A methodology for designing hierarchical scheduling systems

Giuseppe Lipari and Enrico Bini

***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.**

I. Introduction

Thanks to the recent advances in the field of computer architectures, computers are getting faster and faster. It is now possible to concurrently execute different real-time applications in the same system. In this paper, an *application* is a set of concurrent tasks that implement a software program.

The motivation for executing different applications in the same system is in cost reduction and in the re-use of legacy applications on new, faster systems. This trend can be observed both in the general purpose computer area and in the embedded system area.

When executing many 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. In this way 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 an 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 [1].

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.

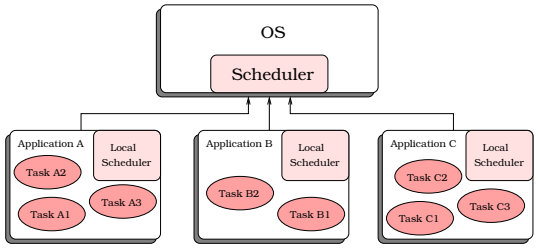

Fig. 1. Hierarchical scheduler structure.

The global scheduler assigns each application 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) [2], [3], then the bandwidth allocation would be easy. Unfortunately, the GPS algorithm is a theoretical abstraction that cannot be implemented in practice, but only approximated [4], [5], [6].

In this paper, we consider the class of algorithms that can be described by the *periodic server abstraction*. In the real-time literature, a *server* is an algorithm that is used to schedule *non-periodic* activities together with hard real-time periodic tasks. Many algorithms have been proposed that belong to the class of *periodic servers* [7], [8], [9], [10], [6]. The concept of *server* was later extended to indicate also the set of parameters that must be assigned to each application. According to a periodic server algorithm, each application is assigned a pair (Q, P), with the meaning that the application will receive Q units of execution every P units of time. The global scheduling mechanism decides when to schedule the applications; the selected application, by using the local scheduling mechanism,

Giuseppe Lipari is with Scuola Superiore Sant'Anna, Pisa (Italy), e-mail: lipari@sssup.it

Enrico Bini is with Scuola Superiore Sant'Anna, Pisa (Italy), e-mail: e.bini@sssup.it

decides which 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. [11] presented a response time analysis for the schedulability of such a hierarchical system.

However, an open issue remains to be answered: *given a real-time application, what is the "best" pair (Q, P) that can be assigned to the application so that it is schedulable?*.

In this paper we present a technique that answers the previous question for applications consisting of periodic (or sporadic) tasks, scheduled by a fixed priority local scheduler. Given an application, 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 I-A), the system model and most of the concept and definition used in the paper are presented in Section II. In Section III, we present our analysis and propose our method. Finally in Section VI, we draw our conclusion and propose some future development.

A. 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 [12], [13]. The basic idea, formalized by Rajkumar et al. [14], 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 [9], [8], [15], [7], [6]. 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. [11] 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. Deng and Liu in [16], [17] proposed a two-level hierarchical architecture, which uses the EDF as global scheduler and a dedicated Total Bandwidth Server [7] 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. [18] for using RM as global scheduling algorithm, but the authors assume that all tasks are periodic with harmonic periods.

Lipari and Baruah in [19], [20] 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 [21] 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 Feng and Mok's 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 inverse problem: given an application, scheduled by a local fixed priority scheduler, how to select the "best" server.

Finally, Almeida et. al [22] presented a methodology for computing the parameters for scheduling messages on the FTT-CAN network protocol. Their approach can be seen as a preliminary version of our design methodology applied to the CAN bus.

II. System model and definitions

An *application* is a set Γ_n of n periodic or sporadic tasks, ordered by decreasing priority. Every task τ_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 assume tasks to be independent. 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 [8], [10], [7], the Constant Bandwidth Server [6], etc. All these algorithms have different peculiarities. Since we do not want to concentrate on one particular mechanism, in Section II-B 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, etc. In the following subsection, we give an overview of the different mechanisms for providing partitions.

A. Partitions

A partition is a function $\mathbf{\Pi}(t)$ that has values in $\{0, 1\}$. If $\mathbf{\Pi}(t) = 1$, then the resource is allocated to the application at time t. A partition is periodic if it exists $P > 0$ such that $\mathbf{\Pi}(t) \equiv \mathbf{\Pi}(t + P)$ (see [23], [24]).

