The Distributed Deadline Synchronization Protocol for Real-Time Systems 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*—Many distributed and multiprocessor real-time applications consist of pipelines of tasks that must complete before their end-to-end deadlines. Different schedulability analyses have been proposed for both Fixed Priority and Earliest Deadline First scheduling. All the schedulability analyses proposed so far assume that a global clock synchronization protocol is used to synchronize the deadlines of jobs allocated on different processors. This assumption may limit the applicability of EDF to such systems.**

In this paper, we propose the Distributed Deadline Synchronization Protocol (DDSP) for computing the absolute deadlines of jobs. The protocol is a non-trivial extension of the Release Guard Protocol proposed for fixed priority systems. DDSP does not require a global clock synchronization, yet existing schedulability analyses are valid for schedules generated by DDSP.

I. Introduction

Many distributed real-time embedded applications consist of cyclically activated pipelines of tasks executing on different processors. For example, multimedia streaming applications are typically structured as sequences of different stages, each one performed by a task executing on a (possibly) different processor; in an encoder, the pipeline is periodically activated at each frame period, while in a decoder it is activated every time a new frame is available. Every pipeline must be completed before its *end-to-end* (EE) deadline relative to the activation of the first task. In the real-time literature, a pipeline of real-time tasks is also called *real-time transaction*. This model was first proposed in [1].

In general, an application can consist of many of these pipelines, each one with its timing parameters and EE deadline. An example of system with three pipelines executing on four processors is shown in Figure 1.

Please notice that two tasks of the same pipeline may be allocated to the same processor: in the figure, tasks τ_{12} and τ_{14} reside on processor 2, and τ_{13} and τ_{15} reside on processor 3.

Real-time pipelines can be used to model distributed as well as multi-core systems. In the first case, the networks can be modeled as special *processors* and messages can be modeled as special *communication tasks* flowing through the network. The operating system schedules the tasks on each node, while a MAC protocol resolves contention among messages over the network.

The research leading to these results has received funding from the European Communitys Seventh Framework Programme FP7 under grant agreement n.248465 "S(o)OS Service-oriented Operating Systems."

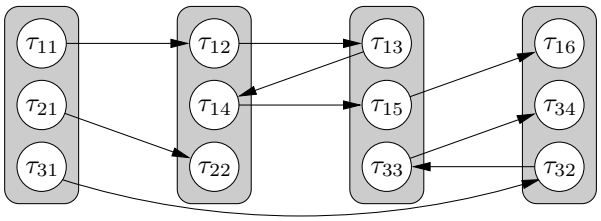

Figure 1: An example of 3 pipelines over 4 processors.

When a fixed priority scheduler is assumed on each node, such model has been extensively studied in the real-time literature. From the point of view of the analysis, the holistic analysis, first proposed by Tindell [1], is used to test the feasibility of the system, i.e. to see if all pipelines will complete before their EE deadlines. Recently, other analyses have been proposed based on event streams [2], or on the reduction to a single-processor problem [3], [4].

Similar methodologies have been developed when Earliest Deadline First (EDF) is used as a scheduler [5]–[7]. In EDF, each task's instance (or *job*) is assigned an *absolute deadline* that is used both for checking feasibility and as dynamic priority of the job.

However, the deadline assigned to each job is relative to the activation of the first task in the pipeline, which may be mapped onto a different node. Hence a common time reference is required between nodes, which can be possibly provided by a clock synchronization protocol.

This requirement limits the applicability of EDF to distributed systems where the overhead of a global clock synchronization may be unacceptable. In this paper we propose a new protocol, called Distributed Deadline Synchronization Protocol (DDSP) that removes the requirement of a global clock synchronization. Systems that use DDSP for assigning absolute deadlines can be analyzed using the classical holistic analysis [5], [6] or the processor demand [7]. The idea is similar to the one behind the Release Guard Protocol (RGP) by Sun and Liu [8]. However, the extension of RGP to EDF is not trivial, as shown in Section V.

Chatterjee and Strosnider [9] described a framework for decomposing real-time pipelines over a distributed system. They also investigate the assignment of real-time parameters. However the synchronization between nodes is assumed to

be granted.

After describing the problem in Section II, we introduce the notation in Section III, and recall the existing schedulability analysis for real-time pipelines of tasks in Section IV. Then, in Section V we propose a simple protocol and we show that it does not solve the problem when the EE deadline is larger than the period of the pipeline. In Section VI we present our protocol, and prove its correctness. Finally, in Section VII we present our conclusions.

II. Problem formulation

In our model, the first task in a pipeline is activated periodically or by external events with minimum inter-arrival times. The release of successor tasks can be done in two ways.

Event-driven activation. When an intermediate task completes, it sends a signal (or a message) to the following task in the pipeline, which is immediately activated upon the reception of the signal. This is the simplest method as it does not require any specific protocol or time synchronization. Its drawback is that it may introduce a large release jitter, because the activation instant depends on the completion time of the previous task, which in turn depends on the interference of other unrelated tasks in the system. The schedulability analysis can be carried out using the *Holistic Analysis*, first proposed by Tindell et al. [1].

