On-line Scheduling of Real-Time Distributed Computers With Complex Communication Constraints

Pascal RICHARD, Francis COTTET, Michaël RICHARD
Laboratory of Applied Computer Science (LISI)
National School of Engineers in Mechanics and Aeronautics (ENSMA)
Téléport 2 - BP 40109
F-86961 Futuroscope
{richardp ,cottet,richardm}@ensma.fr

Abstract

We consider the scheduling of periodic tasks running on distributed computers. Every execution of a task must meet its deadline. Response time analysis of the tasks is used to prove the schedulabilty of hard real-time distributed systems according the on-line priority rules that schedule the processors and the network. Its main advantage is to take into account the precedence dependencies of the schedules of the tasks on the processors and the messages sent on the network(s). Past works have addressed the issue of tasks related by asynchronous communication constraints with the senders and the receivers working at the same rate. In this paper we study more general relations among tasks when the rates of dependent tasks are not equal. We call such relations generalized communication constraints. Usually distributed systems are scheduled using a synchronization protocol and an on-line scheduling algorithm by processor. We present in this paper a graph theoretical approach to this schedulability analysis. Our algorithm transforms complex communication relations into classical ones, so that the classical scheduling analysis can be fully applied. That transformation is independent of the architecture of the distributed systems and no assumption is made on the synchronization protocol considered.

1. Introduction

Hard real-time systems are computing systems that must react within precise time constraints to events coming from the environment. Typically, distributed computers controlling a physical devices or processes involve following basic functions: at periodic intervals and responding to different scenarios by sending signals to actuators with a time-bound. A hard real-time system is programmed as multitask software that is a set of synchronized, communicating tasks sharing critical resources [4] [5][21]. Every task is statically assigned to a processor at the design step, arrives periodically in the system and must met their deadlines. In most of hard real-time systems, due to the strict deadlines that must be met, communications among tasks are implemented in a completely deterministic manner. In other words, computational activities cannot be executed in arbitrary order but have to respect some communication relations defined at the design stage. Past works have addressed the issue of tasks related by simple communication relations. So, if a task τ_i has to communicate the result of its processing to another task τ_j, these tasks have to be scheduled in such a way that the execution of k^{th} instance of the task τ_i precedes the execution of the k^{th} instance of the task τ_j. A more general problem concerns tasks related by generalized precedence relations where n instances of a task can precede one instance of another task or one instance of task precedes m instances of another task.

On the other hand, the use of real-time scheduling algorithms based on the temporal task parameters not only helps in producing a valid sequence, but may also aid the temporal validation of the application, either by analytic checks or by simulation. Usually real-time scheduling is based on preemptive priority-based policies and a set of periodic tasks. Scheduling decisions are taken on-line by every active sub-system (processor or network interface). A task τ_i is schedulable if its worst-case response time (time between the arrival and the completion of the task) is lower than or equal to its deadline (relative to its arrival).

When distributed systems are considered, the worst-case response times of the tasks are mutually dependent because of the messages exchanged by them. The analysis must take into account the synchronization protocol of the communicating tasks [23, 22], and also the scheduling policies of the messages on the network. It is assumed hereafter that the messages are sent at the end of the tasks and received at their beginning. The network is a shared resource for every communicating task. As a consequence no optimal and on-line scheduling policy can be found for scheduling a distributed system. Since only worst-case execution times are known for the tasks, it is not possible to compute the exact sending dates of the messages on the network. As a consequence some priority inversions can occurred if a message is sent before another that would block a higher priority task.

The holistic analysis is a well-known response time analysis of a hard real-time distributed systems [24]. The dependencies between them are taken into account while computing the worst-case release jitters of the tasks and the messages. Results are known for the fixed-priority policy [9, 10, 11, 17, 27], FIFO and round-robin policies [7, 8], and also for the earliest deadline first [18, 19, 20]. Different kinds of networks have also been studied in this context [1, 6, 15, 16, 25, 26, 28]. Furthermore, some results have been extended when offsets are considered on task arrivals [14, 13, 24].

We study in this paper the holistic analysis of complex communicating tasks having different periods in a distributed system. We call the corresponding precedence relations: generalized precedence constraints. As we will see, the holistic analysis only focus to the underlying precedence constraints generated by the asynchronous communications. We prove that the problem dealing with generalized precedence constraints can always be transformed to an equivalent problem with classical precedence constraints among tasks working at the same rate. The remainder of the paper is organized as follows: section 2 presents the background of the holistic analysis, section 3 formally defines the generalized precedence relations among tasks. In section 4 we propose a transformation technique based on the precedence graph unfolding, that allows to use directly the holistic analysis.

Notations:

- τ_i is the task number i.
- C_i is the worst-case computation time of τ_i on each release.
- D_i is the deadline of τ_i, measured relative to the arrival time of the task.
- T_i is the period of the task τ_i.
- J_i is the worst-case release jitter of the task τ_i (i.e. the worst-case delay between the arrival of a task and its release).
- B_i the worst-case blocking time of τ_i, according to a given concurrency control protocol.
- R_i is the worst-case response time of the task τ_i (i.e. the worst-case delay between the arrival of a task and its completion).
- $prec(i)$ is the set of predecessors of τ_i in the precedence graph.

