Analysis and Implicit Deadline Synchronization of Distributed Transactions Scheduled by EDF

Nicola Serreli, Giuseppe Lipari, Enrico Bini
Scuola Superiore Sant'Anna, Pisa, Italy
Email: {n.serreli,g.lipari,e.bini}@sssup.it

***Abstract*—In real-time distributed systems, it is common to model an application as a set of transactions, i.e. chains of tasks activated periodically, that must complete before an end-to-end deadline.**

In this paper, (i) we extend the analysis of transactions scheduled by EDF to account for sporadic activations and (ii) we propose a protocol for assigning deadlines to tasks that does not rely on a global time reference.

First, we show that the scenario of strictly periodic activations is not the worst when the transactions are activated sporadically. For this reason we extend the demand bound function (dbf) to sporadic transactions and we propose a suitable schedulability analysis. Then, we propose IDSP (Implicit Deadline Synchronization Protocol) to assign the absolute deadlines to jobs at run time. The protocol does not require synchronization between nodes and uses only local information. We guarantee that the demand that can be generated at run time is always bounded by the sporadic dbf computed off-line.

I. Introduction

Distributed real-time systems are often modeled as a set of real-time *transactions* [1]. Each transaction is a chain of real-time tasks, and each task is allocated on a (possibly different) computational node. The first task in the transaction is activated periodically, or by external events characterized by a minimum interarrival time. The other tasks are activated upon the completion of the preceding one. All tasks in the transaction must complete within an *end-to-end* deadline relative to the activation time of the transaction. We allow the end-to-end deadline to be larger than the period. This situation is quite common in real applications. For example, in multimedia streaming, the period at which video frames are generated and processed may be lower than the end-to-end deadline for delivering the frames to the user.

An important problem is to check the schedulability of the system, i.e. to test if all transactions will complete before their end-to-end deadline under worst-case conditions. In fixed priority systems, the *holistic analysis* [1], [2] consists in reducing the overall distributed schedulability problem into p single-node problems that can be solved using classical schedulability analysis. Each task is assigned a priority, and task parameters like offsets, jitters, response times are calculated so that the precedence constraints are automatically guaranteed. Since all schedulability problems depend on one another (i.e. the activation of an intermediate task, and hence its jitter, depends on the response time of the preceding task), the analysis is iterated until either a fixed-point solution is found or the set is deemed not schedulable. Similar techniques have been applied to EDF scheduling [3]–[6]. In this case, each task must be assigned an intermediate deadline instead of a fixed priority. Holistic analysis also allows to mix different schedulers on different nodes, as long as the designer is able to compute the worst-case response time of every task.

However, the holistic analysis is *global*, in the sense that it can be performed once the designer knows the parameters of all transactions. Moreover, any variation in one parameter (computation time, priority of an intermediate task,...) can influence the temporal behavior of all the system.

In a component-based approach, instead, it is desirable to perform the analysis in two steps: in the first step (local) we analyze each transaction in isolation, summarizing its temporal behavior with a (possibly small) set of temporal parameters. In the second step (integration), we must verify that the overall system is schedulable by only considering the temporal parameters derived in the first step.

Such an approach facilitates sensitivity analysis, increases robustness of the solution, allows to easily substitute one component with another, reduces the complexity of dynamic on-line admission control, etc.

In the local analysis it is not possible to use the response times of the tasks, because they depend on the presence of all other transactions. Therefore, in this paper we use the *slicing* approach [7]. Each task is assigned an *execution window*, and the execution windows of any two tasks of the same transaction are not overlapping. This can be done by assigning appropriate offsets and deadlines to every task.

Following the slicing method, under EDF the temporal characteristics of the transactions are summarized by a set of *demand bound functions* (dbfs) [8], one for each node. The integration analysis then consists on summing all the dbfs for every node, and check that the resulting function never exceeds the computational power of the node. In [9], we used such a methodology for periodic transactions, and we proposed a method for assigning intermediate deadlines to minimize a function of the dbfs.

However, two problems need to be solved to apply the slicing methodology to EDF. The first problem concerns with extending the analysis to sporadic transactions. In fact,

unlike single tasks scheduled on uniprocessors, in the case of end-to-end deadlines larger than the period the computation of the dbf of a sporadic transaction is not trivial.

The other problem concerns with the run-time support. When scheduling transactions with EDF, we must assign an activation offset and an absolute deadline to each job. These, in turn, depend on the activation of the transaction, which may happen on a different node. Moreover, if activations are sporadic, we cannot predict the activation window of future jobs. Therefore, it may seem that a precise global clock synchronization protocol is needed. Fortunately, the global clock synchronization can be avoided in fixed priority systems by using the Release Guard Protocol [10]. The idea is to release the constraint on separating the execution windows of jobs residing on different nodes, guaranteeing instead only the correct separation between two jobs on the same processor. The protocol is simple and effective, but it was conceived for fixed priority schedulers only.

A. Contributions of this paper.

In this paper, we present two original contributions. First, we propose an algorithm to correctly compute the dbfs of a sporadic transaction on each node, and prove its correctness. Second, we propose Algorithm IDSP (Implicit Deadline Synchronization Protocol) to assign absolute deadlines to jobs in an EDF scheduler. Our protocol guarantees that, under certain conditions, the run-time demand of the jobs on each node never exceeds the dbf computed off-line.

II. Related work

The use of the demand bound function was initially proposed by Baruah et al., for testing the schedulability of set of tasks scheduled by EDF on single processors [8]. This methodology is also known as "Processor Demand Criterion" [11]. The computation of the dbf was later extended to more complex task models, such as the generalized multiframe tasks [12]. Recently, Zhang and Burns [13] proposed a technique to reduce the number of points to check during analysis based on demand bound function.

The processor demand criterion has been extended to the analysis of distributed real-time transactions by Rahni et al. [6]. However, their methodology is still based on the holistic analysis: the activation time of a task is set equal to the finishing time of the previous task in the transaction.

In [7] authors proposed a methodology to analyze the schedulability of task graphs. The methodology also computes intermediate deadlines by using an heuristic approach, and it is based on the *slicing* approach: each task is assigned a slice that does not overlap with the slices of other tasks. Later [14] uses time slices to decouple the schedulability analysis of each node, reducing the complexity of the analysis. Such an approach improves the robustness of the schedule, and allows to analyze each transaction in isolation. We recently proposed a new algorithm for assigning intermediate task deadlines based on the slicing approach [9]. Our methods enables a component-based analysis.

Gantman et al. [15] presented a survey on synchronization protocols for real-time distributed systems. Among the many algorithms presented in the survey, the Release Guard Protocol (originally proposed in [10]) achieves a smaller average end-to-end response time, greatly reduces start-time jitter, and does not require a global clock synchronization. The protocol uses only local information regarding the minimum separation time between instances of the same task, and appropriately delays future instances so to guarantee that higher priority tasks do not interfere too much with lower priority tasks. However, the Release Guard Protocol only works with fixed priority schedulers, and assumes that the end-to-end deadline does not exceed the period. The protocol has been enhanced by Zhang et al. [16] to deal with sporadic transactions, again on fixed priority schedulers.

