International Journal of Computer Applications (0975 - 8887)
Volume 62 - No. 14, January 2013

Schedulability Analysis of Distributed Real-Time applications under Dependence and Several Latency Constraints

Omar Kermia
CDTA
Algiers, Algeria

ABSTRACT

This paper focuses on the analysis of real-time non preemptive multiprocessor scheduling with precedence and several latency constraints. It aims to specify a schedulability condition which enables a designer to check a priori -without executing or simulating- if its scheduling of tasks will hold the precedences between tasks as well as several latency constraints imposed on determined pairs of tasks. It is shown that the required analysis is closely linked to the topological structure of the application graph. More precisely, it depends on the configuration of tasks paths subject to latency constraints. As a result of the study, a sufficient schedulability condition is introduced for precedences and latency constraints in the hardest configuration in term of complexity with an optimal number of processors in term of applications parallelism. In addition, the proposed conditions provides a practical lower bounds for general cases. Performances results and comparisons with an optimal approach demonstrate the effectiveness of the proposed approach.

General Terms:

Distributed Systems, Real-Time

Keywords:

Real-Time Systems, Multiprocessor Scheduling, Schedulability Analysis, Combinatorial Problems, Latency Constraints

1. INTRODUCTION

Nowadays, computer applications in which computation must satisfy stringent timing constraints are widespread. In such applications, failure to meet the specified deadlines can lead to a serious degradation of the system, and can also result in catastrophic loss of life or property. The increasing of computing requirements leads to the distribution of real-time applications over multi-core platforms. However, in addition to the complexity of parallelizing such applications, system designers are faced to the problem of how to deal with applications parameters in such a way that their temporal constraints are met. Yet, the formalization of the performance of parallelisable applications date to year 1967 with the Amdahl law [2] and which was followed by a large number of works one of them is in [23].
The challenge is to ensure that the real-time requirements of distributed applications are satisfied by providing formal methods. In order to schedule, a scheduling algorithm is required which includes a set of rules defining the execution of tasks at the system runtime. At the same time, it is important to provide a schedulability analysis, which determines, whether a set of tasks with parameters describing their temporal behavior will meet their temporal constraints. The result of such a test is typically a yes or a no. This answer indicates whether, the constraints will be satisfied or not. These schemes and tests demand precise assumptions about task properties, which hold for the entire system lifetime. In addition, a set of processors are available for executing a set of distributed real-time applications or software. Each computing element might be a processor in a multi-processor architecture, a host or a core in a multi-core machine. Without loss of generality, the term 'processor' is used in the present paper instead of the other ones.
In this paper, a theoretical study is performed for solving the problem of analyzing a system of real-time tasks under precedence and several latency constraints. Latency constraints addressed in this work are that imposed by the system designer between predefined pairs among tasks of the application graph. Latency constraints analysis can be used to test, both at design time and for on-line execution, whether the time lapses between tasks pairs executions does not exceed an already specified values and, so, meet their deadlines. It constitutes a serious alternative to extensive testing and simulation by providing analytical latency bounds which contribute considerably in process monitoring and control applications required by real-time performance guarantees.
As it is mentioned previously, the paper is interested in non-preemptive scheduling. This choice is motivated by a variety of reasons including [18]:

— In many practical real-time scheduling problems such as I/O scheduling, properties of device hardware and software either make preemption impossible or prohibitively expensive. The preemption cost is either not taken into account or still not really controlled;
— Non-preemptive scheduling algorithms are easier to implement than preemptive algorithms, and can exhibit dramatically lower overhead at runtime;
— The overhead of preemptive algorithms is more difficult to characterize and predict than that of non-preemptive algorithms. Since scheduling overhead is often ignored in scheduling models, an implementation of a non-preemptive scheduler will be closer to the formal model than an implementation of a preemptive scheduler.

For these reasons, designers often use non-preemptive approaches, even though elegant theoretical results on preemptive approaches do not extend easily to them [4]. Designers also choose directed acyclic graphs (DAG) to model different kinds of structures in mathematics and computer science. Indeed, in many real time systems, applications are developed using DAGs [21]

where vertices represent sequential code segments and edges represent precedence constraints. Throughout the paper, it is explained that the latency constraint is strongly linked to the topology of the applications graph or more accurately to the parts of the graph concerned by latency constraints.
There is a large literature in the real-time community on scheduling tasks on multi-processor architectures. Sporadic and aperiodic real-time tasks are considered in respectively [5] and [6] whereas energy-efficient scheduling is proposed in [17]. In [1] QoS management is proposed and [8] targets to minimize either the overall bandwidth consumption or the required number of cores. However, to our knowledge, schedulability analysis dealing with several latency constraints (as it is defined in this paper) has not been considered. In fact, Among the constraints addressed in real-time scheduling issues, latency constraints are less studied comparing with the periodicity constraint for example [19]. Nevertheless, latency is a major concern in several fields such as in embedded signal processing applications [14] for example. In the literature, most often, authors talk about an end-to-end deadline which ensures that the time lapse from sensors and actuators does not exceed a certain value [16]. The main differences between latency and end-to-end deadline is that latency constraints are as much as system designer wants meaning that they can be imposed between any pair of connected tasks in the system (not necessarily sensor and actuator tasks only). In [11], a definition of this constraint is given and the existence of a link between deadlines and latency is proven. In addition, distributed architectures involve inter-processor communications the cost of which must be taken into account accurately. Furthermore, concerning synchronization cost reduction, the approach proposed in [9] is efficient in term of finding a minimal set of interprocessor synchronization, however, this approach assumes that some dependence can be removed even though data are exchanged. Moreover, it is not suitable for latency constraints satisfaction because it imposes a tasks scheduling not exploiting the potential tasks parallelism which is essential in minimizing their total execution time. Moreover, it was not possible to exploit results from parallelism community, essentially because of precedence constraints which are not taken into account [10].
The main contributions of this paper are the proposition of a schedulability conditions for latency constraints in the hardest configuration with an optimal number of processors in terms of application parallelism. This configuration stands for the hardest configuration among the other possible configurations because of the interdependence of latency constraints. Also, from these conditions, practical lower bounds for latency constraints values were deduced, the efficiency and the rapidity of which were showed by evaluation tests.
The paper is organized as follows: Section 2 introduces the model and defines the latency constraint. Section 3 introduces the schedulability analysis through the different possible cases. Section 4 describes the performance evaluation.

