The Space of Feasible Execution Times for Asynchronous Periodic Task Systems using Definitive Idle Times

Thomas Chapeaux and Paul Rodriguez
Université Libre de Bruxelles / ECE Paris
Avenue Franklin Roosevelt 50
1050 Brussels, Belgium
tchapeau@ulb.ac.be, paurodri@ulb.ac.be

Laurent George
University of Paris-Est / ECE Paris
Quai de Grenelle, 37
75015 Paris
lgeorge@ieee.org

Joël Goossens
Université Libre de Bruxelles
Avenue Franklin Roosevelt 50
1050 Brussels, Belgium
joel.goossens@ulb.ac.be

***Abstract*—Sensitivity analysis for real-time systems provides efficient methods to determine feasibility conditions in the case of changes in task parameters, such as different WCET values when porting a system to a different platform. The C-space defines the space of execution times for which a system is feasible, which can help system designers to more easily check if a task set is feasible on a set of candidate platforms. This paper extends previous works generalizing the description of the C-space for Earliest Deadline First scheduling to systems with initial task offsets. The feasibility gain offered by offsets is then expressed as the volume ratio between the C-space of an asynchronous system and its corresponding synchronous system when offsets are randomly chosen. The problem requires solving multiple similar instances of the Chinese Remainder Problem, for which an efficient algorithm in this context is provided. We show that even for random offsets, the gain can reach 15%.**

I. Introduction

Abstract models for real-time systems often require the determination of the worst-case execution time of a job (WCET), a value highly dependent on the platform on which the application is deployed. Feasibility conditions allow us to decide whether a system can be correctly scheduled (i.e. without violating any timeliness constraint) based on task parameters, including the WCET.

The C-space of a real-time system is, intuitively, the space of WCET values for which this system is feasible [1], [2]. A precise and efficient description of the C-space of a system would allow us to efficiently decide whether or not it is feasible on a set of candidate platforms.

In section II, the description of the C-space of synchronous constrained systems, which has been mostly covered by [2], is revisited. This description is based on the Demand Bound Function (DBF) feasibility condition described in [3], [4] and is found with exponential complexity using the notion of definitive idle times (DIT). The DIT notion provides a feasibility interval for EDF that does not depend of the WCET of the tasks.

In section III, we present an extension of the concept of DIT adapted to the asynchronous case called first periodic DIT (t_d) along with an algorithm to determine its existence and value for any system, then we show the influence of some factors on its existence.

In Section IV, we show how to describe the C-space of an asynchronous system based on the demand bound function test. Then we show why applying this test on the interval starting at t_d and spanning an hyperperiod is sufficient when t_d exists. A numerical example is also given.

Finally, in section VI, we use a simulation to quantify the feasibility gain of asynchronous systems in terms of volume of the C-space.

II. Related works

A. Model

We consider a discrete timeline and tasks systems comprised of periodic tasks with constrained deadlines on uniprocessor platforms. Each task τ_i is represented by a tuple (O_i, C_i, D_i, T_i) where O_i is the offset value, C_i the WCET, D_i the relative deadline and T_i the period, which is the exact time between two consecutive arrivals. The constrained deadline property implies that $D_i \leq T_i$. In the following, we differentiate between *synchronous systems* (where $O_i = 0$ for all tasks) and *asynchronous systems* (which are not synchronous).

For both synchronous and asynchronous systems, the preemptive *Earliest Deadline First* (EDF) algorithm, in which jobs are prioritized according to their absolute deadline, has been shown to be optimal [5], which means it correctly schedules any feasible system within that group.

Furthermore, we define:

- $\tau = \{\tau_1, \cdots, \tau_n\}$ is called the task system
- $H = \text{lcm}(T_1, \cdots, T_n)$
- $O_{max} = \max(O_1, \cdots, O_n)$
- An *idle time* is an instant t at which each job that arrived strictly before t has finished its execution.
- A *busy period* is a time interval between two consecutive idle times in which at least one job is executing.
- The interval between the initial time ($t = 0$) and the first idle time after $t = 0$ is called the *first busy period*.

B. The DBF feasibility test

We recall the necessary and sufficient feasability condition on uniprocessor platforms based on the demand-bound function [3], [4].

Definition 1: The **demand-bound function (DBF)** defined for a task set τ in a time interval $[t_1, t_2]$ and denoted $\mathrm{dbf}(t_1, t_2)$, is equal to the cumulated execution time of jobs of τ contained in the closed interval $[t_1, t_2]$.