III. System model and notation

A distributed real-time application is modeled by a set of transactions $\{\mathcal{T}_1, \ldots, \mathcal{T}_m\}$. To simplify the presentation, since our work investigates each transaction in isolation, throughout the paper we drop the index of the transactions.

Transaction $\mathcal{T}$ is composed by a set of n tasks $\{\tau_1, \ldots, \tau_n\}$. Task τ_i, with $i > 1$, is activated upon the completion of the preceding one τ_{i-1} and it has a computation time C_i. The first task τ_1 of the ℓ^{th} instance of the transaction is activated at Φ^ℓ, that is called *absolute activation*. We denote by τ_i^ℓ the ℓ^{th} instance of the task τ_i. We consider sporadic transactions with minimum iterarrival time T. Hence we have

$$\Phi^\ell - \Phi^{\ell-1} \geq T. \tag{1}$$

To describe a possible scenario of activations for the sporadic transaction under analysis, we need to list the possible values of absolute activations Φ^ℓ. We label the instance of the transaction under analysis by 0. Moreover, we operate a time translation, so to set the activation of this transaction at time reference 0. Therefore, we set $\Phi^0 = 0$.

The *successive instances* will be denoted by positive indexes $\ell > 0$, and their absolute activations by $\Phi^1, \Phi^2, \ldots$ Similarly, the *previous instances* will be denoted by negative indexes $\ell < 0$, and their absolute activations by $\Phi^{-1}, \Phi^{-2}, \ldots$

The following vector represents the *sporadic activation pattern*:

$$\overline{\Phi} = (\Phi^{-k_0}, \ldots, \Phi^{k_1}) \tag{2}$$

where k_0 and k_1 depend on the number of instances we need to consider in the analysis (see Section V). Finally, Γ is the set of all possible sporadic activation patterns.

We remark that, similarly to what it happens in multiprocessor scheduling [17], activating the transactions as early as possible (i.e. periodically) **is not the worst-case** for the activation pattern. In Section V we show this by an example.

Each transaction $\mathcal{T}$ has an *end-to-end deadline* D that is the maximum tolerable time from the activation of the first task τ_1 to the completion of the last task τ_n. Since the analysis of the constrained deadline ($D \leq T$) is a straightforward extension of the classic analysis, throughout the paper we always assume $D > T$. In such a case, it may happen that a task is activated before its previous instance has completed. In this paper, we assume that the different activations of each task are served in a FIFO order.

The application is distributed across p processing nodes, and each task τ_i of the transaction $\mathcal{T}$ is mapped onto computational node $x_i \in \{1, \ldots, p\}$. Hence, we define $\mathcal{T}_k = \{\tau_i \in \mathcal{T} : x_i = k\}$ as the subset of tasks in $\mathcal{T}$ mapped onto node k and n_k as the cardinality of $\mathcal{T}_k$.

The delay due to network communication can be easily taken into account by considering the network as a special processing node, and messages as tasks. The methodology presented in this paper is valid also when different scheduling policies are used on the processing nodes. However, to simplify the presentation, in this paper we make two assumptions: we neglect the delay due to network communication (for example, restricting to a multiprocessor system with shared memory); and we assume EDF as the only scheduling algorithm in the system. A more general investigation will be presented in a future work.

Each task is assigned an *intermediate deadline* $\overline{D}_i$, that is the interval of time between the activation of the transaction and the absolute deadline of the task. Hence, using the notation introduced so far, the absolute deadline of the ℓ^{th} instance of τ_i, is

$$d_i^\ell = \Phi^\ell + \overline{D}_i. \tag{3}$$

We enforce the precedence relationship between tasks by the slicing technique [7]: for each task we set the *activation offset* ϕ_i, relative to the activation of the transaction Φ^ℓ, equal to the intermediate deadline of the preceding one:

$$\phi_1 = 0, \quad \phi_i = \overline{D}_{i-1} \quad i = 2, \ldots, n \tag{4}$$

Clearly, the task absolute activation is

$$a_i^\ell = \Phi^\ell + \phi_i. \tag{5}$$

Moreover, we define the task *relative deadline* D_i as

$$D_i \stackrel{\text{def}}{=} \overline{D}_i - \phi_i.$$

The relationship between activation offsets and relative deadlines is depicted in Figure 1. Clearly,

$$\sum_{i=1}^{n} D_i = D \tag{6}$$

The values of $T, \Phi^\ell, D, C_i, \overline{D}_i, D_i, \phi_i$ are all real numbers. Finally, we use the notation $(\cdot)_0 \stackrel{\text{def}}{=} \max\{0, \cdot\}$.

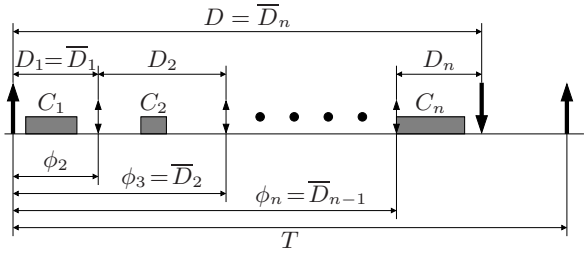

Figure 1: Notation for tasks.

IV. Periodic demand bound function

First, we recall the concept of demand bound function for a transaction that is strictly periodic (i.e. $\forall \ell,\ \Phi^\ell = \ell T$). Then, in the next section we extend the demand bound function to the sporadic case.

The computational requirement of the subset $\mathcal{T}_k$ of tasks allocated on node k is modeled by its *demand bound function* (dbf).

Definition 1: The *demand function* on node k, denoted by $\mathsf{df}_k(t_0, t_1)$, is the total computation time of all the instances of the tasks in $\mathcal{T}_k$, having activation time and deadline within $[t_0, t_1]$.

For periodic transaction, the demand function can be computed as follows [8]:

$$\mathsf{df}_k(t_0, t_1) \stackrel{\text{def}}{=} \sum_{\tau_i \in \mathcal{T}_k} \left(\left\lfloor \frac{t_1 - \overline{D}_i}{T} \right\rfloor - \left\lceil \frac{t_0 - \phi_i}{T} \right\rceil + 1 \right)_0 C_i \tag{7}$$

As suggested by Rahni et al. [6], the overall *demand bound function* of $\mathcal{T}_k$ in an interval of length t, is defined as:

$$\mathsf{dbf}_k(t) \stackrel{\text{def}}{=} \max_{t_0} \mathsf{df}_k(t_0, t_0 + t) \tag{8}$$

A necessary and sufficient schedulability test for non-concrete transactions (i.e. periodic transactions with free initial offset), scheduled by EDF consists in checking that the demand never exceeds the length of the interval on every processor

$$\forall k = 1, \ldots, p \ \ \forall t > 0 \ \ \sum_{\mathcal{T}} \mathsf{dbf}_k(\mathcal{T}, t) \leq t \tag{9}$$

where the sum is made over all the transactions in the system, and $\mathsf{dbf}_k(\mathcal{T}, t)$ denotes the demand bound function of $\mathcal{T}$ on node k. In this case, first the dbf is computed for each transaction and for each node (applying the max operator), and then we sum all the dbf together to compute the overall computational requirement on node k.

