A Best-Response Algorithm for Multiprocessor Periodic Scheduling

A. Al Sheikh[1,2], O. Brun[1,2], P.-E. Hladik[1,2], B.J. Prabhu[1,2]
[1]CNRS ; LAAS ; 7 avenue du colonel Roche, F-31077 Toulouse Cedex 4, France
[2]Université de Toulouse ; UPS, INSA, INP, ISAE ; UT1, UTM, LAAS ; F-31077 Toulouse Cedex 4, France
{aalsheik,brun,pehladik,bjprabhu}@laas.fr

***Abstract*—The problem of scheduling strictly periodic tasks, that is tasks that have to be executed at constant time intervals over an infinite time horizon, naturally arises in real-time video signal processing and in the design of critical embedded systems. We address this problem assuming that the objective is to find a schedule that maximizes the idle times between the task executions while ensuring that they do not overlap in time. This will allow an evolution margin for task budget times, should it be required in the future.**

We first consider the uniprocessor scheduling problem for which we propose an approximation algorithm inspired from game theory. In this algorithm, tasks take turns in some fixed order and at its turn a task selects its offset to maximize its own utility function (which is related to evolution margins of the tasks). We prove the convergence of the algorithm to an equilibrium point where no task has any incentive to unilaterally deviate. Although the equilibrium point need not necessarily be unique, we prove that there exists at least one equilibium point that is also optimal. We also provide an efficient scheme to compute the best-response offset of a task. We then show that this best-response algorithm can be naturally extended to the multiprocessor case by allowing tasks to select a processor in addition to an offset on this processor. Numerical experiments show that this algorithm is far much quicker than the exact algorithm presented in [1] while at the same time it generates periodic schedules with modest relative errors.

***Index Terms*—real-time scheduling, strict periodicity, best response, Nash equilibrium.**

I. Introduction

This paper is devoted to the problem of scheduling strictly periodic tasks, i.e. , tasks that have to be executed at constant time intervals over an infinite time horizon. Periodic scheduling problems naturally arise in many real-time applications, including real-time video signal processing and in the design of critical embedded systems.

Our interest in periodic scheduling originate from the field of avionic systems. Today avionic systems are a complex mix of computers, sensors, actuators, and control and display units that have to be interconnected and integrated. They have to work in "real time" whilst maintaining high standards of safety and reliability in a high stress environment. Traditional avionic systems have been built according to a federated architecture, which leads to numerous equipment boxes, one for each subsystem. To reduce the weight and the power consumption and shorten the design-cycle times, the avionic industry has adopted the Integrated Modular Avionics (IMA). IMA architectures allow the execution of avionic functions on a shared computing platform while respecting hard segregation constraints, not only in terms of functionality, but also regarding the memory and temporal allocation, in order to avoid interference between applications.

The transition from federated architectures to IMA architectures has given rise to a complex task allocation and scheduling problem. Indeed, tasks are executed strictly periodically, which means that the time separating two successive executions (instances) of the same task is strictly equal to the associated period. Since tasks allocated to the the same computing module share the overall execution time (i.e. only one task can be executed by the module at a time), the system designer has to ensure the temporal segregation of these tasks by designing a proper schedule. The intrinsic periodic nature of the tasks gives rise to a periodic scheduling formulation.

We aim to keep the discussion as general as possible by proposing a solution strategy that is also applicable in other application areas. We therefore focus on the essence of the periodic scheduling problem and omit on purpose many application-specific details of our avionic problem (see [1]). For instance, in practical applications some tasks may be prohibited from executing on a given set of processors for reliability reasons, or certain couples of tasks may be prohibited from executing on the same processor to ensure redundancy. Although they are omitted in the present paper, we emphasize that such constraints can easily be incorporated in the proposed approach.

A. Related work

Most of the literature on scheduling periodic tasks in time is restricted to preemptive scheduling. It is known that the problem of deciding whether a periodic schedule meeting all task deadlines exists on m processor is NP-complete, even for $m = 1$ [2]. In the uniprocessor case, Liu and Layland [3] and Labetoulle [4] prove that if a feasible schedule exists, then it is obtained by a dynamic-priority algorithm that schedules tasks with earliest deadlines first. Liu and Layland also introduced the rate-monotonic scheduling which finds a feasible schedule whenever a feasible fixed-priority schedule exists. After their pioneering work, a lot of works have been devoted to the analysis of preemptive scheduling algorithms under various scenarios.

In comparison, non-preemptive periodic scheduling has received less attention in the literature [5]. Most studies on

the scheduling problem consider loose periodicity, assuming that some slack is allowed in the time between successive executions of a periodic task [6]. One of the early work on strictly periodic scheduling is that of Korst [7] who was motivated by real-time video signal processing. He considers the problem of nonpreemptively scheduling periodic tasks on a minimum number of processors, assuming that the tasks have to be executed strictly periodically, and proposes a greedy heuristic to solve this problem. Korst also establishes a necessary and sufficient condition for the schedulability of two periodic tasks. Korst *et al.* show that the problem is NP-complete in the strong sense, even in the case of a single processor, but that it is solvable in polynomial time if the periods and execution times are divisible [8]. They also propose an approximation algorithm, which is based on successively assigning tasks to processors according to some priority rule.

In the last couple of years, several research papers were devoted to non-preemptive periodic scheduling. Some of these works are focused on obtaining sufficient schedulability or non-schedulability conditions in the uniprocessor case [9], [10]. Kermia *et al.* consider a periodic multiprocessor scheduling problem. The objective is to minimize the cycle time while ensuring strict periodicity, latency and precedence constraints. For this purpose, a heuristic was proposed alongside an exact branch-and-cut algorithm. In [11], Eisenbrand *et al.* consider the problem of scheduling strictly periodic tasks on a minimum number of processors. They show that if the periods are harmonic, then there exists a 2-approximation for this minimization problem and that this result is tight. In [12], the authors handle the same problem with additionnal constraints in an avionic context. Assuming harmonic periods, they propose an Integer Linear Programming (ILP) formulation and primal heuristics that together solve real-world instances to optimality. Motivated by the same avionic application, the authors of the present paper propose in [1] a mixed integer linear programming formulation for the case of general periods. This formulation is going to be superficially discussed in this paper.

B. Our contribution

We address the non-preemptive and strictly periodic scheduling problem. Our objective is to find a schedule that maximizes the idle times between the task executions while ensuring that they do not overlap in time. This will allow an evolution margin for task budget times, should it be required for future evolution of the processing done by tasks.

We first consider the uniprocessor scheduling problem for which we propose an approximation algorithm inspired from Game Theory. Indeed, we assume that tasks are allowed to independantly select their offsets to maximize their own utility function. The offset of a task defines its strategy. The utility that a task observes from a particular strategy takes into account the impact of this strategy both on its own evolution margin and on the evolution margins of the other tasks. We assume that at every iteration only one task is allowed to move and that it selects its best-response strategy. The game is assumed to be played sequentially with tasks taking turns in some fixed order.