2. DEFINITIONS AND MODEL

The paper deals with systems of real-time tasks with precedence and several latency constraints. A task t_i is characterized by a worst case execution time (WCET) $C(t_i) \in \mathbb{N}$. The precedences between tasks are represented by a directed acyclic graph (DAG) denoted $\mathcal{G}$ such that $\mathcal{G} = (\mathbb{V}, \mathbb{E})$. $\mathbb{V}$ is the set of tasks characterized as above, and $\mathbb{E} \subseteq \mathbb{V} \times \mathbb{V}$ the set of edges which represent the precedence (dependence) constraints between tasks. Therefore, the directed pair of tasks $(t_a, t_b) \in \mathbb{E}$ means that t_b must be scheduled, only if t_a was already scheduled and t_a is called a predecessor of t_b. The set of tasks belonging to all paths from t_a to t_b including t_a and t_b is denoted by $\mathbb{V}'$. Note that the architecture plate-form is composed of identical processors.
A communication cost is involved when dependent tasks are scheduled on two processors, whereas, the communication cost is considered to be negligible if dependent tasks are scheduled on the same processor. In our study the overall communication overhead involved by the interaction between processors is taken into account. If $\mathcal{M}$ is the function of time needed for communication then $\mathcal{M}$ can vary linearly with the number of processors: $\mathcal{M}(m) = Q.(m-1)$ where Q is a constant dependent on the architecture and stands for an average communication cost between a pair of processors and m is the number of processors. In addition, $\mathcal{M}$ can, also, vary logarithmically since communications can be designed in order to get a logarithmic impact on the total execution time. For example, communications can be parallelized in the case of hierarchical topology architectures and function $\mathcal{M}$ becomes $\mathcal{M}(m) = Q.\log m$. Nevertheless, it is important to notice that in targeted applications, granularity is chosen in such a way to get high computation to communication ratio. Because, when the granularity is large the computation cost becomes dominant and the relatively small (but non-negligible) communication cost actually encourages the use of more processors to help the reduction of scheduling time. This implies more opportunity for performance increase but, nevertheless, involves hard efficient load balancing [7].
Each task t_i has a start time $S(t_i)$ determined by the scheduling algorithm. A latency constraint is defined only between two tasks connected in the tasks graph which means that it exists at least one path connecting the two tasks. By imposing a latency constraint $L(t_a, t_b)$, the time elapsing from the execution start of t_a and the execution start of t_b must be less or equal than an integer denoted also by $L(t_a, t_b)$ and which is already known. As in the graph tasks t_a and t_b are connected by one or several paths, hence, $\boldsymbol{\mathcal{P}(t_a, t_b)}$ denotes the set of paths $\boldsymbol{p_i}$ which connect t_a to t_b. Hence, $\mathcal{P}(t_a, t_b)$ is also a set of sets of tasks meaning that $t_i \in (p_j \in \mathcal{P}(t_a, t_b))$.
The length of p_i is denoted by $\boldsymbol{|p_i|}$ such that $|p_i| = \sum_{t_j \in p_i} C(t_j)$. Among paths p_i, $\boldsymbol{lp}$ denotes the longest one.
More formally, a latency constraint $L(t_a, t_b)$ is met if and only if:

$$S(t_b) - S(t_a) \leq L \tag{1}$$

In the tasks graph of the figure 1 $\mathcal{P}(t_1, t_7)$=$\{p_1, p_2, p_3, p_4, p_5, p_6\}$ such that: p_1 = $\{t_1, t_2, t_3, t_4, t_5, t_6, t_7\}$, p_2= $\{t_1, t_8, t_9, t_4, t_5, t_6, t_7\}$, p_3=$\{t_1, t_2, t_3, t_4, t_5, t_{10}, t_7\}$, p_4= $\{t_1, t_8, t_9, t_4, t_5, t_{10}, t_7\}$, p_5=$\{t_1, t_2, t_{11}, t_4, t_5, t_6, t_7\}$, p_6=$\{t_1, t_2, t_{11}, t_4, t_5, t_{10}, t_7\}$ and p_7=$\{t_1, t_2, t_{11}, t_6, t_7\}$.

