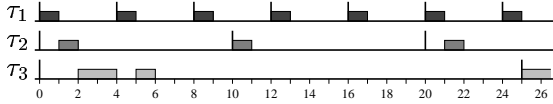


Figure 6. Worst-case schedule of  $\Gamma_3$

By expanding Equation (9) for  $\tau_1$  we obtain the following inequality:

$$\begin{aligned} \mathcal{P}_0(4) &= \{4\} \\ \Delta &\leq 4 - 1/\alpha \end{aligned}$$

Doing the same for  $\tau_2$ , we obtain:

$$\begin{aligned} \mathcal{P}_1(10) &= \{8, 10\} \\ \Delta &\leq \max\{8 - 3/\alpha, 10 - 4/\alpha\} \end{aligned}$$

and, finally, for the last task  $\tau_3$ :

$$\begin{aligned} \mathcal{P}_2(25) &= \{20, 24, 25\} \\ \Delta &\leq \max\{20 - 10/\alpha, 24 - 12/\alpha, 25 - 13/\alpha\} \\ \Delta &\leq 24 - 12/\alpha. \end{aligned}$$

In order to make all the three tasks schedulable, all the inequalities must hold at the same time, as stated in Theorem 3. It follows that:

$$\Delta \leq \min\left\{4 - \frac{1}{\alpha}, \max\left\{8 - \frac{3}{\alpha}, 10 - \frac{4}{\alpha}\right\}, 24 - \frac{12}{\alpha}\right\} \quad (12)$$

In Figure 7, we plotted the set of  $(\alpha, \Delta)$  pairs defined by Equation (12) as a gray area whose upper boundary is drawn by a thick line. This boundary is a piece-wise hyperbole, because it is the minimum between inequalities, each one of them is an hyperbole (see Equations (9) and (10)). Notice that, in this particular case, the schedulability condition for task  $\tau_2$  does not provide any additional constraint in the final function.

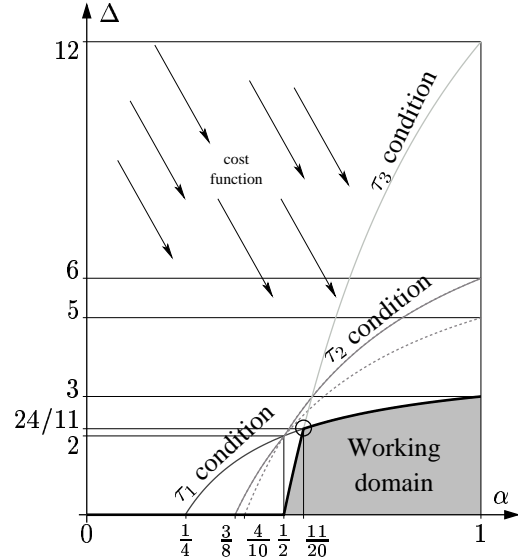


Figure 7. Worst-case schedule of  $\Gamma_3$

In Figure 7, we also plotted a qualitative cost function that is increasing as  $\alpha$  increases, and decreasing as  $\Delta$  increases (see Equation 11). If we minimize this qualitative function on the domain expressed by Equation (12), the solution is  $\alpha = \frac{11}{20}$  and  $\Delta = \frac{24}{11}$ . We can now find the period  $P$  and the budget  $Q$  of the server corresponding to the selected solution:

$$\Delta = 2(P - Q) \quad \alpha = \frac{Q}{P}$$

then:

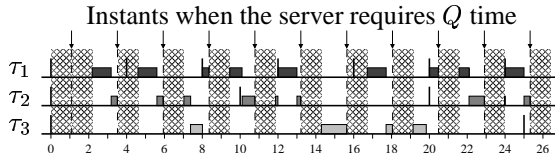
$$P = \frac{\Delta}{2(1 - \alpha)} \quad Q = \alpha P$$

By substitution, we obtain the server parameters:  $P = \frac{80}{33} \approx 2.424$  and  $Q = \frac{4}{3} \approx 1.333$ .

Finally, in Figure (8), we show the schedule for the example application, obtained by considering the worst-case scenario both for the time requested by tasks and for the time provided by the server. The shaded areas represent intervals where the server does not receive any allocation by the global scheduler. As expected, all tasks complete within their deadlines.

## 4. Conclusions and future work

In this paper, we presented a methodology for computing the “best” server parameters in an hierarchical scheduling



**Figure 8. Worst-case schedule of  $\Gamma_3$  on a server with the computed parameters**

system, when the application is scheduled by a fixed priority local scheduler. The proposed methodology is not optimal, as we use the function  $\xi(t)$  rather than  $Z_S(t)$ . However, it can easily be extended to other kind of schedulers, and this will be the subject of future work.

We also believe that the concept of “partition” is quite general and will allow us to extend this methodology in other directions in the next future. For example, we would like to analyse other kind of global allocation mechanisms that cannot be included in the server category. We also believe that this work can be used as a basis for analysing the composition of arbitrary kinds of scheduling mechanisms.

Finally, it would be interesting to understand if it is possible to devise a “optimal” allocation strategy, to avoid the “loss” of resource utilisation due to the delay  $\Delta$ .