In Figure 2 we illustrate the definitions introduced in this section by an example. Consider a transaction whose parameters are: period $T = 5$, end-to-end deadline $D = 8$, task deadlines $D_1 = 2$ and $D_2 = 6$, computation time $C_1 = 1$ and $C_2 = 3$. Both tasks are assigned to a single node. In the lower part of Figure 2, we show three consecutive instances of the transaction on three different

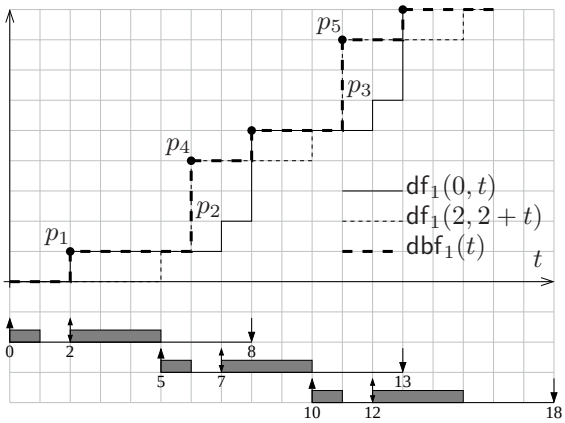

Figure 2: Example of demand bound function.

lines. In the upper part, we show the values of 3 functions: the demand in $[0,t]$; the demand in $[2,2+t]$; and the demand bound function. We represent the points where the dbf has a step by a thick dot. The steps are tightly related to task deadlines. For example in the figure, the points p_1, p_2, p_3 depend on the deadlines of task τ_1, while the points p_4, p_5 depend on the deadlines of τ_2.

To compute the dbf of a periodic transaction, it is sufficient to consider the value of the demand functions obtained on the intervals that start with the activation of a task, as shown in [6]. Also, the dbf has a periodic pattern: its value for a generic large interval t can be computed as $\mathsf{dbf}(t')+jC$, where $C=\sum_{\tau_i\in\mathcal{T}_k} C_i$, $j\geq 0$ and $t'=t-jT$ (see Section 4.1 in [6]).

V. Sporadic demand bound function

Unfortunately, for sporadic transactions, the worst case does not occur with periodic activations. Consider the following transaction with 3 tasks on 2 processors. The transaction has period $T=5$ and end-to-end deadline $D=12$. The task parameters are reported in Table I.

Task	C_i	proc.	D_i
τ_1	1	0	3
τ_2	3	1	4
τ_3	3	0	5

Table I: Parameters for the example

In Figure 3, we show two possible activation patterns. The first one corresponds to a periodic activation ($\Phi^1=T$): in this case, it is easy to see that the maximum demand on processor 0 in any interval of length 5 is at most 3 units of computation.

In the second activation pattern we delay the activation of the second instance by 2 units of time ($\Phi^1=T+2$). As a consequence, the demand in interval $[7,12]$ becomes 4 units of time, because one extra instance of τ_1 enters the interval. Thus, delaying an instance can increase the demand.

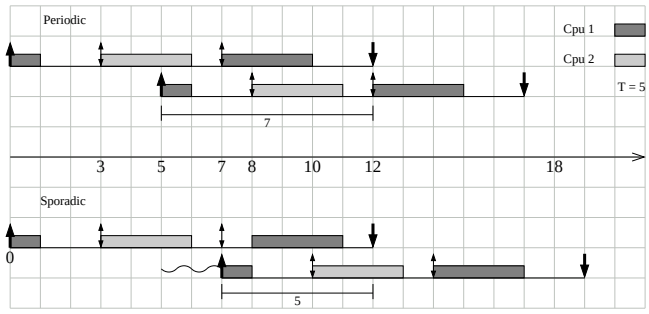

Figure 3: Example of sporadic transaction.

Hence, the analysis based on the classic periodic demand bound function is not applicable if transactions are sporadic. One of the contributions of this paper is to extend the demand bound function to sporadic transactions.

A job τ_i^ℓ in $\mathcal{T}_k$, runs inside interval $[t_0,t_1]$ if its absolute deadline d_i^ℓ is not smaller than t_1

$$t_1 \geq d_i^\ell = \overline{D}_i + \Phi^\ell \tag{10}$$

and its activation is not earlier than t_0

$$t_0 \leq a_i^\ell = \phi_i + \Phi^\ell \tag{11}$$

By introducing the function

$$\mathsf{step}(x) = \begin{cases} 0 & \text{if } x<0 \\ 1 & \text{if } x\geq 0 \end{cases} \tag{12}$$

we can define the following binary-valued function

$$\mathsf{jobIn}_i^\ell(t_0,t_1) \stackrel{\text{def}}{=} \mathsf{step}(t_1-\overline{D}_i-\Phi^\ell)\cdot\mathsf{step}(\phi_i+\Phi^\ell-t_0) \tag{13}$$

that returns 1 if the job τ_i^ℓ has both activation and deadline in $[t_0,t_1]$, and it returns 0 otherwise.

Hence, the demand of all the tasks belonging to the transaction $\mathcal{T}_k$ can be expressed as:

$$\mathsf{df}_k(t_0,t_1) \stackrel{\text{def}}{=} \max_{\overline{\Phi}\in\Gamma} \sum_{\ell=-k_0}^{k_1} \sum_{\tau_i\in\mathcal{T}_k} \mathsf{jobIn}_i^\ell(t_0,t_1) C_i \tag{14}$$

where k_0 and k_1 are indexes of transaction instances (later determined in Eq. (18)) that may have an effect on the demand in $[t_0,t_1]$.

The sum on all the transaction instances ℓ can be split in three parts: the first part is the sum over the indexes corresponding to the *past instances* (from $-k_0$ to -1); the second part is the *current instance* (with $\ell=0$), and the third part is the sum over the *future instances* (from 1 to

k_1). Hence Equation (14) becomes

$$\begin{aligned}\mathsf{df}_k(t_0,t_1) &= \max_{\overline{\Phi}\in\Gamma}\Big\{\sum_{\tau_i}\mathsf{jobln}_i^0(t_0,t_1)C_i + \sum_{\ell=-k_0}^{-1}\ldots + \sum_{\ell=1}^{k_1}\ldots\Big\} \\ &= \sum_{\tau_i}\ldots + \max_{(\Phi^{-k_0},\ldots,\Phi^{-1})\in\Gamma^-}\sum_{\ell=-k_0}^{-1}\sum_{\tau_i}\ldots \\ &\quad + \max_{(\Phi^1,\ldots,\Phi^{k_1})\in\Gamma^+}\sum_{\ell=1}^{k_1}\sum_{\tau_i}\ldots \qquad (15)\end{aligned}$$

where Γ^- and Γ^+ are the sets of the possible activation patterns of the past and the future instances respectively. Although Eq. (15) is apparently more complex than Eq. (14), it will be more useful for our purposes because it has the advantage of decoupling the dependecy on past and future instances (see Sections V-A and V-B).

