

The case for FIFO real-time scheduling

Sebastian Altmeyer, Sakthivel Manikandan Sundharam, Nicolas Navet

Angaben zur Veröffentlichung / Publication details:

Altmeyer, Sebastian, Sakthivel Manikandan Sundharam, and Nicolas Navet. 2016. "The case for FIFO real-time scheduling." Luxembourg: University of Luxembourg. <http://hdl.handle.net/10993/24935>.

Nutzungsbedingungen / Terms of use:

Dieses Dokument wird unter folgenden Bedingungen zur Verfügung gestellt: / This document is made available under the following conditions:

Sonstige Open-Access-Lizenz

Weitere Informationen finden Sie unter: / For more information see:
https://www.bibliothek.uni-augsburg.de/opus/lic_sonst.html

licsonst



The Case for FIFO Real-Time Scheduling

Sebastian Altmeyer
University of Luxembourg
FSTC/Lassy
sebastian.altmeyer@uni.lu

Sakthivel Manikandan Sundharam
University of Luxembourg
FSTC/Lassy
sakthivel.sundharam@uni.lu

Nicolas Navet
University of Luxembourg
FSTC/Lassy
nicolas.navet@uni.lu

Abstract—Selecting the right scheduling policy is a crucial issue in the development of an embedded real-time application. Whereas scheduling policies are typically judged according to their ability to schedule task sets at a high processor utilizations, other concerns, such as predictability and simplicity are often overlooked. In this paper, we argue that FIFO scheduling with offsets is a suitable choice when these concerns play a key role. To this end, we examine the predictability of FIFO, present a schedulability analysis for it and evaluate both, performance and predictability of FIFO scheduling with and without offsets. Our results show that FIFO with offsets exhibits competitive performance for task with regular periods, at an unmatched predictability.

I. INTRODUCTION

First in, first Out (FIFO) scheduling — also referred to as, first come, first serve (FCFS) — executes jobs in the exact order of job arrival. No re-ordering or pre-emption can occur, meaning that FIFO scheduling is arguably the simplest scheduling policy with minimal scheduling overhead. This simplicity comes at the cost of performance. FIFO is non-preemptive by nature and provides no means to account for task priorities. Schedulability under FIFO can only be guaranteed for under-utilized systems with uniform period ranges. FIFO, on the other hand, guarantees fairness, which plays an important role in general-purpose applications, but not in the real-time domain. Instead, the focus of the research community has often been on achieving schedulability under unfavorable conditions and high task-set utilizations. As a consequence, FIFO is only considered an option for soft real-time systems, if at all. While we agree with this assessment, we are interested in other concerns than pure performance.

The research on FIFO scheduling is motivated by recent developments in the area of model-based design. We are interested in devising a model-based design environment called CPAL [22, 1], short for Cyber-Physical Action Language, that eases the design and verification of embedded real-time systems. The motivation is to provide an environment where also non-experts are able to quickly model and deploy complex embedded systems without having to master real-time scheduling and resource-sharing protocols. Especially irreproducible faults due to different timing behaviors or race conditions are a nightmare to debug. We acknowledge that techniques to avoid these problems exist, but they constitute a major obstacle for newcomers and make both design and code more complex and error-prone. When processing power is sufficient, as it is increasingly the case with today's hardware,

other concerns than performance such as simplicity and predictability become important.

In this context, we re-visit FIFO scheduling under modified conditions and make a case for FIFO scheduling with strictly periodic task activation and release offsets to increase the predictability and to improve the performance. The contributions of our paper are threefold:

- We show that FIFO with offsets is unique in the sense that it is both work-conserving and exhibits a single, well-defined execution order.
- We provide a schedulability analysis for FIFO, both with and without offsets.
- We evaluate the performance of FIFO scheduling in terms of schedulable task sets, and compare the predictability of FIFO against the two well-known non-preemptive scheduling policies fixed-priority non-preemptive scheduling (FPNS) and non-preemptive earliest deadline first (EDF_{np}) in terms of distinct execution orders.