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. 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 $\mathbf{\Pi}(t)$, we define the characteristic function $Z_{\mathbf{\Pi}}(t)$ as the *minimum amount of time* provided by the partition in every time interval of length $t \geq 0$:

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

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

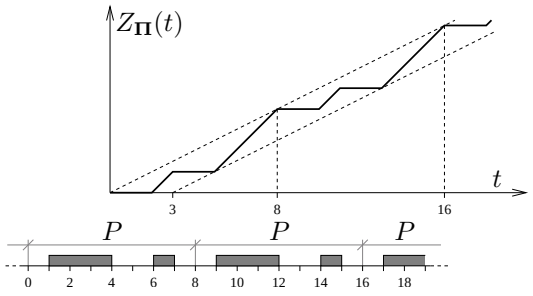

Fig. 2. An example of $Z_{\mathbf{\Pi}}(t)$.

B. The server abstraction

The concept of server algorithm was originally devised for minimizing the response time of aperiodic tasks when scheduled together with hard real-time tasks [8], [10]. Recently, some server algorithm has been used for providing resource reservations [14], [6]. 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 important differences) all the algorithms cited so far.

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

a) 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 periodic server algorithms presented in literature 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 [6]. 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.

C. 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 trade-off between a large P and

a small $\alpha = \frac{Q}{P}$. In fact, as we will see in more detail in Section IV, a small P may cause a high number of context switches between servers, whereas a large P leads to a high utilisation $\frac{Q}{P}$ and thus 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 $\mathbf{S}$, we define $\mathsf{legal}(\mathbf{S})$ as the set of partitions $\mathbf{\Pi}$ that can be generated by the server algorithm.

Definition 3: Given a server $\mathbf{S}$, $Z_{\mathbf{S}}(t)$ is the *minimum amount of time* provided by the server $\mathbf{S}$ in every time interval of length $t \geq 0$.

$$Z_{\mathbf{S}}(t) = \min_{\Pi \in \mathsf{legal}(\mathbf{S})} Z_{\mathbf{\Pi}}(t)$$

To understand the importance of the function $Z_{\mathbf{S}}(t)$, consider the schedulability problem of a single task τ_i on the server $\mathbf{S}$. If $Z_{\mathbf{S}}(D_i)$, 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 τ_i and all its higher priority tasks in the same interval, then task τ_i is feasible on the server $\mathbf{S}$.

In Figure 3, we plot the characteristic function $Z_{\mathbf{S}}(t)$ 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. In this case, the first instant of time at which they will receive execution is at $2(P - Q)$.

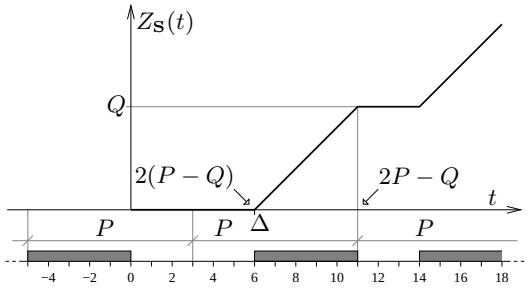

Fig. 3. General case of periodic server.

Theorem 1: Given a server algorithm defined by the rules of Section II-B, and with parameters (Q, P), and defined $k = \left\lceil \frac{t-(P-Q)}{P} \right\rceil$, its characteristic function $Z_{\mathbf{S}}(t)$ is:

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

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 $Z_{\mathbf{S}}(t)$[1].

■

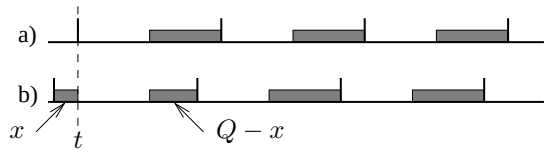

Fig. 4. Worst-case allocation for the server.

Function $Z_{\mathbf{S}}(t)$ is complex to analyse. For this reason, we will first consider a simple lower bound function $\xi(t)$:

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

where α and Δ are defined as follows:

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

The relationship between the characteristic function $Z_{\mathbf{S}}(t)$ 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 [24], [23] by Feng and Mok to define a class of equivalent resource partitions.