## 5. Acknowledgments

We would like to thank Guillem Bernat for the useful ideas and suggestions that helped us in improving the paper.

## References

- [1] L. Abeni and G. C. Buttazzo. Integrating multimedia applications in hard real-time systems. In *Proceedings of the 19<sup>th</sup> IEEE Real-Time Systems Symposium*, Madrid, Spain, December 1998.
- [2] S. Baruah, N. Cohen, C. Plaxton, and D. Varvel. Proportionate progress: A notion of fairness in resource allocation. *Algorithmica*, 6, 1996.
- [3] G. Bernat and A. Burns. New results on fixed priority aperiodic servers. In *Proceedings of the 20<sup>th</sup> IEEE Real-Time Systems Symposium*, 1999.
- [4] E. Bini and G. C. Buttazzo. The space of rate monotonic schedulability. In *Proceedings of the 23<sup>rd</sup> IEEE Real-Time Systems Symposium*, Austin, Texas, U.S.A., December 2002.
- [5] Z. Deng and J. W. S. Liu. Scheduling real-time applications in open environment. In *Proceedings of the IEEE Real-Time Systems Symposium*, San Francisco, December 1997.
- [6] Z. Deng, J. W. S. Liu, and J. Sun. A scheme for scheduling hard real-time applications in open system environment. In *Proceedings of the 9<sup>th</sup> Euromicro Workshop on Real-Time Systems*, 1997.
- [7] X. Feng and A. K. Mok. A model of hierarchical real-time virtual resources. In *Proceedings of the 23<sup>rd</sup> IEEE Real-Time Systems Symposium*, pages 26–35, Austin, TX, USA, December 2002.
- [8] T. Ghazalie and T. Baker. Aperiodic servers in a deadline scheduling environment. *Journal of Real-Time System*, 9, 1995.
- [9] K. Jeffay, F. D. Smith, A. Moorthy, and J. Anderson. Proportional share scheduling of operating systems services for real-time applications. In *Proceedings of the 19th IEEE Real-Time Systems Symposium*, pages 480–491, Madrid, Spain, december 1998. IEEE.
- [10] H. Kopetz, A. Damm, C. Koza, M. Mulazzani, W. Schwabla, C. Senft, and R. Zainlinger. Distributed fault-tolerant real-time systems: The mars approach. *IEEE Micro*, 9(1), February 1989.
- [11] T.-W. Kuo and C.-H. Li. Fixed-priority-driven open environment for real-time applications. In *Proceedings of the IEEE Real Systems Symposium*, December 1999.
- [12] J. P. Lehoczky, L. Sha, and J. Strosnider. Enhanced aperiodic responsiveness in hard real-time environment. In *Proceedings of the 8<sup>th</sup> IEEE Real-Time Systems Symposium*, pages 110–123, San José, December 1987.
- [13] G. Lipari. *Resource Reservation in Real-Time Systems*. PhD thesis, Scuola Superiore S. Anna, Pisa, Italy, 2000.
- [14] G. Lipari and S. Baruah. Efficient scheduling of multi-task applications in open systems. In *IEEE Proceedings of the 6th Real-Time Systems and Applications Symposium*, June 2000.
- [15] C. W. Mercer, R. Rajkumar, and H. Tokuda. Applying hard real-time technology to multimedia systems. In *Workshop on the Role of Real-Time in Multimedia/Interactive Computing System*, 1993.
- [16] A. K. Mok and X. Feng. Towards compositionality in real-time resource partitioning based on regularity bounds. In *Proceedings of the 22<sup>nd</sup> IEEE Real-Time Systems Symposium*, 2001.
- [17] A. K. Mok, X. Feng, and D. Chen. Resource partition for real-time systems. In *Proceedings of the 7<sup>th</sup> IEEE Real-Time Technology and Applications Symposium*, pages 75–84, 2001.
- [18] A. K. Parekh and R. 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.
- [19] A. K. Parekh and R. G. Gallager. A generalized processor sharing approach to flow control in intergrated services networks: the multiple node case. *IEEE/ACM Transactions on Networking*, 2:137–150, April 1994.
- [20] R. Rajkumar, K. Juvva, A. Molano, and S. Oikawa. Resource kernels: A resource-centric approach to real-time and multimedia systems. In *Proceedings of the SPIE/ACM Conference on Multimedia Computing and Networking*, January 1998.
- [21] D. Reed and R. F. (eds.). *Nemesis, the kernel – overview*, May 1997.
- [22] S. Saewong, R. Rajkumar, J. P. Lehoczky, and M. H. Klein. Analysis of hierarchical fixed-priority scheduling. In *Proceedings of the 14<sup>th</sup> IEEE Euromicro Conference on Real-Time Systems*, June 2002.
- [23] B. Sprunt, L. Sha, and J. Lehoczky. Aperiodic task scheduling for hard-real-time systems. *Journal of Real-Time Systems*, 1, July 1989.
- [24] M. Spuri and G. Buttazzo. Scheduling aperiodic tasks in dynamic priority systems. *Journal of Real-Time Systems*, 10(2), 1996.