We analyze in detail this algorithm and provide an efficient scheme to compute the best-response strategy of a task. We also prove the convergence of the algorithm to an equilibrium point where no task has any incentive to unilaterally deviate. Although the equilibrium point is not necesseraly unique, we prove that there exists at least one optimal equilibium point. We then show that this best-response algorithm can be naturally extended to the multiprocessor case by allowing tasks to select a processor in addition to an offset on this processor.

As will be shown numerically, the main merit of this algorithm is that it is several orders of magnitude quicker than the exact ILP algorithm proposed in [1] while at the same time it generates periodic schedules with modest relative errors with respect to optimal solutions. In particular, it is remarkable that this best-response algorithm was able to find feasible solutions to aircraft sized problems provided by one of our industrial partners in few minutes.

C. Organization

In Section II we formally state the periodic scheduling problem in the case of a single processor. We also present and analyze a best-response algorithm to solve this uniprocessor problem. In section III, we show that the best-response approach naturally extends to the multiprocessor case. In Section IV we demonstrate the performances of the the proposed methods, to finally conclude and give our perspective on future work in Section V.

II. Uniprocessor Scheduling

Before addressing the multiprocessor scheduling problem, we shed light on the uniprocessor one. Given a set $\Pi = \{1, \dots, N\}$ of N strictly periodic tasks, we seek for providing a non-preemptive schedule that avoids overlap between executions. Each task $i \in \Pi$ is characterized by the following attributes:

- T_i, a strict period,
- b_i, a time budget, i.e. the duration of task execution (WCET in most cases).

Denote $\mathcal{T}_i = \{0, 1, 2, \dots, T_i - 1\}$. We denote by $t_i \in \mathcal{T}_i$ the time offset at which task i executes for the first time. We define $\mathbf{t} = [t_1, \dots, t_N]$ to be an offset vector, and $\mathcal{T} = \times_{i=1}^{N} \mathcal{T}_i$ to be the set of offset vectors.

The problem we consider amounts to finding a start time for the first execution of each task such that their executions do not overlap in time.

Due to strict periodicity, for a given $\mathbf{t}$ the kth instance of task i (kth execution) executes in the interval

$$I_k^i(t_i) = [t_i + k\,T_i, t_i + k\,T_i + b_i]\,. \tag{1}$$

The executions of tasks i and j do not overlap in time if and only if $I_k(t_i) \cap I_l(t_j) = \emptyset$ for all $k, l \in \mathbb{Z}$. We therefore have the following definition for a feasible offset vector.

Definition 1: An offset vector $\mathbf{t} \in \mathcal{T}$ is said to be feasible if and only if

$$I_k^i(t_i) \cap I_l^j(t_j) = \emptyset, \ \forall k, l \in \mathbb{Z}, \tag{2}$$

for all tasks $i, j \in \Pi$.

Intuitively, for there to be no overlap in the executions of tasks, the minimal distance between two executions of tasks should be larger than the duration of execution of tasks. We shall formalize this argument to arrive at a necessary condition for the feasiblity of an offset vector.

In the following, we let $g_{i,j}$ denote the *greatest common divisor* of T_i and T_j. We also use the symbol $\%$ as a shorthand notation for the modulo operator, i.e. $a\%b$ is to be read $a \mod b$.

Lemma 1:

$$\min_{k,l \in \mathbb{Z}} |(t_j + lT_j) - (t_i + kT_i)| = \min [\ (t_j - t_i)\%g_{i,j}\ , \ (t_i - t_j)\%g_{i,j}] \tag{3}$$

Proof: Assume that $t_j + lT_j \geq t_i + kT_i$. Let $q = \left\lfloor \frac{t_j - t_i}{g_{i,j}} \right\rfloor$. The distance between the two executions is then

$$\begin{aligned}(t_j + lT_j) - (t_i + kT_i) &= (t_j - t_i)\%g_{i,j} + qg_{i,j} + lT_j - kT_i \\ &= (t_j - t_i)\%g_{i,j} + [q + ln_j - kn_i]\ g_{i,j},\end{aligned}$$

where $T_i = n_i g_{i,j}$ and $T_j = n_j g_{i,j}$. Since $0 \leq (t_j - t_i)\%g_{i,j} < g_{i,j}$, the condition $t_j + lT_j \geq t_i + kT_i$ implies that $q + ln_j - kn_i \geq 0$ and thus that

$$(t_j + lT_j) - (t_i + kT_i) \geq (t_j - t_i)\%g_{i,j}.$$

According to the Bachet-Bézout theorem, there exist $k, l \in \mathbb{Z}$ such that $qg_{i,j} = kT_i - lT_j$ and thus the above inequality can be satisfied as an equality. The proof in the case $t_j + lT_j < t_i + kT_i$ is symmetric. ■

In fact, $(t_j - t_i)\%g_{i,j}$ represents the execution time task i is allowed to occupy without interfering with task j execution. Similarly, $(t_i - t_j)\%g_{i,j}$ represents the execution time task j is allowed to occupy without interfering with task i execution. Thus, we have the following necessary and sufficient condition for the feasibility of an offset vector of two tasks.

Theorem 1: Two tasks i and j do not overlap if and only if

$$\begin{aligned} b_i &\leq (t_j - t_i)\%g_{i,j}, \\ b_j &\leq (t_i - t_j)\%g_{i,j}. \end{aligned} \tag{4}$$

Proof: Let $k, l \in \mathbb{Z}$ and assume that $t_j + lT_j \geq t_i + kT_i$. From $(t_j + lT_j) - (t_i + kT_i) \geq (t_j - t_i)\%g_{i,j}$, we conclude that a necessary and sufficient condition for $I_k^i(t_i) \cap I_l^j(t_j) = \emptyset$ is $(t_j - t_i)\%g_{i,j} \geq b_i$.

Similarly, if $t_j + lT_j \leq t_i + kT_i$, a necessary and sufficient condition for $I_k^i(t_i) \cap I_l^j(t_j) = \emptyset$ is $(t_i - t_j)\%g_{i,j} \geq b_j > 0$. ■

Corollary 1: Two tasks i and j do not overlap if and only if $b_i \leq (t_j - t_i)\%g_{i,j} \leq g_{i,j} - b_j$.

The statement in the corollary follows from that fact that $(-a)\%b = a - a\%b$ for $a\%b > 0$.

Remark 1:
1) This theorem in the form of the corollary first appeared in [7]. Here, we provide an alternate and shorter proof of this theorem.
2) For a problem with more than two tasks, condition (4) is necessary but not sufficient to guarantee a feasible schedule.

The term $\frac{(t_j - t_i)\%g_{i,j}}{b_i}$ is the maximum factor by which b_i can be multiplied without interfering with the executions of task j. It can be thought of as the evolution margin for task i with respect to task j.

It may happen that one or more feasible schedules exist. In our work, we seek schedules that maximize the idle times between tasks while respecting scheduling constraints. This will allow an evolution margin for task budget times, should it be required for future expansion of tasks.