Also, function $\xi(t)$ can be used to model other kinds of algorithms and partitions. Therefore, in the remaining of the paper we will first develop our methodology using function $\xi(t)$ (see Sections III and IV). In this way, the methodology is easily extensible to other kind of allocation algorithms. Then, we will analyse what changes in our methodology when we use directly the characteristic function $Z_{\mathbf{S}}(t)$.

III. Analysis of a fixed priority local scheduler

Now, we are going to analyse the schedulability of a task set Γ_n on a server $\mathbf{S}$. This problem has been already approached in different ways. For example Saewong et al. [11] compute the worst-case response time of every task in the presence of a 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.

[1]Please note that Bernat et al. [15] and Saewong et al. [11] found the same kind of relationship for the Deferrable Server when scheduled by a fixed priority global scheduling algorithm.

When analysing the schedulability of task τ_i, we study the situation at the *critical instant*, which corresponds to the time in which all higher priority tasks are released. Saewong et al. [11] proved that this is indeed the worst case for task τ_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 $Z_{\mathbf{S}}(t)$, 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 the time requested by the tasks. In the following, we will characterize the worst-case workload requested by the task set.

A. 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 $1/\alpha$:

$$\forall i = 1, \ldots, n \quad \widetilde{C}_i = C_i/\alpha. \tag{2}$$

The problem is to find the minimum speed $\alpha_{\min}$, keeping the system schedulable. In [25], 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 C_i.

Theorem 2 (Theorem 3 in [25]): A task set $\Gamma_n = \{\tau_1, \tau_2, \ldots, \tau_n\}$ is schedulable **if and only if**:

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

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

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

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

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

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

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

and finally:

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

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

Now we introduce the delay Δ 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 τ_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 τ_i is schedulable because:

$$\exists t^* \in (0, D_i] : \quad \alpha_{\min} t^* \geq W_i(t^*).$$

The presence of a delay Δ forbids us to allocate time slots for an interval of length Δ. This interval can start, in the worst case, at the critical instant for task τ_i, i.e. when τ_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 (α, Δ). Therefore, when introducing Δ, task τ_i is schedulable on server $\mathbf{S}$, characterized by function $\xi(t)$, if:

$$\exists t^* \in (0, D_i] : \quad \xi(t^*) \geq W_i(t^*). \tag{6}$$

Notice that, as Δ increases, the tangent point t^* may change. By using Equation (6), and increasing Δ we can find all possible α that make the task τ_i schedulable.

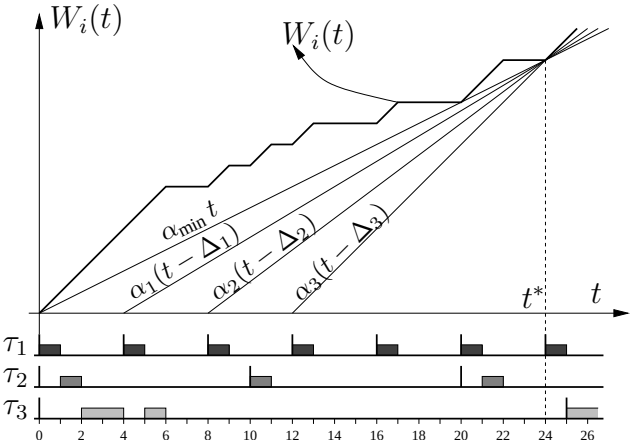

Fig. 5. Workload and the $\alpha_{\min}$ speed.

In order to find a closed formulation for the relation between α and Δ expressed by Equation (6), we need the following Lemma proved in [25].

Lemma 1 (Lemma 4 in [25]): Given a task subset $\Gamma_i = \{\tau_1, \ldots, \tau_i\}$ 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:

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

can be rewritten as follows:

$$\forall i = 1 \ldots n$$

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

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 (α, Δ) is now explicit, we can manipulate the previous expression to obtain a direct relationship between α and Δ. In fact, the schedulability condition of the single task τ_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 τ_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:

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

Proof: If Δ 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. ∎

IV. How to design a server

In our process of designing a server for an application Γ_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 (α, Δ) pairs is obtained. On this class, which guarantees by definition the schedulability of application Γ_n, we perform the server selection by optimizing a desired cost function. One possible cost function is the overhead of the scheduler. In fact, 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.

Thus a typical cost function to be minimised may be the following:

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

where T_{Overhead} is the global scheduler context switch time, P is the server period, α is the fraction of bandwidth, and c_1 and c_2 are two designer defined constants. Moreover some additional constraints in the (α, Δ) 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 I
AN EXAMPLE: Γ_3 DATA.