Related Work: FIFO scheduling has received limited attention in the real-time community, probably due to its inferior performance. George and Minet [11] presented a scheduling analysis for FIFO on a distributed system assuming sporadic task releases, and Leontyev and Anderson [18] presented a tardiness analysis for FIFO scheduling, also assuming a distributed system and sporadic task releases. To the best of our knowledge, FIFO with offsets has not yet been analyzed.

Adding offsets to improve schedulability, has been proposed by Tindell [26] for fixed-priority pre-emptive scheduling (FPPS) and has since been extended to earliest deadline first (EDF) [23, 16], to distributed systems [17] and to non-preemptive EDF (EDF_{np}) [5].

All of these scheduling policies are work-conserving. Non-work-conserving algorithms, as for instance recent work by Nasri and Fohler [21], introduce idle times to delay long tasks that would otherwise block tasks with a shorter deadline. Despite the fundamental difference to FIFO (work-conserving versus non-work-conserving), the motivation to introduce idle-time is the same as to introduce offsets, namely to establish schedulability of an otherwise unschedulable system.

Structure: This paper is structured as follows. In Section II, we explain our system and task model and introduce basic properties of FIFO scheduling, and Section III provides a schedulability test for FIFO with and without offsets. In Section IV, we evaluate FIFO in terms of performance and predictability, and Section V concludes the paper.

II. REAL-TIME SCHEDULING UNDER FIFO

A. Execution Model

We now define the task set and execution parameters. We assume a task set Γ made up of n tasks $\{\tau_1, \dots, \tau_n\}$ running on a single processor. Each task τ_i is represented by a tuple

$$\tau_i: (O_i, C_i, T_i, D_i),$$

where O_i is the task's release offset, C_i the worst-case execution time, T_i the task's period and D_i the deadline. The task instances, also referred to as jobs, are scheduled non-preemptively in order of their arrival. To this end, the scheduler maintains a FIFO queue with ready jobs waiting for dispatch.

In case of simultaneous job arrival, jobs indices are used as a tie breaker: George and Minet [11] have shown that in case of simultaneous job arrival, deadline-monotonic order is optimal. We thus assume without loss of generality that tasks are indexed in deadline monotonic order. The job with lowest index, *i.e.*, with the shortest deadline, is queued first.

We assume constrained deadlines, *i.e.*, $\forall \tau_i: D_i \leq T_i$ and constrained offsets, *i.e.*, $\forall \tau_i: O_i \leq T_i$. The task utilization is defined as

$$U_i = C_i/T_i \quad (1)$$

and the utilization of the complete task set by

$$U_\Gamma = U_i \quad (2)$$

The hyperperiod H of the task set is given by least common multiple of all periods

$$H_\Gamma = \text{lcm}_i\{T_i\} \quad (3)$$

and denotes the time after which the same arrival pattern repeats.

A task produces an infinite sequence of jobs τ_i^j with $j \in \mathbb{N}$. A job's release time is denoted by r_i^j and f_i^j denotes its finishing time. The response time R_i of task τ_i is then given by the maximal delay between the release and completion of a job of τ_i , *i.e.*:

$$R_i = \max_j \{f_i^j - r_i^j\} \quad (4)$$

We distinguish between strictly periodic and sporadic job releases:

1) *Sporadic Release*: In case of sporadic job releases, period T_i is interpreted as the minimal inter-arrival time. The job release times are thus constrained as follows:

$$r_i^0 \geq O_i \quad (5)$$

$$r_i^j \geq r_i^{j-1} + T_i \quad (6)$$

A job's deadline D_i is interpreted relative to the job's release time:

$$d_i^j = r_i^j + D_i. \quad (7)$$

2) *Periodic Release*: In case of strictly periodic release, the job release time r_i^j of job τ_i^j is given by

$$r_i^j = O_i + jT_i \quad (8)$$

and its absolute deadline by

$$d_i^j = O_i + jT_i + D_i. \quad (9)$$

Independently of the release pattern, FIFO scheduling is work-conserving in the sense that it does not introduce any idle times when work is pending. This means that prior to any deadline miss, there must be a busy period in which the processor is not idling. The busy period can be bounded using various independent and incomparable methods:

George et al. [12] presented a bound based on the tasks' deadline and the utilization of the task set:

$$L_U := \max_i \left\{ D_i, D_2, \dots, D_n, \frac{\sum_{i=1}^n (T_i - D_i) U_\Gamma}{1 - U_\Gamma} \right\} \quad (10)$$

Ripoll et al. [24] presented a bound based on the following recursive equation:

$$L_R^{a+1} := \sum_{i=1}^n \frac{L_R^a}{T_i} C_i \quad (11)$$

Since both bounds L_R and L_U are independent, we can take the minimum of both as the task set's busy period L :

$$L := \min\{L_R, L_U\} \quad (12)$$

Naturally, the busy period is only bounded if the task set utilization U_Γ is less than or equal to one.

B. Basic Properties of FIFO Scheduling

In this section, we formulate and prove basic properties of FIFO scheduling. As noted by Leontyev and Anderson [18], non-preemptive execution is implied by the use of a FIFO queue. This represents a stark contrast to other scheduling policies such as EDF and DM, which exist in two variants, preemptive and non-preemptive. Such a distinction is not possible for FIFO as pre-emption would contradict the very nature of FIFO scheduling, and can thus only be realized outside the FIFO scheduling regime.

For the sake of completeness, we repeat the argument of George and Minet [11] that deadline monotonic priority assignment is an optimal tie breaker in case of synchronous task arrival.

Lemma 1. *Any task set schedulable with FIFO scheduling and any arbitrary tie breaker is also schedulable with deadline monotonic as tie breaker.*

Correctness of Lemma 1 is obvious. For any two synchronously released jobs, executing the job with the smallest relative deadline first will not render a schedulable task set unschedulable.

Next, we argue about the sustainability [4] of FIFO scheduling:

Lemma 2. *FIFO schedulable is sustainable with respect to execution times: A schedulable task set will remain schedulable if a task's execution time decreases.*

The execution order of FIFO solely depends on the task release times. Consequently, the tasks will be executed exactly

in the order in which the tasks are released (assuming a potentially unbounded FIFO queue). Reducing a task's execution time can thus only reduce, but never increase, other jobs finishing times. In case of strictly periodic task releases, we can formulate an even stronger version of Lemma 2:

Lemma 3. *Deterministic execution order: Non-preemptive FIFO scheduling with strictly periodic job releases and a deterministic tie breaker enforces a unique execution sequence, which corresponds to sequence of job releases.*

A job's τ_i^d release time r_i^d solely depends on statically defined parameters (see Equation (8)). Any job released prior to r_i^d has already been dispatched or waits in an earlier slot in the FIFO queue. The deterministic tie breaker completes the argumentation. Hence, simulation (assuming worst-case execution times) serves as a valid schedulability analysis, but may be prohibitively slow. We note that the uniqueness of the execution sequence does not entail a fully static schedule (e.g., static cyclic scheduling based on schedule tables) which is non work-conserving. In contrast, FIFO is work-conserving and executes jobs whenever work is pending. An increase in a task period, however, may render a schedulable task set unschedulable as illustrates in Figure 1. Hence, FIFO scheduling is not sustainable in the task's periods.

Lemma 4. *In case of constrained deadlines, a FIFO queue of size is n , i.e., the number of tasks in the system, is sufficient for any schedulable task set.*

The correctness of Lemma 4 can be seen by observing that each task may have at most one job in the FIFO queue at any time. Two jobs of the same task within the FIFO queue immediately implies a deadline miss since we assume constrained deadlines, i.e., $\forall_i D_i \leq T_i$.

C. Advantages of FIFO Scheduling

As confirmed in the experiments of Section IV, FIFO is not a contender to most common real-time scheduling policies with respect to the ability to produce feasible task schedules, especially with long tasks. Only in special cases, FIFO will be able to compete with the two dominant policies, fixed-priority pre-emptive scheduling or earliest deadline first. Even amongst the set of non-preemptive work-conserving scheduling policies without offsets, where non-preemptive EDF is optimal [9], FIFO is unlikely to achieve the same performance.