Introducing

$$d_{ij}(\mathbf{t}) = \min\left(\frac{(t_j - t_i)\%g_{i,j}}{b_i}, \frac{(t_i - t_j)\%g_{i,j}}{b_j}\right), \tag{5}$$

the scheduling problem can be stated as follows

$$\begin{aligned} &\text{maximize } \min_{i,j \neq i} d_{ij}(\mathbf{t}), \\ &\text{subject to } \ \mathbf{t} \in \mathcal{T}. \end{aligned} \tag{OPT}$$

An ILP formulation for this uniprocessor scheduling can be represented as follows (*cf.* [1] for further details):

$$\begin{aligned} \text{maximize} \quad & \alpha \\ \text{subject to} \quad & \\ & \mathbf{t} \in \mathcal{T}, \\ & (t_j - t_i) - q_{j,i}\, g_{i,j} \geq \alpha\, b_i, \ \forall (i,j) \in \Pi^2, \\ & (t_j - t_i) - q_{j,i}\, g_{i,j} \leq g_{i,j} - \alpha\, b_j, \ \forall (i,j) \in \Pi^2, \\ & t_i \in [0, T_i), \ \forall i \in \Pi, \end{aligned}$$

where $q_{j,i} = \left\lfloor \frac{t_j - t_i}{g_{i,j}} \right\rfloor$.

Although the above ILP could be solved numerically, only small problems can be solved with reasonable time – recall that the problem is NP-complete in the strong sense. In the following section we discuss a relatively fast algorithm for generating offset vectors that are optimal under certain conditions. As will be shown numerically, the main merit of this algorithm is that it is much quicker than the exact method for solving the ILP proposed in [1] while at the same time it generates offset vectors with modest relative errors.

A. Uniprocessor Best Reponse

The best-response algorithm we propose is inspired from an algorithm of the same name in Non-cooperative Game Theory (*cf.* [13] for details). In a game, the best-response of player is defined as its optimal strategy conditioned on the strategies of

the other players. It is, as the name suggests, the best response that the player can give for a given strategy of the others. The best-response algorithm then consists of players taking turns to adapt their strategy based on the most recent known strategy of the others.

The proposed best-response algorithm converts the optimization problem (OPT) into the following game. Let us think of tasks as players. The game is assumed to be played sequentially with players taking turns in some fixed order until the strategies (offsets) of the players converge.

Let t_i^n denote the strategy of player i at the beginning of the nth turn. In its turn, player i computes its offset so as to maximise its relative distances with the other tasks, that is, player i solves the following problem :

$$\begin{aligned} &\text{maximize } \min_{j\neq i} d_{i,j}(x, \mathbf{t}_{-i}^n) \qquad \text{(SCHD-}i\text{)} \\ &\text{subject to } \ x \in \mathcal{T}_i, \end{aligned}$$

where in standard Game Theory notation, $\mathbf{t}_{-i} = [t_1, t_2, \ldots, t_{i-1}, t_{i+1}, \ldots, t_N]$ is the vector of offsets of players other than player i. The player i then sets t_i^{n+1} to that value of x that gives the solution. In case the best-response is not unique, then it retains the smallest offset from the set of best-responses. Note that player i solves the same problem as (OPT) except that it takes into account only the terms that are affected by its offset.

Remark 2: In a usual non-cooperative game each task seeks to maximize its own objective function. In that setting the natural objective of a task would be to determine the offset that maximizes the factor by which it can increase its own duration, which would amount to solving the problem

$$\begin{aligned} &\text{maximize } \min_{j\neq i} \left(\frac{(t_j - x)\%g_{i,j}}{b_i} \right) \\ &\text{subject to } \ x \in \mathcal{T}_i. \end{aligned}$$

The objective function in (SCHD-i) is more related to ones in cooperative games in which the players seek to maximize a common objective.

In the following, let us define

$$\alpha_i^n = \min_{j\neq i} d_{i,j}(\mathbf{t}^n) \tag{6}$$

$$\mathcal{S}_i(\mathbf{t}_{-i}^n) = \operatorname*{argmax}_{x\in\mathcal{T}_i} \min_{j\neq i} d_{i,j}(x, \mathbf{t}_{-i}^n), \tag{7}$$

where α_i^n is the utility of player i after the nth iteration, and $\mathcal{S}_i(\mathbf{t}_{-i})$ is the set of all the best-responses of player i.

We shall assume that if the best-response of a player i does not improve its current utility α_i^n, then the player does not change its strategy, i.e. $t_i^{n+1} = t_i^n$. This assumption although not restrictive will allow us to prove the convergence of the algorithm.

The pseudocode for the best-response algorithm is given in Algorithm 1. In step 4 of the algorithm, $n\%N + 1$ gives the index of the player whose turn it is in the nth iteration.

Algorithm 1 Uniprocessor best-response

Require: $\mathbf{t}^0$

1: $n \leftarrow 0$
2: **repeat**
3: **for** $i = 1$ to N **do**
4: **if** $i = n\%N+1$ **and** $\max_x \min_{j\neq i} d_{i,j}(x, \mathbf{t}_{-i}^n) > \alpha_i^n$ **then**
5: $t_i^{n+1} \leftarrow \min \operatorname{argmax}$ (SCHD-i)
6: **else**
7: $t_i^{n+1} \leftarrow t_i^n$
8: **end if**
9: **end for**
10: $n \leftarrow n + 1$
11: **until** $\mathbf{t}^n \neq \mathbf{t}^{n-N}$.
12: **return** $\mathbf{t}^n$

B. Properties of the best-response algorithm

In the rest of this section we shall make the following assumption.

Assumption 1: The offsets can take on non-integral values, i.e. $\mathcal{T}_i = [0, T_i)$.

This assumption will allow us to prove two important properties of the best-response algorithm. The first property states that the algorithm converges. Indeed, for recursive algorithms like the best-response, convergence of the algorithm is a desired property. There however need not be a unique equilibrium point. The second property states that one or more of the equilibrium points correspond to the solutions of (OPT). Consequently, if the initial point is chosen appropriately, the best-response algorithm will converge to an optimal solution.

Lemma 2: Assume that at iteration n, player i updates its strategy. If $t_i^{n+1} = t_i^n$, then $\boldsymbol{\alpha}^{n+1} = \boldsymbol{\alpha}^n$. Otherwise, we have for all $j \neq i$

$$\alpha_j^n < \alpha_i^n \Rightarrow \alpha_j^{n+1} = \alpha_j^n \tag{8}$$

$$\alpha_j^n = \alpha_i^n \Rightarrow \alpha_j^{n+1} \geq \alpha_j^n \tag{9}$$

$$\alpha_j^n > \alpha_i^n \Rightarrow \alpha_j^{n+1} > \alpha_i^n \tag{10}$$

Proof: To be more concise, we use here d_{ji}^n for $d_{ji}(t^n)$. Clearly, if $t_i^{n+1} = t_i^n$ then $\mathbf{t}^{n+1} = \mathbf{t}^n$ and thus $\boldsymbol{\alpha}^{n+1} = \boldsymbol{\alpha}^n$. Let us therefore assume that $t_i^{n+1} \neq t_i^n$, i.e. $\alpha_i^{n+1} > \alpha_i^n$. Consider a task $j \neq i$ such that $\alpha_j^n < \alpha_i^n$. Since $t_k^{n+1} = t_k^n$ for all $k \neq i$, we have

$$\begin{aligned} \alpha_j^{n+1} &= \min\left(\min_{k\neq i,j} d_{jk}^{n+1}, d_{ji}^{n+1} \right) = \min\left(\min_{k\neq i,j} d_{jk}^{n}, d_{ji}^{n+1} \right) \\ &= \alpha_j^n, \end{aligned}$$

where the last equality is obtained thanks to the following inequalities : $\min_{k\neq i,j} d_{jk}^n = \alpha_j^n < \alpha_i^n < \alpha_i^{n+1} \leq d_{ji}^{n+1}$.