To clarify the methodology consider the following example. Suppose we have a set of three tasks Γ_3 with the data shown in table I (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 α cannot definitively be smaller than $47/100$. The schedule corresponding to the worst-case scenario (i.e. the critical instant) when the application is scheduled alone on the processor is shown in Figure 6.

By expanding Equation (9) for τ_1 we obtain the following inequality:

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

Doing the same for τ_2, we obtain:

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

and, finally, for the last task τ_3:

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

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:

$$\Delta \leq \min\{4 - \frac{1}{\alpha}, \max\{8 - \frac{3}{\alpha}, 10 - \frac{4}{\alpha}\}, 24 - \frac{1}{2}\alpha\} \quad (12)$$

In Figure 7, we plotted the set of (α, Δ) 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 τ_2 does not provide any additional constraint.

In Figure 7, we also plotted a qualitative cost function that increases as α increases, and decreases as Δ increases (see Equation 11). If we minimize this qualitative function on the domain expressed by Equation (12), the solution is $\alpha = \frac{11}{20}$

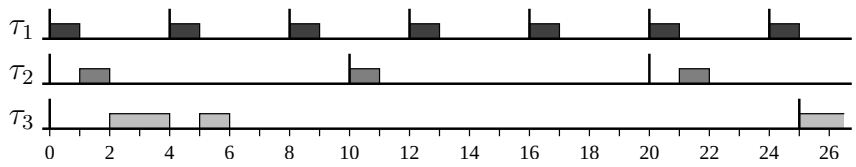

Fig. 6. Worst-case schedule of Γ_3.

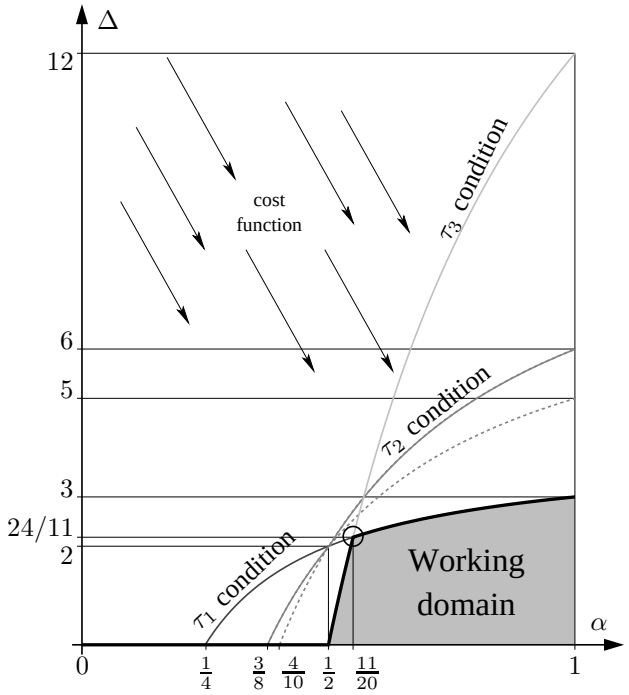

Fig. 7. Γ_3 example: server parameters in the (α, Δ) domain.

and $\Delta = \frac{24}{11}$. We can now find the period P and the budget Q of the server corresponding to the selected solution:

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

then:

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

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.

V. Improving the methodology

In previous sections, we described a methodology for obtaining the parameters of a periodic server given an application with fixed priority as a local scheduler. We used function $\xi(t)$ which gives an lower bound on the characteristic function of the server. It follows that the methodology is quite general and can be applied also to other kind of on-line algorithms. For example, it is possible to bound the characteristic function of a static partition with a function of the same form $\xi(t)$.

However, when considering periodic server algorithms, the presented methodology is not optimal because we use only a lower bound of the characteristic function of the server. To be more precise, we will consider directly function $Z_{\mathbf{S}}(t)$.

