Sufficient FTP Schedulability Test for the Non-Cyclic Generalized Multiframe Task Model

Vandy Berten
Université Libre de Bruxelles
Brussels, Belgium

Joël Goossens
Université Libre de Bruxelles
Brussels, Belgium

October 27, 2011

1 Introduction

In this paper we consider the schedulability of the Non-Cyclic Generalized Multiframe Task Model System Model defined by Moyo et al. [7] (NC-GMF in short). Moyo et al. provided a sufficient EDF schedulability test, Baruah extended the model by considering the Non-cyclic recurring task model [1] and provided an exact feasibility test. Stigge et al. further extend this model and consider Digraph real-time task model [8], and consider the feasibility (or, equivalently, the EDF-schedulability) of their task model.

Formally, an NC-GMF task set \( \tau \) is composed of \( M \) tasks \( \{ \tau_1, \ldots, \tau_M \} \). An NC-GMF task \( \tau_i \) is characterized by the 3-tuple \(( C_i, D_i, T_i )\) where \( C_i \), \( D_i \), and \( T_i \) are \( N \)-ary vectors \([ C_1^i, C_2^i, \ldots, C_N^i ]\) of execution requirement, \([ D_1^i, D_2^i, \ldots, D_N^i ]\) of (relative) deadlines, and \([ T_1^i, T_2^i, \ldots, T_N^i ]\) of minimum separations, respectively. Additionally each task makes an initial release not before time-instant \( O_i \).

The interpretation is as follows: each task \( \tau_i \) can be in \( N_i \) different configurations, where configuration \( j \) with \( j \in \{1, \ldots, N_i\} \) corresponds to the triplet \(( C_j^i, D_j^i, T_j^i )\). We denote by \( \ell_i^k \in \{1, \ldots, N_i\} \) the configuration of the \( k \)-th frame of task \( \tau_i \). The first frame of task \( \tau_i \) has an arrival time \( a_1^i \geq O_i \), an execution requirement of \( C_1^i \), an absolute deadline \( a_1^i + D_1^i \) and a minimum separation duration of \( T_1^i \) with the next frame. The \( k \)-th frame has an arrival time \( a_k^i \geq a_{k-1}^i + T_{k-1}^i \), an execution requirement of \( E_k^i \), an absolute deadline \( a_k^i + D_k^i \) and a minimum separation duration of \( T_k^i \) with the next frame.

It is important to notice that \( \ell_i^k \) can be any value in \( \{1, \ldots, N_i\} \) for \( i \in \{1, \ldots, M\} \) and \( k \in \mathbb{N}_0^+ \). I.e., the exact order in which the different kinds of jobs are generated is not known at design-time, it is non-cyclic. In this way this model generalizes the GMF model [2].

In this research we consider the scheduling of NC-GMF using Fixed Task Priority (FTP) schedulers and without loss generality \( i < j \) implies that \( \tau_i \) has a higher priority than
\section{Critical Instant}

First notice that the popular critical instant notion (see Definition 1) is not really applicable in our task model.

\textbf{Definition 1.} A critical instant of a task \(T_i\) is a time instant such that: the job of \(T_i\) at this instant has the maximum response time of all job of \(T_i\).

This notion is difficult to extend to non-cyclic model in the sense that the workload (task requests) is not unique, contrary to the Liu and Layland or the (cyclic) GMF task model.

Anyway, the critical instant is important because it provides the worst-case scenario from \(\tau_i\) standpoint, in the next section we will show that the synchronous case remains the worst case for the more general model namely the NC-GMF.

\textbf{Definition 2.} Let us consider an NC-GMF task set \(\{\tau_i : i \in [1, \ldots, M]\}\), where \(\tau_i\) is described by

\[\{(C^1_{i}, D^1_{i}, T^1_{i}), \ldots, (C^N_{i}, D^N_{i}, T^N_{i})\}\].

