A FIFO worst case analysis for a hard real-time distributed problem with consistency constraints *

L. George, P. Minet
Reflecs project, INRIA
78153 Rocquencourt cedex, France

Abstract

A solution for a hard real-time scheduling problem in a distributed system is designed and proved. The constraints of our problem are first to preserve consistency even in presence of concurrency and second to preserve the order of tasks releases, provided that tasks release times differ more than clock precision. That is achieved by a FIFO based scheduling. The feasibility conditions resulting from the worst-case response time analysis of each task set are given. The solution complexity is shown to be pseudo-polynomial.

1 Introduction

In this article we are concerned with hard real-time scheduling preserving consistency in a distributed system. Hard real-time means that a violation of real-time constraints by any task may have severe consequences (human lives endangered, financial losses). Real-time constraints are expressed as follows: any task τ_i released at time t must be at the latest executed by $t+D_i$, where D_i is called the relative deadline and $t+D_i$ the absolute deadline. In such a system, it is required to prove that real-time tasks meet their deadlines. That proof must be given at design time and in any case before implementation time [1]. We are more particularly interested in scheduling a set of non-concrete periodic or sporadic tasks reading/writing distributed objects bound by consistency constraints. Moreover all the executions produced must preserve consistency and conflicting tasks whose release times differ more than the clock precision must be executed chronologically. We establish the feasibility conditions of a task set by providing the worst case response time of each task and showing that it is less than the relative deadline. In section 2, we define the scheduling problem more precisely. In the uniprocessor case, FIFO scheduling is an obvious solution. The solution, given in section 3 for the distributed case, makes use of FIFO scheduling based on the release times of tasks so that the scheduling order of tasks (on the servers) reflects the release order (on the clients). The section 4 shows how it is possible to derive from a uniprocessor scheduling analysis, worst case response times of tasks in a distributed system. First the classical concepts and known results are given in sub-section 4.1. FIFO scheduling is studied in the uniprocessor context (sub-section 4.2). Sub-section 4.3 shows how to take into account distribution first considering perfect global time then removing this assumption, clients and servers clocks being synchronized within the clock precision. Finally, a complexity analysis is proposed : the general problem can be solved in pseudo polynomial time.

2 The Scheduling Problem

We investigate the problem of scheduling a set $\tau = \{\tau_1, \ldots, \tau_n\}$ of n tasks in a distributed system. All tasks τ_i, $\forall i \in [1, n]$ are independent and are critical (hard real-time) : all of them must be completed by their absolute deadline. Furthermore tasks are assumed to be periodic or sporadic. Sporadic tasks were formally introduced in [2] (although already used in some papers, e.g., [3]) and differ only from periodic tasks in the release time: the $(k+1)^{th}$ instance of a periodic task τ_i is released at time $t_i^{k+1} = t_i^k + T_i$, where T_i stands for the period of the task, while that instance is released at $t_i^{k+1} \geq t_i^k + T_i$ if the task is sporadic. Hence T_i represents the minimum interarrival time between two successive release times for sporadic tasks. We do not assume any relation between the periods T_i and the relative deadlines D_i of the tasks.

A task τ_i, $\forall i \in [1, n]$, is released by a client upon receipt of an external/internal event. We can assume

without loss of generality that a task τ_i, $\forall i \in [1,n]$, is always released by the same client (tasks can be identified to ensure that). A task is executed on one or more distributed servers. The set of servers is represented by $\{s_1, \ldots, s_K\}$. In a real-time system where servers can fail by crashing, high criticity tasks are replicated. The replication degree is a function of the task criticity. One solution is to structure each task in independent subtasks, each subtask being executed by a server. The subtask to server mapping is assumed to be given. We know for each task τ_i and for each server s_j executing a subtask of τ_i, the duration of that subtask on s_j as C_i^j. Notice that even if the subtasks are the same, the processing times C_i^j can differ because servers can be heterogeneous. The overhead due to context switching and scheduling is considered to be included in the duration of subtasks. A subtask can access persistent local objects stored on that server. Objects are bound by consistency constraints. Each task, taken alone, preserves consistency constraints.

The transmission delay between clients and servers has a known upper bound Max. Both servers and clients have synchronized clocks, such that the difference between any two clocks is not higher than the precision ε. Furthermore in this study, we do not consider network failures, however a bounded number of omissions can be taken into account in Max.

The real-time application requires the following properties of the scheduling algorithm :

- *(P1) Tasks in τ whose release times differ more than clock precision are scheduled chronologically (ie. according to their release time order).*
- *(P2) Any execution of tasks in τ preserves consistency.*
- *(P3) All tasks in τ meet their deadlines.*

Property (P1)can be required for instance by applications concerned with human computer interaction. Moreover property (P1) ensures that causal dependencies between tasks are preserved by scheduling, provided that the interarrival time of two tasks, causally related, is higher than ε.