Finally, as for the periodic dbf, the sporadic dbf is the maximum among all the sporadic demand functions computed on intervals with the same length:

$$\mathsf{dbf}_k(t) \stackrel{\text{def}}{=} \max_{t_0}\mathsf{df}_k(t_0,t_0+t) \qquad (16)$$

Figure 4 shows that, for the same parameters of Table I, the sporadic dbf computed from Eq. (16) is larger than the periodic dbf (Eq. (8)).

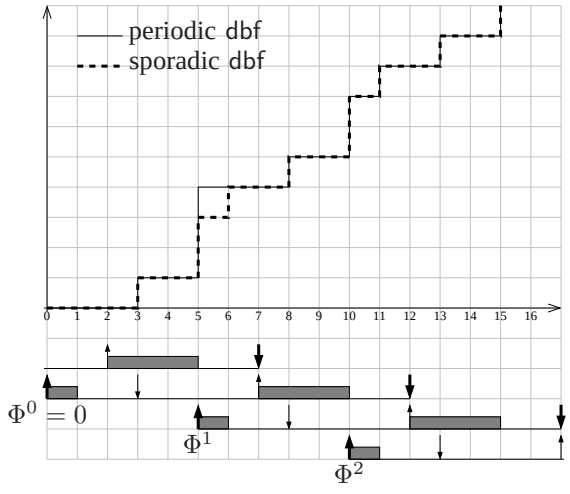

Figure 4: An example of sporadic dbf.

Equation (16) is a nice and compact expression of the dbf. It is however unclear how such a dbf should be practically computed: how many instances ℓ of the transaction should we consider in the sum of Eq. (15)? How many values of t_0 should we consider in the maximum of Eq. (16)?

We follow a strategy similar to the one used for computing the dbf of periodic transaction. The strategy consists in the algorithm reported in Figure 5. First (at line 2), we compute the list intervalSet of all the *significant* intervals $[t_0,t_1]$, i.e. the intervals such that $\forall\varepsilon>0$ both the demands $\mathsf{df}_k(t_0+\varepsilon,t_1)$ and $\mathsf{df}_k(t_0,t_1-\varepsilon)$ are **strictly less** than $\mathsf{df}_k(t_0,t_1)$. In

```
1: intervalSet ← ∅                          ▷ initialize the set of intervals
2: STOREINTERVALS                           ▷ store intervals in intervalSet
3: sort intervalSet by increasing t1 − t0
4: lastDBFval ← 0
5: for each [t0, t1] ∈ intervalSet do ▷ loop on all intervals
6:     Γ−, Γ+ ← ∅   ▷ init sets of past and future patterns
7:     SPANPATTERNS(t0, t1)                 ▷ store all patterns
8:     curDBFval ← df_k(t1, t0)                          ▷ Eq. (15)
9:     if curDBFval > lastDBFval then                    ▷ Eq. (16)
10:        store the point (t1 − t0, curDBFval)
11:    else
12:        do nothing (dominated by previous point)
13:    end if
14: end for
```

Figure 5: Algorithm for computing the dbf.

Section V-A we describe the procedure STOREINTERVALS for performing this step. After sorting the intervals $[t_0,t_1]$ in intervalSet by increasing t_1-t_0 (at line 3), we search for the activation pattern $\overline{\Phi}$ that maximizes the demand in $[t_0,t_1]$. In Section V-B we describe the procedure SPANPATTERNS that computes the set Γ of all possible activation patterns.

A. Enumerating the intervals

The first stage requires to enumerate all the intervals $[t_0,t_1]$. The pseudocode of this stage is reported in Figure 6. First, we claim that t_0 must coincide with the activation of some job. In fact, if this does not happen then we could increase t_0 achieving a shorter interval with the same demand. Hence we set t_0 equal to the activation of the job τ_i^0, i.e. t_0 spans on $\{\phi_i : \tau_i \in \mathcal{T}_k\}$ (see line 4 of the algorithm). Notice that, without loss of generality, we label by 0 the transaction instance which this job belongs to.

Regarding the possible values of t_1, it is easy to see that it is sufficient to test only the absolute deadlines d_j^h. In fact if $t_1 = d_j^h$ for some task $\tau_j \in \mathcal{T}_k$ and some transaction instance h, then a reduction of t_1 by an arbitrary small amount ε will decrease the demand df by at least C_j. However, the main difficulty here is that the absolute activations are not fixed, hence we do not know where the deadlines are until we fix the sporadic activation pattern $\overline{\Phi}$.

First, we list the values of t_1 associated with the absolute deadlines of the instance 0 (see lines 5–9). Then we invoke the recursive procedures FUTUREDEADLINE and PASTDEADLINE that list the absolute deadlines of the future and past instances, respectively.

These two procedures explore the possible activation patterns $\overline{\Phi}$ such that task activations are aligned with t_0. For each pattern the values of absolute deadlines are recorded as candidate values for t_1.

We conclude the section by showing that, after a transient that is long at most $D+T$, the dbf becomes periodic.

```
1: procedure STOREINTERVALS
2:     intervalSet ← ∅                                   ▷ initialize
3:     for each τ_i ∈ T_k do                             ▷ loop on t_0
4:         t_0 ← φ_i
5:         for τ_j ∈ T_k do
6:             if D̄_j > t_0 then
7:                 store [t_0, D̄_j] in intervalSet
8:             end if
9:         end for
10:        FUTUREDEADLINE(t_0, 1, 0)
11:        k_0 ← ⌈(D − t_0)/T⌉ − 1
12:        PASTDEADLINE(t_0, −k_0, −2k_0(D + T))
13:    end for
14: end procedure

15: procedure FUTUREDEADLINE(t_0, ℓ, Φ^{ℓ−1})
16:    for all Φ^ℓ ∈ {Φ^{ℓ−1} + T} ∪ {t_0 − φ_i : t_0 − φ_i > Φ^{ℓ−1} + T, τ_i ∈ T_k} do
17:        for each τ_i ∈ T_k do
18:            t_1 = Φ^ℓ + D̄_i
19:            if t_1 > t_0 then
20:                store [t_0, t_1] in intervalSet
21:            end if
22:        end for
23:        if ℓ < ⌈(D + 2T)/T⌉ then
24:            FUTUREDEADLINE(t_0, ℓ + 1, Φ^ℓ)
25:        end if
26:    end for
27: end procedure

28: procedure PASTDEADLINE(t_0, ℓ, Φ^{ℓ+1})
29:    for all Φ^ℓ ∈ {ℓT, Φ^{ℓ+1} + T} ∪ {t_0 − φ_i : Φ^{ℓ+1} + T < t_0 − φ_i < ℓT, τ_i ∈ T_k} do
30:        for each τ_i ∈ T_k do
31:            t_1 = Φ^ℓ + D̄_i
32:            if t_1 > t_0 then
33:                store [t_0, t_1] in intervalSet
34:            end if
35:        end for
36:        if ℓ < −1 then
37:            PASTDEADLINE(t_0, ℓ + 1, Φ^ℓ)
38:        end if
39:    end for
40: end procedure
```

Figure 6: Algorithm for enumerating intervals.