Mathematically,

$$\mathrm{dbf}(t_1, t_2) = \sum_{i=1}^{n} n_i(t_1, t_2)\, C_i \tag{1}$$

where $n_i(t_1, t_2)$ is the number of jobs of task τ_i whose arrival times and deadlines are both in the closed interval $[t_1, t_2]$.

The values of the n_i are given by [4]

$$n_i(t_1, t_2) = \mathrm{J}\left\lfloor \frac{t_2 - O_i - D_i}{T_i} \right\rfloor - \left\lceil \frac{t_1 - O_i}{T_i} \right\rceil + 1\mathrm{K}_0 \tag{2}$$

where $\mathrm{J}x\mathrm{K}_0$ stands for $\max\{0, x\}$.

The following theorem is a necessary and sufficient condition of feasibility based on the DBF, given in [3]:

Theorem 1:

$$\tau \text{ is feasible } \iff \mathrm{dbf}(t_1, t_2) \leqslant t_2 - t_1 \ \forall t_1, t_2 \mid 0 \leq t_1 \leq t_2$$

C. C-space of synchronous systems

The authors of [2] give the following definition of the C-space (based on [1]):

Definition 2: The **C-space** of a task system τ is a region of n dimensions (where each dimension denotes the possible C_i of a task of τ) such that for any vector $C = \{C_1, \cdots, C_n\}$ in it, τ is feasible.

As explained in [6], a region of $\mathbb{N}_0^n$ such as the C-space can be described by a *polytope*, or a list of parametric linear constraints (called the *formulation*). The constraints must be conditions which can unequivocally decide if a point is included in the region or not. In the sequel we give a formulation for the C-space in $\mathbb{N}_0^n$ (For a formulation in $\mathbb{N}^n$, we can add $C_i \geqslant 1\ \forall \tau_i \in \tau$ to the list of constraints).

The DBF test gives us the following list of inequations on the WCET values of the task set which, taken together, are a necessary and sufficient condition of feasibility:

$$\sum_{i=1}^{n} n_i(t_1, t_2) \cdot C_i \leq t_2 - t_1 \ \forall t_1, t_2 \mid 0 \leq t_1 \leq t_2 \tag{3}$$

Note that for given values of (t_1, t_2), the $n_i(t_1, t_2)$ are constant (w.r.t. the C_i). Those inequations are thus indeed linear.

However, this description contains an infinite number of constraints, most of which are redundant. While this does not harm the precision of the test, a finite description is needed in practice. The description should also be as small as possible as this can greatly reduce the computation time of practical feasibility tests. In Section II-D a method to reduce the size of this set of constraints is given.

D. Removing redundant constraints

In the synchronous case, is has been shown in [4] that the constraints corresponding to every interval $[0, t]$ (with t a job deadline happening before H) are necessary and sufficient to test the feasibility of a system.

Even then, the number of constraints to consider is exponential in the number of tasks. While it has been shown [4] that constraints that cover intervals ending after the end of the first busy period are also redundant, this value depends on the C_i and is therefore not usable for the purposes of this paper. The following concept, initially described by [7], is thus introduced. It gives an upper bound for the end of the first busy period and does not depend on the execution times.

Definition 3: A **definitive idle time** (DIT) [7] is a time t (if it exists) such that every job released strictly before instant t has its absolute deadline before or at instant t.

Definition 4: The earliest DIT (if it exists) occurring in the system strictly after $t = 0$ is called the **first DIT**, denoted t_d.

Remark 1: (from previous definitions):

- Contrary to other idle times, DITs are independent of the scheduling or the execution times of the jobs.
- In a feasible system, every DIT is an idle time.
- In a feasible system, the first DIT happens after the end of the first busy period if it exists.
- In synchronous systems any kH with $k \geqslant 0$ is a DIT and we also have $t_d \leqslant H$.

The authors of [2] give a method to find the value of t_d for cases in which periods are pairwise coprime.

For synchronous systems in which the execution times are not known, the first DIT t_d is an upper limit on the end of the busy period regardless of the WCET values. As it was sufficient to check the DBF test in intervals $[0, t]$ before the end of the first busy period, a condition valid for any WCET is as follows:

$$\mathrm{dbf}(0, t) \leqslant t\ \forall t \leqslant t_d \iff \text{The system is feasible} \tag{4}$$

To summarize, the description of the C-space is reduced from an infinite number of constraints to only those where $t_1 = 0$, $t_2 \leqslant t_d$ and values of t_2 correspond to job deadlines. Note that the number of constraints is still exponential in the number of tasks in the worst-case (which is when $t_d = H$).