Let us now consider a task $j \neq i$ such that $\alpha_j^n = \alpha_i^n$. Observe that $d_{ij}^{n+1} \geq \alpha_i^{n+1} > \alpha_i^n = \alpha_j^n$. Moreover

$$\min_{k\neq i,j} d_{jk}^{n+1} = \min_{k\neq i,j} d_{jk}^{n} \geq \alpha_j^n. \quad (11)$$

We thus conclude that

$$\alpha_j^{n+1} = \min(\min_{k\neq i,j} d_{jk}^{n+1}, d_{ij}^{n+1}) \geq \alpha_j^n.$$

Finally, let us consider a task $j \neq i$ such that $\alpha_j^n > \alpha_i^n$. We have $d_{ij}^{n+1} \geq \alpha_i^{n+1} > \alpha_i^n$. Moreover equation (11) holds and implies that $\min_{k\neq i,j} d_{jk}^{n+1} > \alpha_i^n$. Thus, we can conclude that $\alpha_j^{n+1} > \alpha_i^n$. ∎

For each vector $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_N)$, we let $\hat{\boldsymbol{\alpha}}$ denote the vector obtained from $\boldsymbol{\alpha}$ by sorting the values in the increasing order. We define $\boldsymbol{\alpha} \succ \boldsymbol{\beta}$ as $\hat{\boldsymbol{\alpha}}$ greater than $\hat{\boldsymbol{\beta}}$ in the lexicographic order, i.e.

$$\exists k \leq N \quad \hat{\alpha}_i = \hat{\beta}_i \ \forall i < k \quad \text{and} \quad \hat{\alpha}_k > \hat{\beta}_k \quad (12)$$

In the following we shall demonstrate that the vector $\boldsymbol{\alpha}$ increases in the lexicographic sense when a player updates its strategy.

Proposition 2:

$$\alpha_i^{n+1} > \alpha_i^n \Rightarrow \boldsymbol{\alpha}^{n+1} \succ \boldsymbol{\alpha}^n \quad (13)$$

Proof: Note that since $\alpha_j^{n+1} = \alpha_j^n$ for all j such that $\alpha_j^n < \alpha_i^n$, the vectors $\hat{\boldsymbol{\alpha}}^n$ and $\hat{\boldsymbol{\alpha}}^{n+1}$ have a common prefix (maybe of length 0). Consider the multiplicity of the value α_i^n in the vector $\hat{\boldsymbol{\alpha}}^n$. According to Lemma 2, the multiplicity of this value decreases by at least one when task i changes its strategy. Since $\alpha_i^{n+1} > \alpha_i^n$ and $\alpha_j^{n+1} > \alpha_i^n$ for all j such that $\alpha_j^n > \alpha_i^n$, it concludes the proof. ∎

Theorem 3: The best-response algorithm converges.

Proof: The sequence $\boldsymbol{\alpha}^n$ is increasing in the lexicographic order. Since it is bounded from above, we conclude that it converges. ∎

We note that there can be multiple equilibrium points. In Game Theory, an equilibrium point is also known as a Nash Equilibrium Point after John Nash who made major contributions to the theory of Non-Cooperative Games.

Theorem 4: There exists at least one equilibrium point that is also a solution of (OPT).

Proof: Let $\mathbf{t}^0$ be a solution of (OPT). From Proposition 2, $\hat{\boldsymbol{\alpha}}^n$ is non-decreasing lexicographically, that is $\alpha_1^n \geq \alpha_1^0$. Since $\mathbf{t}^0$ has been assumed to be an optimal offset vector, α_1^0 is maximal. Thus, $\alpha_1^n = \alpha_1^0$, and consequently, $\mathbf{t}^n$ is also an optimal offset vector for all n. From Theorem 3, we know that the sequence $\mathbf{t}^n$ converges. Hence, we can conclude that there is at least one equilibrium point that is also a solution of (OPT). ∎

C. Computing the best-response

The best-response of task i can be computed using linear search (at least when the offsets are restricted tointegers) which requires $O(T_i)$ computations. We now propose a method to reduce the computational complexity of the best-response.

Let

$$\mathcal{I}_i(\mathbf{t}_{-i}) = \bigcup_{(j,k)\in(\Pi\setminus\{i\})^2} \left\{ x : \frac{(x-t_j)\%g_{i,j}}{b_j} = \frac{(t_k-x)\%g_{i,k}}{b_i} \right\}$$

be the set of points where two tasks interfere.

Theorem 5: $\mathcal{S}_i(\mathbf{t}_{-i}) \subset \mathcal{I}_i(\mathbf{t}_{-i}) \subset \mathcal{T}_i$.

Proof: Assume on the contrary that

$$\frac{(t_j - t_i^*)\%g_{i,j}}{b_i} \neq \frac{(t_i^* - t_k)\%g_{i,k}}{b_k}, \quad \forall j,k \neq i. \quad (14)$$

where t_i^* denotes one of the best-responses of task i.

Let $j \neq i$ be such that $\alpha_i(t_i^*, t_{-i}) = \min d_{ij}(t_i, \mathbf{t}_{-i})$. By assumption, we know that $\frac{(t_j-t_i^*)\%g_{i,j}}{b_i} \neq \frac{(t_i^*-t_j)\%g_{i,j}}{b_j}$. Let us assume that $\frac{(t_i^*-t_j)\%g_{i,j}}{b_j}$ is the smaller of the two terms. We thus have for all $k \neq i$

$$\frac{(t_i^* - t_j)\%g_{i,j}}{b_j} \leq \frac{(t_i^* - t_k)\%g_{i,k}}{b_k}, \quad (15)$$

$$\frac{(t_i^* - t_j)\%g_{i,j}}{b_j} < \frac{(t_k - t_i^*)\%g_{i,k}}{b_i}. \quad (16)$$

Equation (15) comes from the definition of j as the task achieving the minimum in (6). The strict inequality in (16) is obtained using (14).

With $k = j$ in (16), we obtain $(t_i^* - t_j)\%g_{i,j}/b_j < (t_j - t_i^*)\%g_{i,j}/b_i$, which implies that $(t_j - t_i^*)\%g_{i,j} > 0$. With (15), it implies that $(t_i^* - t_k)\%g_{i,k} > 0$ for all $k \neq i$ and thus that there exists $q_{ik} \in \mathbb{Z}$ such that $t_i^* \in (t_k + q_{ik}g_{i,k}, t_k + (q_{ik}+1)g_{i,k})$.