The \textit{scenario} of task \(\tau_i\), denoted by \(S_i\), is a value \(a^1_i \geq O_i\) (the initial release time) associated with an infinite sequence of tuples \(\{(\ell^k_i, \pi^k_i) : k = 1, 2 \ldots \}\), with:

- \(\ell^k_i \in \{1, \ldots, N_i\}\),
- \(\pi^k_i \geq T^{\ell^k_i}_i\).

If task \(\tau_i\) follows the scenario \(S_i\), then:

- the first job is released at time \(a^1_i\), has an execution requirement of \(C^{\ell^1_i}_i\) and a relative deadline \(D^{\ell^1_i}_i\),
- the \(k\)th job \((k > 1)\) is released at time \(a^k_i = a^{k-1}_i + \pi^{k-1}_i\), has an execution requirement of \(C^{\ell^k_i}_i\) and a relative deadline \(D^{\ell^k_i}_i\).

In this work we consider offset independent scenarios with the following definition:
Definition 3. A scenario \( S_i \) is said to be offset independent if it depends neither on \( a^1_i \), nor on the schedule. In other words, if we shift by \( \Delta \) the first arrival, all arrivals will be shifted the same way, and the jobs will keep the same characteristics.

A scenario \( \hat{S}_i \) obtained in shifting \( S_i \) by \( \Delta \) is described by:

\[
\begin{align*}
\hat{a}^1_i &= a^1_i - \Delta \\
\hat{\ell}^k_i &= \ell^k_i \forall k \\
\hat{\pi}^k_i &= \pi^k_i \forall k
\end{align*}
\]

Definition 4 (From [4]). An idle instant occurs at time \( t \) in a schedule if all jobs arriving strictly before \( t \) have completed their execution before or at time \( t \) in the schedule.

Theorem 5. For any task \( \tau_i \) the maximum response time occurs when all higher priority tasks release a new task request synchronously with \( \tau_i \) and the subsequent requests arrive as soon as possible.

Proof. We consider a job \( J_i \) in a given scenario and we will show that, by shifting the job releases of higher priority tasks, in the synchronous case the response time of the job \( J_i \) does not decrease. Since that property holds whatever the scenario we can conclude that the worst case response time occurs in a synchronous scenario, hence the property.

Let \( t_0 \) be the \( J_i \) release time. Let \( t_{-1} \) be the latest idle instant for the scheduling of \( \tau_1, \ldots, \tau_i \) at or before \( t_0 \). Let \( t_R \) denote the time instant when \( J_i \) completes. (See Figure 1.)

First, if we (artificially) redefine \( J_i \) release time to be \( t_{-1} \) then \( t_R \) remains unchanged but the \( J_i \) response time may increase.

Assume now that \( \tau_i \) releases a job at \( t_{-1} \) and some higher priority task \( \tau_j \) does not. If we left-shift \( \tau_j \) such that its first job is released at time \( t_{-1} \) then \( J_i \) response time remains

---

This proof is based upon lecture slides, the authors thank S. Funk and J. Anderson. It corrects [5, 6] for the popular sporadic task model.
unchanged or is increased (since we consider offset independent scenarios). We have
constructed a portion of the schedule that is identical to the one which occurs at time $t_{-1}$
when $\tau_1, \ldots, \tau_i$ release job together. This shows that asynchronous releases cannot cause
larger response time than synchronous release (for the same scenario).

It remains to show that the left-shift does not impact on $t_{-1}$. It may be noticed that
no job shift pasts $t_{-1}$, every job that starts after $t_{-1}$ continues to do so after the shift by
construction. Similarly, any job that starts before $t_{-1}$ continues to do so after the shift and
$t_{-1}$ remain an idle instant: for each interval $[x, t_{-1}]$, no job shift into this interval, no
left shifts cross $t_{-1}$, therefore the total demand during $[x, t_{-1}]$ cannot increase, therefore
the last completion time prior to $t_{-1}$ cannot be delayed.