Yet, it is a common understanding that performance is not the only criterion to select a scheduling policy. Furthermore, real-world task sets often differ strongly from the ones used to evaluate the performance of scheduling policies [10, 3]. In the following, we describe the non-performance related properties of FIFO that can be considered an advantage for FIFO:

- **Simplicity:** FIFO scheduling requires little more than a FIFO queue to store pending jobs and is arguably one of the scheduling policies with lowest implementation overhead. Tasks are executed non-preemptively, which also simplifies the runtime environment and the timing verification.

- **Starvation-Free:** Unless the buffer size is exceeded and jobs are dropped, each job is eventually executed. We note that fairness may be also be considered a dis-advantage since it does not provide a native solution to implement different task criticalities.
- **Work-Conserving:** Similar to EDF or priority-driven scheduling algorithms, FIFO scheduling executes ready jobs and does not introduce unnecessary idle times.

A common assumption in real-time systems are sporadic tasks with minimal inter-arrival times. We deviate from this task model and assume strictly periodic task activation with offsets. Under this restriction, FIFO also exhibits the following advantages:

- **Deterministic Execution Order:** The execution order of FIFO scheduling with offset and strictly periodic task activation is uniquely and statically determined. This means that whatever the execution platform and the task execution times, be it in simulation mode in a design environment or at run-time on the actual target, the task execution order will remain identical. Beyond the task execution order, the reading and writing events that can be observed outside the tasks occur in the same order. This property, leveraged by the CPAL design flow [22], provides a form of timing equivalent behavior between development and runtime phases which eases the implementation of the application and the verification of its timing correctness.
- **Execution Time Sustainability:** FIFO scheduling is sustainable in the tasks' execution times, meaning that if a task set is deemed schedulable and the execution times of the tasks are reduced, the task set remains schedulable.

These properties greatly simplify verification and even enable simulation as a valid (even though in many cases too timing consuming) schedulability test. This is in stark contrast to most non-preemptive scheduling policies, where scheduling anomalies [14] prohibit the use of system simulation for validation purposes.

Furthermore, we have found that FIFO is unique in that it is the only scheduling policy that combines these properties:

Theorem 1. *FIFO scheduling with strictly periodic task releases is the only scheduling policy that is both work-conserving and exhibits a unique execution order.*

Proof. We observe that only upper bounds on the task's execution time are explicitly defined. A lower bound of 0 is implicitly assumed. Consequently, the tasks' actual execution times at runtime can be arbitrary close to 0. In this case, any work-conserving scheduling policy will eventually execute jobs in order of job arrival, which corresponds to FIFO scheduling. Any re-ordering of job executions would violate the uniqueness of the execution order. Hence, we can conclude that FIFO with strictly periodic job release is the only work-conserving scheduling policy with a unique execution order. \square

III. SCHEDULABILITY ANALYSIS

We first revisit the schedulability analysis for sporadic release times presented in [11], correct an assumption about the critical

instance and then provide a schedulability analysis for the more restricted setting of strictly period job release times.

A. Sporadic Release Times

The schedulability test in [11] relies on the hypothesis that the critical instance is given when all jobs are released (i) synchronously and (ii) all subsequent jobs are released at their highest rate, *i.e.*, strictly periodic. The schedulability tests checks for all task release within the busy period whether or not a deadline miss occurs. Whereas a synchronous job arrival indeed constitutes the critical instance for many scheduling policies, FIFO scheduling is an exception. Figure 1 demonstrates the optimism using a task set with harmonic periods and shows that the critical instance is not given in the case of synchronous task release.