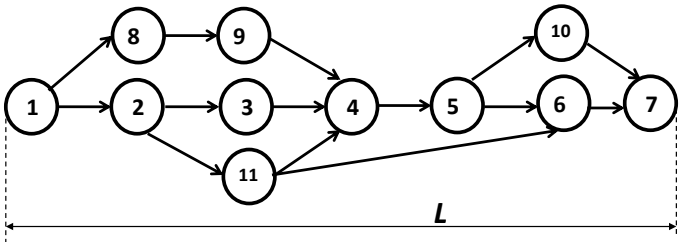

Fig. 1: Tasks under latency constraint

3. SCHEDULABILITY STUDY

The studied problem is close to the problem "P | prec | C_{max}" (using Lenstra's 3-fields notation [15]) which is known to be NP-hard [15]. The "P | prec | C_{max}" problem aims to minimize maximum completion time of all tasks whereas the objective is to determine the schedulability of the graph tasks by findingl whether

a scheduling of all tasks of the graph on a multiprocessor platform, satisfying the precedence and latency constraints, exists or not. Consequently, our problem in a one latency case is also NP-hard. Moreover, in the several latency constraints case, the problem becomes NP-hard in the strong sens because of links between latency constraints .
Since the studied problem is NP hard, no algorithm can resolve it in a polynomial time (unless NP=P) and this is, also, true for the schedulability condition. This means that, in a general case, it is impossible to propose a necessary and sufficient condition allowing to check if a set of tasks under a latency constraint is schedulable or not in a polynomial time.

3.1 One latency Constraint Case

The matter of dealing with a latency constraint is closely linked to the structure of the graph. That is the reason why a partitioning method is proposed considering graph paths. Without loss of generality, in the present paper it is considered that the whole graph is under the latency constraint $L(t_a, t_b)$ which means that the considered graph has one root vertice t_a and one leaf vertice t_b (see figure 1). In the case of graphs with large tasks and edges numbers, the number of paths is also very large. However, determining all paths is not an NP hard problem [25]. Besides, according to [24], it exists several approaches for determining all paths of a graph, among which the topological sort of the graph can be mentioned. However, in practice, the number of paths is less than the number of vertices in a graph. Even in a simple design with a small quantity of components, the number of vertices in $\mathcal{G}$ is more than 10 times the number of paths in the architecture [22].
The allocation algorithm (Algorithm 1) has as inputs all paths of the graph and as outputs the selection of some of them which, each one, will be associated to a distinct processor. First, the algorithm begins by sorting paths in $\mathcal{P}(t_a, t_b)$ according to a decreasing order of their lengths then it selects them one by one and it allocates paths tasks to a processor to which it is associated. After that, at each step, tasks belonging to a path p_i and which were not allocated before via another path (the case of tasks belonging to several paths) will be allocated to the processor to which p_i is associated. The algorithm stops when all tasks under a latency constraint are allocated meaning that all paths will not be necessarily selected.
As a result, each task of the application graph will be allocated to only one processor. Also, an integer m is returned equivalent to the number of selected paths which returns the number of required processors. In other words, Algorithm 1 parallelizes the execution of the application by allocating its tasks to a set of processors. Besides, this parallelization follows the configuration of paths which compose the application graph.

Algorithm 1 Allocation Algorithm

1: $m \leftarrow 0$
2: Sort paths in $\mathcal{P}$ in a decreasing order of length
3: Select lp and initialize a set of tasks $\Phi = lp$
4: **while** $\Phi \neq \mathbb{V}'$ **do**
5: For each path p_i not already selected :
$\lambda(p_i) = \sum_{t_j \in p_i \wedge t_j \notin \Phi} C(t_j)$
6: Select p_i such that $\lambda(p_i) = \max(\lambda)$
7: $\Phi = \Phi \cup p_i$ (include p_i's tasks in Φ)
8: $m \leftarrow m + 1$
9: **end while**

An example of Algorithm 1 application is illustrated in figure 2. Processors P_1, P_2 and P_3 were required whereas seven paths were detected (see example of section 2). For this example it is assumed that the execution times of tasks are equal. From now

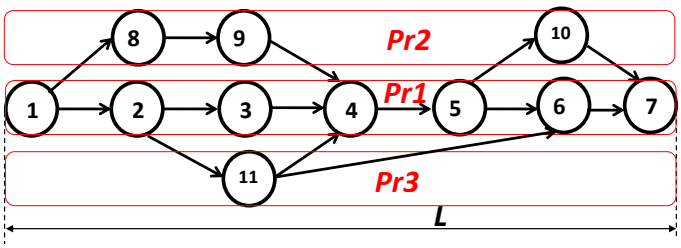

Fig. 2: Paths Allocation

the set of paths $\mathcal{P}(t_a, t_b)$ is considered composed of m paths (the ones selected by Algorithm 1). Also, we notice by $\widehat{p_i}$ the set of tasks exclusively belonging to p_i, more formally, if $t_i \in \widehat{p_i}$ then $\forall p_j \in \mathcal{P}(t_a, t_b) \setminus p_i, t_i \notin p_j$.
One can ask what makes the number of tasks returned by Algorithm 1 so distinctive. The answer is that the value of m represents the optimal number of processors since it allows to exploit the total parallelism inherent to the application graph. This means that if two tasks are not linked by a path in the graph (no one is the predecessor or the successor of the other) then they are allocated to distinct processors. Moreover, Adding other processors than the m processors required by Algorithm 1 does not improve the exploitation of the parallelism inherent to the application graph. Proposition 1 introduces the optimality of m.