III. DIT in asynchronous systems

A. Definition

In an asynchronous system, the first DIT may happen before O_{max}, i.e. before all tasks are in the system. We thus introduce the following notion:

Definition 5: The **first periodic DIT** (FPDIT) of a system is the earliest DIT occurring strictly after O_{max}.

By a slight abuse of notation, it will also be noted t_d. Note that in the synchronous case the first periodic DIT is also the first DIT.

B. Existence condition

As in the synchronous case, the problem of existence of the FPDIT can be written as a system of modular equations:

$$\begin{array}{l} \exists?\ t_d\ s.t.\ \forall \tau_i \in \tau, \exists a_i \in [D_i, T_i]\ : \\ \begin{cases} t_d > O_i \\ t_d - O_i \equiv a_i\ (mod\ T_i) \end{cases} \end{array} \tag{5}$$

Where $a \equiv b\ (mod\ n)$ means that a and b are congruent modulo n.

This system is equivalent to multiple instances (one per combination of a_i values) of the Chinese Remainder Problem (CRP) where the goal is to find the value of t_d such that it is congruent to $a_i + O_i$ modulo T_i for all tasks τ_i in τ. To find the FPDIT it is to the best of our knowledge necessary to solve the problem on all combinations of a_i values and then to compute the minimum solution. We define $A_i = [D_i, T_i]$, $X_i = \{x_i = a_i + O_i \mid a_i \in A_i\}$ and $n_i = T_i$ for clarity in the context of the CRP. One instance of this problem is guaranteed to have at least one solution modulo H for given values of a_i if and only if Condition 6 is satisfied [8]:

$$x_i \equiv x_j\ (\text{mod}\ \gcd(n_i, n_j))\ \forall \tau_i, \tau_j \in \tau \tag{6}$$

Condition 6 means that DITs might still exist even though in the general case task periods will not necessarily be pairwise coprime.

C. Solving the modular system

A sketch of the construction algorithm to compute t_d can be found in Algorithm 1 (note that $x^{-1}\ (mod\ y)$ means the modular inverse of x in the naturals modulo y). The congruences must first be modified in order to obtain pairwise coprime moduli. Each n_i is factored into prime powers $p_1^{k_1} p_2^{k_2} \cdots p_m^{k_m}$ with k_j the highest integer s.t. $p_j^{k_j}$ divides $n_i\ \forall j\ \mid 1 \leqslant j \leqslant m$. Each original constraint $t_d \equiv x_i\ (mod\ n_i)$ is replaced by the following equivalent set of constraints [8]:

$$\begin{array}{c} t_d \equiv x_i\ (mod\ p_1^{k_1}) \\ t_d \equiv x_i\ (mod\ p_2^{k_2}) \\ \vdots \\ t_d \equiv x_i\ (mod\ p_m^{k_m}) \end{array}$$

This operation might create duplicate constraints, which are removed. We call C the set of resulting constraints. We define two functions on the set of constraints, assuming contraint c of the form $t_d \equiv x\ (mod\ p)$ we have $p(c) = p$ and $x(c) = x$. As shown in [8], the resulting system can be solved using Algorithm 1.

Algorithm 1: Gauss's CRP Algorithm

Require: H is the hyperperiod of the system
for all $c \in C$ **do**
  $Mc \leftarrow \frac{H}{p(c)}$
  $invMc \leftarrow Mc^{-1}\ (mod\ p(c))$
  $e_c \leftarrow Mc \cdot invMc$
end for
return $\sum_{c \in C} x(c) \cdot e_c\ (mod\ H)$

Algorithm 1 finds one DIT given one combination of a_i values. However using it in practice is very inefficient as there are exponentially many different combinations of a_i values in the number of tasks (the cartesian product of all the A_i sets), and resolving exponentially many CRP instances is prohibitively slow.

We propose a faster method for finding the FPDIT which consists in merging all these CRP instances into one problem (finding all DITs), which can be solved faster by Algorithm 2, as a significant part of the intermediate results are shared between instances. First, the same C set is obtained, but instead of one x_i value per task, all variants in X_i spawn a set of constraints. All duplicate and useless constraints are removed. We then define P the set of all powers of prime numbers appearing as moduli in the congruences (the $p_j^{k_j}$ values). For all $p \in P$, we define C_p the set of constraints where p is the modulo.