Let $z = \min_{k\neq i} t_k + (q_{ik}+1)g_{i,k}$. The function $x \to \min_{k\neq i}(x-t_k)\%g_{i,k}/b_k$ is continuous and strictly increasing on the interval (t_i^*, z). Therefore, we have

$$\min_{k\neq i} \frac{(x-t_k)\%g_{i,k}}{b_k} > \frac{(t_i^* - t_j)\%g_{i,j}}{b_j} \quad \forall x \in (t_i^*, z). \quad (17)$$

Moreover, the function $x \to \min_{k\neq i}(t_k - x)\%g_{i,k}/b_i$ is continuous and strictly decreasing on the interval (t_i^*, z). Thus, (16) implies that there exists $\epsilon > 0$ such that

$$\min_{k\neq i} \frac{(t_k-x)\%g_{i,k}}{b_i} > \frac{(t_i^* - t_j)\%g_{i,j}}{b_j} \quad \forall x \in (t_i^*, t_i^* + \epsilon). \quad (18)$$

From (17) and (18), we conclude that $\alpha_i(x, \mathbf{t}_{-i}) > \alpha_i(t_i, \mathbf{t}_{-i})$ for all $x \in (t_i^*, t_i^* + \epsilon)$, which is clearly a contradiction.

The proof in the case $(t_j - t_i^*)\%g_{i,j}/b_i < (t_i^* - t_j)\%g_{i,j}/b_j$ is symmetric. ■

Thus, the solution of (SCHD-i) can be obtained by restricting the search over the set $\mathcal{I}_i(\mathbf{t}_{-i})$. Since, the original problem is defined only for integer values of the offsets, the best-response algorithm described in Algorithm 1 will search for best-response only on floor and ceiling integers of points in $\mathcal{I}_i(\mathbf{t}_{-i})$ instead of searching all the T_i points.

D. Computing the intersection points

We now give an algorithm to compute the elements of $\mathcal{I}_i(\mathbf{t}_{-i})$ and, if possible, its cardinality.

An offset $x \in \mathcal{I}_i(\mathbf{t}_{-i})$ satisfies

$$t_j - x = mg_{i,j} + (t_j - x)\%g_{i,j}, \tag{19}$$

$$x - t_k = ng_{i,k} + (x - t_k)\%g_{i,k}, \tag{20}$$

$$\frac{(t_j - x)\%g_{i,j}}{b_i} = \frac{(x - t_k)\%g_{i,k}}{b_k}, \tag{21}$$

for some m and n in $\mathbb{Z}$. From which we conclude that

$$t_j - t_k - ((t_j - x)\%g_{i,j})\left(1 + \frac{b_k}{b_i}\right) = lg_{i,j,k}, \tag{22}$$

for some l in $\mathbb{Z}$.

Since $(t_j - x)\%g_{i,j} \in [0, g_{i,j})$, $(x - t_k)\%g_{i,k} \in [0, g_{i,k})$ and the equality (21) is true,

$$(t_j - x)\%g_{i,j} \in \left[0, \min\left(g_{i,j}, g_{i,k}\frac{b_i}{b_k}\right)\right),$$

that is,

$$t_j - t_k - ((t_j - x)\%g_{i,j})\left(1 + \frac{b_i}{b_k}\right) \in \tag{23}$$

$$(t_j - t_k - \mu_{j,k}, t_j - t_k]. \tag{24}$$

where we have defined $\mu_{j,k} = \min\left(\frac{g_{i,,j}}{b_i}, \frac{g_{i,k}}{b_k}\right)(b_i + b_k)$. From (22) and (24), $t_j - t_k - \left((t_j - x)\%g_{i,j}\right)\left(1 + \frac{b_i}{b_k}\right)\Big)$ can potentially be any integer multiple of $g_{i,j,k}$ in the interval $(t_j - t_k - \mu_{j,k}, t_j - t_k]$.

Let $L(j,k)$ be the number of multiples of $g_{i,j,k}$ in the interval

$$(t_j - t_k - \mu_{j,k}, t_j - t_k],$$

that is,

$$L(j,k) = \left\lfloor \frac{t_j - t_k}{g_{i,j,k}} \right\rfloor - \left\lfloor \frac{t_j - t_k - \mu_{j,k}}{g_{i,j,k}} \right\rfloor. \tag{25}$$

From (22) and (25),

$$(t_j - x)\%g_{i,j} = \frac{t_j - t_k - lg_{i,j,k}}{1 + \frac{b_k}{b_i}}, \text{ for } l = \left\lfloor \frac{t_j - t_k}{g_{i,j,k}} \right\rfloor, \ldots, \left\lfloor \frac{t_j - t_k - \mu_{j,k}}{g_{i,j,k}} \right\rfloor + 1,$$

that is,

$$(t_j - x)\%g_{i,j} = \frac{(t_j - t_k)\%g_{i,j,k} + lg_{i,j,k}}{1 + \frac{b_k}{b_i}}, \quad \text{for } l = 0, \ldots, L(j,k) - 1. \tag{26}$$

Note that $(t_j - x)\%g_{i,j} < \min\left(g_{i,j}, g_{i,k}\frac{b_i}{b_k}\right)$ as is necessary for (21) to be satisfied.

Finally, in order to determine x, we need to determine m, which we shall now proceed to do. Let $\hat{m}$ and $\hat{n}$ be the Bézout coefficients of the pair $g_{i,j}$ and $g_{i,k}$, that is

$$\hat{m}g_{i,j} + \hat{n}g_{i,k} = g_{i,j,k}.$$

These coefficients can be determined using the extended Euclid algorithm [14]. Then,

$$l\hat{m}g_{i,j} + l\hat{n}g_{i,k} = lg_{i,j,k}.$$

Let $c_{i,j,k}$ be the least common multiple of $g_{i,j}$ and $g_{i,k}$. From the Bachet-Bézout theorem we know that the $m = l\hat{m} + q\frac{c_{i,j,k}}{g_{i,j}}$, for some $q \in \mathbb{Z}$. Thus, for a given l, we need to determine q in order to compute x.

Let

$$q(l) = \max\{q : t_j - l\hat{m} - qc_{i,j,k} - (t_j - x)\%g_{i,j} > 0\}.$$

That is, $q(l)$ is that value of q that results in the smallest non-negative x that satisfies (19) and (20). We shall denote this smallest value of t by $\tau(l)$. Thus,

$$\tau(l) = (t_j - l\hat{m}g_{i,j} - (t_j - x)\%g_{i,j})\%c_{i,j,k}.$$

If $\tau(l)$ is an intersection point, then so is $\tau(l) + rc_{i,j,k}$. Thus, for each l, there are $\lfloor \frac{T_i - \tau(l)}{c_{i,j,k}} \rfloor + 1$ values of r for which $\tau(l) + rc_{i,j,k} \in \mathcal{T}_i$ which give the same intersection heights.

In Algorithm 2, we summarize the above steps for computing the elements of the set $\mathcal{T}_i$.