2. Holistic analysis background

Tindell and Clark have proposed a nice Response Time Analysis method for hard real-time distributed systems [27] called the holistic analysis. In their approach, the network is viewed as an additional processor that executes fictitious tasks that model the messages. In that way, communication dependencies are modeled as precedence constraints among this new set of tasks. Their approach is also useful for dealing with distributed systems based on several networks. The principle of the method is to compute iteratively the worst-case release jitters of the tasks as the maximum worst-case response times of the predecessors in the precedence graph. In that way when the tasks are released, their input messages have been received by the node executing them. Computation stops when the values computed in two successive iterations are equal (i.e. when the fix point of the system has been reached). The system of equations is the following:

$$1 \le i \le n \left\{ \begin{array}{lcl} J_i^{(0)} & = & 0 \\ R_i^{(k)} & = & ResponseTime\left(i, J_i^{(k-1)}\right) \\ J_i^{(k)} & = & \max\limits_{j \,\in\, prec(i)} \left(R_j^{(k)}\right) \end{array} \right.$$

$$R_i = R_i^{(k)} = R_i^{(k-1)}$$

The method always converges if the functions ”ResponseTime” are enforced to be non-decreasing functions of the release jitters. In the remainder of the paper we consider that the messages are fictitious tasks and that the network is an additional fictitious processor (possibly several additional processors if the distributed system uses more than one network). The response time function computes the worst-case response time of a given task and takes into account the higher priority tasks. So to every processor is associated a specific non-decreasing response-time function that takes in parameters the number of a task and the current release jitters of the tasks running on that processor.

In that way communication relations are modeled by a set of precedence relations. Results in the literature assume that the communicating tasks work at the same rate. But more complex asynchronous communicating relations can be found when not all the instances of a task are submitted to the precedence relations. For instance a task can be released when a set of values has been put in the buffer by an another task. In the next section, we present a general technique to handle such complex but common precedence relations in real applications.

3. Complex Communication Relations

3.1. Generalized Precedence Constraints

The real-time distributed tasks are described by a directed acyclic graph, where the tasks and the messages are modeled by the vertices and precedence relations among them are modeled by the edges. Since communications are assumed to be asynchronous, to every communication is associated two precedence constraints (i.e. two edges): one from the sender to the message and another from the message to the receiver (messages are considered as tasks). This precedence graph defines a partial order on the task set. If the task τ_i is connected by a path in the precedence graph to the task τ_j then $\tau_i \prec \tau_j$. It means that every execution of the task τ_j must be preceded by an execution of the task τ_j. Figure 1 illustrates the model of a hard real-time distributed system by a directed acyclic graph that describes the precedence relation for tasks and messages.

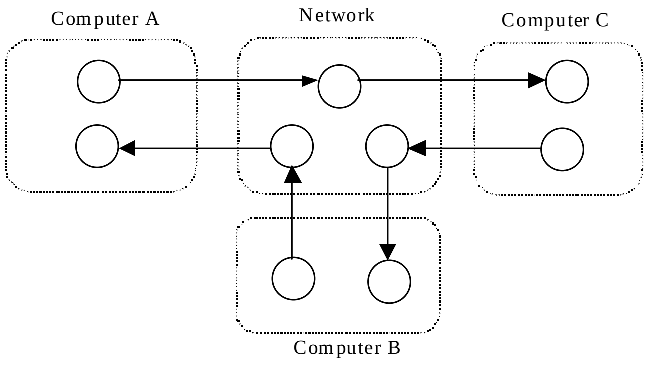

Figure 1. Model of a hard real-time distributed system.

In the case of a simple precedence constraint (dependent tasks work at the same rate), all the tasks belonging to a connected component of the precedence graph must have the same periods. Because if the periods of the tasks are different, sooner or latter both tasks will run at the lowest rate. As a consequence the task with the shortest period will miss its deadline (we do not consider cyclical asynchronous buffers of messages [4]).

Some applications require more involved communication relations. Figure 2 gives two examples where the rates of the communicating tasks are not equal. We say that the precedence relations are generalized. In order to distinguish the classical communication relations to the complex ones, the last ones are represented by dotted arrows. As indicated figure 2 in case 1 ($kT_i = T_j, k \in \mathbb{N}$) k instances of a task can precede one instance of a task and in case 2 ($T_i = kT_j, k \in \mathbb{N}$) one instance of a task τ_i precedes k instances of an another task τ_j. Figure 3 presents the Gantt chart associated to these both cases. There is furthermore more complex cases if k is no longer an integer but a rational number. In these latter cases, the generalization of the techniques used for simple communication constraints to analyze schedulability of the tasks (as in the holistic analysis) is not straightforward.