The same argument (i.e., the left-shift) can be used to complete the proof and show
that the worst case response time occurs when the subsequent requests arrive as soon as
possible. \hfill $\square$

3 Schedulability Test

3.1 Request Bound Function

In order to provide a schedulability test, we will use a similar approach to the one used by [9, 3, 8], in order to bound the computation demand that a task can request. From the
previous section, we know that a scenario has its worst demand when the first request
arrives at time 0, and all other requests arrive as soon as possible. I.e., $a_i^k = 0$ and
$\tau_i^k = T_i^{\ell_i^k}$. This kind of scenario will be called a dense scenario.

In the following, we define a sporadic scenario as a scenario were exact release times
are let free. A sporadic scenario is then determined by a sequence of configuration
numbers $\{\ell_1^i, \ell_2^i, \ldots\}$ with $\ell_i^k \in \{1, \ldots, N_i\}$ $\forall i, k$. Theorem 5 says then amongst all the
possible arrival pattern of a sporadic scenario, the dense scenario is the worst.

From this, we provide a new definition:

Definition 6. Given a sporadic scenario $S_i$ of a task $\tau_i$, we define the Request Bound
**Function** $\text{RBF}_{\sigma_i}(t)$ as the maximal total demand that $\tau_i$ can request in the interval $[0, t]$. Formally,

\[ \text{RBF}_{\sigma_i}(t) = \sum_{k=1}^{\alpha(t)} C^k_i \text{ with } \alpha(t) = \min\{\beta \geq 0 \mid \sum_{k=1}^{\beta} T^k_i \geq t\}. \]

In the single frame model, where there is only one possible (sporadic) scenario, this corresponds to the Request Bound Function \[3\]. In the framework of feasibility for dynamic priorities, authors of \[8\] consider Demand Bound Function, instead of Request Bound Function.

Some example of such request bound functions are shown in Figure 2(a)–2(c). We can observe that a request bound function is always composed of a sequence of connected steps, where the $k^{\text{th}}$ step is composed of a vertical jump of $C^k_i$, followed by a horizontal segment of length $T^k_i$.

Let us now assume that the (sporadic) scenario is fixed for all tasks $\tau_1, \ldots, \tau_{i-1}$, is proved to be schedulable, and we want to know if the system is still schedulable if we add a task $\tau_i$. Adapting the results from \[9\], we need to check that, if starting synchronously with all higher priority tasks, $\tau_i$ can still meet its deadline. As $\tau_i$ can have different configurations, we need to find the smallest solution to the following $N_i$ inequalities:

\[ t = C^k_i + \sum_{j<i} \text{RBF}_{\sigma_j}(t) \quad \forall k \in \{1, \ldots, N_i\}. \]

(1)

If we now want to check the schedulability of any task system, we need to raise the hypothesis that the scenarios are known, and consider all the possible combinations of scenarios. But checking the schedulability through this method would be impossible within a reasonable amount of time, even for small problem instances.

In the following, we propose to bound all the possible request bound functions for a given task, and to use this bound instead of all possible scenarios. This will make the problem tractable, but at the price of loosing the exactness of the schedulability test (as explained in Section 3.3).

### 3.2 Maximum Request Bound Function

In the following, we will focus on the maximal request that a task can produce in the interval $[0, t)$, whatever the scenario.

**Definition 7.** The Maximum Request Bound Function $\text{MRBF}_i(t)$ of a task $\tau_i$ represents an upper bound on the demand that task $\tau_i$ can request up to any time $t$. Formally,

\[ \text{MRBF}_i(t) = \max_{\sigma \in \{S_i\}} \text{RBF}_\sigma(t) \]

where $\{S_i\}$ represents all the possible dense scenarios of $\tau_i$. 