3 The adopted scheduling algorithm

3.1 Which Scheduling Policy

Let us first solve that problem in the uniprocessor case, where there is no transmission delay ($Max = 0$) and there is only one processor clock ($\varepsilon = 0$). Scheduling theory applied to hard real-time systems has been widely studied in the last twenty years. A lot of results have been achieved for the problem of non-idling scheduling over a single processor. The most effective real-time schedulers make use of the HPF (Highest Priority First) on-line policy, two approaches are identified: fixed priority driven schedulers [4], [5], [6], [7], [8] that were designed for easy implementation, and dynamic priority driven schedulers that were considered better theoretically [4], [9], [10], [11], [8], [12]. In the uniprocessor non idling class, an optimal dynamic priority driven scheduler has been extensively studied: the Earliest Deadline First algorithm (EDF). EDF is optimal in the class of uniprocessor non idling scheduling either in preemptive and non-preemptive context [13], [3], [14], [8]. However neither EDF, nor a fixed priority scheduling meets property (P1), which is obviously met by FIFO (First in First Out) scheduling.

Our purpose is now to show how FIFO scheduling in the uniprocessor case can be extended to the distributed case. First let us prove that the simplest extension does not work. Let us consider the following example where servers s_1 and s_2 execute two conflicting tasks τ_i and τ_j, (ie accessing the same object in read/write or write/write mode) in a different order. On the left side, τ_i (resp. τ_j) is received on s_1 (resp. s_2) after a transmission delay $< Max$. A short time later, (right side of figure 2), τ_j (resp. τ_i) is received on s_1 (resp. s_2) after a transmission delay of Max. Hence a violation of property (P2).

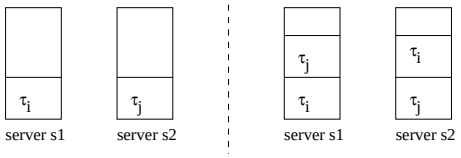

Figure 1: Different orderings of conflicting tasks τ_i and τ_j by s_1 and s_2.

Hence the tasks receipt order on the servers cannot be used to achieve (P2). It is the same for any local order. That is why a global order [15] must be applied by all servers. Servers use FIFO scheduling based on the release time of the tasks on the clients.

3.2 The Scheduling Algorithm

Each server uses two waiting queues :

- the Pending Queue (P.Q) contains the pending tasks (tasks received from the clients) not already scheduled. P.Q is sorted according to the release time of the tasks.
- the Running Queue (R.Q) contains all scheduled tasks not entirely executed. R.Q is served FIFO.

We can now give the basic principles of the scheduling algorithm:

- A client, upon receipt of an external/internal event releases the associated task τ_i and multi-

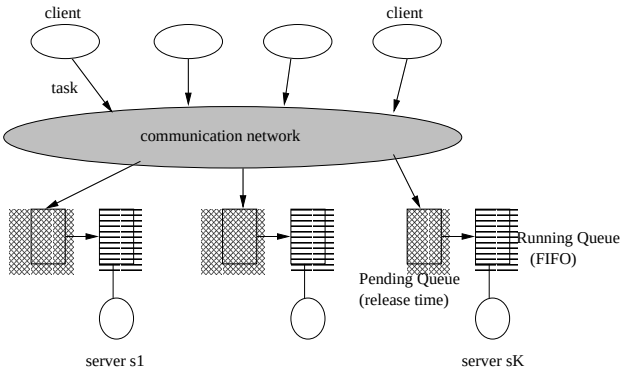

Figure 2: Distributed system model.

casts τ_i, timestamped with its release time t_i, to the servers involved by τ_i.

- When a server receives (from a client) a task τ_i released at time t_i, it inserts it in its P.Q. The rank of τ_i in P.Q depends on t_i (P.Q is sorted by increasing release times).
- At local time $t_i + Max + \varepsilon$, the server moves τ_i from its P.Q to the end of its R.Q. If several tasks have the same release time, they are inserted in R.Q according to their deadline (Deadline Monotonic), (and then if they have the same deadline, according to their client identifier).
- A server executes the first task of its R.Q. When that task is locally completed, it extracts it from its R.Q.

3.3 Properties of the Scheduling Algorithm

Let us prove that this algorithm meets the properties (P1), (P2) and (P3). The demonstration of property (P3) is done in section 4.

Property (P1) is met.