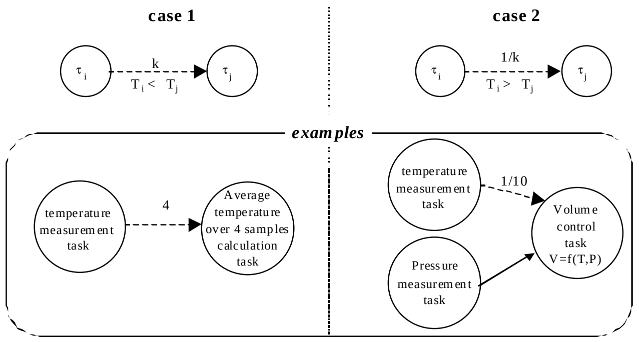
case 1
τ_i
k
$T_i < T_j$
τ_j
case 2
τ_i
1/k
$T_i > T_j$
τ_j
examples
temperature measurement task
4
Average temperature over 4 samples calculation task
temperature measurement task
1/10
Volume control task V=f(T,P)
Pressure measurement task

Figure 2. Two cases of generalized precedence relations among tasks.

A simple precedence constraints $\tau_i \prec \tau_j$ is satisfied if, and only if, at every date, the number of instances of τ_i that have been finished is greater than or equal to the number of instances of τ_j that have been started. When generalized precedence constraints are considered, the condition on the rates of dependent tasks must be verified, as described in the following definition.

Definition 1 *Let $B_i(t)$ (resp.$E_i(t)$) be the number of instances of τ_i that have been started (resp. finished) at the date t, then the generalized precedence constraint $\tau_i \prec \tau_j$ is satisfied if, and only if:*

$$E_i(t) \times T_i \geq B_j(t) \times T_j \qquad \forall t \in \mathbb{N}$$

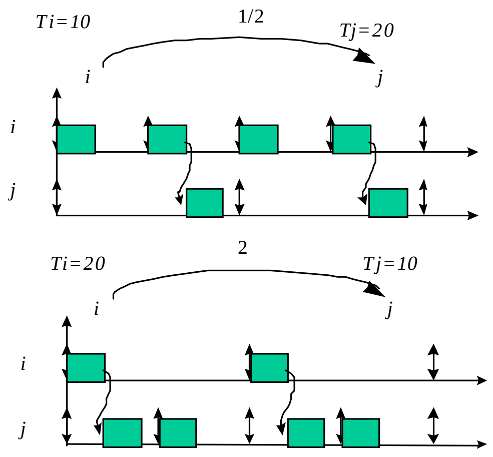

Figure 3. Gantt charts of cases presented on figure 2.

3.2. Related Precedence Relations

Similar precedence constraints to these presented in the previous subsection have been studied in the literature. We report the two most important that are closely related to our generalized precedence relations. Then, we focus on the difference between these two kinds of precedence constraints, on the one hand, with those that are considered in this paper, on the other hand.

In [12] is studied the basic cyclic scheduling problem with linear precedence constraints. The objective of the basic cyclic scheduling problem is to compute optimal frequencies of generic tasks in the earliest schedule (asymptotic behavior of the system). Tasks are subjected the strongly connected precedence graph, where the vertices model tasks and edges model linear precedence relations. No resource constraint is considered (infinite number of processors). In this study, the difference of iteration indices between the execution of two tasks subjected to a linear precedence constraint is not constant (as in classical precedence relations), but a linear function of the execution indices. If we consider two tasks τ_i and τ_i connected by a linear precedence relation (e), and the respective cyclic executions of the tasks are $\langle\tau_i,1\rangle,...,\langle\tau_i,n\rangle$ and $\langle\tau_j,1\rangle,...,\langle\tau_j,n\rangle$, then instances in precedence are given by:

$$\langle\tau_i, a(e)\times k + b(e)\rangle \quad \prec \quad \langle\tau_j, c(e)\times k + d(e)\rangle$$

where $\prec$ denotes the classical precedence relation and $a, b, c,$ and d are integers associated to the edge e. The problem is solved using an expansion of the linear precedence graph into a classical precedence graph. We will use the same approach to solve our problem with acyclic generalized precedence graph.

In [2] is studied the feasibility problem of (acyclic) dataflow graphs used in signal processing algorithms. As usual, vertices model recurring tasks and edges model buffers used to exchange data between tasks. A task can produce or consume several data simultaneously in a buffer. As a consequence, the precedence relation between tasks are generalized precedence constraints. But the buffer in [2] has a specificity: a threshold amount that denotes the number of messages that must be present in a buffer in order to allow the next task to execute. As a direct consequence, the feasibility problem (schedulability and respect of the upper bound on buffer sizes) is NP-hard in the strong sense. In practice, the threshold amount of a buffer lead to introduce offsets of the tasks before consuming the queued data. It is well-known that feasibility problem for recurring tasks with offsets is NP-hard in the strong sense [3], even for one processor real-time systems.

A generalized precedence is a particular case of linear precedence constraint and the graphs consider in this paper are acyclic while those consider by [12] are cyclic. These differences lead to different expansion of graphs. On the other hand, dataflow graphs consider in [2] lead exactly to generalized precedence constraints studied in this paper, but the threshold amounts associated to the buffers lead to computer intractability for the feasibility problem. It will not be the case in our study.