First, we rewrite Equation (10) substituting (α, Δ) with (Q, P). Recall the relationship between (Q, P) and (α, Δ):

$$\Delta = 2(P - Q) \qquad \alpha = \frac{Q}{P}. \tag{13}$$

Now, equation (10) can also be written by using the logical *and*, *or* operators:

$$\bigwedge_{i=1\ldots n} \bigvee_{t \in \mathcal{P}_{i-1}(D_i)} \Delta \leq t - \frac{1}{\alpha} \sum_{j=1}^{i} \left\lceil \frac{t}{T_j} \right\rceil C_j. \tag{14}$$

To simplify the notation, we define the processor *demand* of the first i higher priority tasks as $Y_i(t) = \sum_{j=1}^{i} \left\lceil \frac{t}{T_j} \right\rceil C_j$. Then we substitute (13) in the previous equation:

$$\Delta \leq t - \frac{Y_i(t)}{\alpha}$$
$$2(P - Q) \leq t - \frac{P\, Y_i(t)}{Q}$$
$$2Q^2 + (t - 2P)Q - P\, Y_i(t) \geq 0$$
$$Q \geq \frac{\sqrt{(t-2P)^2 + 8PY_i(t)} + 2P - t}{4}$$

and then:

$$Q \geq \max_{i=1\ldots n} \min_{t \in \mathcal{P}_{i-1}(D_i)} \frac{\sqrt{(2P-t)^2 + 8PY_i(t)} + 2P - t}{4} \tag{15}$$

This expression provides the same server parameters as Eq. (10), since simply obtained by substituting (α, Δ) with (Q, P). However, it is useful to compare this result with the one we will get next.

Now, we directly use $Z_{\mathbf{S}}(t)$. By following the same reasoning of Section III, task τ_i is schedulable **iff**:

$$\exists t^* \in (0, D_i] : \quad Z_{\mathbf{S}}(t^*) \geq Y_i(t^*). \tag{16}$$

Therefore, the following Theorem holds.

Theorem 4: A task set $\Gamma_n = \{\tau_1, \tau_2, \ldots, \tau_n\}$ is schedulable by a periodic server characterized by function $Z_{\mathbf{S}}(t)$ **iff**:

$$Q \geq \max_{i=1\ldots n} \min_{t \in \mathcal{P}_{i-1}(D_i)} q(i, t, P) \tag{17}$$

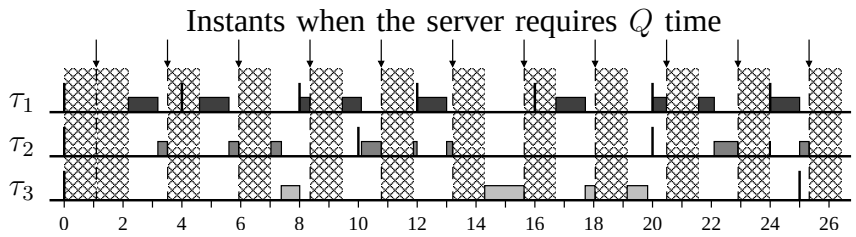

Fig. 8. Worst-case schedule of Γ_3 on a server with the computed parameters.

where:

$$q(i,t,P) = \begin{cases} \dfrac{Y_i(t)}{k-1} & \frac{t+2Q}{k+1} \le P < \frac{t+Q}{k} \\ P - \dfrac{t - Y_i(t)}{k+1} & \frac{t+Q}{k+1} \le P < \frac{t+2Q}{k+1} \end{cases}$$

and, as in Theorem 3:

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

Proof: From [26], [25], we know that a task set is schedulable **iff**:

$$\bigwedge_{i=1\ldots n} \bigvee_{t \in \mathcal{P}_{i-1}(D_i)} Y_i(t) \le t, \tag{18}$$

meaning that for every task, the available time t must be greater than or equal to the required time in at least one time instant within the task deadline.

If only a fraction of the processor time is available, expressed by function $Z_{\mathbf{S}}(t)$, the necessary and sufficient condition becomes:

$$\bigwedge_{i=1\ldots n} \bigvee_{t \in \mathcal{P}_{i-1}(D_i)} Y_i(t) \le Z_{\mathbf{S}}(t). \tag{19}$$

By substituting the analytical expression for $Z_{\mathbf{S}}(t)$, given by Theorem 1, in $Y_i(t) \le Z_{\mathbf{S}}(t)$ we obtain:

$$\bigwedge_{i=1\ldots n} \bigvee_{t \in \mathcal{P}_{i-1}(D_i)} Q \ge q(i,t,P) \tag{20}$$

Then, by one simple step we get:

$$Q \ge \max_{i=1\ldots n} \min_{t \in \mathcal{P}_{i-1}(D_i)} q(i,t,P) \tag{21}$$

as required. ■

It is now possible to evaluate the benefit of using the tighter Equation (17) instead of Equation (15). Since both the relationships are obtained by assembling in the same way two different functions of t, i and P, we will just compare the two composing functions. Consider, using again the example of Section IV, the second task ($i = 2$) and fix t equal to 10. For this task set the processor demand $Y_2(10)$ is 4. Finally we plot $q(2, 10, P)$ and the function given by the substitution in the (α, Δ) model.