Proof: for any given task τ_i, all the involved servers have the knowledge of the same release time t_i, the one given by the client. At time $t_i + Max + \varepsilon$, a server knows all the tasks (it has to execute) and that have been released at a time $t \leq t_i$. Indeed according to the model described in section 2, the maximum transmission delay is Max, but as the servers clocks and the client clocks are synchronized within ε, a server is sure that a delay Max has elapsed for task τ_i released at time t_i when at its clock it is $t_i + Max + \varepsilon$. At this time, it will never more receive tasks released before t_i. Otherwise those tasks would have suffered a transmission delay higher than Max: a contradiction with our model. Consequently at time $t_i + Max + \varepsilon$ the task can be put in the running queue. Notice that because of non perfect clocks, the release times given by the clients can differ from the release times in the absolute time.

We can show that if two releases differ more than ε in the absolute time, the associated tasks are scheduled according to their release time order as seen in the absolute time. Let us assume that an external observer first sees the release of task τ_i and after a delay $d > \varepsilon$ the release of task τ_j. The release times t_i and t_j given by the clients always meet $t_i < t_j$ even if clock of client releasing task τ_i is in advance of ε with regard to clock of client releasing task τ_j. $\diamond$

However tasks in τ whose release times are closer than clock precision are scheduled in an arbitrary order but the same for all servers. The worst case identified in sub-section 4.3 takes into account that behavior.

Property (P2) is met.

Proof: in the absence of failure, serialisability has been proved [15] to be a sufficient condition for preserving consistency, each task taken alone preserving consistency. As all servers order conflicting tasks in the same way, the resulting execution produces the same results as a serial execution according to the release times (given by the clients) order. Hence consistency constraints are met. $\diamond$

Property (P3) is met if the task set meets the feasibility conditions given in sub-section 4.3.

Proof: see the next section (section 4). $\diamond$

4 Real-time Scheduling : Worst Case Analysis

In this chapter we show how to establish feasibility conditions and response times. In the literature, two approaches can be identified to solve a problem of feasibility in hard real-time scheduling. First depending on the context (preemptive, non-preemptive, the scheduling parameters considered ,...) and on the scheduling policy, we can exhibit:

a) a simple (polynomial time) algebric condition to check [4].

b) if not possible (because of intrinsic complexity), we first try to determine the worst case(s) scenario(s). Each scenario is examined on a certain interval to compute the response time of each task. The aim is to reduce the length of each interval.

The approach used in this paper is approach b). For that purpose, we first show how a uniprocessor analysis of FIFO scheduling can be extended to compute the worst-case response times in a distributed context. Let us notice that the considered tasks have the following properties:

- (P4) because of the tasks structure, the execu-

tion of a task on a server is not delayed by the execution of any other task on any other server.

- (P5) the response time of a task once placed in R.Q does not depend on the order of the tasks already in R.Q (obvious from (P4)).

From (P4) it follows that we can reason on every server independently. A task is thus feasible if and only if it is feasible on every involved server.

4.1 Classical concepts and known results

Let us now recall some classical concepts used in hard real-time scheduling:

- The worst case response time of a task τ_i is denoted r_i.
- The first release time of task τ_i is denoted t_i^0. A task is said to be concrete if its release time is known a priori.
- The scheduling of a concrete task set ω is said to be valid if and only if no task instance ever misses its absolute deadline. A concrete task set ω is said to be feasible with respect to a given class of schedulers if and only if there is at least one valid schedule that can be obtained by a scheduler of this class. Similarly, a non-concrete task set τ is said to be feasible with respect to a given class of schedulers if and only if every concrete task set ω that can be generated from τ is feasible in this class [14]. In this article, we only consider non-concrete task sets that are more general and more realistic in our context.
- A scheduler is said to be optimal with respect to a given class of schedulers if and only if it generates a valid schedule for any feasible task set in this class.
- Time is discrete (task releases occur and tasks executions begin and terminate at clock ticks; the parameters used are expressed as multiples of the clock tick); in [16], it is shown that there is no loss of generality with respect to feasibility results by restricting the schedules to be discrete, once the task parameters are assumed to be integers (multiple of the clock tick) i.e a discrete schedule exists if and only if a continuous schedule exists.

We now recall classical concepts used in the uniprocessor context for periodic or sporadic tasks. A task τ_i is defined in that context by a maximum duration C_i, a deadline D_i and a period T_i. We will show in sub-section 4.3 how to use those concepts in the distributed case.

The busy period notion provides a powerful way to characterize either fixed or dynamic priority non-idling schedulers as it identifies activity periods of the processor. The aim of the analysis given in 4.2 and 4.3 is to identify the worst case busy periods (where the maximum response time of any task can be obtained).