Notice the \textit{MRBF} is not the maximum of all the \textit{RBF}'s (such a maximum does not exist in general), but a function which, for each \( t \), gives the maximum of all \( \text{RBF}(t) \).

Examples of such \textit{MRBF} functions can be seen in Figures 2(d)-2(e). Before explaining how this \textit{MRBF} can be practically computed, here is how we use it:

\textbf{Theorem 8.} An NC-GMF task set \( \tau \) is schedulable using an FTP assignment if, \( \forall i \in \{1, \ldots, M\}, \forall k \in \{1, \ldots, N_i\} \), the smallest positive solution to the equality

\[ t = C_i^k + \sum_{j<i} \text{MRBF}_j(t) \]

is not larger than \( D_i^k \).

\textit{Proof.} This is a consequence of Theorem 5 and the worst-case response time computation for FTP schedulers [9]. \hfill \Box

\textbf{3.3 Pessimism}

As we stated before, this technique introduces some pessimism. Here is a simple example explaining why. Let us consider a system with two tasks, where \( \tau_1 \) is described in Figure 2(a)-2(d), and \( \tau_2 \) contains only one configuration with \( C_2^1 = 1 \) and \( T_2^1 = D_2^1 = 3 \).

A way of proving that this system is schedulable is to consider the \textit{RBF} of all possible scenarios for \( \tau_1 \), and to see if \( \tau_2 \) can process its single unit of work within the three first units of time. For every scenario \( \sigma \), it boils down to find a \( t \leq 3 \) such that \( \text{RBF}_\sigma(t) + C_2^1 \leq t \). This is obviously the case for Figure 2(a) \( (t = 2) \), (b) \( (t = 3) \) and (c) \( (t = 2) \). We let the reader check the other cases. However, we could not find such a \( t \) with \textit{MRBF}, which implies that our test could not conclude the schedulability of the system.

As a future work, we would like to quantify this pessimism, i.e., study how much exactness we loose by considering \textit{MRBF} instead of all the \textit{RBF}'s separately. We hope to be able to bound this pessimism by a constant \( C \): if our test attests the non-schedulability of a task system \( \tau \) on a unit speed CPU, then it is for sure not schedulable on a CPU of speed \( C \).

\textbf{3.4 Algorithm}

Computing \( \text{MRBF}_i(t) \) can be done in a very similar way as in the context of dynamic priorities [8]. We present the algorithm (Algorithm 1) here for the sake of completeness.

The basic idea is to store in a queue (\( Q \)) points where a (dense) scenario can pass (i.e., one step stops at this point. In Figure 2, this corresponds to the rightmost point of a vertical segment). We first add the point \((0, 0)\) (line 2), and then all the points that can be reached from \((0, 0)\), i.e., \((T_1, C_1^1), (T_2, C_2^2), \ldots, (T_N, C_N^N)\) (lines 6 through 10). Of course, we only add a point if it is not already in the queue, as observed in \([8]\) (line 9).

Notice that we only need to get \textit{MRBF} until the maximal deadline of tasks with a lower index, represented by \( \Delta \) (line 5).
Algorithm 1: Computing \( \text{MRBF}_i \)

**Data:** \([C^1, C^2, \ldots, C^N], [T^1, T^2, \ldots, T^N], \Delta\)

1. \( \text{MRBF}(t) = 0, \forall t \geq 0; \)
2. \( Q \leftarrow (0, 0); \)
3. **while** \( Q \) not empty **do**
   4. \( (t, v) \leftarrow Q; \) // \( Q \) is ordered on \( t \)
   5. **if** \( t < \Delta \) then
      6. **foreach** \( C^k, T^k \) do
         7. \( t' \leftarrow t + T^k, v' \leftarrow v + C^k, \)
         8. \( \forall x \geq t : \text{MRBF}(x) \leftarrow \max\{\text{MRBF}(x), v'\}; \)
         9. **if** \( (t', v') \notin Q \) then
            10. \( Q \leftarrow (t', v'); \)