Time-driven activation. Intermediate tasks are activated periodically, with period equal to the period of the pipeline, and an offset (or *phase*) equal to the worst-case completion time of the preceding task. This method is called *Phase Modification* (PM) protocol and has been first proposed by Bettati and Liu [10]. In order to set the offsets so that the precedence between tasks is respected, the protocol requires global clock synchronization. However, this protocol is only valid for periodic pipelines (not sporadic ones), and it is not robust to any variation of the parameters, like the task computation times. In fact, when a task completes later than expected, the successor task may start before such completion, violating the precedence constraint. In the same paper, Bettati and Liu also proposed the *Modified Phase Modification* (MPM) protocol to overcome such limitation.

The advantage of using a time-driven protocol is that the analysis of the distributed system can be decoupled into m single-processor schedulability analysis. In fact, the release time of a job does not depend on the completion of the preceding jobs.

A. Synchronization protocols for fixed priority

The *Release Guard Protocol*, proposed by Sun and Liu [8], takes the best of the two approaches. The idea is to control the release of a task so that the inter-arrival time between two consecutive jobs is no shorter than the pipeline period. The protocol maintains a variable a_i for each task τ_i that stores the activation time of the last job. When an activation signal arrives from the preceding task at time t, the difference $t - a_i$ is compared against the pipeline period T: if $t < a_i + T$, then the task release is delayed until $a_i + T$, otherwise it is immediately released.

In this case (as with the PM and MPM protocols), the schedulability analysis reduces to m single processor analyses. The protocol requires one timer for each task.

B. Synchronization protocols with EDF

Under EDF scheduling, we have the additional problem of setting the absolute deadline d_i^ℓ of every job τ_i^ℓ. In single processor systems, the absolute deadline d_i^ℓ can be simply computed as the job release time a_i^ℓ plus the task relative deadline D_i, which is a fixed parameter: $d_i^\ell = a_i^\ell + D_i$.

In distributed systems, if we choose a time-driven activation strategy the absolute deadline can simply be computed as the release time plus the relative deadline. Also, we can use any single processor schedulability analysis, like the one based on the demand bound function, as we suggest in [7]. However, we still have the same problems as in fixed priority: the need for a global clock synchronization and the lack of robustness.

Conversely, if we choose a purely event-driven activation strategy without a clock synchronization protocol, the absolute deadline of a job depends on its release time, which in turn depends on the completion time of the preceding task, which can be subject to a large jitter. Therefore, the analysis becomes quite pessimistic, because we must take the worst-case jitter into account.

In this paper, we propose the *Distributed Deadline Synchronization Protocol* (DDSP), a protocol to correctly set the absolute deadlines of the jobs that does not require global clock synchronization. The protocol allows us to use both an analysis based on the demand bound function, as the one proposed in [7], and the holistic analysis for EDF [5], [6]. The protocol uses an idea similar to the Release Guard Protocol by Sun and Liu for EDF scheduling. However, extending the Release Guard Protocol to EDF is not trivial, as we will discuss in Section V.

III. System model and notation

A distributed real-time application is modeled by a set of pipelines. To simplify the presentation, if not otherwise specified, since our analysis focuses on each pipeline in isolation, we denote a pipeline by $\mathcal{T}$.

The pipeline $\mathcal{T}$ 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 pipeline is activated at Φ^ℓ, which is called *absolute activation*. We consider sporadic pipelines with minimum inter-arrival time T. Hence we have

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

We denote by τ_i^ℓ the ℓ^{th} instance (job) of task τ_i. Each pipeline $\mathcal{T}$ has an *end-to-end (EE) 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.

When $D > T$, 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 pipeline $\mathcal{T}$ 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 $\mathcal{T}_k$.

Each task is assigned an *intermediate deadline* $\overline{D}_i$, that is the interval of time between the activation of the pipeline 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

$$d_i^\ell = \Phi^\ell + \overline{D}_i. \tag{2}$$

For each task we define the *offset* ϕ_i, relative to the activation of the pipeline Φ^ℓ, equal to the intermediate deadline of the preceding one:

$$\phi_1 = 0, \quad \phi_i = \overline{D}_{i-1} \quad i = 2, \ldots, n. \tag{3}$$

The *absolute offset* for job τ_i^ℓ is simply:

$$\phi_i^\ell = \Phi^\ell + \phi_i. \tag{4}$$

We define the task *relative deadline* D_i as

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

The relationship between offsets and relative deadlines is depicted in Figure 2. Clearly,

$$\sum_{i=1}^{n} D_i = D. \tag{5}$$

The release time of a job τ_i^ℓ depends on the activation method. If we use a time-driven activation, then the release time will be equal to the pipeline release time plus the offset.

$$a_i^\ell = \Phi^\ell + \phi_i. \tag{6}$$

If we use an event-driven activation, the release time will be equal to the completion time of the preceding job in the chain. If all tasks complete no later than their deadlines, we have:

$$\forall i, \ell \quad a_i^\ell \leq \Phi^\ell + \phi_i. \tag{7}$$

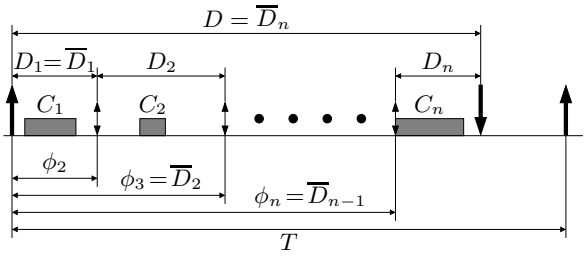

Figure 2: Notation for tasks.