4. Generalized precedence graph unfolding

In order to validate a hard real-time distributed system with generalized communications, we show in this section that every graph that models the system with generalized precedence constraints can be unfolded into a graph containing only simple precedence constraints (i.e. such that all dependent tasks have the same rate). In that way, the holistic analysis can be applied without any change to analyze the schedulability of the hard real-time system. The method is presented in two steps: first we present the unfolding of a generalized precedence constraint, and in second we deal with the complete graph unfolding. Complexity issues are later discussed. A generalized precedence constraint can be modeled by a set of simple precedence constraints.

4.1. Unfolding one precedence constraint

Every task τ_i is duplicated by n_i tasks $\tau_i^1,...,\tau_i^n$ that are simply called hereafter the duplicates. The duplicates have the same computation times than the original tasks. The n^{th} execution of the duplicate τ_i^k models the $((n-1)n_i+k)^{th}$ execution of the task τ_i. Obviously we fix that:

$$\tau_i^k \prec \tau_i^{k+1} \qquad 1 \le k \le n-1$$

We now study the precedence relation between the duplicates of different tasks. Theorem 1 establishes a relation between the rates of the tasks and the number of duplicates.

Theorem 1 *Let n_i and n_j be the number of the duplicates of τ_i and τ_j respectively, the generalized precedence constraint $\tau_i \prec \tau_j$ can be modeled by a simple precedence constraint between two duplicates if, and only if, it exists $n_i \in \mathbb{N}^*, n_j \in \mathbb{N}^*$ such that:*

$$n_i \times T_i - n_j \times T_j = 0$$

Proof: we consider two duplicates of the tasks τ_i and τ_j, respectively τ_i^k and τ_i^l . We give a necessary and sufficient condition of existence of a simple precedence constraints between respectively τ_i^k and τ_i^l ($\tau_i^k \prec \tau_i^l$):
- The l^{th} instance of τ_j can start after the k^{th} instance of τ_i:

$$k \times T_i - l \times T_j \geq 0 \qquad (1)$$

- the $(l-1)^{th}$ instance of τ_j can start after the k^{th} instance of :

$$(k-1) \times T_i - (l-1) \times T_j \geq 0 \qquad (2)$$

- the l^{th} instance of τ_j cannot start after the $(k-1)^{th}$ instance of τ_i:

$$(k-1) \times T_i - l \times T_j \geq 0 \qquad (3)$$

Grouping (1), (2) and (3), we can state that ($\tau_i^k \prec \tau_i^l$) if, and only if:

$$T_i > k \times T_i - l \times T_j \geq max(T_i - T_j, 0) \qquad (4)$$

The previous relation must hold for every instance of τ_i^k and τ_j^l. Since by definition of the duplicates the n^{th} instance of τ_j^k (resp. τ_j^l) models the $((n-1)n_i + k)^{th}$ instance of τ_i (resp. $((n-1)n_j + l)^{th}$ instance of τ_j) then (4) must also be verified for these numbers:

$$\begin{aligned} T_i > ((n-1)n_i + k) \times T_i - ((n-1)n_j + l) \times T_j \\ \geq max(T_i - T_j, 0) \end{aligned} \qquad (5)$$

Then it follows that (5) is verified if, and only if, it exists $n_i \in \mathbb{N}^*, n_j \in \mathbb{N}^*$ such that:

$$n_i \times T_i - n_j \times T_j = 0$$

Theorem 1 allows computing the numbers of duplicates. Using the necessary and sufficient condition (5), the simple precedence among the duplicates can be computed by theorem 2.

Theorem 2 *Let $\tau_i \prec \tau_j$ be a generalized precedence constraint, n_i and n_j be the numbers of duplicates of the tasks τ_i and τ_j. Then it induces n_i simple precedence constraints if $T_i > T_j$:*

$$\forall k \in \{1..n_i\}, \tau_i^k \prec \tau_j^{a_k}, a_k = \lfloor (k-1)T_i/T_j \rfloor + 1$$

n_j simple precedence constraints otherwise:

$$\forall k \in \{1..n_j\}, \tau_i^{b_k} \prec \tau_j^k, b_k = \lceil kT_j/T_i \rfloor$$

Proof: Let τ_i^k and τ_j^l be two duplicates of the tasks τ_i and τ_j. We consider two cases:

- If $T_i > T_j$, we need to compute a_k the number of duplicates of τ_j (i.e. $\tau_j^{a_k}$) that is preceded by the k^{th} duplicate of τ_i. In the proof of theorem 1 it is shown that if, and only if:

  $$\begin{aligned} T_i > ((n-1)n_i + l)T_i - ((n-1)n_j) + a_k)T_j \\ \geq T_i - T_j \end{aligned}$$

  Since $n_iT_i - n_jT_j = 0$, then the previous expression becomes: $T_i > kT_i - a_kT_j \geq 0$. It follows that the unique integer solution is: $a_k = \lfloor (k-1)T_i/T_j \rfloor + 1$.

- If $T_i \leq T_j$, in the same way we compute the number b_k of the duplicate of τ_j that precedes the k^{th} duplicate of τ_j. By theorem 1, $\tau_i^{b_k} \prec \tau_j^k$ if, and only if:

  $$T_i > ((n-1)n_i + b_k)T_i - ((n-1)n_j + k)T_j \geq 0$$

  Using the same reasoning than in the previous case, we obtain that the unique integer solution of the previous expression is: $b_k = \lceil kT_j/T_i \rceil$.