- An idle time t is defined on a processor as a time such that there are no tasks released before time t pending at time t. An interval of successive idle times is called an idle period.
- A busy period is defined as a time interval $[a, b)$ such that there is no idle time in $]a, b)$ (the processor is fully busy) and such that both a and b are idle times.
- Given a non-concrete task set, the synchronous busy period is defined as the time interval $[0, L)$ delimited by two distinct idle times in the schedule of the corresponding synchronous concrete task set [11]. It turns out that the value of L does not depend on the scheduling algorithm, as far as it is non-idling, but only on the task arrival pattern. L is called the length of the synchronous busy period.
- Given a critical instant at time 0 ($\forall i \in [1, n], t_i^0 = 0$), the workload $W(t)$ is the amount of processing time requested by all tasks whose release time are in the time interval $[0, t)$ [9]: $W(t) = \sum_{i=1}^{n} \lceil t/T_i \rceil C_i$.
- $U = \sum_{i=1}^{n} C_i/T_i$ is the processor utilization factor, i.e., the fraction of processor time spent in the execution of the task set [4]. An obvious necessary condition for the feasibility of any task set is $U \leq 1$ (this is assumed in the sequel).
- The value of the synchronous busy period L can be computed using the following recursive equation: $L^{k+1} = W(L^k)$ where $L^0 = \sum_{i=1}^{n} C_i$. The recursion ends when $L^{k+1} = L^k = L$.
- Let $R_i = \{R_{i,t_i^0}, t_i^0 \in [0, T_i - 1]\}$ where R_{i,t_i^0} denotes the scenario such that task τ_i is first released at time $t_i^0 \in [0, T_i - 1]$ and all other tasks at time 0.
- Let $B_i = \{B_{i,t_i^0}, t_i^0 \in [0, T_i - 1]\}$ where B_{i,t_i^0} is the length of the first busy period of the scenario $R_{i,t_i^0}, t_i^0 \in [0, T_i - 1]$. In sub-section 4.3.2 we show how to compute B_i.

4.2 The uniprocessor case

We now consider the problem of scheduling n sporadic tasks on a uniprocessor. In that case there is only one subtask per task. We prove that:

- the worst case response time for any task is obtained when tasks are released at their maximum rate i.e periodically (see lemma 1).
- the longest busy period is the synchronous busy period (see lemma 2).
- the worst case response time is reached in the first busy period of these scenarios (see lemma 3).

Notice that these three lemma do not depend on the scheduling algorithm used. The following two lemma explicitly depend on the scheduling algorithm:

- the optimality of the fixed tie-breaking rule, deadline monotonic, used with FIFO (see lemma 4).
- the response time of a task in a given scenario (see lemma 5).

The following lemma shows that for any scheduling algorithm, for any task τ_i in a busy period *bp*, the worst case occurs for τ_i when all the other tasks are released synchronously, at the begining of the busy period *bp*. This interesting property will enable us to restrict the number of scenarios needed to compute the response time of task τ_i.

Lemma 1- *The worst case response time of a task τ_i, $i \in [1,n]$, is found in a busy period where all tasks are at their maximum rate and all tasks τ_k, $k \neq i \in [1,n]$ are released synchronously at the beginning of the busy period.*

Proof: Let us consider a busy period containing an instance of task τ_i, we set time 0 at the beginning of the busy period. Let t_k^0, be the first time task τ_k, $k \neq i$ is released in the busy period. If you consider the scenario where all tasks are at their maximum rate and t_k^0 is changed to 0 then all the instances of task τ_i see either the same number or a greater number of tasks scheduled before them. From property (P5), it follows that their response time is either increased or left unchanged. Thus if the deadline of task τ_i is ever missed then it is necessarily missed in a busy period where all tasks are at their maximum rate and all tasks but τ_i are released synchronously at the beginning of the busy period. $\diamond$

Hence for task τ_i, we only need to consider the scenarios where all the other tasks are synchronous. Notice that this property is valid for any scheduling algorithm. We now focus on the busy period analysis. This notion is an interesting notion that enables to limit the length of the scenarios tested to compute the response time of the tasks. This notion has already been studied by [5], [17], [8], [12]. In lemma2, we show that the longest busy period is the synchonous busy period L. Thus the length of any tested scenario is bounded by L: a first improvement.

Lemma 2- *The longest busy period is the synchronous busy period L*

Proof: Suppose there exists a busy period different from the synchronous busy period and longer than the synchronous busy period. Let time 0 be the beginning of that busy period. Let t_k^0 be the first time task τ_k, $k \in [1,n]$ is released in the busy period. If $\forall k \in [1,n]$, t_k^0, is changed to 0 then the resulting busy period can only see more or the same number of tasks and thus its size cannot decrease invalidating the former hypothesis. $\diamond$

We now show how to lower the number of scenarios needed to compute the response time of τ_i.