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 $(\cdot)_0 \stackrel{\text{def}}{=} \max\{0, \cdot\}$.

IV. Schedulability analysis

In this section, we recall the two main schedulability analyses used in the literature for analyzing distributed real-time pipelines.

A. Holistic analysis

The holistic analysis was initially proposed for fixed priority scheduling [1], and later extended to EDF [5], [6], [11].

The basic idea is to analyze the system iteratively until the analysis converges to a fixed point. The WCDO algorithm by Palencia and Gonzalez [5] works as follows: every task is assigned a *release time* and a *release jitter* equal to the best-case and worst-case response times of the preceding task in the chain, respectively. Absolute deadlines of jobs are set in accordance to Eq. (2).

Once the parameters of every task have been set, the response times are computed using a single processor schedulability analysis. In the case of EDF, the algorithm for computing the response time has been originally proposed by Spuri [11]. The new response times are used to set the new release times and jitters. The computation is iterated until a fixed point is reached, or the EE deadline is missed.

Pellizzoni and Lipari proposed the algorithm MDO-TO [6], an improvement of WCDO that simply ignores the jitter and only uses the release time, which is set equal to the worst-case response time of the preceding task in the chain. Moreover, MDO-TO uses a less pessimistic single processor analysis for EDF that was proposed in [12].

B. Processor Demand Analysis

Holistic analysis is *global*, in the sense that the designer must know the parameters of all pipelines in the system before being able to start analyzing its schedulability.

Recently, it has been proposed a *component-based approach* analysis of distributed real-time pipelines [7]. In this analysis, the computational requirement of the subset $\mathcal{T}_k$ of tasks allocated on node k is modeled by its *demand bound function* $\mathsf{dbf}_k(t)$ that is the maximum execution requirement in any interval of length t. An algorithm for computing the dbf was developed [7]. We highlight that, differently from what one may expect, the dbf of sporadic pipelines may be larger than the corresponding periodic one.

The demand based analysis has the advantage of enabling a component-based approach to the analysis of complex distributed systems, since the demand of each pipeline does not depend on other pipelines. However, it is more pessimistic than the holistic analysis, due to the fact that the activation of every job is set to $a_i^\ell = \Phi^\ell + \phi_i$, while holistic analysis can take advantage of early completion of tasks. Nonetheless, at run-time we can release jobs earlier as long as the absolute deadline is correctly set to $d_i^\ell = a_1^\ell + \overline{D}_i$.

Lemma 1: An early release of any job τ_i^ℓ without modifying its absolute deadline does not increase the dbf.

Proof: Let $\phi_i^\ell = \Phi^\ell + \phi_i$ be the offset of job τ_i^ℓ, d_i^ℓ its absolute deadline and let $a_i^\ell < \phi_i^\ell$ be its actual release at run-time.

The demand on any interval $[t_0, t_1]$ that contains $[a_i^\ell, d_i^\ell]$ is the same as demand in $[\phi_i^\ell, d_i^\ell]$. Instead, the demand on any interval $[t_0, t_1]$ that does not contain $[a_i^\ell, d_i^\ell]$ may be lower when $[t_0, t_1] \supseteq [\phi_i^\ell, d_i^\ell]$.

Hence activating jobs earlier cannot increase the demand bound function. ■

Thanks to Lemma 1, we can analyze pipelines assuming a time-driven activation, while at run-time we can safely use an event-driven activation method.

While the demand-based analysis has the advantage of enabling composability, it requires indeed a common time reference among the nodes. This can be achieved through a global synchronization protocol. However, to avoid the additional cost of such a protocol, we describe below a distributed implicit synchronization protocol.

V. Simple Synchronization protocol

In this section we describe a simple synchronization protocol that is a direct extension of the Release Guard Protocol (RGP) by Sun and Liu [8]. We show that, in the case of $D > T$, such protocol does not guarantee a correct absolute deadline computation. Hence in Section VI we fix this problem by proposing a new protocol.

Let us first define an *optimal protocol* $\mathcal{A}$ that relies on global synchronization. As the first task in the pipeline is activated at Φ^ℓ, the optimal protocol $\mathcal{A}$ communicates this instant to all nodes so that on each node job deadlines can be correctly set equal $d_i^\ell = \Phi^\ell + \overline{D}_i$.

In order to correctly schedule jobs, any other protocol $\mathcal{B}$ must possess the following properties.

1) Under the assumption that all jobs complete before their deadlines, the absolute deadline of a job assigned by $\mathcal{B}$ must be less than or equal to the corresponding absolute assigned by the optimal protocol $\mathcal{A}$. This is required to avoid the violation of the EE pipeline deadline.
2) The *demand* generated on-line by the protocol $\mathcal{B}$ must not exceed the demand computed off-line (corresponding to deadlines assigned by $\mathcal{A}$). This is required to ensure schedulability.