Lemma 1: For large values of t, the $\mathsf{dbf}(t)$ has a periodic pattern. More formally:

$$\forall t > D + T \quad \mathsf{dbf}_k(t + T) = \mathsf{dbf}_k(t) + C^k.$$

where $C^k = \sum_{\tau_i \in \mathcal{T}_k} C_i$.

Proof: Let t_0 and $\overline{\Phi}$ be the instant and activation pattern that give the value of $\mathsf{dbf}_k(t)$ in Equations (16) and (15) respectively, and let us set $t_1 = t_0 + t$.

We identify with ℓ the first transaction instance with activation $\Phi^\ell > t_0$, hence $\Phi^{\ell-1} \leq t_0$. Since we are in the worst case and $\Phi^\ell > t_0$, then

$$\forall h \geq \ell \quad \Phi^h - \Phi^{h-1} = T \tag{17}$$

otherwise, we could anticipate all Φ^h with $h \geq \ell$ without removing any job from the interval. On the contrary, the deadline of a job may enter the interval, and the worst-case activation pattern cannot be $\overline{\Phi}$ anymore, causing a contradiction.

From (17) and the definition of ℓ, we notice that the instance ℓ of the transaction ends earlier than t_1. Clearly this is also true for all instances before ℓ. Formally

$$\Phi^{\ell-1} \leq t_0 \quad \Rightarrow \quad \Phi^\ell \leq t_0 + T$$
$$\Phi^\ell + D \leq t_0 + T + D < t_1.$$

From (17), it follows that any interval of length T starting after $\Phi^\ell + D$ contains exactly one activation and one deadline of each task. Hence the demand generated in the interval $[t_0, t_1 + T]$ increases by one job for all tasks in $\mathcal{T}_k$, i.e. C^k.

Suppose by absurd that $\mathsf{dbf}_k(t + T) > \mathsf{dbf}_k(t) + C^k$. Then, it exists an interval $[t'_0, t'_0 + t + T]$ with demand larger than $\mathsf{dbf}_k(t) + C^k$. Let $\overline{\Phi}'$ be its activation pattern, and let us call ℓ' the first instance with $\Phi^{\ell'} > t'_0$. Followong the same reasoning as above, the demand in $[t'_0, t'_0 + t]$ decreases by C^k. However, this is absurd because we obtain a new interval with the same length t but with demand higher than in $[t_0, t_0 + t]$. ■

Since, thanks to the lemma, the transient part of the dbf lasts for no longer than $D + T$ and the periodic part is long T, it is possible to compute the dbf only for lengths of intervals less than $D + 2T$.

Now we present an algorithm for computing the activation patterns that determines the maximum demand in a given interval $[t_0, t_1]$.

B. Algorithm for enumerating the activation patterns

In this section we explain the procedure SPANPATTERNS(t_0, t_1) (see line 7 of the algorithm in Figure 5) that checks all possible sporadic activation patterns of past and future instances that may have an impact on the interval $[t_0, t_1]$. Therefore, we are interested only in transaction instances that may overlap with the interval $[t_0, t_1]$. The indexes of these transactions are from $-k_0$ to k_1, where

$$k_0 = \left\lceil \frac{D - t_0}{T} \right\rceil - 1 \quad k_1 = \left\lceil \frac{t_1}{T} \right\rceil - 1. \tag{18}$$

Hence the sum of transactions instances of Eq. (15) has to be made for $\ell = -k_0, \ldots, k_1$.

For the example of Table I (see also Figure 4 for a timeline representation of the instances), if we set $t_0 = \phi_1 = 0$ and

$t_1 = 13$, we find $k_0 = 2$ and $k_1 = 2$, meaning that in the analysis of the demand in the interval $[5, 12]$ we consider the instances $-2(= -k_0), -1, 0, 1, 2(= k_1)$ of the transaction.

In the exploration of the activation patterns we distinguish between future instances (with index $\ell > 0$) and past instances (with index $\ell < 0$). The guideline for the exploration of absolute activations of future instances is to align some task activation $a_i^\ell = \Phi^\ell + \phi_i$ with t_0. This is possible by setting

$$\Phi^\ell = t_0 - \phi_i. \tag{19}$$

However, this is a valid absolute activation only if it respects the contraints of being a sporadic transaction with minimum interarrival T, that is

$$\Phi^\ell \geq \Phi^{\ell-1} + T. \tag{20}$$

This condition introduces a recurrent dependency between all the values $\Phi^0, \Phi^1, \Phi^2, \ldots, \Phi^{k_1}$. The procedure COMPUTEFUTURE for testing future istances is reported in Figure 6.

The same rationale is applied to past instances (the ones with index $\ell < 0$). In this case however, we aim at finding the absolute activation Φ^ℓ such that some absolute deadline is aligned with t_1. The full algorithm that explores the activation patterns is reported in Figure 7.

In the example of Figure 4, if we assume $t_0 = 0$ then Φ^1 should be tested with the values of $5(= T)$. Instead, if $t_0 = \phi_3 = 7$ then Φ^1 is checked both when it is 5 and when it is $t_0 - \phi_1 = 7$, meaning that we align the activation of the instance 1 with the offset $\phi_3 = t_0 = 7$.

C. Complexity analysis

We start by analysing the complexity of procedure STOREINTERVALS. The outer loop (line 3) is executed n_k times. After adding the intervals for instance 0, procedures FUTUREDEADLINE and PASTDEADLINE are invoked.

Procedure FUTUREDEADLINE explores a number of instances at most equal to $k_2 = \lceil \frac{D+2T}{T} \rceil - 1$. Of this, the first $\lfloor \frac{t_0}{T} \rfloor$ instances may vary their activation time, while for the successive ones, the worst-case corresponds to interarrival times equal to T. The number of possible combinations of activations (line 16) is then $n_k^{\lfloor \frac{t_0}{T} \rfloor}$. For each combination, $n_k k_2$ deadlines are generated.

Procedure PASTDEADLINE is very similar. The number of instances is k_0 (see Eq. (18)). The maximum number of elements generated for each combination of past activations is $n_k k_0$. Finally, the maximum number of combinations (line 29) is $(n_k + 2)^{k_0}$.