PROPOSITION 1. *The application of Algorithm 1 on an application graph returns the optimal number of processors allowing the task parallelism exploitation.*

Proof Algorithm 1 allocates tasks according to paths to which they belong. Notice that the considered paths are those which include, at least, a task which does not belong to any other path. Let assume that for a given graph G algorithm 1 returned m processors. Also, let assume that, it exists a number of processors m‘ such that m‘ $< m$ for which the exploitation of the parallelism of the graph G is optimal. This means that each pair of tasks (t_i, t_j) not linked by a path in G are allocated to two distinct processors among the m‘ processors. As assessed earlier, the graph G has only one root task t_a and only one leaf task t_b and, hence, it exist two distinct paths which link t_a and t_b and include t_i for the first and t_j for the second (This is due to the fact that t_i and t_j are not linked). This implies that all distinct paths in G will be concerned. Consequently, $(m - m$‘$)$ processors are missing in order to parallelize all pairs (t_i, t_j) $\square$
From now on, Algorithm 1 is systematically applied to allocate tasks.
The following proposition introduces a necessary and sufficient schedulability condition in the case of one latency constraint.

PROPOSITION 2. *Let L be a latency constraint imposed on the tasks pair (t_a, t_b). Latency constraint $L(t_a, t_b)$ is met if and only if:* $\forall p_j \in \mathcal{P}(t_a, t_b)$,

$$\sum_{t_i \in \bigcap p_j} C(t_i) + \max_{p_j}\Big(\sum_{t_i \in \widehat{p_j}} C(t_i)\Big) + \mathcal{M}(m) \leq L \qquad (2)$$

Proof this result is quite intuitive and can be obtained by examining the inequality $S(t_b) - S(t_a) \leq L$. Indeed $S(t_b) - S(t_a)$ which is the scheduling time of tasks under latency constraint L is equal to the sum of execution times of:

(1) Tasks which are non-parallelisable with any other tasks (sequential tasks which are linked by a path in the application graph). These are represented by tasks shared between all paths in $\mathcal{P}(t_a, t_b)$ $(t_i \in \{\bigcap p_j\})$,

(2) Among parallel tasks, the longest sub-path is selected from the m paths. On each processor m_i tasks of the set $\widehat{P}_i$ are allocated and the largest sum of executions time of tasks of each $\widehat{P}_i$ is kept. This is due to the precedence between tasks which prevents of distributing parallel tasks between processors to get a more balanced distribution such as $\frac{\sum_{t_i \in \mathcal{V}'} C(t_i)}{m}$ ($\mathcal{V}'$ is the set of tasks which are in parallel in the graph application),

(3) Communication overhead□

3.2 Several Latency Constraints Case

In [12], authors have stated that all possible combinations for two pairs of tasks under, each one, a latency constraint can be covered by three cases:

— In parallel, when there is no path linking tasks under the first latency constraint to those under the second latency constraint.

— In Z, when there is one (or more) path linking tasks under the first latency to those under the second latency or vice versa.

— In X, there is one (or more) path linking tasks under the first (resp. second) latency to those under the second (resp. first) latency.

For the Z and parallel relations the schedulability study can be performed as for the one latency case. This statement issues from the fact that latency constraints in these cases can be addressed one after the other in order to check the schedulability of the whole system. In addition, the X configuration is the hardest one to be studied because the two latency constraints are dependent. In fact, satisfying one of these latencies is not related to the scheduling of tasks under this constraint only but it is related, also, to some tasks which are under other latency constraints. Usually, in this case, it is about multi-objective optimization and the problem becomes harder than in a single optimization case [13].
Let's take an example of a tasks graph subject to a pair of latency constraints in X. The figure 3 depicts a pair of latency constraints L_1 and L_2 in X imposed between (t_1, t_4) and (t_9, t_{11}).

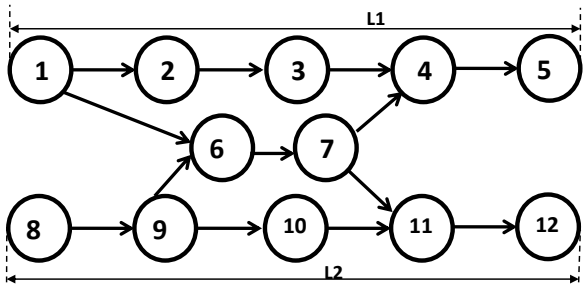

Fig. 3: A pair of latency constraints in X

The following proposition introduces a necessary and sufficient schedulability condition in the case of two latency constraints in X.

PROPOSITION 3. *Let (L_1, L_2) be two latency constraints in X imposed, respectively, on tasks pairs (t_a, t_b) and (t_c, t_d). Latency constraints $L_1(t_a, t_b)$ and $L_2(t_c, t_d)$ are met if and only if:*

(*1*) *Condition of proposition 2 is met for tasks under L_1 and m_1 processors and for tasks under L_2 and m_2 processors*

(*2*) *and*

$$\begin{cases} \max\limits_{p_i \in \mathcal{P}(t_c, t_b)} |p_i| + \mathcal{M}(m) \leq L_1 \\ \max\limits_{p_i \in \mathcal{P}(t_a, t_d)} |p_i| + \mathcal{M}(m) \leq L_2 \end{cases} \quad (3)$$

m, m_1 and m_2 are obtained by applying Algorithm 1 on the graph under latency constraints L_1 and L_2. m_1 is the number of processors to which tasks under L_1 are allocated, m_2 the number of ones to which tasks under L_2 are allocated and m represents all required processors. Notice that $m < m_1 + m_2$ because there exist tasks under the two latency constraints.