To eliminate the need for global clock synchronization, the basic idea is to reason on a per processor basis. The Very Simple Protocol (VSP) that we propose has the following 3 rules.

1) The distance between the absolute deadlines of two consecutive jobs of the same task must always be greater than or equal to the pipeline period T; in formula, $d_i^\ell - d_i^{\ell-1} \geq T$.
2) The distance between the release time and the absolute deadline of a job must always be greater than the job relative deadline D_i; in formula, $d_i^\ell - a_i^\ell \geq D_i$.

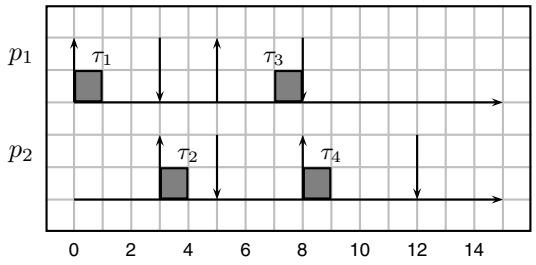

Figure 3: Deadlines assigned by the optimal protocol.

3) The distance between the absolute deadlines of two jobs of the same pipeline on the same processor is never less than the same distance computed off-line. In formula, for every task τ_j that is a predecessor of τ_i, and that is allocated on the same processor k, it must hold that $d_i^\ell - d_j^\ell \geq \overline{D}_i - \overline{D}_j$.

The protocol VSP assigns a job the minimum absolute deadline that respects the three rules above, that is

$$d_i^\ell = \max_{\tau_j \in \mathcal{T}_k} \{d_i^{\ell-1} + T, a_i^\ell + D_i, d_j^\ell + \overline{D}_i - \overline{D}_j\}. \quad (8)$$

Notice that we do not require that the distance between the absolute deadline of tasks allocated on different processors is maintained. This distance can indeed be less than the one computed by the optimal algorithm $\mathcal{A}$ with clock synchronization, as we show in the example below.

A. Example

Consider a pipeline with 4 tasks on 2 processors: tasks τ_1 and τ_3 are allocated on processor 1, whereas tasks τ_2 and τ_4 are allocated on processor 2. The EE deadline is equal to 12, and the relative deadlines D_i are $3, 2, 3, 4$, respectively, while computation times of all tasks is $\forall i, C_i = 1$. Job τ_1^1 is activated at time $a_1^1 = 0$ with deadline $d_1^1 = 3$. According to the optimal protocol $\mathcal{A}$, the absolute deadline of τ_2^1 is at $d_2^1 = a_1^1 + \overline{D}_2 = 5$ (see Figure 3). However, if we apply VSP and we assume that τ_1^1 completes at time 1 (and then releases job τ_2^1 on processor 2), the absolute deadline is set at $d_2^1 = a_2^1 + D_2 = 3$, in accordance with Eq. (8). The we assume job τ_2^1 completes at time $t = 2$ and signals job τ_3^1 on processor 1. Following the rules of the protocol, the minimum deadline is $d_3^1 = d_1^1 + \overline{D}_3 - \overline{D}_1 = 8$. Notice that, in this case, the absolute deadline set by VSP is the same as the deadline set by the optimal protocol $\mathcal{A}$.

Finally, task τ_3 completes at time $t = 8$ due to interference by jobs of other pipelines (not shown in the figures). Thus, task τ_4 is activated on processor p_2 at time $t = 8$, and VSP assigns it an absolute deadline equal to $d_4^1 = t + D_4 = 12$, the same as the optimal protocol $\mathcal{A}$.

Notice that the deadlines assigned by VSP are never later than the deadlines computed by the optimal protocol.

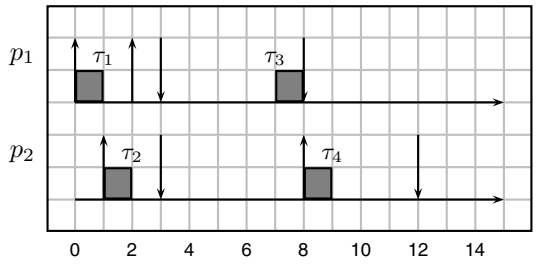

Figure 4: Deadlines assigned by the simple protocol.

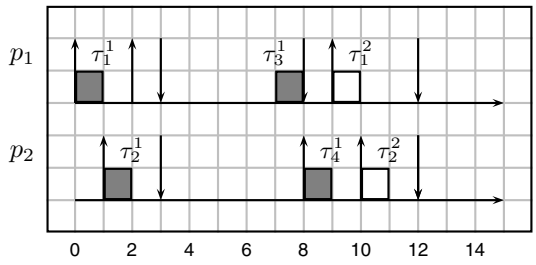

Figure 5: Counterexample for the simple protocol.