We detail an example of the unfolding method. Let τ_i and τ_j be two tasks with respectively periods 30 ms and 40 ms. The number of duplicates according to the theorem 1 follows: $n_i \times 30 - n_j \times 30 = 0$. The minimal solution within integer numbers is $n_i = 4$ and $n_j = 3$ (the solutions are linked to the LCM of the periods as we will latter see). Notice that $T_i < T_j$, then applying theorem 2 we obtain 3 simple precedence constraints:

$$\begin{aligned} k = 1, b_1 = \lceil 1 \times 40/30 \rceil = 2 \quad &\Rightarrow \quad \tau_i^2 \prec \tau_j^1 \\ k = 2, b_2 = \lceil 2 \times 40/30 \rceil = 3 \quad &\Rightarrow \quad \tau_i^3 \prec \tau_j^2 \\ k = 3, b_3 = \lceil 3 \times 40/30 \rceil = 4 \quad &\Rightarrow \quad \tau_i^4 \prec \tau_j^3 \end{aligned}$$

Figure 4 presents the generalized precedence constraint and the according unfolded graph.

4.2. Unfolding a graph

We now detail the unfolding of a complete generalized precedence graph using the technique just presented. Without loss of generality, we only consider acyclic graph, since if there is a circuit in the graph then an execution of a task will depend of itself. We first need some definitions.

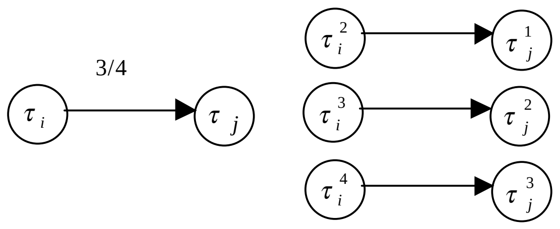

Figure 4. A generalized precedence constraint and the corresponding unfolded graph.

Definition 2 *Let p be an edge connecting τ_i to τ_j in the graph G, the height of p, noted $H(p)$, is:*

$$H(p) = T_i/T_j$$

Definition 3 *Let ρ be a path in the graph G. The height of ρ is defined by the product of the heights of the edges belonging to ρ :*

$$H(\rho) = \prod_{p \in \rho} H(p)$$

From these definitions, it can be easily shown a strong property of the generalized precedence graph.

Property 1 *Let τ_i and τ_j be two vertices of the generalized precedence graph then every path ρ from τ_i to τ_j has the same height $H(\rho) = T_i/T_j$.*

Proof: We proceed by induction on the length of an arbitrary path $\tau_i e_1 e_2 ... e_k \tau_j$ connecting τ_i to τ_j (noted hereafter ρ_k).
Base: if $k = 0$, then $\rho_0 = \tau_i \tau_j$ and then $H(\rho_0) = T_i/T_j$.
Step: Assume that the induction hypothesis is verified for ρ_k. At step $k+1$, the path is $\rho_{k+1} = \tau_i e_1 e_2 ... e_k e_{k+1} \tau_j$. By definition of the height of a path:

$$H(\rho_{k+1}) = \frac{T_i}{T_{e_1}} \times \frac{T_{e_1}}{T_{e_2}} \times ... \times \frac{T_{e_k}}{T_{e_{k+1}}} \times \frac{T_{e_{k+1}}}{T_j} = H(\rho_k)$$

As a consequence, every circuit of a precedence graph has a height equal to 1. But every execution of a task belonging to a circuit need to be preceded by itself. So this task will never be released and cannot meet its deadline. Clearly, a graph can be unfold if, and only if, the precedence graph contains no circuit. This property can be checked in polynomial time.

The relation given in the theorem 1 must be verified for every edge of the graph G in order to allow the unfolding process. So we must solve the system of equations $\Sigma(G)$:

$$\Sigma(G): \quad n_i T_i = n_j T_j \qquad \tau_i \prec \tau_j, n_i \in \mathbb{N}^*, n_j \in \mathbb{N}^*$$

Theorem 3 proves that every acyclic generalized precedence graph can be unfolded. It also proves how to compute the numbers of duplicates while considering the whole precedence graph. Its proof is based on the fact that the number of duplicates can be computed while considering an arbitrary spanning tree belonging to the graph G. We illustrate these principles through an example: the computation of the minimal number of duplicates for the graph given in figure 5. There is two paths joining τ_1 to τ_5. It is easy to see that the edge (τ_1, τ_4), corresponding to the equation of $\Sigma(G)$: $n_1 T_1 = n_4 T_4$ (5), is a linear combination of the others equations:

$$\begin{aligned} n_1 T_1 &= n_3 T_3 \\ n_2 T_2 &= n_4 T_4 \\ n_3 T_3 &= n_5 T_5 \\ n_4 T_4 &= n_1 T_1 \end{aligned}$$

So (5) is not useful to solve $\Sigma(G)$, and repeating this process leads to a spanning tree belonging to G.