Algorithm 2 Computing the set of intersection points

Require: : $\mathbf{t}_{-i}$
  $\mathcal{I}_i(\mathbf{t}_{-i}) = \emptyset$
  for all $j \neq i$ **do**
    for all $k \neq i$ **do**
      for $l = \left\lfloor \frac{t_j - t_k - \min\left(\frac{g_{i,j}}{b_i}, \frac{g_{i,k}}{b_k}\right)(b_i + b_k)}{g_{i,j,k}} \right\rfloor + 1$ to $\left\lfloor \frac{t_j - t_k}{g_{i,j,k}} \right\rfloor$ **do**
        $\tau(l) = (t_j - l\hat{m}g_{i,j} - \frac{t_j - t_k - lg_{i,j,k}}{1 + \frac{b_k}{b_i}})\%c_{i,j,k}$
        for $r = 0$ to $\left\lfloor \frac{T_i - \tau(l)}{c_{i,j,k}} \right\rfloor$ **do**
          $\mathcal{I}_i(\mathbf{t}_{-i}) \leftarrow \mathcal{I}_i(\mathbf{t}_{-i}) \cup (\tau(l) + rc_{i,j,k})$
        end for
      end for
    end for
  end for

An upper bound on the cardinality of $\mathcal{I}_i(\mathbf{t}_{-i})$ can be computed as

$$|\mathcal{I}_i(\mathbf{t}_{-i})| \leq \sum_{j \neq i}\sum_{k \neq i} \frac{\min\left(\frac{g_{i,j}}{b_i}, \frac{g_{i,k}}{b_k}\right)(b_i + b_k)}{g_{i,j,k}} \frac{T_i}{c_{i,j,k}}$$

$$= \sum_{j \neq i}\sum_{k \neq i} \frac{\min\left(\frac{g_{i,j}}{b_i}, \frac{g_{i,k}}{b_k}\right)(b_i + b_k)T_i}{g_{i,j}g_{i,k}}$$

If we assume that the tasks are at least schedulable in pairs, that is $(b_i + b_j) \leq g_{i,j}$, $\forall i, j$, then a sufficient condition for this algorithm to check fewer points than linear search is

$$b_i > K^2,$$

where K is the total number of tasks.

Remark 3: When the offsets are restricted to integers, we could potentially be computing several intersection points between two integers. And, for all these intersection points the best-response algorithm shall check for only the two integers closest to them. Although, theoretically, there could be instances where there could be more intersection points than the period of the task, in practice, this will not have much impact on the efficiency of the best response algorithm.

III. Multiprocessor Scheduling

In this section we further extend our model to the multiprocessor scheduling problem. Our schedule is now not only represented by the temporal scheduling of tasks $\mathbf{t}$, but also by the processor allocation for each of the tasks. Let $\mathcal{P} = \{1, \ldots, P\}$ be a set of P processors where processor k is characterized by the available memory capacity M_k, and the maximum number of tasks H_k it can host.

An allocation can be represented as a vector of binary variables $\mathbf{a} = (a_{i,k} : \forall i \in \Pi, \forall k \in \mathcal{P})$ such that:

$$a_{i,k} = \begin{cases} 1 & \text{if task } i \text{ is assigned to processor } k, \\ 0 & \text{otherwise.} \end{cases}$$

The set of all possible allocations is denoted by $\mathcal{A}$. The associated ILP formulation can be represented as follows (non-fundamental constraints are omitted, additional constraints such as exclusion and communication delays are further discussed in [1]):

$$\text{maximize}_{\mathbf{a},\mathbf{t}} \alpha \tag{27}$$

$s.t.$

$$\sum_{p_k \in \mathcal{P}} a_{i,k} = 1 \quad , \forall i \in \Pi, \tag{28}$$

$$\sum_{i \in \Pi} a_{i,k}\, m_i \leq M_k \quad , \forall k \in \mathcal{P}, \tag{29}$$

$$\sum_{i \in \Pi} a_{i,k} \leq H_k \quad , \forall k \in \mathcal{P}, \tag{30}$$

$$(t_j - t_i) - q_{j,i}\, g_{i,j} \geq \alpha\, b_i - (2 - a_{i,k} - a_{j,k})\, Z \quad , \forall k, \forall (i,j), \tag{31}$$

$$(t_j - t_i) - q_{j,i}\, g_{i,j} \leq g_{i,j} - \alpha\, b_j + (2 - a_{i,k} - a_{j,k})\, Z \quad , \forall k, \forall (i,j), \tag{32}$$

$$a_{i,k} \in \{0, 1\} \quad , \forall k, \forall i, \tag{33}$$

$$t_i \in [0, T_i) \quad , \forall i \in \Pi, \tag{34}$$

$$q_{j,i} \in \mathbb{Z} \quad , \forall (i,j), \tag{35}$$

As in the uniprocessor setting, the objective (27) aims at maximizing the evolution margins of the tasks,

Constraint (28) prevents tasks from executing on more than one processor. Constraints (29) and (30) represent the maximum memory and task capacities of processors. Constraints (31) and (32) represent condition (4) for task couples executing on the same processor. Z is a large constant that ensures that these constraints are not active unless $a_{i,k} = a_{j,k} = 1$ as inspired from [15], i.e. tasks i and j execute both on processor k. Constraints (33), (34) and (35) specify the variables' domains.

This ILP formulation appeared to be inefficient for large scale systems, as was shown in [1]. For fairly limited ones (e.g. limited number and flexible temporal properties of tasks) this method is able to give optimal results in acceptable execution times. In modern and upcoming complex systems, however, satisfactory results would not be acquired in convenient amounts of time.

In the multiprocessor setting, at its turn a task sequentially computes its best response on each of the processors. It then selects the processor which improves its current evolution margin the most. Just as in the uniprocessor case, we shall assume that a task does not switch its processor and change its offset unless it can strictly improve its evolution margin (or, its utility).

Following the definition (6), for the multiprocessor setting let us define

$$\alpha_i^n = \min_{\{j : j \neq i, a_{i,k}^n = a_{j,k}^n \forall k\}} d_{i,j}(\mathbf{t}^n), \tag{36}$$

which is the evolution margin of the task i with respect to those that are scheduled on the same processor as itself for a given offset vector $\mathbf{t}^n$ and allocation vector $\mathbf{a}^n$. We shall also define $\mathbf{a}_i^n$ to be the allocation vector of task i after the nth iteration.

The pseudocode for the multiprocessor best-response algorithm is given in Algorithm 3.

IV. Results

In this section we demonstrate some experimentations used to evaluate the performance of the best response algorithms proposed in Section II and III. The ILP formulation represented in Section III serves as an exact method, for both the uniprocessor and multiprocessor problems, and is solved using the linear program solver CPLEX [16] from the ILOG community. The machine used is based on an Intel® Core™2 Quad CPU Q6700 @ 2.66GHz with 4MB of cache and 4GB of system memory. It was noticed that for all of the experimentations, no more than one cpu core is used.