B. Failure of VSP

Although VSP is a direct translation of the RGP, when $D > T$ it may generate an interval in which the demand is higher than the demand computed according to the optimal protocol $\mathcal{A}$. Consider the pipeline of the previous example, and suppose that it is periodically activated with period equal to $T = 9$. If we use the optimal protocol $\mathcal{A}$ to assign deadlines, we expect the jobs of τ_2 to be assigned deadlines $5, 14, 23, 32, \ldots$ while those ones belonging to τ_4 deadlines $12, 21, 30, 39, \ldots$

Suppose that the second job of task τ_1 completes at $t = 10$ and immediately activates τ_2 (Figure 5 shows this scenario, jobs of the second instance are in white). According to VSP, the deadline is set at $d_2^2 = a_2^2 + D_2 = 12$. However in this circumstance, the interval $[8, 12]$ contains the two jobs τ_4^1 and τ_2^2 with an overall demand of 2 units of time. With the deadlines assigned by the optimal protocol $\mathcal{A}$ there is no interval of length 4 that can contains two units of time. Hence the protocol VSP is not correct because it may generate a demand higher than $\mathcal{A}$.

Fortunately this wrong behavior can be fixed by setting an additional rule that prevents this phenomenon to happen, as explained in the next section.

VI. Distributed Deadline Synchronization Protocol

Before describing the protocol, we introduce the following definition.

Definition 1: The *precedence set* $\mathcal{P}_i^\ell$ of job τ_i^ℓ allocated on processor k is the set of all jobs τ_j^h of tasks $\tau_j \in \mathcal{T}_k$, with $h \in \{\ell, \ell-1, \ell-2, \ldots \ell-l_0\}$ that, under any possible activation patterns, have

$$\begin{cases} d_j^h < d_i^\ell \\ \phi_j^h < \phi_i^\ell \end{cases} \tag{9}$$

where the number l_0 of past instances is:

$$l_0 = \left\lceil \frac{D}{T} \right\rceil - 1. \tag{10}$$

In practice, to avoid the problem highlighted earlier, the deadline d_i^ℓ must necessarily follow all the deadlines of jobs $\mathcal{P}_i^\ell$ under all possible activation patterns. Intuitively, it is clear that we only need to look at l_0 instances in the past. Jobs of earlier instances, in fact, cannot overlap with τ_i^ℓ. In the example of Figure 5, $\mathcal{P}_2^2 = \{\tau_4^1, \tau_2^1\}$.

The following lemma guarantees that we only have to examine periodic activation patterns.

Lemma 2: If Equation (9) holds for job τ_j^h when we assume a periodic activation pattern (i.e. all pipeline instances from h to ℓ are separated by T), then it is valid for any other feasible activation pattern.

Proof: The proof follows from the definition of absolute offset and absolute deadline (see Equations (4) and (2)). Under the assumption of periodic activation, $\Phi^\ell = \Phi^h + (\ell - h)T$, while a sporadic activation pattern leads to $\Phi'^\ell \geq \Phi^h + (\ell - h)T$. Then

$$\begin{aligned} d_j^h < d_i^\ell &= \Phi^\ell + \overline{D}_i \\ &= \Phi^h + (\ell - h)T + \overline{D}_i \\ &\leq \Phi'^\ell + \overline{D}_i = d'^\ell_i. \end{aligned}$$

The same steps hold for the release time:

$$\begin{aligned} \phi_j^h < \phi_i^\ell &= \Phi^\ell + \phi_i \\ &= \Phi^h + (\ell - h)T + \phi_i \\ &\leq \Phi'^\ell + \phi_i = \phi'^\ell_i. \end{aligned}$$

■

The precedence set of a job τ_i^ℓ does not depend on the instance index ℓ, as stated by the following lemma.

Lemma 3: If $\tau_j^h \in \mathcal{P}_i^\ell$, then $\forall u < h \leq l$, $\tau_j^{h-u} \in \mathcal{P}_i^{\ell-u}$.

Proof: Follows directly from the definition. ■

Therefore, by appropriately shifting job indexes, the precedence sets of all jobs of the same task can be made congruent (except for the first l_0 instances in which some element is missing). Hence, the precedence set $\mathcal{P}_i$ for a task τ_i can be computed off-line: after fixing a generic instance index $\ell > l_0$, we compute the list of jobs of previous instances that are part of the set, and store them in the set using relative instance indexes. Then, $\mathcal{P}_i^\ell$ can be obtained from $\mathcal{P}_i$ by shifting the job indexes by ℓ.

The main property of the precedence set is expressed by the following lemma.

Lemma 4: The distance between d_i^ℓ and the deadline d_j^h of any of the jobs in $\mathcal{P}_i^\ell$ has to satisfy the inequality

$$d_i^\ell - d_j^h \geq (\ell - h)T + \overline{D}_i - \overline{D}_j \tag{11}$$

Proof: Follows directly from the definitions. In fact

$$\begin{aligned} d_i^\ell - d_j^h &= \Phi^\ell + \overline{D}_i - \Phi^h - \overline{D}_j \\ &\geq \Phi^h + (\ell - h)T + \overline{D}_i - \Phi^h - \overline{D}_j \\ &= (\ell - h)T + \overline{D}_i - \overline{D}_j \end{aligned}$$

■

A. The protocol

We now present the Distributed Deadline Synchronization Protocol (DDSP) to compute the absolute deadline of a job τ_i^ℓ at the instant of its release a_i^ℓ. The protocol consists of four simple rules.