Algorithm 2: Modified Gauss Algorithm

Require: P, C_p, $x(c)$: as defined above
Require: H is the hyperperiod of the system
$sumchunks \leftarrow$ empty mapping
for all $p \in P$ **do**
  $taskTerms \leftarrow$ empty list
  $Mp \leftarrow \frac{H}{p}$
  $invMp \leftarrow Mp^{-1}(mod\ p)$
  for all $c \in C_p$ **do**
    append $(x(c) \cdot Mp \cdot invMp)$ to $taskTerms$
  end for
  $sumchunks[p] \leftarrow taskTerms$
end for
$results \leftarrow$ empty list
append 0 to $results$
for all $p \in P$ **do**
  $newresults \leftarrow$ empty list
  for all $chunk \in sumchunks[p]$ **do**
    for all $r \in results$ **do**
      append $(r + chunk)$ to $newresults$
    end for
  end for
  $results \leftarrow newresults$
end for
return $r\ (mod\ H)$ for all $r \in results$

In the first loop over P of Algorithm 2, all the required intermediate values are computed. In the second loop, those intermediate values are added in every possible way. Both Algorithm 1 and Algorithm 2 require the computation of one modular inversion per constraint, which is done in a linear number of modular inversions per task. However, to find all DITs Algorithm 1 has to be executed once for each possible combination of a_i values, which is exponential in the number of tasks. Algorithm 2 has the same theoretical complexity as the computation of all DITs still requires an exponential number of answers in the number of tasks, but it is nonetheless significantly faster in practice.

One drawback of Algorithm 2 is that the replacement of each constraint by sets of constraints with powers of primes as moduli is not equivalent. Algorithm 2 might output values that

are not DITs. However, as we only need the FPDIT, it remains practical to check those solutions one by one from the lowest to the highest, and stop as soon as the first correct result is found.

D. Simulation

This simulation aims to quantify how much system have a FPDIT and which factor can influence this quantity.

We thus checked the existence condition on 10000 systems generated by:

- Generating utilization values with the `UUniFast` algorithm described in [9], with:
  - System utilization chosen uniformly between 0.25 and 0.75
  - The number of tasks set to several values (see Fig. 1)
- Choosing the T_i uniformly between 5 and 20 (in order to have a manageable value of H), which also gives us the $C_i = \lfloor U_i\, T_i \rfloor$
- Choosing the O_i with a normal distribution of mean T_{min} and standard deviation $\frac{T_{max}-T_{min}}{2}$
- Choosing the D_i uniformly in the interval $[T_i - CDF \cdot (T_i - C_i), T_i]$, where CDF varies between 0 and 1 and is called the *Constrained Deadline Factor*. It allows us to generate systems where the deadlines are closer to the period.

The number of tasks and the Constrained Deadline Factor is shown to have a direct influence on the density of FPDIT. Indeed, Fig 1 show the percentage of systems having no FPDIT for different values of CDF.

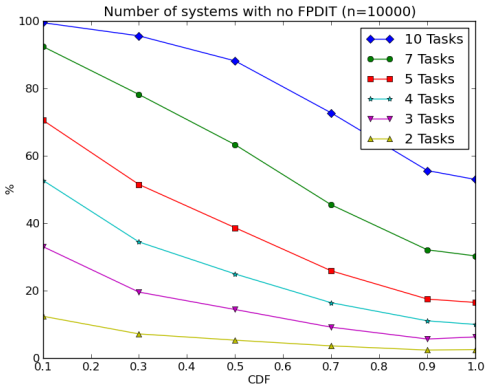

Fig. 1. Evolution of the existence of FPDIT with the CDF

We observe that when the CDF is close to zero, fewer systems have a FPDIT. This is easily explained by the fact that a high CDF means more idle instants for each task, and thus more opportunities for DITs to happen. Similarly, as all tasks must be synchronized for a DIT to happen, a higher number of tasks results in a decrease in FPDIT density.