For example generation, harmonic (H) periods, i.e. every period divides all other periods of greater value, were chosen uniformly from the set $\{1500, 3000, 6000, 12000, 24000\}$, whereas for non-harmonic (NH) periods, i.e. general values may exist, tasks were uniformly allocated one of five periods chosen from the set $\{2^x\, 3^y\, 50 : x \in [0,4], y \in [0,3]\}$, as was inspired from [12]. For task time budgets, it was generated

Algorithm 3 Multiprocessor best-response

Require: $\mathbf{t}^0, \mathbf{a}^0$
$n \leftarrow 0$
repeat
  for $i = 1$ to N **do**
    if i = n % N + 1 **then**
      $\mathbf{a}_i^{n+1} \leftarrow \mathbf{0}$
      for $k = 1$ to M **do**
        $z \leftarrow \max_x \min_{\{j:j\neq i, a_{j,k}=1\}} d_{i,j}(x, \mathbf{t}_{-i}^n)$
        if $z > \alpha_i^n$ **then**
          $\alpha_i^n \leftarrow z$
          $c \leftarrow k$
          $t_i^{n+1} \leftarrow \min \operatorname{argmax} \min_{\{j:j\neq i, a_{j,k}=1\}} d_{i,j}(x, \mathbf{t}_{-i}^n)$
        end if
      end for
      $a_{i,c}^{n+1} = 1$
    else
      $t_i^{n+1} \leftarrow t_i^n$
      $\mathbf{a}_i^{n+1} \leftarrow \mathbf{a}_i^n$
    end if
  end for
  $n \leftarrow n + 1$
until $\mathbf{t}^n \neq \mathbf{t}^{n-N}$ **and** $\mathbf{a}^n \neq \mathbf{a}^{n-N}$.
return $\mathbf{t}^n$

following an exponential distribution and averaging at about 20% of the task's period. Memory and task capacities for processors and memory requirements for tasks were generated so as not to greatly constraint the problem, in other words, they were generated to limit the processors with a twenty task cap. The influence of memory resources on the scheduling problem is left for future work. Finaly, among the architectural constraints (for multiprocessor problems), only task exclusions were considered (i.e. couples of tasks that cannot execute on the same processor). 40% of tasks were chosen uniformly to be implicated in an exclusion constraint.

We start off by evaluating the algorithm's performance in the uniprocessor case. Twenty uniprocessor examples, with fifteen tasks each, were generated with harmonic periods. As can be seen in figure 1, the relative error on the optimized evolution coefficient α averaged at about 7.8%. This is interesting given that the average execution time for the best response algorithm was around 2.7 seconds versus 20 minutes for the exact method.

A particular situation arises when the problem becomes more complex and the exact method fails to prove optimality in an adequate amount of time. Consider for instance a non-harmonic example constituted of 20 tasks whose temporal attributes are indicated in Table I–these are not generated as mentioned earlier but randomly to make the problem even harder–, task time budgets remain small compared to their periods.

For this example the best response algorithm gave $\alpha = 1.41$ in 2.83 seconds where the exact method failed to supply a value superior to $\alpha = 1.11$ within a one hour limit. In this

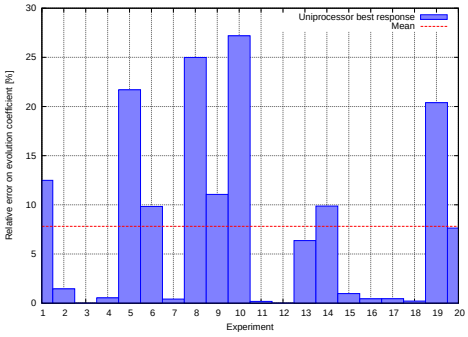

Fig. 1. Relative error on α for uniprocessor examples.

TABLE I
A UNIPROCESSOR EXAMPLE WITH 20 TASKS OF GENERAL NON-HARMONIC PERIODS. LCM BETWEEN PERIODS IS 756000

Task	1	2	3	4	5	6
Time budget	10	30	30	10	10	10
Period	1200	1200	3600	1200	1200	1500

7	8	9	10	11	12	13
10	10	30	10	10	45	40
4200	1000	2000	4000	1200	2400	2000

14	15	16	17	18	19	20
40	60	80	30	10	60	40
4000	3000	3000	2700	200	1800	1800

case, we notice the benefit of the proposed algorithm. It is true that the final optimal solution may be greater than the obtained value but one has to take into account the time cost required to do so.

For the multiprocessor scheduling problem, 4 experiment sets, each consisting of 100 instances, were generated. Instances of the first and second sets are constituted of 4 processors and 20 tasks with harmonic and non-harmonic periods respectively. Instances of the third and fourth sets are composed of 4 processors and 40 tasks with harmonic and non-harmonic periods respectively. The attributes were generated as indicated in the beginning of this section.

When solving the multiprocessor problem to optimality using the exact ILP formulation, a time limit of thirty minutes was defined for instances of the first and second sets. As for the third and fourth sets, this limit was increased to one hour and task offsets were considered continuous ($t_i \in \mathbb{R}$). This decreases execution times significantly [1]. To limit the minor difference in the optimal solution, all temporal attributes were divided by 100. For what concerns the porposed best response algorithm, the instances were not modified and no time limit was imposed.

Table II demonstrates the relative error on the evolution coefficient α for the multiprocessor best response algorithm with respect to the exact method. Execution times for the exact method are not included as we have already shown in [1] that exceeding certain limits, they may be inconvenient. It is obvious that execution times as well as the quality of obtained

TABLE II
RELATIVE ERROR ON OPTIMALITY FOR THE MULTIPROCESSOR BEST RESPONSE ALGORITHM. xPyT DESIGNATES INSTANCES WITH x PROCESSORS AND y TASKS.

Experiment set	Nature of periods	Mean relative error on α	Mean CpuTime
4P20T-H	harmonic	13.9%	0.85s.
4P20T-NH	non-harmonic	14.5%	1.96s.
4P40T-H	harmonic	9.37%	13.9s.
4P40T-NH	non-harmonic	9.22%	26.4s.

TABLE III
THREE SCHEDULING PROBLEMS WITH SIZES SIMILAR TO THOSE IN AVIONIC TASK SCHEDULING PROBLEMS

Problem	Evolution coefficient α	CpuTime
6P36T	2.93	1.61s.
6P79T	1.47	142s.
24P419T	1.14286	2.4hrs.

solutions are quite interesting. For all of the considered examples, harmonic and non-harmonic, mean relative error on the evolution coefficient α, as compared to the exact solution, remains below 15%. For the third set '4P40T-H', the relative error averages at about 9%, the reason why this average is lower than that of '4P20T-H' is that the exact method often required more than one hour to reach or even prove optimality, and hence, within this time limit, α obtained by the best response algorithm was superior to that of the exact method. The same analysis applys to '4P40T-NH'.

A. Large scale and industrial applications

We hereafter test the aforementioned algorithm on examples depicting real world scenarios. Our work, being supported by an industrial project, is aimed at avionic platforms. Avionic applications represented by tasks need to be scheduled on available modules or in other words processors. Three types of tasks and processors actually exist in an airplane, without discussing the nomenclature and side locations (modules are distributed between two sides), this leads to the definition of three distinct scheduling problems, one per type. We inspired from this to generate three sets of problems depicting those on an airplane. A set composed of 6 processors and 36 tasks '6P36T', another of 6 processors and 79 tasks '6P79T' and a last one composed of 24 processors and 419 tasks '24P419T'. All periods were considered harmonic and temporal attributes were generated as before. Exclusions between tasks were neglected for these three sets. Table III demonstrates the results obtained.