11. **return** \( \text{MRBF}; \)

3.5 Bounding \( \text{MRBF} \)

In order to efficiently compute \( \text{MRBF}_i \), as well as to give some approximation, it is useful to bound this function. Before giving some bound, we will first define some value. The maximal slope is defined as \( U^\text{max}_i \) def \( \max_k \{C^k_i / T^k_i\} \). Notice that \( U^\text{max}_i \) could correspond to several configurations of \( \tau_i \). The maximal amount of work is given by \( C^\text{max}_i \) def \( \max_k \{C^k_i\} \).

We also need to define \( C^U^\text{max}_i \), the computation requirement corresponding to the maximal slope (or the maximum of them if several configurations correspond to the maximal slope): \( C^U^\text{max}_i \) def \( \max_k \{C^k_i : C^k_i / T^k_i = U^\text{max}_i\} \).

**Property 9.** \( \forall t : \text{MRBF}_i(t) \leq \text{UB}_i(t) \) def \( U^\text{max}_i \times t + C^\text{max}_i \).

**Proof.** In order to prove that \( \text{MRBF}_i \) is always below the line \( \text{UB}_i \), we will show that no scenario can ever have any point above this line. First, we remind that a scenario is a set of connected steps, each of them composed of one vertical jump of some value \( C^k_i \), followed by a horizontal segment of the corresponding \( T^k_i \). One step represents the contribution of one job of \( \tau_i \) in its configuration \( k \).

The proof is done by contradiction. Let us assume that a scenario contains a point \((t, v)\) above the line, i.e., such that \( \text{UB}_i(t) < v \). Without loss of generality, we can assume that this point is the left-most point of the segment it belongs to. By construction, this point is in the middle of the contribution of some configuration \( k \) of \( \tau_i \), and this contribution starts at \((t, v - C^k_i)\). Note that as \( C^k_i \leq C^\text{max}_i \), this point is above the line \( f(t) = U^\text{max}_i \times t \). We should now find a sequence \( k_1, k_2, \ldots, k_n \) such that \( \sum_j C^k_{ij} = v - C^k_i \) and \( \sum_j T^k_{ij} = t \). This is impossible, as it would mean that, in average, the contributions of the jobs would have a slope larger than \( U^\text{max}_i \), which is a contradiction. \( \square \)
Property 10. \( \forall t : \text{MRBF}_i(t) \geq \text{LB}_i(t) \overset{\text{def}}{=} U_{i\text{max}} \times t \).

Proof. In order to prove this property, we simply build a scenario \( S \) which respects the condition \( \forall t : S(t) \geq \text{LB}_i(t) \). As by definition, \( \text{MRBF}_i(t) \geq S(t) \) for any scenario \( S \) of \( \tau_i \), this will make the proof. The scenario \( S \) consists in choosing infinitely many times the configuration with the highest slope, i.e., the one corresponding to \( U_{i\text{max}} \). Trivially, this scenario “touches” \( \text{LB}_i \) on the right side of each horizontal segment.

With a similar argument, we could refine our lower bound:

\[
\text{LB}_i(t) \overset{\text{def}}{=} U_{i\text{max}} \times t + (C_{i\text{max}} - C_{iU_{\text{max}}}).
\] (2)

We let the formal proof for a longer version of this paper.

Property 11. No scenario going through \( (t, v) \) such that \( \text{LB}_i(t) - C_{i\text{max}} > v \) can ever coincide with \( \text{MRBF}_i \) after \( t \).

Proof. From Prop. 9, a point below \( \text{LB}_i - C_{i\text{max}} \) will never be part of a scenario reaching \( \text{LB}_i \), for the same reason that no scenario of \( \tau_i \) starting at \( (0, 0) \) can ever reach \( \text{UB}_i \). As \( \text{MRBF}_i \) is always above \( \text{LB}_i \) (Prop. 10), a point below \( \text{LB}_i - C_{i\text{max}} \) will never be part of a scenario contributing in \( \text{MRBF}_i \).