Lemma 3- *The worst case response time of a task τ_i, $i \in [1,n]$, is obtained for at least one instance of τ_i released at time $t_i \in [0,L)$ in the first busy period of a scenario of R_i.*

Proof: Suppose that there exists a scenario R^* such that an occurrence of τ_i released at time t_i misses its deadline. Consider the last busy period including the task missing its deadline. Set time 0 to the beginning of that busy period. Let t_k^0 be the first time task τ_k, $k \in [1,n]$ is released in R^*. From lemma 1, setting $\forall k \neq i$, $t_k^0 = 0$ will produce a scenario $R_{i,t_i^0} \in R_i$ where task τ_i still misses its deadline, t_i^0 being the time of the first occurrence of τ_i in R_{i,t_i^0}. From its definition, R_{i,t_i^0} is necessarily a busy period as R^* is a busy period and R_{i,t_i^0} contains a number of activations before t_i larger or equal. From lemma 2 we necessarily have $t_i \in [0,L)$. $\diamond$

We then show that when some tasks are released at the same time, an optimal fixed tie breaking rule is Deadline Monotonic [6] (i.e the task with the shortest deadline is scheduled first). Such a scheduling is denoted FIFO/DM.

Lemma 4- *FIFO/DM is optimal in the class of FIFO scheduling*

Proof: We prove this by the classical interchange argument. Let F be the fixed tie breaking rule associated with the FIFO scheduling, the resulting scheduling is called FIFO/F. Tasks in τ are ordered by decreasing priority according to F. Suppose τ feasible by FIFO/F scheduling. Let *bp* be a busy period (time=0 at the beginning of *bp*) such that for two consecutive tasks τ_i and τ_k, $(k = i+1)$, $D_i > D_k$. Suppose that τ_i and τ_k are released at the same time. According to F, τ_i is executed first. As τ is feasible, the execution of task τ_k is performed before $t + D_k < t + D_i$. Suppose that the fixed priority of tasks τ_i and τ_k in F are interchanged then it is easy to show that the resulting scheduling is still feasible. Indeed, the workload in $[0,t)$ is unchanged after the interchange. Applying

this interchange argument to all the tasks in any busy period, we prove the lemma. $\diamond$

In the following, we assume, without loss of generality, FIFO/DM scheduling. We show how to compute the response time of a task τ_i released at time t in a scenario $R_{i,t_i^0} \in R_i$. The analysis given hereafter can be adapted to any fixed tie breaking rule by replacing $H(D_i - D_k)$ by $H(Priority(\tau_i) - (Priority(\tau_k))$ with: $H(Priority(\tau_i) - (Priority(\tau_k)) = 1$ if $Priority(\tau_k) \geq Priority(\tau_i)$, and 0 otherwise.

Lemma 5- *The response time $r_{i,t}$ of a task τ_i released at time t in the first busy period of the scenario $R_{i,t_i^0} \in R_i$ with $t_i^0 = t - \lfloor t/T_i \rfloor T_i$, is given by: $r_{i,t} = w_{i,t} - t$ where $w_{i,t}$ is the termination time of task τ_i with $w_{i,t} = \sum_{k=1}^{n} \lceil (t + H(D_i - D_k))/T_k \rceil C_k$, with $H(x) = 1$ if $x \geq 0$ and 0 otherwise.*

Proof: Let τ_i be a task released at time t according to the scenario $R_{i,t_i^0} \in R_i$ with $t_i^0 = t - \lfloor t/T_i \rfloor T_i$. Let $w_{i,t}$ be the termination time of task τ_i. According to FIFO scheduling used with the optimal fixed tie breaking rule, a task τ_k, $k \neq i$ released at time t_k has a higher priority than task τ_i if:

$t_k < t + 1$ when $D_k \leq D_i$

$t_k < t$ when $D_k > D_i$

The maximum number of instances of task τ_k with a priority higher than task τ_i is at most $\lceil (t + H(D_i - D_k))/T_k \rceil$. For task τ_i, all activations in $[t_i^0, t]$ must be considered. Hence the number of activations is exactly $1 + \lfloor (t - t_i^0)/T_i \rfloor = 1 + \lfloor t/T_i \rfloor = \lceil (t+1)/T_i \rceil$ (because of discrete scheduling). As task τ_i is released in a busy period, it terminates exactly at time $w_{i,t} = \sum_{k=1}^{n} \lceil (t + H(D_i - D_k))/T_k \rceil C_k$. Hence $r_{i,t} = w_{i,t} - t$. $\diamond$

The set B_i containing the length of the first busy period for every scenario R_{i,t_i^0} can be computed recursively (see sub-section 4.3.2).