IV. C-space of asynchronous systems

A. Description using DBF

The description based on [3] and presented in Section II-C was general and is still relevant. Therefore the C-space is still accurately described by the following infinite number of constraints:

$$\sum_{i=1}^{n} n_i(t_1, t_2)\, C_i \leq t_2 - t_1 \;\forall\, 0 \leq t_1 \leq t_2 \tag{7}$$

However, the method used in the synchronous case to remove redundancy does not apply here. A more general approach is presented in the following section.

B. Removing redundant constraints

The first two following sections contain well-known results of feasibility analysis. The third one is a new result based on the FPDIT and the last section contains a general but heavy method for removing constraints redundancies.

1) Job arrival and deadline: Consider a time interval $[t_1, t_2]$. If t_2 is an instant without any job deadline, then by definition the value of $\mathrm{dbf}(t_1, t_2)$ will be equal to $\mathrm{dbf}(t_1, t_2^*)$, where t_2^* is the latest deadline before t_2.

A similar argument can be made if t_1 is an instant without any job arrival (and t_1^* is the earliest arrival after t_1). We can thus restrict ourselves to intervals where t_1 is an instant with at least one job arrival, and t_2 is an instant with at least one job deadline.

2) Feasibility interval: First let us recall how an asynchronous system behaves under an optimal scheduler, for example EDF.

The system begins in the *incomplete period*, where all tasks are not in the system. Then, after O_{max} time units, every task has arrived in the system and the *transient period* begins. Then, after $O_{max} + H$ time units, the system is faced with the same pattern of arrival as in O_{max}. However, because in $O_{max} + H$ every task is in the system and in O_{max} some were missing, the state of the system may differ. The *stationary period* then begins. At instant $O_{max} + 2H$ it is guaranteed that the system is in the exact same situation it was at instant $O_{max} + H$.

As shown in [10], it is sufficient to apply the demand-bound function test in intervals included in the interval $[O_{max}, O_{max} + 2H]$, which has an exponential length in the number of tasks. A better interval is given in [11] using the notion of cyclic idle times (denoted t_c), which are idle times occurring at fixed intervals of length H and whose value depends of the tasks execution time, therefore they cannot be used directly to characterize the C-Space. Their interval is $[0, t_c + H]$ and is valid for every cyclic idle time t_c.

In the following section, we show how the latter interval can be used with the previously defined notion of periodic DIT, a particular case of cyclic idle times which does not depend on the execution times.

3) Using the first periodic DIT: The FPDIT, if it exists, must occur after in the transient period. Indeed, $t_d > O_{max}$ by definition, and if it occurred in the k^{th} stationary period, another DIT should have occurred in the transient period at instant $t_d - kH$ (a contradiction).

Despite t_d being in the theoretical transient period, the system will be in the same state at instant $t_d + H$ (A rigorous proof is given in [11] for any cyclic idle time). We can thus restrict the DBF test to intervals included in the interval $[t_d, t_d + H]$ if the FPDIT exists. This will always be a smaller interval than $[O_{max}, O_{max} + 2H]$ and if the first periodic DIT does not exist (which can be verified as explained in Section III-B), the latter interval can be used.

4) Redundancy as an integer linear problem: Previous sections give us a list of (integer) linear constraints accurately describing the C-space, composed of the DBF test applied to the required intervals. We now give a method to remove redundant constraints from this list.

The general problem of redundancy of some linear constraint $C \cdot X \leq d$ w.r.t. a set of previous linear constraints $A \cdot X \leq B$ in an ILP can itself be written as the ILP presented in Fig. 2.

$$z = \max C \cdot X$$

s.t.

$$\begin{array}{rcl} A \cdot X & \leq & B \\ C \cdot X & \leq & d+1 \end{array}$$

with

$$\begin{array}{rcl} A & : & [n,k] \text{ matrix} \\ B & : & [n,1] \text{ matrix} \\ C & : & [n,1] \text{ matrix} \\ X & : & [1,n] \text{ matrix} \\ d & : & \text{integer} \end{array}$$

Fig. 2. Redundancy of $C \cdot X \leq d$ w.r.t. $A \cdot X \leq B$ as an ILP

Indeed, if the maximization returns $z = d + 1$, it means that there are some integer values accessible with the previous constraints that the new constraint forbids. On the other hands, if the maximization returns $z \leq d$, it means that the limitation expressed in the new constraint is already enforced by the previous constraints. Therefore $z \leq d$ is a necessary and sufficient condition of redundancy of the new constraint.