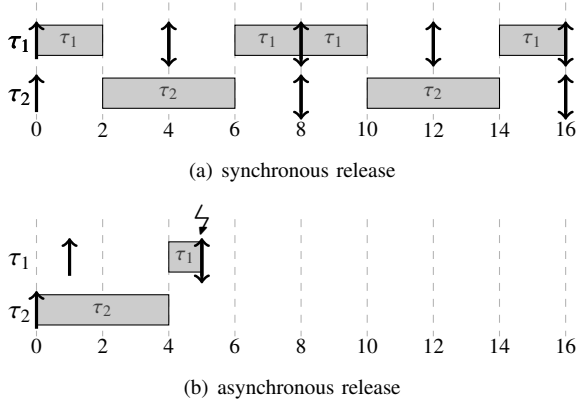


Fig. 1. Synchronous task release is not the critical instance for FIFO scheduling. Indeed, τ_1 meets its deadline in the synchronous case and exceeds it in an asynchronous case. $\Gamma = \{\tau_1, \tau_2\}$, $C_1 = 2$, $C_2 = 4$, $D_1 = T_1 = 4$, $D_2 = T_2 = 8$.

The construction of the critical instance is the key to the schedulability test for FIFO scheduling. Yet, there is one critical instance per task, instead of one instance for all tasks. In case of sporadic task releases, any arrangement of task releases is possible.

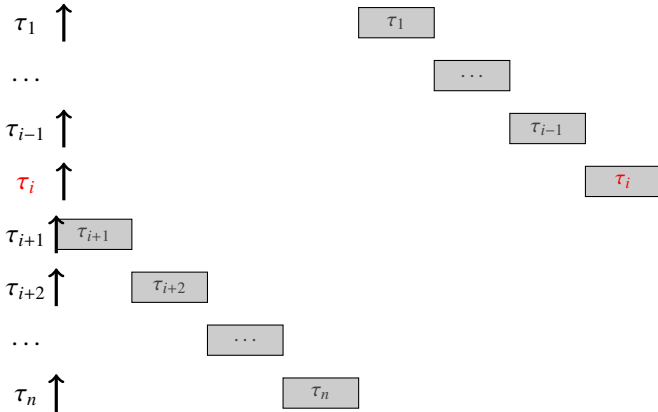


Fig. 2. Critical Instance for task τ_i . Tasks with lower priority than i are released synchronously with i , tasks with lower priority are released ϵ before.

The critical instance for task τ_i occurs when the job τ_i^j is positioned in the very last place in the FIFO queue, *i.e.* when

the queue already contains a job of each other task. Let r_i^j be the release time of job τ_i^j . Tasks with higher priority ¹ have released a job synchronously, *i.e.*

$$\forall_{l < i} \exists k: r_i^j = r_l^k \quad (13)$$

and all tasks with a lower priority have released a job just an $\epsilon > 0$ before, *i.e.*:

$$\forall_{l > i} \exists k: r_l^k = r_i^j - \epsilon \quad (14)$$

Theoretically ϵ can be arbitrary small. In any realistic environment, however, the smallest interval between two job arrivals of different tasks is non-negligible and determined by the implementation of the execution environment and the scheduler.

Consequently, the response time R_i of task τ_i^j in case of FIFO scheduling with sporadic job releases and constrained deadlines is given as follows:

$$R_i = \begin{cases} \sum_{j=\{1, \dots, n\}} C_j - \epsilon & \text{if } i < n \\ \sum_{j=\{1, \dots, n\}} C_j & \text{if } i = n \end{cases} \quad (15)$$

The response time computation is exact and exhibits linear complexity in the number of tasks. It also highlights the low performance of FIFO scheduling: only highly underutilized systems can be deemed schedulable.

B. Strictly Periodic Release Times

Even though we are not aware of any work on FIFO scheduling with offsets, we were able to construct a schedulability analysis for this policy using already established schedulability results. In particular, the schedulability test for EDF with offsets presented by Pellizzoni and Lipari [23].