Proof As expected, the one latency case schedulability condition (condition 2) becomes a necessary condition in the case of two latency constraints in X. Indeed, if one of the two latency constraints is not met then all the system is considered as non-schedulable. Then, in order to prove the sufficiency of the condition proposed here, equations (3) is assumed as satisfied, and constraints L_1 and L_2 are, nevertheless, not met. The constraints L_1 and L_2 are not met means that $S(t_b) - S(t_a) > L_1$ and $S(t_d) - S(t_c) > L_2$.

$S(t_b) - S(t_a) > L_1$ means that:
Either,
$\exists p_i \in \mathcal{P}(t_a, t_b),\ |p_i| + \mathcal{M}(m_1) > L_1$. This hypothesis is in contradiction with the condition 2 because:
$(2) \Rightarrow \forall p_i \in \mathcal{P}(t_a, t_b),\ |p_i| \leq L_1$

Or,
as t_c is a predecessor of the task t_b, hence, the start execution of t_b is related to the execution of t_c and other tasks which are under the latency constraint L_2. Therefore, in the present case, the start execution of t_b is delayed by the execution of tasks under latency constraint L_2 whereas all predecessor tasks of t_b under latency constraint L_1 were executed. This is, more formally, described by the following inequality:
$\exists t_x \in (\mathcal{P}(t_a, t_b) \cap \mathcal{P}(t_c, t_d))$,

$$\max_{p_i \in \mathcal{P}(t_c, t_x)} |p_i| + \mathcal{M}(m_2) < \max_{p_i \in \mathcal{P}(t_a, t_x)} |p_i| + \mathcal{M}(m_1) \quad (4)$$

Furthermore,
$S(t_b) - S(t_a) > L_1$ and (4) $\Rightarrow$
$\exists p_j \in \mathcal{P}(t_c, t_x)$ and $\exists p_k \in \mathcal{P}(t_x, t_b)$,

$$|p_j| + \mathcal{M}(m_1) + |p_k| + \mathcal{M}(m_2) > L_1 \quad (5)$$

Otherwise, it is clear that:

$$|p_j| + \mathcal{M}(m_1) + |p_k| + \mathcal{M}(m_2) \leq \max_{p_i \in \mathcal{P}(t_c, t_b)} |p_i| + \mathcal{M}(m) \quad (6)$$

from condition 3, equation 5 is in contradiction with equation 6.

The same reasoning can be followed to prove that $(S(t_c) - S(t_d) > L_2)$ is in contradiction with the assumption that the constraint L_2 is met□

The result of proposition 3is easily generalizable to a tasks graph subject to n latency constraints, two by two, in X configuration. Indeed, It suffices to check conditions of proposition 3 for each pair of latency in X then to conclude the schedulability of the whole system. So, using results of propositions 2 and 3 any application graph can be dealt with whatever the number of imposed latency constraints is and whatever these latency constraints are configured.

The schedulability study performed earlier introduces schedulabilty conditions over a processors number which stands for the optimal number to exploit all the parallelism inherent to the application graph, but the proposed conditions does not fit a system with a static architecture (i.e., the number of processors is

known beforehand and fixed). When system designers face such systems, they tend towards fast analysis methods even thought these methods are not as exact as optimal methods. So, knowing that the targeted problem is NP-hard in the strong sens the schedulability analysis of such systems throughout optimal approaches or even heuristics takes a very long time. Instead of an optimal schedulability analysis, conditions the paper proposes practical lower bounds for latency constraints values L_i whatever the number of processors is. Hence, system designers can refer to the proposed conditions to adjust the latency constraints values while saving a considerable time. The following proposition introduces lower bounds for latency constraints values according to the different configurations.

PROPOSITION 4. *1. if L is a latency constraint imposed on the tasks pair (t_a, t_b). The lower bound of $L(t_a, t_b)$ is:*

$$L^{lb} = \sum_{t_i \in \bigcap p_j} C(t_i) + \max_{p_j}\Big(\sum_{t_i \in \widehat{p_j}} C(t_i)\Big) + \mathcal{M}(m) \quad (7)$$

2. If (L_1, L_2) are two latency constraints in X imposed, respectively, on tasks pairs (t_a, t_b) and (t_c, t_d). The lower bounds of $L_1(t_a, t_b)$ and $L_2(t_c, t_d)$ are:

$$L_1^{lb} = \mathbf{max}\Big(\sum_{t_i \in \bigcap p_j}^{p_j \in \mathcal{P}(t_a,t_b)} C(t_i) + \max_{p_j \in \mathcal{P}(t_a,t_b)}\Big(\sum_{t_i \in \widehat{p_j}} C(t_i)\Big) + \mathcal{M}(m_1), \max_{p_j \in \mathcal{P}(t_c,t_b)} |p_j| + \mathcal{M}(m)\Big) \quad (8)$$

$$L_2^{lb} = \mathbf{max}\Big(\sum_{t_i \in \bigcap p_j}^{p_j \in \mathcal{P}(t_c,t_d)} C(t_i) + \max_{p_j \in \mathcal{P}(t_c,t_d)}\Big(\sum_{t_i \in \widehat{p_j}} C(t_i)\Big) + \mathcal{M}(m_2), \max_{p_j \in \mathcal{P}(t_a,t_d)} |p_j| + \mathcal{M}(m)\Big)$$