Theorem 1- *The worst case response time r_i of a task τ_i is given by: $r_i = max_t(r_{i,t})$ with $t \in [0, B_{i,t_i^0}) \cap \{t_i^0 + pT_i, p \in N\}$ and $t_i^0 \in [0, T_i - 1]$.*

Proof: From lemma 4, the worst response time of task is obtained in at least one of the busy period of a scenario R_{i,t_i^0}, $t_i^0 \in [0, T_i - 1]$. In such a scenario, task τ_i is first released at time t_i^0 and is then periodic. Thus computing $r_{i,t}$ for every release time of task τ_i in the first busy period of the scenarios in R_i will lead to the worst response time of task τ_i. $\diamond$

Theorem 2- *A general task set $\tau = \{\tau_1, \ldots, \tau_n\}$ is feasible, using FIFO/DM if and only if: $\forall i \in [1, n]$, $r_i \leq D_i$ and $U \leq 1$.*

The procedure Response-time which computes the worst case response time of a task τ_i is only given in the distributed case (see sub-section 4.3.2). We now focus on distribution.

4.3 The distributed case

In this section, we extend the worst-case response time analysis developed in the uniprocessor case to the distributed case. We now reason on a server s_j and the notation used in the uniprocessor case must be adapted on s_j as follows:

- C_i^j the duration on s_j of the sequence of τ_i.
- $w_{i,t}^j$ is the workload duration, on server s_j, until completion of task τ_i released at time t.
- r_i^j is the worst case response time of τ_i on s_j.
- U^j is the processor utilisation on server s_j. We necessarily have : $\forall j \in [1, K]$, $U^j \leq 1$.
- $B_{i,t_i^0}^j$ is the length on server s_j of the first busy period of scenario R_{i,t_i^0}.
- L^j is the length of the synchronous busy period on s_j.

4.3.1 Accounting for transmission delays and clock precision

Effect on the servers: Network delay and servers clock precision. When distribution is considered, a task τ_i released at time t on a client is received by a server s_j involved by τ_i, at a variable time $t + network\_delay$ such that the $network\_delay \leq Max$. And as the clock precision is ε, this task has to wait until time $t + Max + \varepsilon$ before being inserted in R.Q. The worst-case analysis given in 4.2 must be adapted. The response time of any task τ_i released at time t on a client, must account for the network delay and the clock precision on server s_j, $r_{i,t}^j = Max + \varepsilon + w_{i,t}^j - t$.

Effect on the clients: clients clock precision. The clock precision on the clients can be treated with the release jitter approach [7], [18], [19]. Indeed as the clients timestamp the tasks with their release time, for two tasks released on two clients at the same global time, there can be a difference in values $\leq \varepsilon$. A priority inversion may occur and must be accounted for in the worst case analysis. This can be expressed as a release jitter. A task τ_i released at local time t_i on client1 can be delayed by another task τ_k released on any client2 distinct from client1 at global time t (at client2 time $t - \varepsilon$) with $t \leq t_i + \varepsilon$ if:

- there is a server involved both in τ_i and in τ_k.
- the clock of client2 is ε late compared to the clock of client1.

The clock precision on the clients has two impacts:

- First, the formula giving $w^j_{i,t}$ for server s_j and task τ_i released at time t becomes: $w^j_{i,t} = \sum_{k=1}^{n} \lceil (t+\varepsilon_k+H(D_i-D_k))/T_k \rceil C^j_k$ where $\varepsilon_k = \varepsilon$ if $\tau_k \supset s_j$ and the client releasing τ_k is not the one releasing τ_i, $\varepsilon_k = 0$ otherwise. The transmission delay is taken into account in $r^j_{i,t}$, with $r^j_{i,t} = Max + \varepsilon + w^j_{i,t} - t$.
- Second, for the length of busy periods B^j_i we must use the new formula of $w^j_{i,t}$ to compute all the busy periods lengths.

The feasibility of the problem can thus be treated applying procedure *Response_time*(τ_i, s_j) (given in section 4.3.2) on every server s_j involved by τ_i to compute the worst case response time on s_j. Then the maximum response time is given by: $r_i = max_{j\in[1,K]}(r^j_i)$, with $\tau_i \supset s_j$. Then an obvious necessary and sufficient condition for the general problem is:

Theorem 3- *A general task set* $\tau = \{\tau_1, \cdots, \tau_n\}$ *is feasible in a distributed system, using FIFO/ DM if and only if:* $\forall i \in [1,n]$, $\forall j \in [1,K]$ *such that* $\tau_i \supset s_j$, *we have* $r^j_i \leq D_i$ *and* $U^j \leq 1$.

4.3.2 Algorithmic solution

The length of the busy period B^j_{i,t^0_i} on the server s_j in the scenario R_{i,t^0_i} is computed by the function $Busy(R_{i,t^0_i}, s_j)$ using the following recursive equations $\forall t^0_i = T_i - 1$ down to 0, where by convention, $B^j_{i,T_i} = \sum_{k=1,k\neq i}^{n} C^j_k$.

At step 0, $B^j_{i,t^0_i}(0) = max(b, B^j_{i,t^0_i+1})$ where $b = \sum_{k=1}^{n} C^j_k$ if $t^0_i = 0$ and $b = 0$ otherwise.