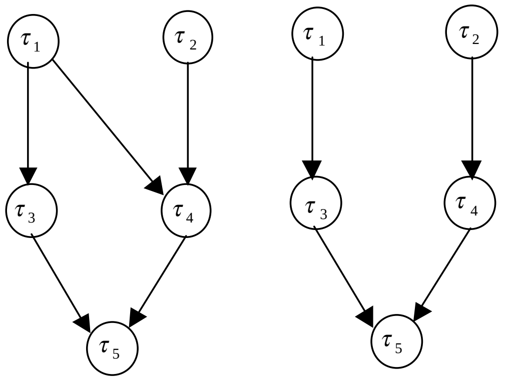

Figure 5. A generalized precedence graph and a spanning tree of G.

Theorem 3 *Every acyclic generalized precedence graph can be unfolded and the minimum solution of $\Sigma(G)$ is given by:*

$$n_i = \frac{lcm(T_1, ..., T_m)}{T_i} \qquad 1 \leq i \leq m$$

Proof: Without loss of generality, we assume the graph G to be a connected graph. If it is not the case, then every connected component would be separately studied. Let m be the number of vertices in G, and n be the number of edges, we verify $n \geq m - 1$, since the graph is assumed to be connected. Consider two paths joining the vertices τ_i and τ_j, then by the property 1, they have the same height. So it exists in $\Sigma(G)$ an equation that is a linear combination of the others. Deleting the corresponding equation let unchanged the solutions of $\Sigma(G)$. Repeating this process for every path joining couple of vertices, then $n - m + 1$ edges will be deleted. Let $\overline{G}$ be the obtained graph by deleting redundant equations, it has exactly m vertices and $m - 1$ edges. So $\overline{G}$ is a spanning tree of G. We also verify that $\Sigma(G) \Leftrightarrow \Sigma(\overline{G})$. Let $X = (n_i)$ be an integer vector, since $\Sigma(\overline{G})$ has m variables and $m - 1$ equations, then every integer vector $\lambda X, \lambda \in \mathbb{N}$ is also a solution (i.e. the solutions are proportional). Then it follows that the minimal integer solutions of $\Sigma(G)$ are given by:

$$n_i = \frac{lcm(T_1, ..., T_m)}{T_i} \qquad 1 \leq i \leq m$$

In order to complete the unfolding process, we must define the parameters of the duplicates. It directly follows from theorem 2 and 3 that S and S^* are equivalent (i.e the same deadlines, workloads and precedence relations among the instances of the tasks).

Theorem 4 *Let $S = \{\tau_i(r_i, C_i, D_i, T_i), 1 \leq i \leq m\}$ be a set of tasks, $\prec$ be an partial order on S, n_i be the number of duplicates of the tasks $\tau_i, 1 \leq i \leq m$, and $S^* = \{\tau_i^k(r_i^k, C_i^k, D_i^k, T_i^k), 1 \leq i \leq m\, 1 \leq k \leq n_i\}$ be a set of tasks such that:*

$$1 \leq i \leq m, 1 \leq k \leq n_i \quad \begin{cases} r_i^k &= r_i + (k-1)T_i \\ T_i^k &= n_i T_i \\ D_i^k &= D_i \\ C_i^k &= C_i \end{cases}$$

then S under the partial order $\prec$ is schedulable if, and only if, S^* is schedulable under the unfolded partial order.

According to the generalized precedence graph of figure 3 and theorem 4, then a feasible schedule is presented figure 6.

4.3. Complexity issue

The complete algorithm of the method is presented figure 7. The next theorem gives the complexity of the unfolding process.

Theorem 5 *Let $G = (T, P)$ be an arbitrary generalized precedence graph then complexity of the unfolding algorithm is:*

$$O(|T| \sum_{i \in T} n_i + |P| \sum_{(i,j) \in P} \min{(n_i, n_j)})$$

where T is the set of vertices, P is the set of edges of G, and n_i (resp. n_j) is the number of duplicates of τ_i (resp. τ_j).

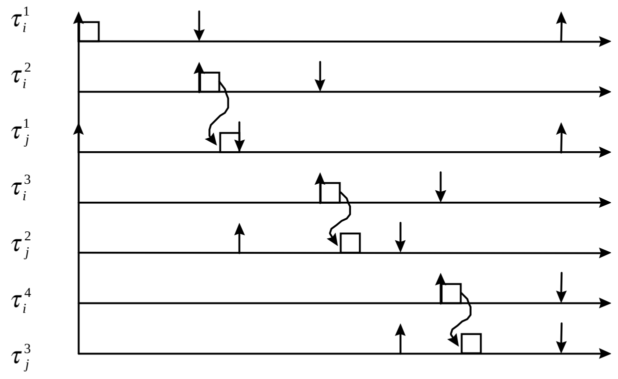

Figure 6. Gantt chart of dependent tasks of figure 3.

Proof: The algorithm contains two exponential-time parts: duplicates creation, on one hand, and precedence constraint construction, on the other hand. Duplicates are created in $O(|T| \sum_{i \in T} n_i)$. In the last part of the algorithm, since $n_i \times T_i = n_j \times t_j$ for every edge (τ_i, τ_j) in the precedence graph, then we verify that $T_i > T_j \Rightarrow n_i < n_j$. The number of iterations done in the loops are dependent of the number of the duplicates. So the complexity of the second part of the algorithm is $O(|P| \sum_{(i,j) \in P} \min{(n_i, n_j)})$.