Proof
L^{lb} represents a lower bound to the scheduling time between t_a and t_b $(S(t_b) - S(t_a))$. This means that the value that system designer will give to $L(t_a, t_b)$ must not be lower than L^{lb} otherwise the latency will necessarily be not met. As high computation applications are targeted, the use of more processors involves the reduction of scheduling time. Reciprocally the reduction of the number of processors will increase the scheduling time. This proves that l^{lb} in the different seen configurations is a minimum of scheduling time for systems where the number of processors is less than m.
In addition, as m represents the optimal number of processors to get the optimal parallelism within the application graph, the fact of using more processors than m processors does not lead to reduce the scheduling time□

3.3 Performance Evaluation

In order to evaluate the performances of applying the schedulability condition of proposition 3 we implemented an application designated as the proposed approach which, for a given graph of tasks under a pair of latency constraints in X, checks conditions of proposition 3 and outputs, following the obtained result, the schedulability of the system. Then, two kinds of tests are performed:

— an evaluation of time performances of the proposed solution,
— a comparison with solutions provided by the constraint programming approach.

Tasks graphs (DAGs) used for the evaluation were generated randomly according to the two following parameters: number of tasks and density. In our case the graph density is a ratio between the number of edges in the graph and the number of possible edges (in the complete graph). For example, a graph of 12 tasks with 0.5 density has 33 edges whereas a complete graph of 12 tasks has 66 edges. Notice that the number of edges in a complete graph is $\frac{n(n-1)}{2}$, where n is the number of tasks.
Inside the graph, 40 % of tasks is put under the constraint L_1 and 40 % under the constraint L_2. Next, the remaining 20% are put under the two constraints L_1 and L_2. An example of a generated graph with 12 tasks and 0.25 of density (17 edges) is given in figure 4: 5 tasks are exclusively under the constraint L_1, 5 other tasks are exclusively under L_2 and 2 tasks are under both of L_1 and L_2.
In the generated graph the number of edges is determined by the density (as explained in the previous paragraph) whereas the configuration of these edges is defined randomly as follows:

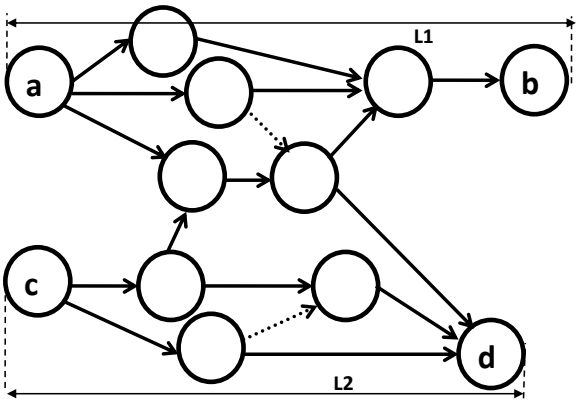

Fig. 4: Example of generated 12 tasks graph

— a set of randomly generated edges within the restriction of ensuring the X configuration of latency constraints (the edges in continued line in the graph of figure 4),
— a set of randomly generated edges between tasks under the same latency constraints (the edges in discontinued line in the graph of figure 4) and which satisfy the DAG properties of the graph.

The first test concerns time performances of the proposed approach functions of the graph's number of tasks and the graph's density. The diagram of figure 5 depicts the evolution of the runtime by a 3d curve. It showed that the increasing density has a more important impact, than those of the number of graph tasks, on the runtime of the proposed approach. This is mainly explained by the fact that the number of paths increases when the graph has a higher density. Moreover, the runtime of the proposed approach are very reasonable even when the density is hight. Notice that the runtime follows a logarithmic scale and results were collected on a machine with a 3,4 GHz Intel Core i7 processor and 10GB main memory.
The second test targets the efficiency of the proposed approach in term of schedulability and lower bounds. To do so, we chose to use the constraint programming for resolving the latency constraints scheduling problem and to compare the obtained results to the proposed approach results.
The constraint programming is a programming language that is oriented to relationships or constraints among entities [3]. The most important reason is that constraint programming has a rich modeling language which is very convenient to express the problem. Moreover, the underlying CP solver is relatively robust with respect to the addition of new constraints, and the search can be controlled entirely by the user.
Our problem was solved using ILOG OPL Studio commercial software according to the following CP formulation. The objective is to minimize the scheduling of tasks under L_1 by minimizing the start time of t_b and in the same time minimizing the