Unfortunately, comparison with the exact formulation was not possible due to its limitations. The quality of an obtained solution has to assessed based on its utility, a value of $\alpha > 1$ implies in essence a feasible schedule. Any greater value gives more flexibility when increasing time budgets. Execution times are nevertheless convenient, the largest problem with 419 tasks taking about two and a half hours.

TABLE IV
THREE SCHEDULING PROBLEMS SUPPLIED AS A SMALL BENCHMARK

Problem	Evolution coefficient α	CpuTime
12P36T	9.99	0.13s.
12P112T	3.12	11.3s.
48P636T	1.56	1.17hrs.

We further investigate the utility of the algorithm with a small benchmark supplied by one of the industrial partners of the project. An example architecture constituted of three sub-problems were supplied, the first of 12 processors and 36 tasks '12P36T', the second of 12 processors and 112 tasks '12P112T' and the last one of 48 processors and 636 tasks '48P636T'. Task periods belong to the set $\{60000, 120000, 240000\}$ (in μseconds). Processor segregation constraints (exclusions) were also supplied, totalling 176 constraints (implicating tasks of the architecture). Additional resource attributes were also supplied (e.g. RAM, NVM, etc.). Table IV demonstrates the obtained results.

The results appear to be, as before, convenient. We notice a difference in execution times between '48P636T' at about 1 hour and the previous one '24P419T' (Table III) at 2.4 hours. One reason can be related to the fact that in '48P636T' the number of tasks per processor averages at about 13 whereas in '24P419T' this average is 17, and thus less computation per processor in '48P636T'. The decreased number of distinct periods and simpler temporal attributes is another factor that may have played a role in decreasing the number of iterations required for the algorithm to converge.

A final experimentation we carried out was the division of temporal attributes (periods and time budgets) by 1000, as usually a precision in the order of milliseconds is sufficient. Surprisingly the algorithm converged for '48P636T' in about 4 minutes. This is caused from reducing the precision on the time horizon and hence reducing several computations throughout the algorithm.

V. CONCLUSION

In this paper we have proposed a best-response algorithm for the non-preemptive and strictly periodic multiprocessor scheduling problem. This game theoretic approach appeared to be extremely efficient in solving this scheduling problem, the quality of the supplied solutions is promising and execution times are highly convenient given the complexities dealt with. Our objective is not classical as we are not looking to minimize the number of machines used, but are searching for possibilities to utilize the available resources in a manner that allow the most for the future expansion of task execution times without having to reconfigure the whole system.

From a theoretical viewpoint, future work includes the analysis of the convergence time of the best-response algorithm as well as the analysis of the error made by this heuristic. For the latter point, our goal will be to derive an upper bound on the so-called Price of Anarchy [17], i.e. the ratio between the global cost obtained by the worst Nash equilibrium and

a global optimal solution. We also plan to design a randomized algorithm on the basis of the best-response algorithm. Numerical experiments have shown that the solution space is separated into a small number of regions of attraction. Started in any point of such a region, the best-response algorithm will converge to the associated equilibrium. We plan to analyze the number of these regions as well as their measures in order to devise a probabilistic algorithm.

From a more practical viewpoint, another worth pursuing research direction concerns the effect of architectural constraints, such as segregation between tasks (should not be allocated the same processor), on the behavior of the proposed algorithm. Bad placement of tasks in the beginning of the algorithm may block several solutions, if not all – especially when there is a large number of segregation constraints. This can be solved by applying the Maximum Independent Set algorithm discussed in [1], where a maximum set of independent tasks are found (they are surely to be allocated different processors). It would also be interesting to investigate adding transmission delay constraints to the algorithm to incorporate data exchange and execution chains between tasks.

ACKNOWLEDGMENT

The work presented in this paper was conducted under the research project SATRIMMAP (SAfety and Time Critical Middleware for future Modular Avionics Platform) which is supported by the French National Agency for Research (ANR).

REFERENCES

[1] A. Al-Sheikh, O. Brun, and P.-E. Hladik, "Partition scheduling on an IMA platform with strict periodicity and communication delays," in *International Conference on Real-Time and Network Systems (RTNS 2010), pp.179-188*, November 2010.

[2] J.-T. Leung and M. Merril, "A note on preemptive scheduling of periodic, real-time tasks," *Information Processing Letters*, vol. 11, pp. 115–118, 1980.

[3] C. Liu and J. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *Journal of the ACM (JACM)*, vol. 20, no. 1, pp. 46–61, 1973.

[4] J. Labetoulle, "Some theorems on real time scheduling," *Computer Architecture and Networks*, pp. 285–293, 1974.

[5] L. George, P. Muhlethaler, and N. Rivierre, "Optimality and non-preemptive real-time scheduling revisited," Rapport de Recherche RR-2516, INRIA, Tech. Rep., 2006.

[6] J. Korst, E. Aarts, and J. K. Lenstra, "Scheduling periodic tasks with slack," *INFORMS JOURNAL ON COMPUTING*, vol. 9, pp. 351–362, 1997.

[7] J. Korst, "Periodic multiprocessor scheduling," Ph.D. dissertation, Eindhoven university of technology, Eindhoven, the Netherlands, 1992.

[8] J. Korst, E. Aarts, Lenstra, and J. Karel, "Scheduling periodic tasks," *INFORMS JOURNAL ON COMPUTING*, vol. 8, pp. 428–435, 1996.

[9] M. Marouf and Y. Sorel, "Schedulability conditions for non-preemptive hard real-time tasks with strict period," November 2010.

[10] P. Meumeu and Y. Sorel, "Non-schedulability conditions for off-line scheduling of real-time systems subject to precedence and strict periodicity constraints," in *Proceedings of 11th IEEE International Conference on Emerging technologies and Factory Automation, ETFA06, WIP*, 2006.

[11] F. Eisenbrand, N. Hahnle, M. Niemeier, M. Skutella, J. Verschae, and A. Wiese, "Scheduling periodic tasks in a hard real-time environment," in *ICALP 2010 Proceeding, Part I*, L. . S. Abramsky et al (Eds.), Ed., 2010, p. 2010.

[12] F. Eisenbrand, K. Kesavan, R. Mattikalli, M. Niemeier, A. Nordsieck, M. Skutella, J. Verschae, and A. Wiese, "Solving an avionics real-time scheduling problem by advanced IP-methods," *Algorithms–ESA 2010*, pp. 11–22, 2010.

[13] D. Fudenberg and J. Tirole, *Game Theory*. MIT Press, 1991.

[14] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*. MIT Press, 2001.

[15] W. Roux, "Une approche coherente pour la planification et l'ordonnacement de systèmes de production complexes," Ph.D. dissertation, LAAS-CNRS, report n^o 97248, 1997.

[16] ILOG CPLEX, http://www.ilog.com/products/cplex/.

[17] E. Koutsoupias and C. H. Papadimitriou, "Worst-case equilibria." in *STACS 1999*, 1999.