We note that FIFO is work-conserving in the sense that it does not introduce any idle times when work is pending. This means that prior to any deadline miss, there must be a busy period in which the processor is not idling. As we assume arbitrary offsets and strictly periodic releases, we do not know when a deadline-miss happens and so, would need to validate all busy periods within twice the hyperperiod. To avoid this prohibitively long search, we construct for each task, a hypothetical critical instance leading to a task's first deadline miss. Let τ_i be the task to miss its deadline, and τ_i^j released at r_i^j the corresponding job. The critical instance happens when all tasks other than τ_i release a job as close to r_i^j as possible. If we can prove that despite this pessimistic assumption, job τ_i^j will finish before its deadline d_i^j , we can conclude that no job of task τ_i will ever miss its deadline. If we can repeat the same argumentation for each task in Γ , we can conclude that the complete task set is schedulable.

1) *Construction of $\hat{\Gamma}$* : Formally, we define for each task τ_i a pseudo task-set $\hat{\Gamma}$ that represents the critical instance for task τ_i . The two task sets Γ and $\hat{\Gamma}$ only differ in the task offsets, the rest of the parameters remaining identical. Let $\hat{\tau}_i^j$ be a job that misses its deadline. As we know that in a work-conserving

¹Higher priority means here a smaller index value which is used as the tie-breaker in case of simultaneous releases.

scheduling algorithm, a deadline miss must be within a busy-period L , we set the release time as follows $\hat{r}_i^j = L$ and its deadline to $\hat{d}_i^j = L + D_i$.

We now select the task parameter of each task $\hat{\tau}_l$ with $l \neq i$ to maximize the likelihood of a deadline miss of job $\hat{\tau}_i^j$. To this end, we postpone the job release of the last job of task $\hat{\tau}_l$ executed before the deadline miss as much as possible. An earlier job release will only increase the slack time and so, reduce the pressure on the finishing time of job τ_i^j .

In case of a higher priority task, *i.e.*, $\hat{\tau}_l$ with $l < i$, the job must be released just before or synchronously with $\hat{\tau}_i^j$, whereas tasks with lower priority must be released strictly before $\hat{\tau}_i^j$. Since we use task priorities as a tie breaker, a lower priority task released synchronously with $\hat{\tau}_i$ would be executed after, and not before task $\hat{\tau}_i$.

Pellizzoni and Lipari presented a computation of the minimum distance between any two release times of two different tasks τ_i and τ_l (see [23], Lemma 2). In contrast to their work, we are not only interested in the minimal distance, but also in the minimal distance larger than zero. We therefore repeat the computation of the minimal distance.

Let δ be distance between j th job of task τ_i and the k job of task τ_l :

$$\delta_{i,l} = j \cdot T_i + O_i - k \cdot T_l + O_l \quad (16)$$

By replacing T_i with $x_i \cdot \text{gcd}(T_i, T_l)$ and T_l with $x_l \cdot \text{gcd}(T_i, T_l)$, we get

$$\begin{aligned} \delta_{i,l} &= j \cdot T_i + O_i - k \cdot T_l + O_l \\ &= j \cdot x_i \cdot \text{gcd}(T_i, T_l) + O_i - k \cdot x_l \cdot \text{gcd}(T_i, T_l) + O_l \\ &= (j \cdot x_i - k \cdot x_l) \text{gcd}(T_i, T_l) + O_i - O_l \end{aligned}$$

Since $j \cdot x_i - k \cdot x_l$ can take any arbitrary value, we replace it by x and get

$$\delta_{i,l} = x \cdot \text{gcd}(T_i, T_l) + O_i - O_l \quad (17)$$

Now, we just need to find smallest $\delta_{i,l} \geq 0$ and the smallest $\delta_{i,l} \geq 1$, which are given by

$$x = \frac{O_l - O_i}{\text{gcd}(T_i, T_l)}$$

and

$$x' = \frac{O_l - O_i + 1}{\text{gcd}(T_i, T_l)}$$

Applying these values to Equation (17), we get

$$\Delta_{i,l} = O_i - O_l + \left\lceil \frac{O_l - O_i}{\text{gcd}(T_i, T_l)} \right\rceil \text{gcd}(T_i, T_l). \quad (18)$$

and

$$\Delta'_{i,l} = O_i - O_l + \left\lceil \frac{O_l - O_i + 1}{\text{gcd}(T_i, T_l)} \right\rceil \text{gcd}(T_i, T_l). \quad (19)$$