At step $p \geq 1$, we have (ε_k is defined in section 4.3.1) $B^j_{i,t^0_i}(p) = \sum_{k=1,k\neq i}^{n} \lceil (B^j_{i,t^0_i}(p-1) + \varepsilon_k)/T_k \rceil C^j_k + max(0, \lceil (B^j_{i,t^0_i}(p-1) - t^0_i)/T_i \rceil) C^j_i$. The recursion ends when $B^j_{i,t^0_i}(p) = B^j_{i,t^0_i}(p-1) = B^j_{i,t^0_i}$. It can be shown that B^j_{i,t^0_i} exists: for any $p \in N$, we have $B^j_{i,t^0_i}(p) \geq B^j_{i,t^0_i}(p-1)$ and $B^j_{i,t^0_i}(p) \leq L^j$.

The procedure *Response_time*(τ_i, s_j) written in pseudo-code shows how to compute the worst case response time of task τ_i on server s_j. This procedure begins with $t^0_i = T_i - 1$ down to 0. Indeed, one can show that $B^j_{i,T_i-1} \leq B^j_{i,T_i-2} \leq \cdots \leq B^j_{i,0}$. Notice that if $\exists t^0_i$ such that $B^j_{i,t^0_i} \leq t^0_i$, then all the scenarios $R_{i,t}$, $t \in [B^j_{i,t^0_i}, t^0_i]$ do not need to be tested on server s_j. Indeed if $B^j_{i,t^0_i} \leq t^0_i$, an idle time has been reached on server s_j before time t^0_i and from lemma 3 adapted to the distributed context, we do not need to compute the response time of τ_i anyfurther.

Response_time(τ_i, s_j)
Begin
  $r^j_i = C^j_i + Max + \varepsilon$
  $t^0_i = T_i - 1$ $B^j_{i,T_i} = \sum_{k=1,k\neq i}^{n} C^j_k$
  While $t^0_i \geq 0$
    /* computation of the busy period */
    $B^j_{i,t^0_i} = Busy(R_{i,t^0_i}, s_j)$
    If $B^j_{i,t^0_i} \leq t^0_i$ then
      /* Idle period before the first release of τ_i */
      $tmp = B^j_{i,t^0_i}$ $t^0_i = B^j_{i,t^0_i} - 1$ $B^j_{i,t^0_i+1} = tmp$
    End if
    $t = t^0_i$
    While $t < B^j_{i,t^0_i}$
      /* cf lemma 5 adapted to distribution */
      $r^j_i = max(r^j_i,$
        $\sum_{k=1}^{n} \lceil \frac{t+\varepsilon_k+H(D_i-D_k)}{T_k} \rceil C^j_k + Max + \varepsilon - t)$
      $t = t + T_i$
    End While
    $t^0_i = t^0_i - 1$
  End While
End

5 Complexity analysis

The aim of this section is to show that the complexity of the problem is pseudo polynomial in the number of computation of the task response times. In section 4, we show that for any task τ_i, on every server s_j involved by τ_i, we must compute the response time of τ_i in the scenarios $R_i = \{R_{i,t^0_i}, t^0_i \in [0, T_i - 1]\}$. Each scenario R_{i,t^0_i} has on server s_j a length B^j_{i,t^0_i} bounded by L^j the length of the synchronous busy period on server s_j. We have $L^j = max_{i\in[1,n],\tau_i\supset s_j}(B^j_{i,0})$. The result of [8] can be extended in our context to show that the complexity of the computation of L^j is in $O(n\frac{\sum_{i=1}^{n} C^j_i}{min_{i=1}^{n} C^j_i})$ when there exists a constant α^j such that $U \leq \alpha^j < 1$. For a task τ_i first released at time $t^0_i \in [0, T_i - 1]$, the number of task instances in the corresponding busy period $[0, B^j_{i,t^0_i})$ is given by: $\lceil (B^j_{i,t^0_i} - t^0_i)/T_i \rceil$. Thus the theoretical number of computation of response time for τ_i is given by: $\sum_{t^0_i=0}^{T_i-1} \lceil (B^j_{i,t^0_i} - t^0_i)/T_i \rceil$. We have to do such computations for any server s_j, and for any task τ_i involving s_j (denoted $\tau_i \supset s_j$). There are K servers. We then obtain: $\sum_{j=1}^{K} \sum_{i=1,\tau_i\supset s_j}^{n} \sum_{t^0_i=0}^{T_i-1} \lceil (B^j_{i,t^0_i} - t^0_i)/T_i \rceil$, which is bounded by: $\sum_{j=1}^{K} \sum_{i=1,\tau_i\supset s_j}^{n} T_i \lceil L^j/T_i \rceil \leq nK\, max_{i\in[1,n]}(T_i) \lceil max_{j\in[1,K]}(L^j)/min_{i\in[1,n]}(T_i) \rceil$.
Hence the complexity of the problem is pseudo polynomial.