From this last property, we can strongly improve Algorithm 1. As soon as a point is too far from \( \text{MRBF}_i \), it can be ignored. Line 9 of Algorithm 1 can be re-written as:

\[
\text{if } t' \times U_{i\text{max}} < v' + C_{i\text{max}} \text{ and } (t', v') \notin \mathcal{Q}
\] (3)

With the same argument as in [8], we may prove that Algorithm 1 is polynomially bounded in \( \Delta \) and \( n_i \).

4 Open Questions and Conclusion

In this work, we provide a sufficient schedulability test for the scheduling of NC-GMF tasks using FTP schedulers.

There is still a lot of work to be done in that direction. Here are a few examples of our plans for the very near future.

- We observed that starting from a value we still need to quantify, \( \text{MRBF}_i \) follows a periodic pattern of size \( C_{iU_{\text{max}}} \times T_{iU_{\text{max}}} \). We hope to find a fast way to compute this pattern and when it starts. This might allow strongly improving the complexity of computing \( \text{MRBF}_i(t) \).

- As already stated, our method introduces some pessimism that we would like to quantify, and hopefully bound.
• We would like to adapt the technique presented in [3] in order to define a polynomial approximation scheme for our schedulability test, combining MRBF for \( k \) steps with UB for higher values. However, the complexity of computing \( k \) steps of MRBF is much more complex than in [3] where computing \( k \) steps is in \( O(k) \). We will need to bound the complexity of getting \( k \) steps of MRBF; the previous item will be a first step before we get this result.

• We also plan to adapt our technique to the more general model of Digraph tasks [8]. We believe that it should not be very difficult, but we will need to adapt the proof of the critical instant and the algorithms.

Acknowledgment

We would like to warmly thank Sanjoy Baruah, from the University of North Carolina (Chapel Hill), for posing the problem and for the very interesting and useful discussions we had together.

References

[1] Baruah, S. The non-cyclic recurring real-time task model. In Proceedings of the 2010 31st IEEE Real-Time Systems Symposium (2010), IEEE Computer Society, pp. 173–182.

[2] Baruah, S., Chen, D., Gorinsky, S., and Mok, A. Generalized multiframe tasks. Real-Time Systems 17, 1 (1999), 5–22.

[3] Fisher, N., and Baruah, S. A polynomial-time approximation scheme for feasibility analysis in static-priority systems with arbitrary relative deadlines. In Proceedings of the EuroMicro Conference on Real-Time Systems (2005), IEEE Computer Society Press, pp. 117–126.

[4] Goosens, J., and Baruah, S. Multiprocessor algorithms for uniprocessor feasibility analysis. In Proceedings of the seventh International Conference on Real-Time Computing Systems and Applications (Cheju Island (South Korea), December 2000), IEEE Computer Society Press.

[5] Liu, C. L., and Layland, J. W. Scheduling algorithms for multiprogramming in a hard-real-time environment. Journal of the Association for Computing Machinery 20, 1 (January 1973), 46–61.

[6] Liu, J. W. S. Real-Time Systems. Prentice Hall, 2000.
[7] MOYO, N., NICOLLET, E., LAFAYE, F., AND MOY, C. On schedulability analysis of non-cyclic generalized multiframe tasks. In 22nd Euromicro Conference on Real-Time Systems (2010), IEEE, pp. 271–278.

[8] STIGGE, M., EKBERG, P., GUAN, N., AND YI, W. The digraph real-time task model. In IEEE Real-Time and Embedded Technology and Applications Symposium (2011), pp. 71–80.

[9] TINDELL, K., BURNS, A., AND WELLINGS, A. An extensible approach for analysing fixed priority hard real-time tasks. Journal of Real-Time Systems 6, 2 (1994), 133–151.