Each generated interval must be inserted in a ordered list, an operation that takes logarithmic time in the size of the list. The size of the list at the end of the procedure is:

$$s = k_2 n_k^{\lfloor \frac{t_0}{T} \rfloor + 1} + n_k k_0 (n_k + 2)^{k_0}$$

and the complexity is $O(\sum_{i=1}^{s} \log(i))$.

```
1: procedure SPANPATTERNS(t_0, t_1)
2:     k_1 = ⌈(t_1)/T⌉ − 1                                   ▷ see Eq. (18)
3:     COMPUTEFUTURE(1, (0, ..., 0) [k_1])
4:     k_0 = ⌈(D − t_0)/T⌉ − 1                               ▷ see Eq. (18)
5:     COMPUTEPAST(−1, (0, ..., 0) [k_0])
6: end procedure

7: procedure COMPUTEFUTURE(ℓ, (Φ^1, ..., Φ^{k_1}))
8:     if ℓ > k_1 then                                       ▷ the exit condition
9:         store (Φ^1, ..., Φ^{k_1}) in Γ^+                  ▷ Γ^+ is global
10:    else
11:        Φ^0 ← 0
12:        for all Φ^ℓ ∈ {Φ^{ℓ−1} + T} ∪ {t_0 − φ_i : t_0 − φ_i >
       Φ^{ℓ−1} + T, τ_i ∈ T_k} do
13:            COMPUTEFUTURE(ℓ + 1, (Φ^1, ..., Φ^{k_1}));
14:        end for
15:    end if
16: end procedure

17: procedure COMPUTEPAST(ℓ, (Φ^{−k_0}, ..., Φ^{−1}))
18:    if ℓ < −k_0 then
19:        store (Φ^{−k_0}, ..., Φ^{−1}) in Γ^−
20:    else
21:        Φ^0 ← 0
22:        for all Φ^ℓ ∈ {Φ^{ℓ+1} − T} ∪ {t_1 − D̄_i : t_1 − D̄_i <
       Φ^{ℓ+1} − T, τ_i ∈ T_k} do
23:            COMPUTEPAST(ℓ − 1, (Φ^{−k_0}, ..., Φ^{−1}));
24:        end for
25:    end if
26: end procedure
```

Figure 7: Algorithm for generating Γ^- and Γ^+.

Notice that, while generating the the values of t_1, it is quite common to obtain many times always the same values. In average, we expect that the final size of the list is much smaller than its upper bound s.

Regarding procedure SPANPATTERNS, we apply a similar reasoning. We address separately future and past instances. Procedure COMPUTEFUTURE builds a tree in which at level 1 sets the value of Φ^1, at level 2 sets the value of Φ^2, and so on. There will be k_1 levels. Each node has at most $n_k + 1$ children. Hence, the number of leafs of such a tree is $(n_k + 1)^{k_1}$. Each leaf corresponds to a different value of $(\Phi^1, \ldots, \Phi^{k_1})$. A similar tree can be built for past instances. Thus the complexity of enumerating all activation patterns is

$$O((n_k + 1)^{k_0} + (n_k + 1)^{k_1}).$$

Finally, the complexity of computing the whole dbf is

$$O\left(\sum_{i=1}^{s}\log(i)+sn_k\left((n_k+1)^{k_0}+(n_k+1)^{k_1}\right)\right).$$

We are aware that the proposed algorithm is very complex. Most of the complexity lies in the sporadicity of the transaction that requires to check all possible scenarios. In this paper, we focused on the exact analysis regardless of its complexity. We leave to future investigations the development of simplified algorithms as well as Fully Polynomial Time Approximation Schemes (FPTAS).

VI. Implicit Deadline Synchronization Protocol

In order to implement a system that uses the transaction model presented in this paper, the scheduler on each node must set the activation times and the deadlines of the jobs so that, if all tasks execute for less than their worst-case execution times C_i, and the difference between two consecutive transaction instances is greater than the minimum interarrival time T,

1) every transaction always respects its end-to-end deadline;
2) in every interval, the sporadic demand of every transaction is always less than or equal to the sporadic demand computed off-line.

At first glance, it may seem that we need a strict clock synchronization protocol to guarantee that the activations and the deadlines are correctly computed. In fact, in a distributed system the timing information are obtained on each node by reading local timer hardware interfaces, and different timers can have different offsets and different speeds. Therefore, a global clock synchronization protocol is often used to synchronize the different timing views to the one taken as reference.

In this paper, instead, we show that it is possible to remove the need for a common time reference. However, we still assume that there is no drift among the timers of different nodes. We plan to extend our analysis to distributed systems with clock drifts in a future work.

Our idea is similar to the Release Guard Protocol (RGP) [10] by Sun and Liu. RGP has been thought for reducing the start-time jitter and guaranteeing a minimum separation time between two task activations in a fixed priority system. RGP works by delaying the activations of a task so that the distance between two consecutive instances is never less than the minimum interarrival time.

The algorithm we propose uses a similar idea to impose a minimum distance between deadlines.

We start by observing that, once we assign the correct deadline to a job, we can also let it start before its offset ϕ_i.

Lemma 2: Anticipating the activation of a task without modifying its absolute deadline does not increase the sporadic dbf.

Proof: Let $\phi_i^\ell = \Phi^\ell + \phi_i$ be the activation of job τ_i^ℓ, d_i^ℓ its absolute deadline and let $a_i^\ell < \phi_i^\ell$ be its actual starting time. Then, $d_i^\ell - a_i^\ell > D_i$.

Let $t > D_i$, and let t_0 be such that $\mathsf{df}(t_0, t_0+t)$ is maximal. If the interval contains ϕ_i^ℓ and d_i^ℓ but not a_i^ℓ, then the dbf may decrease. In all other cases, the dbf in t remains the same. ■

While for the purpose of the analysis we impose that the activation of a task is equal to the deadline of the previous task, thanks to Lemma 2 at run-time we can activate a job at the completion of the previous job, as long as the deadlines are correctly set.

The second observation is that the dbfs of different nodes are not related to each other. If we restrict our attention to a node k, as long as the system is schedulable and all tasks meet their deadlines, we can use the activation of some task in $\mathcal{T}_k$ as a reference time to compute all other parameters.

Before describing the protocol, we need an additional definition.

Definition 2: We define as *precedence set* $\mathcal{P}_i^\ell$ of job τ_i^ℓ the set of all jobs τ_j^h of tasks $\tau_j \in \mathcal{T}_k$, with $h \in \{\ell, \ell-1, \ell-2, \dots \ell-k_0\}$, that have absolute deadline less than d_i^ℓ under all possible activation patterns. The number k_0 of instances to consider is:

$$k_0 = \left\lceil \frac{D}{T} \right\rceil - 1.$$

The precedence sets impose a partial order on the set of jobs belonging to previous instances. In practice, the deadline d_i^ℓ of job τ_i^ℓ must necessarily follow all the deadlines of jobs $\mathcal{P}_i^\ell$ under all possible activation patterns.