6 Conclusion

In this paper, we have established a necessary and sufficient condition for the feasibility in a hard real-time distributed system, of a task set with a FIFO/DM scheduling based on task release times. Such an algorithm has the advantage of preserving consistency. Moreover conflicting tasks are ordered according to the chronological order of their release times (as observed by an external observer), if their release times differ more than the clock precision. Conflicting tasks whose release times are closer than the clock precision are ordered in an arbitrary order, the same for all servers. These results have to be extended to more complex task structures, (e.g synchronization constraints between servers). An open problem is to determine how those results can be adapted to the more general case of fork and join graphs since the general problem is NP-Hard.

References

[1] G. Le Lann, "A methodology for designing and dimensioning critical complex computing systems", *IEEE Int. symposium on the Engineering of Computer Based Systems*, Friedrichshafen, Germany, pp 332-339, March 1996.

[2] A.K. Mok, "Fundamental Design Problems for the Hard Real-Time Environments", *MIT Ph.D. Dissertation*, May 1983.

[3] Kim, Naghibdadeh, "Prevention of task overruns in real-time non-preemptive multiprogramming systems", *Proc. of Perf., Assoc. Comp. Mach.*, pp 267-276, 1980.

[4] C.L Liu, J. W. Layland, "Scheduling Algorithms for multiprogramming in a Hard Real Time Environment", *Journal of the Association for Computing Machinery*, Vol. 20, No 1, Jan. 1973.

[5] J.P. Lehoczky, "Fixed priority scheduling of periodic task sets with arbitrary deadlines", *Proc. 11th IEEE Real-Time Systems Symposium*, Lake Buena Vista, FL, USA, pp. 201-209, 5-7 Dec. 1990.

[6] N. C. Audsley, "Optimal priority assignement and Feasibility of static priority tasks with arbitrary start times", *Dept. Comp. Science Report* YCS 164, University of York, 1991.

[7] K. Tindell, A. Burns, A. J. Wellings, "An extendible approach for analyzing fixed priority hard real time tasks", *Real-Time Systems*, Vol.6, pp. 133-151, 1994.

[8] L. George, N. Rivierre, M. Spuri, "Preemptive and Non-Preemptive Real-Time Uniprocessor Scheduling" INRIA Rocquencourt, France, *Research Report* 2966, Sept. 1996.

[9] S. K. Baruah, A. K. Mok, L. E. Rosier, "Preemptively Scheduling Hard-Real-Time Sporadic Tasks on One Processor", *Proc. of the 11th Real-Time Systems Symposium*, pp. 182- 190, 1990.

[10] S. K. Baruah, R. R. Howell, L. E. Rosier, "Feasibility Problems for Recurring Tasks on one processor", *Theoret. Comput. Sci.*,Vol. 118, pp. 3-20, 1993.

[11] D. I. Katcher, J. P. Lehoczky, J. K. Strosnider, "Scheduling models of dynamic priority schedulers", *Research Report* CMU CDS-93-4, Carnegie Mellon University, Pittsburgh, April 1993.

[12] M. Spuri, "Analysis of deadline scheduled real-time systems", *Research Report* 2772, INRIA, France, Jan. 1996.

[13] M. Dertouzos, "Control Robotics: the procedural control of physical prosessors", *Proc. of the IFIP congress*, pp 807-813, 1974.

[14] K. Jeffay, D. F. Stanat, C. U. Martel, "On Non-Preemptive Scheduling of Periodic and Sporadic Tasks", *IEEE Real-Time Systems Symposium*, San-Antonio, pp. 129-139, Dec. 1991.

[15] P. A. Bernstein, V. Hadzilacos and N. Goodman, *Concurrency Control and Recovery in Database Systems*, Addison Wesley Pub., ISBN 0-201-10715-5, 370 p., 1987.

[16] S. K. Baruah, R. R. Howell, L. E. Rosier, "Algorithms and Complexity Concerning the Preemptive Scheduling of Periodic Real-Time tasks on one processor", *Real-Time Systems*, Vol.2 , pp. 301-324, 1990.

[17] I. Ripoll, A. Crespo, A.K. Mok, "Improvement in Feasibility Testing for Real-Time Tasks," *Real-Time Systems*, Vol.11, pp. 19-39, 1996.

[18] K. Tindell, A. Burns, A. J. Wellings, "Analysis of hard real-time communications", *Real-Time Systems*, Vol.9, pp. 147-171, 1995.

[19] K. Tindell, "Holistic Schedulability Analysis for Distributed Hard Real-Time Systems", *Euromicro Journal*, Special Issue on Embedded Real-Time Systems, Feb. 1995.