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

$$d_i^\ell \geq a_i^\ell + D_i \tag{12}$$

Rule 2 There must be a minimum separation between the deadlines of the jobs of the same task:

$$d_i^\ell \geq d_i^{\ell-1} + T. \tag{13}$$

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 by Lemma 4. In formula:

$$\forall \tau_j^h \in \mathcal{P}_i^\ell, \quad d_i^\ell \geq d_j^h + (\ell - h)T + \overline{D}_i - \overline{D}_j \tag{14}$$

Rule 4 At run-time, it may happen that a job τ_i^ℓ is released before a job of its precedence set, due to the fact that the end-to-end deadline can be larger than the period and previous jobs may complete much earlier than their deadline. In such a case, τ_i^ℓ is suspended because 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 of 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, $\overline{D}_i$ and $\mathcal{P}_i$ for each task).

B. Example revisited

We first apply the DDSP protocol to the simple example of Section V-B. First of all, we compute the precedence set of task τ_2 as $\mathcal{P}_i = \{\tau_4^{-1}, \tau_2^{-1}\}$. Under a periodic activation pattern, $d_4^{\ell-1}$ must always precede d_2^ℓ by at least 2 time units. Therefore, when job τ_4^2 is activated at time $t = 10$, its deadline is computed according to Rule 3 of the protocol:

$$d_2^2 = d_4^1 + T - \overline{D}_4 + \overline{D}_2 = 12 + 9 - 12 + 5 = 14$$

as expected. The situation is depicted in Figure 6.

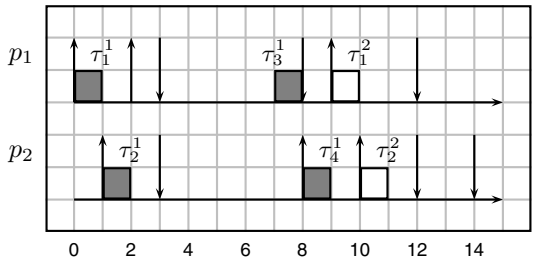

Figure 6: The example of Figure 5 scheduled by DDSP.

C. Minimal Precedence Set

Set $\mathcal{P}_i$ can contain up to $n_k(l_0 + 1)$ jobs. We can reduce such number by using a *minimal set* $\mathcal{P}_i^*$ that contains at most one job per past instance.

Therefore, we now present an algorithm to build a minimal subset $\mathcal{P}_i^*$ for each task τ_i. Then, we show that the reduced set contains all the jobs necessary to implement the DDSP protocol.

Algorithm to compute $\mathcal{P}_i^*$.

- Initially, $\mathcal{P}_i^*$ is empty.
- We fix an arbitrary instance ℓ of the pipeline.
- We start from instance $h = \ell$. We only need to know the deadline of task $\tau_j \in \mathcal{T}_k$ that immediately precedes τ_i in the pipeline. Then, τ_j^ℓ is added to $\mathcal{P}_i^*$. 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 $h \in \ell-1, \ell-2, \ldots, \ell-l_0$. As stated by Lemma 2, we can safely assume that the distance between two consecutive instances of the pipeline, with index from h to ℓ, is equal to the period. We have two sub-cases:
  1) If $\mathcal{P}_i^*$ is empty, we consider the latest job τ_j^h that has absolute deadline smaller than d_i^ℓ and absolute offset less than ϕ_i^ℓ. If such a job exists, it is added to $\mathcal{P}_i^*$. Else, we move to the previous instance.
  2) If $\mathcal{P}_i^*$ contains one or more jobs, let τ_z^u be the job with the largest deadline in $\mathcal{P}_i^*$. Consider the latest job τ_j^h that has absolute deadline in interval (d_z^u, d_i^ℓ) and absolute offset less than ϕ_i^ℓ. If such a job exists it is added to $\mathcal{P}_i^*$. Otherwise we look for jobs that are in the precedence set of τ_i^ℓ but not in the one of τ_z^u; this means that we have to search the latest job τ_j^h that has absolute deadline smaller than d_z^u and absolute offset in interval (ϕ_u^z, ϕ_i^ℓ). If such a job exists, it is added to $\mathcal{P}_i^*$. Else, we move to the previous instance.
- We iterate until instance $h = \ell - l_0$.

The minimal precedence set $\mathcal{P}_i^*$ for a task τ_i allocated on processor k contains at most $\min(l_0 + 1, n_k)$ jobs, where l_0 is defined by Equation (10), and n_k is the number of tasks of the pipeline allocated on processor k.

D. Example

An example of the procedure is shown in Figure 7. To simplify the graphical presentation, in this figure we show one instance of the same pipeline per each time-line.

Consider a pipeline having 6 tasks, with period $T = 10$ and end-to-end deadline $D = 25$. Hence, we need to consider $l_0 = 2$ 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 third line of Figure 7). By setting all preceding release at a 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 no task precedes τ_2 on processor 2;
2) job $\tau_4^{\ell-1}$;
3) job $\tau_6^{\ell-2}$.

E. Equivalence between $\mathcal{P}_i$ and $\mathcal{P}_i^*$

Rule 3 of the protocol 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 $\mathcal{P}_i^*$.