Such a linear problem can be solved by a branch-and-bound approach [6]. However, in our case the problem is not to test for redundancy of a particular constraint but to find all redundant constraints within a set. A naive algorithm would consist of testing every constraint against all others. We present Algorithm 3, which has the same theoretical complexity but is more efficient when most of the constraints are redundant. Indeed, most of the system we generated had a final C-space description size of less than five constraints, even if the initial description often contained more than a thousand.

This algorithm has an exponential complexity in the number of constraints. The initial description of the C-space should thus be as concise as possible.

To summarize, we first reduced the number of constraints from every possible interval $[t_1, t_2]$ to the intervals where t_1 is an arrival time and t_2 is a deadline in $[t_d, t_d + H]$ if t_d exists, or in $[O_{max}, O_{max} + 2H]$ otherwise. This set of constraints is a correct but redundant description of the C-Space. Redundant constraints are removed through a linear integer approach.

Algorithm 3: Removing redundancy from CSPACE

```
1  CSPACE : set of constraints
2  S ← ∅
3  {1st pass: test each cstr against previous ones}
4  for cstr in CSPACE do
5     if cstr is not redundant w.r.t. S then
6        S.push(cstr)
7     end if
8  end for
9  {2nd pass: test remaining cstr against every other}
10 for cstr in S do
11    cstr = S.pop()
12    if cstr is not redundant w.r.t. S then
13       S.push(cstr)
14    end if
15 end for
16 return S
```

V. Numerical Example

Consider the following task system

	O_i	C_i	D_i	T_i
τ_1	8	C_1	7	15
τ_2	0	C_2	2	5

The FPDIT happens at $t = 15$ and we have $H = 15$. The study interval using the FPDIT is thus $[15, 30]$, which contains 11 intervals $[a, d]$ with a being the arrival of a job, d being the deadline of a job, and $15 \leqslant a < d \leqslant 30$. For comparison, the original study interval of $[O_{max}, O_{max} + 2H]$ is $[8, 38]$ which yields 57 intervals.

For each $[a, d]$, we have a constraint

$$\sum_{i=1}^{n} n_i(a,d)\, C_i \leqslant d - a$$

which give a description of the C-space.

We now apply Algorithm 3 to remove redundancies, leaving us with the following constraints:

$$\left\{ \begin{array}{rcl} C_2 & \leqslant & 2 \\ C_1 + C_2 & \leqslant & 7 \end{array} \right. \tag{8}$$

Let us now consider the same system but with a synchronous pattern of arrival ($O_1 = O_2 = 0$). The first DIT happens at time $t_d = 7$. The intervals to consider are thus of the form $[0, t]$ where t is a deadline happening before or at time t_d. This yields two intervals ($[0, 2]$ and $[0, 7]$) and the following constraints describing the C-space, none of them being redundant:

$$\left\{ \begin{array}{rcl} C_2 & \leqslant & 2 \\ C_1 + 2C_2 & \leqslant & 7 \end{array} \right.$$

We see that the offsets allow three more points to be in the C-space ($(C_1, C_2) = (4, 2), (5, 2), (6, 1)$).

VI. Experimental C-space size gain of asynchronous systems

One of the advantages of asynchronous systems over synchronous systems in the context of periodic tasks is that the critical instants of asynchronous systems can be less constraining for the C-space than synchronous instants (if offsets are chosen to prevent synchronous releases). For this reason, modifying task offsets when possible during the design of a real-time system is a useful trick to fit a heavy periodic task set into a given platform.

In Fig. 3 a number of feasible asynchronous task sets of three tasks without synchronous instants were generated as described in Section III-D. The size of their C-space (number of valid WCET vectors) was compared with that of the same task sets with all offsets set to zero. The systems were generated as in section III-D and the integer linear problems used to remove redundancies were solved via the GLPK tool[1].

In other words, fig. 3 evaluates the feasibility gain of adding random offsets w.r.t. a synchronous task set. Offsets eventually leading to synchronous releases have been discarded. The problem of offset assignment has been considered for example in [12] in the case of fixed WCET values. To the best of our knowledge, finding offsets maximizing a C-space region is an open problem.