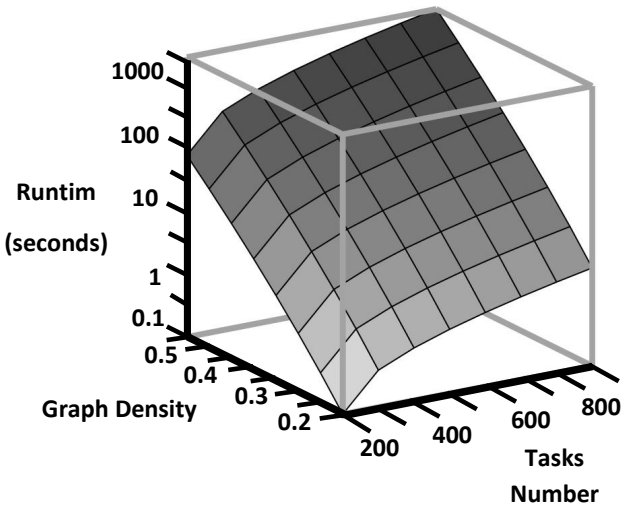

Fig. 5: Proposed approach runtime evolution

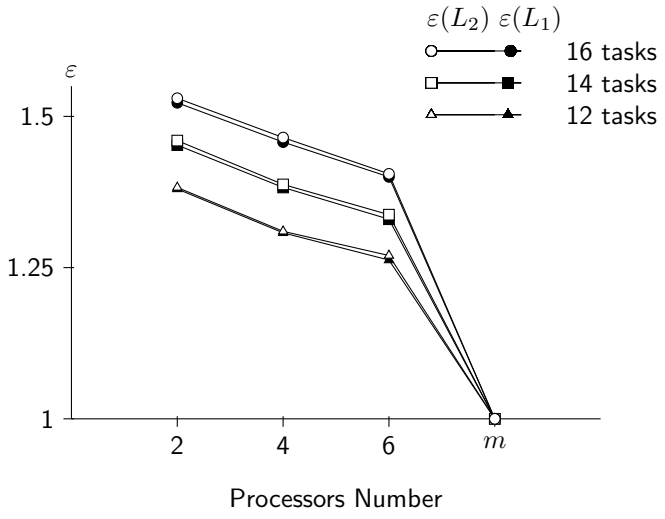

Fig. 6: Proposed approach schedulability performances

Table 1. : Definition of Variables and Domains

Variable	Domain
NbTasks	$\mathbb{N}^+$
NbProcs	$\mathbb{N}^+$
duration(t_i)	$\mathbb{N}^+$
task($t_i, proc_j$)	[StartOf(t_i), EndOf(t_i)] $\subset \mathbb{N}^+$

Table 2. : Definition of Constraints

Constraint	Description
· if $(t_i, t_j) \in \mathbb{E}$ then EndOf(t_i) $\leq$ StartOf(t_j)	· t_j is a predecessor of t_i
·$\forall t_i \in \mathbb{V}, \forall proc_j$, alternative(task($t_i, proc_j$))	· each task needs only one processor to be executed
· $\forall proc_i$, noOverlap($proc_i$)	· no overlap on processors

scheduling of tasks under L_2 by minimizing the start time of t_d (knowing that latency constraint are imposed on (t_a, t_b) and (t_c, t_d)). Hence, the multiple objectives are expressed in a single objective by summing them together and applying weights to each objective to signify its relative importance. It was assessed, first, that the two objectives have the same importance and, consequently, the same weight. But, the runtime of CP approach exploded, even for small graphs. Hence, CP approach minimizes L_1 first then L_2. Thus, the objective function is:

$$\text{Min } (x * \text{StartOf}(t_b) + y * \text{StartOf}(t_d))$$

Where $(x, y) = (1, 0)$ then $(x, y) = (0, 1)$. In addition, variables domains and constraints are given in table 1 and 2. Constraints of table 2 are provided by ILOG OPL Studio for scheduling modeling [20]. The number of processors is defined by Algorithm 1.

To do so, within the CP approach the objective was to look for the scheduling which minimizes the start dates of t_b and t_d then to compute the values of L_1^{opt} = StartOf(t_b) and L_2^{opt}=StartOf(t_d). These values are the optimal (smallest) values that L_1 and L_2 can have. Then, they were compared to the values of L_1^{lb} and L_2^{lb} resulting from the calculation of equations 8.

After that, the value of ρ is computed which the ratio between L_i^{opt} and L_i^{lb} such that $\rho(L_i) = \frac{L_i^{opt}}{L_i^{lb}}$ in order to get an idea of how far are the proposed approach results from the optimal ones. For each case of the tasks number list [12,14,16] until 20 different graphs were generated and both approaches were applied on them. Notice that the chosen density of all tested graphs was 0.4. At the beginning, the two approaches were executed on a m processors architecture (m is given by Algorithm 1). After that, the number of processor was reduced and fixed from the list [4,3,2], and only the optimal approach was executed. Notice that the proposed approach cannot be execute since it fixes itself the number of processors. Results are illustrated by diagrams on figure 6 ($\rho(L_1)$ is marked in black and $\rho(L_2)$ is in white).

As expected, ρ is equal to 1 when the number of processors is equal to m meaning that our approach as the optimal approach return the optimal latency values in the case of m processors. After that, once the number of processors was reduced, ρ values increase meaning that the values returned by CP is larger than L_i^{lb} and L_i^{lb} which confirm their positions of lower bounds. Notice that the values of ρ increase, also, from the first set of tests (12 tasks) to the second set of tests (14 tasks) then increase again in the third set of tests (16 tasks). This is explained by the fact that when the number of tasks increase, the number of paths follows and it leads to increase the value of the optimal number of processors m.

In addition, it emerges that the proposed approach provides an interesting results considering that, among the three sets of tests, optimal approach returned results varying from 1.25 and 1.5 times the proposed approach results. This means that, on all performed experiments, the proposed approach gives a value to L_i which is, at worst, around 1.5 times smaller than the one given by the CP approach. Hence, the proposed lower bounds can be considered as efficient seeing the difference between runtimes of the two approaches. The light difference between $\rho(L_1)$ and $\rho(L_2)$ is explained by the fact that in CP approach priority is given to the minimization of (StartOf(t_b)) at the cost of minimization of (StartOf(t_d)). As with any other optimal method, runtime of CP approach explodes exponentially as soon as the number of tasks becomes more important which prevented us to consider more than 16 tasks graphs.