A stronger property is the following.

Lemma 3: At run time, the distance between d_i^ℓ and any of the jobs in $\mathcal{P}_i^\ell$ cannot be less than

$$d_i^\ell \geq d_j^h + (\ell - h)T + \overline{D}_i - \overline{D}_j \tag{21}$$

Proof: Follows directly from the definitions. ■

A. The protocol

We now present the IDSP algorithm to compute the absolute deadline of a job τ_i^ℓ at the instant of its activation a_i^ℓ. The protocol consists of three simple rules.

Rule 1 The separation between activation and deadline of τ_i^ℓ must always be greater than its relative deadline:

$$d_i^\ell \geq a_i^\ell + D_i \tag{22}$$

Rule 2 The second rule mandates a minimum separation between the deadlines of the jobs of the same task:

$$d_i^\ell \geq d_i^{\ell-1} + T. \tag{23}$$

Rule 3 The distance between d_i^ℓ and any job in $\mathcal{P}_i^\ell$ must not be less than the minimum possible distance as computed off-line. In formula:

$$\forall \tau_j^s \in \mathcal{P}_i^\ell, \quad d_i^\ell \geq d_j^s + (\ell - s)T + \overline{D}_i - \overline{D}_j \tag{24}$$

It may happen that a job τ_i^ℓ is activated before a job in its precedence set, due to the fact that the end-to-end deadline can be larger than period and previous jobs complete much earlier than their deadline. In such a case, the job is suspended and its deadline cannot be computed until we have computed the deadlines of all the jobs in its precedence set.

From the previous rules, the job deadline d_i^ℓ can simply be computed as the maximum among the RHS the three inequalities. Notice that we only use parameters that are local to each node (a_i^ℓ, d_i^ℓ) or statically known (D_i, T, $\overline{D}_i$, $\mathcal{P}_i^\ell$, and the intermediate deadline $\overline{D}_j$ for each job $\tau_j^h \in \mathcal{P}_i^\ell$).

B. Computing the precedence set

Set $\mathcal{P}_i^\ell$ can be very large. We will now show that it is possible to only compute a smaller subset $\mathcal{P}_i^{\ell*}$ that contains at most one job per past instance. First of all, let us show how to compute such reduced subset:

- Set $\mathcal{P}_i^{\ell*}$ contains at most one job per each instance $\ell, \ell-1, \dots \ell-k_0$. Initially, $\mathcal{P}_i^{\ell*}$ is empty.
- Considering instance $h = \ell$, we only need to know the task $\tau_j \in \mathcal{T}_k$ that immediately precedes τ_i in the transaction. Then, τ_j^ℓ is added to $\mathcal{P}_i^{\ell*}$. If τ_i has no preceding task in $\mathcal{T}_k$, we skip this step.
- Then, we enter a cycle in which we compute the job for instance $h \in \ell-1, \ell-2, \dots, \ell-k_0$. Let d_m be the greatest deadline among the jobs already in $\mathcal{P}_i^{\ell*}$. Consider the latest job τ_j^h that has absolute deadline in interval (d_m, d_i^ℓ), under the assumption of periodic distance between all instances from h to ℓ. If such a job exists, its deadline d_j^h is added to $\mathcal{P}_i^{\ell*}$. Else, we skip to the next instance.
- We iterate until instance $k = \ell - k_0$.

An example of the procedure is shown in Figure 8. Consider a transaction having 6 tasks, with period $T = 10$ and end-to-end deadline $D = 25$. Hence, we need to consider $k_0 = 2$ instances. The intermediate deadlines are respectively, $3, 6, 10, 14, 21, 25$. We assume that tasks τ_1, τ_3, τ_5, are allocated on processor $k = 1$, while task τ_2, τ_4, τ_6 are allocated on processor $k = 2$. We want to compute the precedence set of job τ_2^ℓ (the second task in the bottom line of Figure 8). By setting all preceding activations at distance equal to T, we have the activation pattern shown in the figure. The precedence set of τ_2^ℓ contains: 1) no job of instance ℓ, because there is not task preceding τ_2 on processor 2; 2) job $\tau_4^{\ell-1}$; 3) job $\tau_6^{\ell-2}$.

According to rule 3, we must check that $d_2^\ell \geq d_4^{\ell-1} + 2$ and $d_2^\ell \geq d_6^{\ell-2} + 1$. Therefore, d_2^ℓ can be computed as:

$$d_2^\ell = \max\{a_2^\ell + 25, d_2^{\ell-1} + 10, d_4^{\ell-1} + 2, d_6^{\ell-2} + 1\}.$$

In general, the maximum number of elements to maximize is upper bounded by $\min\{k_0, n_k - 1\} + 2$, so it depends on the ratio between end-to-end deadline and period, and in no case is greater than $n_k + 1$.

C. Proof of correctness

Rule 3 mandates that all the deadlines in the precedence set $\mathcal{P}_i^\ell$ must be computed before we can compute deadline d_i^ℓ. The following lemma proves that at run-time it is sufficient to only consider $\mathcal{P}_i^{\ell*}$.

Lemma 4: If all the jobs in $\mathcal{P}_i^{\ell*}$ have been assigned a deadline at run-time, then all the jobs in $\mathcal{P}_i^\ell$ have been assigned a deadline.

Proof: By contradiction. Suppose that a job $\tau_j^h \in \mathcal{P}_i^\ell - \mathcal{P}_i\ell*$ has not been assigned a deadline, and let τ_i^ℓ be the first job for which this happens at run-time.

Since $d_j^h < d_i^\ell$ but τ_j^h does not belong to $\mathcal{P}_i\ell*$, it must exist a job $\tau_z^s \in \mathcal{P}_i\ell*$ such that $h \leq s \leq \ell$ and $d_j^h < d_z^s < d_i^\ell$. Then, $\tau_j^h \in \mathcal{P}_z^s$. Since τ_z^s has been assigned a deadline at run-time, according to rule 3, τ_j^h must have been assigned a deadline, against the hypothesis. ■

The following lemma proves that the absolute deadlines assigned by algorithm IDSP never exceed the absolute deadlines assigned by an algorithm that uses global time.

Lemma 5: Let a^ℓ be the activation of the ℓ-th instance of transaction $\mathcal{T}$. Under the assumption that the transaction is schedulable, the absolute deadline d_i^ℓ of every job τ_i^ℓ, computed dynamically using Equations (22)–(24), is never larger than $a^\ell + \overline{D}_i$.

Proof: The complete proof has been removed for space constraints. We just report here the intuition behind it. The proof is by induction. We first show that the lemma is true for the first job of the first instance. This is evident because rule 1 is the only one that can be applied. Then, we prove the induction step: if the lemma is true for all jobs in $\mathcal{P}_i^\ell$, then it is true for τ_i^ℓ. This can be proved by showing how to compute d_i^ℓ starting from rule 2 and rule 3. ■

Lemma 5 guarantees that, if the transaction is locally schedulable on each node, then no task misses the deadlines that a global algorithm would have assigned on each node.