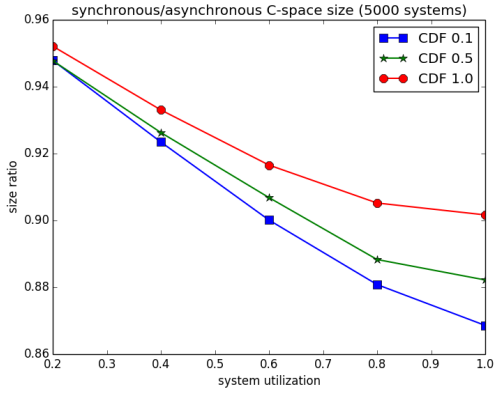

Fig. 3. C-space size ratio between asynchronous and equivalent synchronous systems

The ratio between the size of the synchronous and asynchronous C-spaces of these systems roughly range from 0.95 in the worst case (high utilization, high CDF) to 0.85 in the best case (low utilization, low CDF). Each point on the graph is an average over 5000 tests. Note that the value of the CDF has a greater influence on this ratio than the system's total utilization, but less so when the CDF is higher. It is also important to note that even though a 0.95 ratio may not seem much, the synchronous C-spaces of the systems contain many dominated WCET vectors, and only those that are in the border (i.e. non-dominated) actually matter to the designer of the system. Using the asynchronous C-space instead will relax some constraints and therefore add new acceptable and dominating solutions, increasing the quality of the C-space border.

[1] http://www.gnu.org/software/glpk/

VII. Conclusion

In this paper, we presented a general approach to concisely define the C-space of a constrained periodic task set for EDF on uniform platforms. We then used the C-space to quantify the feasibility gain given by random offset values, with a final result of about 10 % more WCET values being feasible. Possible future works includes an extension to the study of systems with preemption costs or running on multiprocessor platforms, as the value of the DIT does not depend on preemptions or the number of processors.

Future works could also repeat the simulation of section VI with the algorithm described in [12], which determine offset values with optimal feasibility for given WCET values. We could like to study if those heuristics obtained for particular WCETs scenarios can also be interesting for any WCET in the C-Space.

Another direction is to study the shape of the C-space and how it is affected by some factors. An interesting path would be the effect of the CDF on the number of constraints describing the C-space, as we know that in the implicit case this description is simple ($U_{tot} \leqslant 1$). While we quantified the feasibility gain by looking at the ratio between sizes of C-space, another approach would be to look at the ratio between the minimal distance of the border of the C-space, which would give a lower bound of the gain given by the offsets.

References

[1] E. Bini and G. C. Buttazzo, "Schedulability analysis of periodic fixed priority systems," *Computers, IEEE Transactions on*, vol. 53, no. 11, pp. 1462–1473, 2004.

[2] L. George and J.-F. Hermant, "Characterization of the space of feasible worst-case execution times for earliest-deadline-first scheduling," *Journal of Aerospace Computing, Information, and Communication*, vol. 6, no. 11, pp. 604–623, 2009.

[3] S. Baruah, D. Chen, S. Gorinsky, and A. Mok, "Generalized multiframe tasks," *Real-Time Systems*, vol. 17, no. 1, pp. 5–22, 1999.

[4] S. K. Baruah, L. E. Rosier, and R. R. Howell, "Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor," *Real-Time Systems*, vol. 2, no. 4, pp. 301–324, 1990.

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

[6] G. L. Nemhauser and L. A. Wolsey, *Integer and combinatorial optimization*. Wiley New York, 1988.

[7] G. Lipari, L. George, E. Bini, and M. Bertogna, "On the average complexity of the processor demand analysis for earliest deadline scheduling," in *Proceedings of a conference organized in celebration of Professor Alan Burns's sixtieth birthday*, 2013, p. 75, (ISBN 1482707780).

[8] D. E. Knuth, *The Art of Computer Programming*, ser. Seminumerical Algorithms. Addison-Wesley, 1969, vol. 2.

[9] E. Bini and G. C. Buttazzo, "Measuring the performance of schedulability tests," *Real-Time Systems*, vol. 30, no. 1-2, pp. 129–154, 2005.

[10] J. Y.-T. Leung and J. Whitehead, "On the complexity of fixed-priority scheduling of periodic, real-time tasks," *Performance evaluation*, vol. 2, no. 4, pp. 237–250, 1982.

[11] A. Choquet-Geniet and E. Grolleau, "Minimal schedulability interval for real-time systems of periodic tasks with offsets," *Theoretical computer science*, vol. 310, no. 1, pp. 117–134, 2004.

[12] M. Grenier, L. Havet, and N. Navet, "Pushing the limits of can-scheduling frames with offsets provides a major performance boost," 2008.