Thus the unfolding process leads to a pseudo-polynomial time algorithm assuming that the values n_i are entries of the problem. But these values can by huge since they are directly dependent of the lcm of periods. The worst-case scenario is obtained if the periods are prime numbers (but is never the case in practical applications). We must notice that the NP-Hardness of the unfolding of generalized precedence graph is an open problem.

No assumption on the architecture of the distributed system is made, then the holistic analysis can be applied without any changes. Notice that if tasks are first release simultaneously in the initial task set, then it is not the case after the generalized precedence graph unfolding. So, functions calculating the worst-case response times of the duplicates during the holistic analysis must explicitly take into account of the release dates.

4.4. Example

We present an example of the unfolding process on the simple distributed architecture with two computers and one network. Figure 9 presents the generalized precedence graph and the mapping of the tasks for this the application. Four tasks run on the first computer: τ_1, τ_7, τ_8 collect samples from sensors and τ_2 prepares messages to send on the network. In the second computer, τ_3 collects messages and dispatches data to the other tasks, namely $\tau_4, \tau_5, \tau_6, \tau_9$, and

```
Input : graph $G = (T, P)$            /* $T$ : set of edges, $P$ : set of vertices */
Output : graph $S = (E, V)$           /* $E$ : set of edges, $V$ : set of vertices */
Begin
    $E = V = \varnothing$
    $H = \underset{i=1..n}{LCM}(T_i)$
    /* create duplicata */
    For Each $i \in T$ Do             /* For every Edge of $T$ */
        $n_i = \frac{H}{T_i}$
        For $k = 1..n_i$ Do
            $E = E \cup \{i_k\}$
        End For
    End For Each
    /* Create simple communication constraints */
    For Each $e \in P$ Do             /* For every vertice of $P$ */
        $i$=Input_Edge($e$) ; $j$=Output_Edge($e$)
        If $T_i > T_j$ Then
            For $k = 1..n_i$ Do
                $a_k = \left\lfloor \frac{(k-1)T_i}{T_j} \right\rfloor + 1$
                $V = V \cup \{(i_k, j_a)\}$
            End For
        Else
            For $k = 1..n_j$ Do
                $b_k = \left\lceil \frac{kT_j}{T_i} \right\rceil$
                $V = V \cup \{(i_b, j_k)\}$

            End For

        End If

    End For Each
End
```

Figure 7. The unfolding algorithm.

τ_{10}. Figure 9 gives the periods of the tasks and the number of duplicates after the unfolding process of the generalized precedence graph of figure 8.

The lcm of the periods is 800 and the total number of tasks (duplicates) to be considered in the schedulability analysis is 364. The unfolding process requires less than one second of computation on a Personal Computer (Pentium III). Schedulability analysis can be easily done using the holistic analysis since it runs in pseudo-polynomial time.

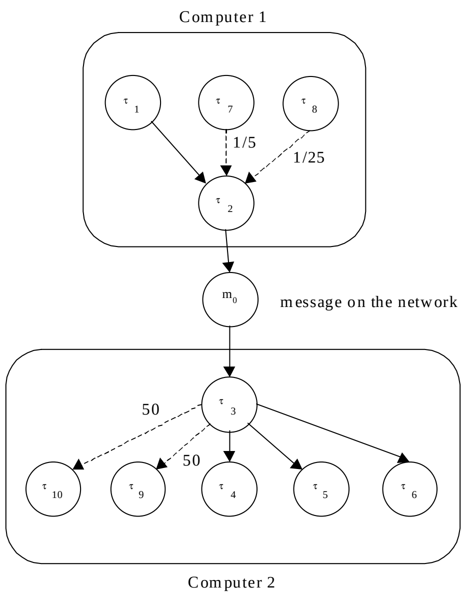

Figure 8. Example of a simple distributed application.

5. Conclusion

We have presented a method to handle complex asynchronous communication relations between tasks in a hard real-time distributed system in order to prove its schedulability using the holistic analysis. The method is based on the unfolding of the generalized precedence graph underlying to the complex communication relations. A new set of tasks and a new set of messages are then created such that all dependent tasks have the same period. This new task set is shown to be equivalent from the schedulability point of view and can be directly use to validate the application with the classical holistic analysis.

Since our method only focus on the precedence relations among the tasks and the messages we made no assumption on the architecture of the hard real-time distributed system, on the scheduling policies of the processors or the network, on the concurrency control protocol and also on the synchronization protocol of the messages. All these parameters of the distributed system are managed in the holistic analysis that uses in entry the problem defined by our algorithm. The method can be applied on single processor problems as well as on complex distributed systems.