The dark gray area represents the pairs (Q, P) that make the application tasks schedulable on the server, but cannot be obtained by using the (α, Δ) model (Equation (15)). For different values of t and i the same functions in Figure 9 are scaled along the two axes, not altering the overall shape.

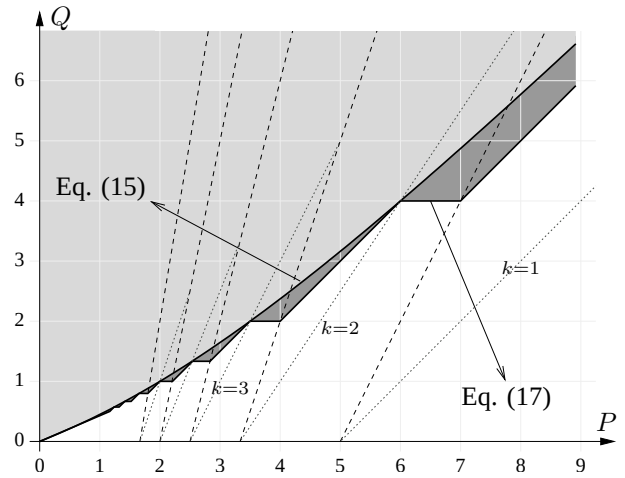

Fig. 9. Comparison between Equations (15) and (17).

It is worth to note that for small values of the period P the difference between the two curves becomes negligible and tend to 0 as P approaches 0. This is justified by the fact that small server periods tend to approximate fluid allocation [2], [3]. As expected, for small periods, the slope of both the curves tends to $Y_i(t)/t$ ($2/5$ in Figure 9).

Of course, the final region is the combination of many curves similar to the one shown in Figure 9. We do not show here the final set of feasible (Q, P) because the resulting figure is quite complex.

As last remark, we show the result of the exact methodology applied on the example task set of Table I. When we calculate the server's parameters as described in Section IV, we have some spare time left due to the fact that the methodology is not optimal. For example, by looking at Figure 8, it is possible to see that the processor is idle just before time instant $t = 20$. This idle time interval is $\frac{2}{11}$ units of time long, and can reduced by selecting the server parameters more carefully.

In fact, if we apply the exact methodology described in this section, we obtain $P = \frac{45}{14} \approx 3.214$ and $Q = \frac{12}{7} \approx 1.714$. By using these parameters, the task set is still schedulable but the period is larger and the required bandwidth ($\frac{Q}{P} = \frac{8}{15} \approx 0.533$) is smaller than those obtained using the (α, Δ) server model.

VI. Conclusions and future work

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

when the application is scheduled by a fixed priority local scheduler.

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.

REFERENCES

[1] 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*, vol. 9, no. 1, February 1989.

[2] 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*, vol. 1, no. 3, pp. 344–357, June 1993.

[3] ——, "A generalized processor sharing approach to flow control in intergrated services networks: the multiple node case," *IEEE/ACM Transanctions on Networking*, vol. 2, pp. 137–150, April 1994.

[4] 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*. Madrid, Spain: IEEE, december 1998, pp. 480–491.

[5] S. Baruah, N. Cohen, C. Plaxton, and D. Varvel, "Proportionate progress: A notion of fairness in resource allocation," *Algorithmica*, vol. 6, 1996.

[6] L. Abeni and G. C. Buttazzo, "Integrating multimedia applications in hard real-time systems," in *Proceedings of the* 19^{th} *IEEE Real-Time Systems Symposium*, Madrid, Spain, December 1998.

[7] M. Spuri and G. Buttazzo, "Scheduling aperiodic tasks in dynamic priority systems," *Journal of Real-Time Systems*, vol. 10, no. 2, 1996.