4. CONCLUSION

The paper presents a theoretical study of the real-time non preemptive multiprocessor scheduling with precedence and several latency constraints. After assessing the NP-hardness of this problem, an algorithm is proposed for allocating application graph tasks to a number of processors allowing the optimal task parallelism exploitation. The schedulability study, proposed here, introduces a first condition in the case of one latency constraint. Then, after giving the different possible configurations in the case of several latency constraints, it introduces a second condition to check the schedulability of latency constraints in the hardest configuration in term of complexity. Finally, from the proposed conditions a practical lower bounds were deduced.
The first phase of tests demonstrates that the proposed approach has a very competitive runtime. In addition, the second phase concerned a comparison with an optimal approach which is the Constraint Programming approach. These tests showed that the proposed approach provides an interesting results in term of schedulability and lower bounds.
The performed study assumes that the number of processors is at least equal to the number of paths selected by the allocation algorithm. Hence, it is plan to explore the possibilities of including the number of processors in the schedulability condition as a fixed parameter.

5. REFERENCES

[1] Luca Abeni, Tommaso Cucinotta, Giuseppe Lipari, Luca Marzario, and Luigi Palopoli. Qos management through adaptive reservations. Real-Time Systems, 29(2-3):131–155, 2005.

[2] G. M. Amdahl. Validity of the single-processor approach to achieving large scale computing capabilities. In AFIPS Conference Proceedings, volume 30, pages 483–485. AFIPS Press, 1967.

[3] Krzysztof Apt. Principles of constraint programming. 2003.

[4] F. Balarin, L. Lavagno, P. Murthy, and A. Sangiovanni-vincentelli. Scheduling for embedded real-time systems. IEEE Design and Test of Computers, 15(1):71–82, 1998.

[5] Sanjoy K. Baruah and Joel Goossens. The edf scheduling of sporadic task systems on uniform multiprocessors. In IEEE Real-Time Systems Symposium, pages 367–374, 2008.

[6] Sanjoy K. Baruah and Giuseppe Lipari. Executing aperiodic jobs in a multiprocessor constant-bandwidth server implementation. In ECRTS, pages 109–116, 2004.

[7] Lawrence Livermore National Laboratory Blaise Barney. Introduction to parallel computing. Web, 2010.

[8] Giorgio C. Buttazzo, Enrico Bini, and Yifan Wu. Partitioning real-time applications over multicore reservations. IEEE Trans. Industrial Informatics, 7(2):302–315, 2011.

[9] H-Yi Chao and M P. Harper. Minimizing redundant dependencies and interprocessor synchronizations. International Journal of Parallel Programming, 23:245–262, 1994.

[10] Tommaso Cucinotta. Optimum scalability point for parallelisable real-time components. In Proceedings of the International Workshop on Synthesis and Optimization Methods for Real-time and Embedded Systems (SOMRES 2011), Vienna, Austria, November 2011.

[11] L. Cucu, N. Pernet, and Y. Sorel. Periodic real-time scheduling: from deadline-based model to latency-based model. Annals of Operations Research, 2007.

[12] L. Cucu and Y. Sorel. Non-preemptive scheduling algorithms and schedulability conditions for real-time systems with precedence and latency constraints. (RR-5403):33, 2004.

[13] Christian Glasser, Christian Reitwiessner, Heinz Schmitz, and Maximilian Witek. Approximability and hardness in multi-objective optimization. In Proceedings of the Programs, proofs, process and 6th international conference on Computability in Europe, CiE'10, 2010.

[14] S. M. Goddard and Jr. On the management of latency in the synthesis of real-time signal processing systems from processing graphs, 1998.

[15] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Ronnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. In Annals of Discrete Mathematics, 1979.

[16] Chih-wen Hsueh and Kwei-jay Lin. Scheduling real-time systems with end-to-end timing constraints using the distributed pinwheel model. IEEE Transactions on Computers, 49(1):51–66, 2000.

[17] Kai Huang, Jian-Jia Chen, and Lothar Thiele. Energy-efficient scheduling algorithms for periodic power management for real-time event streams. In RTCSA (1), pages 83–92, 2011.

[18] K. Jeffay, D. F. Stanat, and C. U. Martel. On non-preemptive scheduling of periodic and sporadic tasks. In Proceedings of the 12 th IEEE Symposium on Real-Time Systems, pages 129–139, December 1991.

[19] O. Kermia. Optimizing distributed real-time embedded system handling dependence and several strict periodicity constraints. Advances in Operations Research, page 10.1155/2011/561794, 2011.

[20] Philippe Laborie. Ibm ilog cp optimizer for detailed scheduling illustrated on three problems. In Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, Lecture Notes in Computer Science. 2009.

[21] Cong Liu and James H. Anderson. Supporting graph-based real-time applications in distributed systems. Real-Time Computing Systems and Applications, International Workshop on, 1:143–152, 2011.

[22] Yuchun Ma, Zhuoyuan Li, Jason Cong, Xianlong Hong, G. Reinman, Sheqin Dong, and Qiang Zhou. Micro-architecture pipelining optimization with throughput-aware floorplanning. In Proceedings of the 2007 Asia and South Pacific Design Automation Conference, 2007.

[23] Yuan Shi. Reevaluating amdahl's law and gustafson's law. Technical report, Temple University, Philadelphia, PA 19122, October 1996.

[24] F Tutzauer. Entropy as a measure of centrality in networks characterized by path-transfer flow. Social Networks, 29(2), 2007.

[25] S. V. N. Vishwanathan, N. Schraudolph, R. Kondor, and K. Borgwardt. Graph kernels. Journal of Machine Learning Research, 11:1201–1242, 2010.