In some particular cases, the generalized precedence graph unfolding is not necessary. For instance if we consider a generalized precedence constraint between two tasks mapped on the same processor and having proportional periods, then one can enforce the scheduling policy to obey the precedence relation . Such result can be achieved by modifi-

tasks	initial period (ms)	#duplicates
τ_1	16	50
τ_2	16	50
τ_3	16	50
τ_4	16	50
τ_5	16	50
τ_6	16	50
τ_7	80	10
τ_8	400	2
τ_9	800	1
τ_{10}	800	1
m_0	16	50

Figure 9. Number of duplicates for the application of figure 8.

cation of the task parameters without creating any duplicate of the initial task . So a perspective of this work is to search new conditions in order to avoid to generalized precedence graph unfolding.

References

[1] N. C. Audsley and A. Grigg. Timing analysis of arinc 629 databuses for real-time applications. *proc. ERA Avioncs Conference*, 1996.

[2] S. Baruah, S. Goddard, and K. Jeffay. Feasibility concerns in pgm graphs with bounded buffers. *proc. Int. Conf. on Engineering of Complex Computer Systems*, pages 130–139, 1997.

[3] S. Baruah, R. Howell, and L. Rosier. Feasibility problems for recurring tasks on one processor. *Theoretical Computer Science*, 118:3–20, 1993.

[4] G. Buttazzo. *Hard Real-Time Computing Systems: Predictable Scheduling, Algorithms and Applications*. Kluwer Academic Publishers, 1997.

[5] F. Cottet, J. Delacroix, C. Kaiser, and Z. Mammeri. *Real-Time Scheduling*. Herms Sciences (in french), 2000.

[6] A. Ermedahl and H. H. M. Sjdin. Response-time garantees in atm networks. *proc. IEEE Real-Time System Symposium*, 1998.

[7] L. George and P. Minet. A fifo worst-case analysis for a hard real-time distributed problem with consistency constraints. *proc. Int. Conf. on Distributed Computing Systems*, pages 441–448, 1997.

[8] J. M. A. Jean-Marie. Timing analysis of scheduling policies: a trajectory based model. *INRIA Research Report 3561*, 1998.

[9] M. Joseph and P. Pandya. Finding response-time in a real-time system. *BCS Computer Journal*, 29(5):390–395, 1986.

[10] M. Klein, T. Ralya, B. Pollak, R. Obenza, and M. G. Harbour. *A practitioner's handbook for real-time system analysis*. Kluwer Academic Publishers, 1993.

[11] J. P. Lehoczky. Fixed priority scheduling of periodic task sets with arbitrary deadlines. *proc. IEEE Real-Time System Symposium*, pages 201–209, 1990.

[12] A. Munier. The basic cyclic scheduling problem with linear precedence constraints. *Discrete Applied Mathematics*, 64:219–238, 1996.

[13] J. C. Palencia, J. Garcia, and M. Harbour. Best-case analysis for improving the worst-case schedulability test for distributed hard real-time systems. *proc. IEEE Euromicro Real-Time Systems*, 1998.

[14] J. C. Palencia and M. Harbour. Schedulability analysis for tasks with static and dynamic offset. *proc. IEEE Real-Time System Symposium*, 1998.

[15] P. Pedro and A. Burns. Worst-case response-time analysis of real-time sporadic traffic in fip networks. *proc. IEEE Euromicro Real-Time Systems*, 1997.

[16] M. Sjodin and H. Hansson. Analysis multimedia in real-time atm networks. *proc. IEEE Real-Time Technology and Application Symposium*, 1998.

[17] M. Sjodin and H. Hansson. Improved response time analysis calculations. *proc. IEEE Euromicro Real-Time Systems*, 1998.

[18] M. Spuri. *Earliest Deadline Scheduling in Real-Time Systems*. PhD Thesis, Scuola Superiore S. Anna, Pisa, 1995.

[19] M. Spuri. Analysis of deadline scheduled real-time systems. *INRIA Reseach Report 2772*, 1996.

[20] M. Spuri. Holistic analysis for deadline scheduled real-time distributed systems. *INRIA Reseach Report 2973*, 1996.

[21] J. A. Stankovic, M. Spuri, K. Ramaritham, and G. Buttazzo. *Deadline Scheduling FOr Real-Time Systems: EDF and Related Algorithms*. Kluwer Academic Publishers, 1998.

[22] J. Sun. *Fixed-Priority End-to-End Scheduling in Distributed Real-Time Systems*. PhD Thesis, University of Illinois at Urbana-Champain, 1997.

[23] J. Sun and J. Liu. Synchronization protocols in distributed real-time systems. *proc. Int. Conf. on Distributed Computing Systems*, 1996.

[24] K. Tindell. *Fixed-Priority Scheduling of Hard Real-Time Systems*. PhD Thesis, University of York, 1994.

[25] K. Tindell, A. Burns, and J. Wellings. Analysis of hard real-time communications. *Real-Time Systems Journal*, 9(2):287–300, 1995.

[26] K. Tindell, A. Burns, and J. Wellings. Calculating controller aera network (can) message response-time. *Control Engineering Practice*, 3(8):163–1169, 1995.

[27] K. Tindell and J.Clark. Holistic schedulability analysis for distributed hard real-time systems. *Microprocessors and Microprogramming*, March 1994.

[28] S. Zhang and A. Burns. Garanteeing synchronous message sets in fddi networks. *Workshop on Distributed Computer Control Systems*, pages 107–112, 1995.