Theorem 1: If all the jobs in $\mathcal{P}_i^*$ 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 $\tau_j^h \in \mathcal{P}_i^\ell - \mathcal{P}_i^*$ 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 $\tau_j^h \in \mathcal{P}_i^\ell$ and $\tau_j^h \notin \mathcal{P}_i^*$, from the algorithm used to build $\mathcal{P}_i^*$, it must exist a job $\tau_u^z \in \mathcal{P}_i^*$ such that $h \leq z \leq \ell$ and $d_j^h < d_u^z < d_i^\ell$ and $\phi_j^h < \phi_u^z$. Then, $\tau_j^h \in \mathcal{P}_u^z$. Since τ_u^z has been assigned a deadline at run-time, according to rule 3, τ_j^h must have been assigned a deadline, against the hypothesis. ■

F. Proof of correctness

The following lemma proves that the absolute deadlines assigned by algorithm DDSP never exceed the absolute deadlines assigned by an optimal algorithm that uses global synchronization.

Lemma 5: Let Φ^ℓ be the release time of the ℓ-th instance of pipeline $\mathcal{T}$. Under the assumption that the pipeline is schedulable, the absolute deadline d_i^ℓ of every job τ_i^ℓ, computed dynamically using algorithm DDSP, is never larger than $a^\ell + \overline{D}_i$.

Proof: We want to prove that

$$\forall \ell, i \quad d_i^\ell \leq a^\ell + \overline{D}_i \tag{15}$$

We prove it by induction on both ℓ (the instance index) and i (the task index).

We start proving the first instance, when $\ell = 0$. Notice that since $\ell = 0$ then only Rules 1 and 3 can be applied. By induction on i. If $i = 1$ then we have to prove that

$$d_1^0 \leq a^0 + \overline{D}_1 \tag{16}$$

In this case only rule 1 (Eq. 12) applies. Hence

$$d_1^0 = a^0 + D_1 = a^0 + \overline{D}_1$$

as required.

Now we prove it for any i assuming it is true for all smaller task indexes. Notice that $\mathcal{P}_i^*$ contains at most one job, related to the same instance, the first one. Let f_i^ℓ be the finishing time of job τ_i^ℓ. If d_i^0 has been computed according to Eq. 12, then

$$\begin{aligned} d_i^0 &= a_i^0 + D_i = f_{i-1}^0 + D_i \leq d_{i-1}^0 + D_i \\ &\leq a^0 + \overline{D}_{i-1} + D_i = a^0 + \overline{D}_i \end{aligned}$$

as required. If, instead Rule 3 applies, then $\mathcal{P}_i^*$ contains one job, τ_j^0 and we have

$$d_i^0 = d_j^0 + \overline{D}_i - \overline{D}_j \leq a^0 + \overline{D}_j + \overline{D}_i - \overline{D}_j = a^0 + \overline{D}_i$$

which concludes the proof for instance $\ell = 0$.

Now, we perform an inductive step on ℓ. Let us assume that Eq. 15 holds true for $\ell - 1$. By induction on i. If $i = 1$, then we have to prove that

$$d_1^\ell \leq a^\ell + \overline{D}_1$$

If Rule 1 is applied, then

$$d_1^\ell = a^\ell + D_1 = a^\ell + \overline{D}_1$$

as required. If, instead Rule 2 is applied, then

$$d_1^\ell = d_1^{\ell-1} + T \leq a^{\ell-1} + D_1 + T \leq a_1^\ell + D_1$$

as required. If Rule 3 is applied, it means that

$$d_1^\ell = \max_{\tau_j^h \in \mathcal{P}_1^\ell} \left\{ d_j^h + (\ell - h)T + \overline{D}_1 - \overline{D}_j \right\}$$

Suppose that τ_u^z is the job that gives the max, then

$$\begin{aligned} d_1^\ell &= d_u^z + (\ell - z)T + \overline{D}_1 - \overline{D}_u \\ &\leq a^z + \overline{D}_u + (\ell - z)T + \overline{D}_1 - \overline{D}_u \\ &\leq a^\ell + \overline{D}_1 \end{aligned}$$

We conclude by proving it for any i, assuming it true for the preceding ones. Again, let f_i^ℓ be the finishing time of job τ_i^ℓ. If d_i^ℓ has been computed according to Rule 1, then

$$\begin{aligned} d_i^\ell &= a^\ell + D_i = f_{i-1}^\ell + D_i \leq d_{i-1}^\ell + D_i \\ &\leq a^\ell + \overline{D}_{i-1} + D_i = a^\ell + \overline{D}_i \end{aligned}$$

as required. If Rule 2 applies then

$$d_i^\ell = d_i^{\ell-1} + T \leq a^{\ell-1} + \overline{D}_i + T \leq a^\ell + \overline{D}_i$$

As before, if Rule 3 is applied, it means that

$$d_i^\ell = \max_{\tau_j^h \in \mathcal{P}_i^\ell} \left\{ d_j^h + (\ell - h)T + \overline{D}_i - \overline{D}_j \right\}$$

Suppose that τ_u^z is the job that gives the max, then we have

$$\begin{aligned} d_i^\ell &= d_u^z + (\ell - z)T + \overline{D}_i - \overline{D}_u \\ &\leq a^z + \overline{D}_u + (\ell - z)T + \overline{D}_i - \overline{D}_u \\ &\leq a^\ell + \overline{D}_i, \end{aligned}$$

which concludes the proof. ■

Lemma 5 guarantees that, if the pipeline 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, if the system results schedulable according to one of the schedulability analyses described in Section IV, then it never misses deadlines at run-time