Now we want to prove that the protocol assigns deadlines so that the sporadic dbf, as computed in Section V-B and V-A, is always respected.

Lemma 6: Let τ_j^h be any job in $\mathcal{P}_i^\ell$. Under the assumption that the transaction is schedulable and all deadlines are assigned using Equations (22)–(24), the distance between d_j^h and d_i^ℓ is never smaller than the distance as computed in Equation (21).

Proof: For $\tau_j^h \in \mathcal{P}_i^{\ell*}$, the lemma follows directly from Rule 3. For the other jobs, it is easy to see that we can apply a similar reasoning to the one in Lemma 4 to derive the thesis. ■

Now the main theorem.

Theorem 1: Under the assumption that all tasks execute for less than their WCET, and that for any interval the sum of the sporadic dbfs computed off-line never exceeds the length of the interval, then the dbf computed on-line by IDSP is always less or equal to the sporadic dbf.

Proof: The proof has been removed for space constraints. We report here only the intuition behind it.

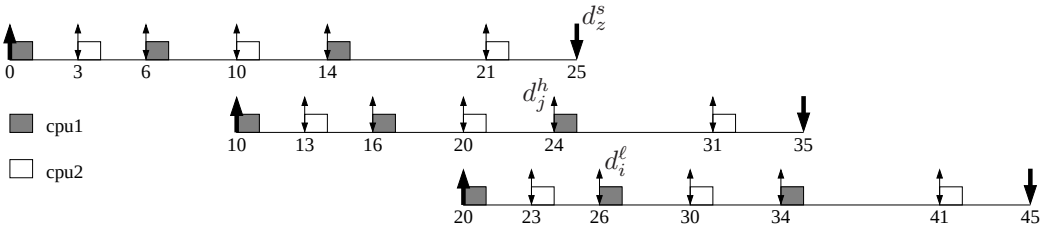

Figure 8: Example of computation of the precedence set.

The proof is by contradiction. Let $[t_0, t_1]$ be the first and smaller interval in which the demand computed on-line exceeds the sporadic dbf. The, t_0 must be coincident with a task activation and t_1 with a deadline, let it be d_i^ℓ. According to Lemma 6, the deadlines in $[t_0, t_1]$ are separated by no less than their worst-case distance as computed by Equation (21). Then, we show that by moving deadline and activations in a conservative way (i.e. without decreasing the on-line demand), we reach one of the situations enumerated by the algorithms in Figures 7 and 6. Hence we obtain a contradiction, and the thesis is proved. ■

VII. Conclusions and future work

In this paper we addressed the problem of analyzing the schedulability of sporadic transactions on a distributed system scheduled by EDF, and how to support our methodology at run time. We proposed an algorithm to compute the *sporadic* dbf offline on each node. We also proposed the IDSP protocol that assigns appropriate deadlines to jobs guaranteeing that the dbf computed on-line never exceeds the dbf computed off-line.

As a future work, we plan to extend the methodology to task graphs. Also, we would like to study the effect of combining different scheduling strategies on different nodes, and the effects of network scheduling.

References

[1] K. W. Tindell, A. Burns, and A. Wellings, "An extendible approach for analysing fixed priority hard real-time tasks," *Journal of Real Time Systems*, vol. 6, no. 2, pp. 133–152, Mar. 1994.

[2] J. C. Palencia and M. González Harbour, "Schedulability analysis for tasks with static and dynamic offsets," in *Proceedings of the* 19^{th} *IEEE Real-Time Systems Symposium*, Madrid, Spain, Dec. 1998, pp. 26–37.

[3] M. Spuri, "Holistic analysis for deadline scheduled real-time distributed systems," INRIA, France, Tech. Rep. RR-2873, Apr. 1996.

[4] J. Palencia and M. G. Harbour, "Offset-based response time analysis of distributed systems scheduled under EDF," in *15th Euromicro Conference on Real-Time Systems*, Porto, Portugal, July 2003.

[5] R. Pellizzoni and G. Lipari, "Holistic analysis of asynchronous real-time transactions with earliest deadline scheduling," *Journal of Computer and System Sciences*, vol. 73, no. 2, pp. 186–206, Mar. 2007.

[6] A. Rahni, E. Grolleau, and M. Richard, "Feasibility analysis of non-concrete real-time transactions with edf assignment priority," in *Proceedings of the* 16^{th} *conference on Real-Time and Network Systems*, Rennes, France, Oct. 2008, pp. 109–117.

[7] M. Di Natale and J. A. Stankovic, "Dynamic end-to-end guarantees in distributed real time systems," in *Proceedings of the* 15^{th} *IEEE Real-Time Systems Symposium*, San Juan, Puerto Rico, Dec. 1994, pp. 215–227.

[8] S. K. Baruah, R. Howell, and L. Rosier, "Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor," *Real-Time Systems*, vol. 2, pp. 301–324, 1990.

[9] N. Serreli, G. Lipari, and E. Bini, "Deadline assignment for component-based analysis of real-time transactions," submitted to 2nd Workshop on Compositional Theory and Technology for Real-Time Embedded Systems, colocated to RTSS '09, available at http://retis.sssup.it/~bini/publications/.

[10] J. Sun and J. W.-S. Liu, "Synchronization protocols in distributed real-time systems," in *In ICDCS*, 1996, pp. 38–45.

[11] G. C. Buttazzo, *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*, 2nd ed. Springer Verlag, 2004, iSBN: 0-387-23137-4.

[12] S. Baruah, D. Chen, S. Gorinsky, and A. K. Mok, "Generalized multiframe tasks," *Real-Time Systems*, vol. 17, no. 1, pp. 5–22, 1999.

[13] F. Zhang and A. Burns, "Schedulability analysis for real-time systems with edf scheduling," Real-Time Systems Group, University of York, techreport YCS-2008-426, February 2008.

[14] S. Jiang, "A decoupled scheduling approach for distributed real-time embedded automotive systems," in *IEEE Real Time Technology and Applications Symposium*. IEEE Computer Society, 2006, pp. 191–198.

[15] A. Gantman, P.-N. Guo, J. Lewis, and F. Rashid, "Scheduling real-time tasks in distributed systems: A survey," University of California San Diego, Tech. Rep., 1998, available at http://cseweb.ucsd.edu/classes/fa98/cse221/OSSurveyF98/p8.pdf.

[16] Y. Zhang, D. K. Krecker, C. D. Gill, C. Lu, and G. H. Thaker, "Practical schedulability analysis for generalized sporadic tasks in distributed real-time systems," in *ECRTS*. IEEE Computer Society, 2008, pp. 223–232.

[17] S. Baruah and K. Pruhs, "Open problems in real-time scheduling," *Journal of Scheduling*, 2009.