[8] B. Sprunt, L. Sha, and J. Lehoczky, "Aperiodic task scheduling for hard-real-time systems," *Journal of Real-Time Systems*, vol. 1, July 1989.

[9] J. P. Lehoczky, L. Sha, and J. Strosnider, "Enhanced aperiodic responsiveness in hard real-time environment," in *Proceedings of the* 8^{th} *IEEE Real-Time Systems Symposium*, San José, December 1987, pp. 110–123.

[10] T. Ghazalie and T. Baker, "Aperiodic servers in a deadline scheduling environment," *Journal of Real-Time System*, vol. 9, 1995.

[11] S. Saewong, R. Rajkumar, J. P. Lehoczky, and M. H. Klein, "Analysis of hierarchical fixed-priority scheduling," in *Proceedings of the* 14^{th} *IEEE Euromicro Conference on Real-Time Systems*, June 2002.

[12] 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.

[13] D. Reed and R. F. (eds.), "Nemesis, the kernel – overview," May 1997.

[14] 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.

[15] G. Bernat and A. Burns, "New results on fixed priority aperiodic servers," in *Proceedings of the* 20^{th} *IEEE Real-Time Systems Symposium*, 1999.

[16] 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^{th} *Euromicro Workshop on Real-Time Systems*, 1997.

[17] Z. Deng and J. W. S. Liu, "Scheduling real-time applications in open envirovment," in *Proceedings of the IEEE Real-Time Systems Symposium*, San Francisco, December 1997.

[18] 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.

[19] 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.

[20] G. Lipari, "Resource reservation in real-time systems," Ph.D. dissertation, Scuola Superiore S.Anna, Pisa, Italy, 2000.

[21] A. K. Mok and X. Feng, "Towards compositionality in real-time resource partitioning based on regularity bounds," in *Proceedings of the* 22^{nd} *IEEE Real-Time Systems Symposium*, 2001.

[22] L. Almeida, P. Pedreiras, and J. A. G. Fonseca, "The FFT-CAN protocol: Why and how," *IEEE Transaction on Industrial Electronics*, vol. 49, no. 6, pp. 1189–1201, dec 2002.

[23] X. Feng and A. K. Mok, "A model of hierarchical real-time virtual resources," in *Proceedings of the* 23^{rd} *IEEE Real-Time Systems Symposium*, Austin, TX, USA, December 2002, pp. 26–35.

[24] A. K. Mok, X. Feng, and D. Chen, "Resource partition for real-time systems," in *Proceedings of the* 7^{th} *IEEE Real-Time Technology and Applications Symposium*, 2001, pp. 75–84.

[25] E. Bini and G. C. Buttazzo, "The space of rate monotonic schedulability," in *Proceedings of the* 23^{rd} *IEEE Real-Time Systems Symposium*, Austin, Texas, U.S.A., December 2002.

[26] J. P. Lehoczky, L. Sha, and Y. Ding, "The rate-monotonic scheduling algorithm: Exact characterization and average case behavior," in *Proceedings of the* 10^{th} *IEEE Real-Time Systems Symposium*, 1989, pp. 166–172.

Giuseppe Lipari is Assistant professor of Computer Engineering at the Scuola Superiore Sant'Anna in Pisa (Italy). He received the Laurea degree in Computer Engineering from the "Università di Pisa" in 1996. He received a Ph.D. degree in Computer Engineering from Scuola Superiore Sant'Anna in 2000.

During 1999 he was a visiting student at the University of Vermont and at the University of North Carolina at Chapel Hill, working with Prof. S. Baruah and Prof. K. Jeffay on novel scheduling algorithms for Resource Reservation on Real-Time systems. His main research interests include real-time operating systems, scheduling algorithm for soft real-time systems, component-based real-time kernels for embedded systems.

Enrico Bini is a PhD student in Computer Engineering at Scuola Superiore Sant'Anna in Pisa (Italy). In 2000, he received the Laurea degree in Computer Engineering from the "Università di Pisa". In the same year he received the Licenza degree from the Scuola Superiore Sant'Anna.

In 1999 he was a visiting student at Technische Universiteit Delft, in the Nederlands. In 2001 he was with Ericsson Lab Italy in Roma. Currently he a visiting student at University of North Carolina at Chapel Hill, where he is collaborating with prof. Sanjoy Baruah

His interests cover scheduling algorithms, real-time operating systems, embedded systems design and linear/non-linear optimization algorithms.