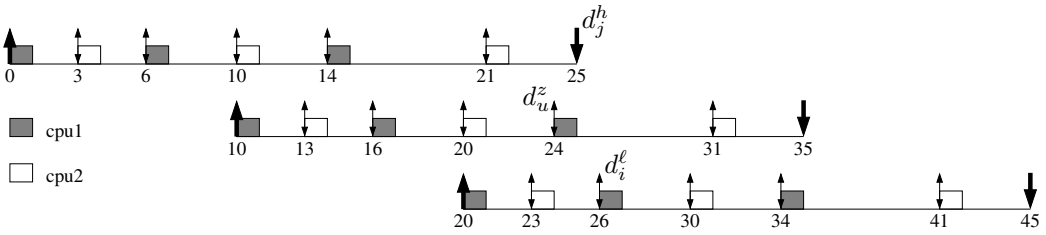

Figure 7: Example of computation of the precedence set.

when absolute deadlines are computed by DDSP. First, an important Lemma.

Lemma 6: Let τ_j^h be any job in $\mathcal{P}_i^\ell$. Under the assumption that the pipeline is schedulable and all deadlines are assigned using DDSP, the distance between d_j^h and d_i^ℓ is never smaller than the distance as computed in Equation (11).

Proof: For $\tau_j^h \in \mathcal{P}_i^*$, 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 1 to derive the thesis. ■

Now the main theorem.

Theorem 2: Consider a system that is feasible according to the processor demand analysis [7]. Assume that all tasks execute for less than their WCET and that, for any pipeline, the distance between any two consecutive releases of the pipeline is never less than the period. If DDSP is used to assign the absolute deadlines of jobs at run-time, then for every interval of time the demand produced by one pipeline never exceeds the demand computed off-line.

Proof: The proof is very long and tedious and has been removed for space constraints. The interested reader can find it in [13].

■

VII. Conclusions and future work

In this paper we presented DDSP, a protocol for computing the absolute deadlines of jobs in distributed real-time systems modeled as pipelines of tasks. The protocol does not require a global clock synchronization, and jobs can be released at the completion of the preceding job in the chain. The system can thus be analyzed using holistic schedulability analysis, or processor demand analysis.

The protocol can easily be extended to more general task topologies, like directed acyclic graphs (DAGs), by slightly modifying the definition of precedence set. As a future work, we also plan to implement the algorithm both in Linux and in ERIKA [14], [15], a real-time operating systems for limited resource platforms.

References

[1] K. Tindell and J. Clark, "Holistic schedulability analysis for distributed hard real-time systems," *Microprocessing and Microprogramming*, vol. 50, pp. 117–134, Apr. 1994.

[2] A. Hamann, M. Jersak, K. Richter, and R. Ernst, "Design space exploration and system optimization with symta/s– symbolic timing analysis for systems," in *RTSS*. IEEE Computer Society, 2004, pp. 469–478.

[3] P. Jayachandran and T. F. Abdelzaher, "Transforming distributed acyclic systems into equivalent uniprocessors under preemptive and non-preemptive scheduling," in *ECRTS*. IEEE Computer Society, 2008, pp. 233–242.

[4] P. Jayachandran and T. Abdelzaher, "Delay composition algebra: A reduction-based schedulability algebra for distributed real-time systems," in *Proceedings of the 29th IEEE Real-Time Systems Symposium*, Barcelona, Spain, Dec. 2008, pp. 259–269.

[5] 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.

[6] 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.

[7] N. Serreli, G. Lipari, and E. Bini, "The demand bound function interface of distributed sporadic pipelines of tasks scheduled by edf," in *Proceedings of the 22nd Euromicro Conference on Real-Time Systems*, Bruxelles, Belgium, Jul. 2010, available at http://retis.sssup.it/~bini/publications/.

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

[9] S. Chatterjee and J. Strosnider, "Distributed pipeline scheduling: A framework for distributed, heterogeneous real-time system design," *The Computer Journal*, vol. 38, no. 4, pp. 271–285, 1995.

[10] R. Bettati and J. W.-S. Liu, "End-to-end scheduling to meet deadlines in distributed systems," in *ICDCS*, 1992, pp. 452–459.

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

[12] R. Pellizzoni and G. Lipari, "Feasibility analysis of real-time periodic tasks with offsets," *Real-Time System Journal*, vol. 30, no. 1, pp. 105–128, 2005.

[13] N. Serreli, "Component-based analysis and synchronization of distributed transactions scheduled by edf," Ph.D. dissertation, Scuola Superiore Sant'Anna, http://retis.sssup.it/?q=node/65, Jan 2010.

[14] P. Gai, E. Bini, G. Lipari, M. Di Natale, and L. Abeni, "Architecture for a portable open source real-time kernel environment," in *Proceedings of the 2nd Real-Time Linux Workshop*, Orlando (FL), U.S.A., Nov. 2000.

[15] "Erika enterprise," http://erika.tuxfamily.org/.