Finally, we can set the release time of the last job $\hat{\tau}_l^k$ of task $\hat{\tau}_l$ executed before $\hat{\tau}_i^j$ as follows:

$$\hat{r}_l^k = \begin{cases} \hat{r}_i^j - \Delta_{i,l} & \text{if } l \leq i \\ \hat{r}_i^j - \Delta'_{i,l} & \text{if } l > i. \end{cases} \quad (20)$$

The offset of task τ_i is given by

$$\hat{O}_i = \hat{r}_i^j \bmod T_i, \quad (21)$$

and for all other tasks $l \neq i$ by

$$\hat{O}_l = \hat{r}_l^k \bmod T_l. \quad (22)$$

The remaining task set parameters, *i.e.*, the relative deadline, period and execution time remain unchanged. Figure 3 illustrates the task set parameters.

Theorem 2. *A deadline miss of task τ_i in Γ entails a deadline miss of job $\hat{\tau}_i^j$ within task set $\hat{\Gamma}$*

$$\exists k: f_i^k > d_i^k \Rightarrow \hat{f}_i^j > \hat{d}_i^j$$

Proof. Let k be the job index, so that $f_i^k > d_i^k$ holds. We know that prior to the deadline miss of $\tau_{i,k}$, there must be a busy period without any idle time. Let t be the length of this busy period. The number of job releases N_l of task τ_l within $[r_i^k - t: r_i^k]$ is given by

$$N_l = \left\lfloor \frac{t - \text{dist}_{i,l}}{T_l} \right\rfloor + 1$$

where $\text{dist}_{i,l}$ is the distance between r_i^k and the last job of τ_l executed before r_i^k . Analogously, the number of job releases \hat{N}_l of task $\hat{\tau}_l$ within $[\hat{r}_i^k - t: \hat{r}_i^k]$ is given by

$$\hat{N}_l = \left\lfloor \frac{t - \hat{\text{dist}}_{i,l}}{T_l} \right\rfloor + 1.$$

Since there is a deadline miss at d_i^k and an idle time before $r_i^k - t$, we know that

$$\sum_l N_l \cdot C_l > t + D_i.$$

By construction, $\hat{\text{dist}}_{i,l} \leq \text{dist}_{i,l}$, and hence $N_l \leq \hat{N}_l$. Therefore, we can conclude that $\sum_l \hat{N}_l \cdot C_l \geq \sum_l N_l \cdot C_l > t + D_i$, and so, there must be a deadline miss at \hat{d}_i^k , which concludes our proof. \square

2) *Schedulability of $\hat{\tau}_i^j$:* Using Theorem 2, it is sufficient to validate the schedulability of $\hat{\Gamma}$: if $\hat{\tau}_i$ in $\hat{\Gamma}$ is schedulable with FIFO, so is τ_i in Γ . Furthermore, since we know which job of task $\hat{\tau}_i$ will miss its deadline in case of a deadline miss, it is sufficient to concentrate on the j th job $\hat{\tau}_i^j$, which allows us to reduce the analysis time. If we are able to prove or disprove a deadline miss of job $\hat{\tau}_i^j$, we can immediately abort the schedulability analysis of task τ_i . Consequently, we concentrate only on job $\hat{\tau}_i^j$ and ignore all others. First, we define the number of job releases that may postpone the completion of task i within a given time interval.

The function $\eta_l^{\text{inc}}(t_1, t_2)$ denotes the number of job releases of task τ_l within the time interval $[t_1: t_2]$, *i.e.*, including t_2 and is given as follows:

$$\eta_l^{\text{inc}}(t_1, t_2) = \left\lfloor \frac{t_2 - \hat{O}_l}{T_l} \right\rfloor + 1 - \left\lfloor \frac{t_1 - \hat{O}_l}{T_l} \right\rfloor \quad (23)$$

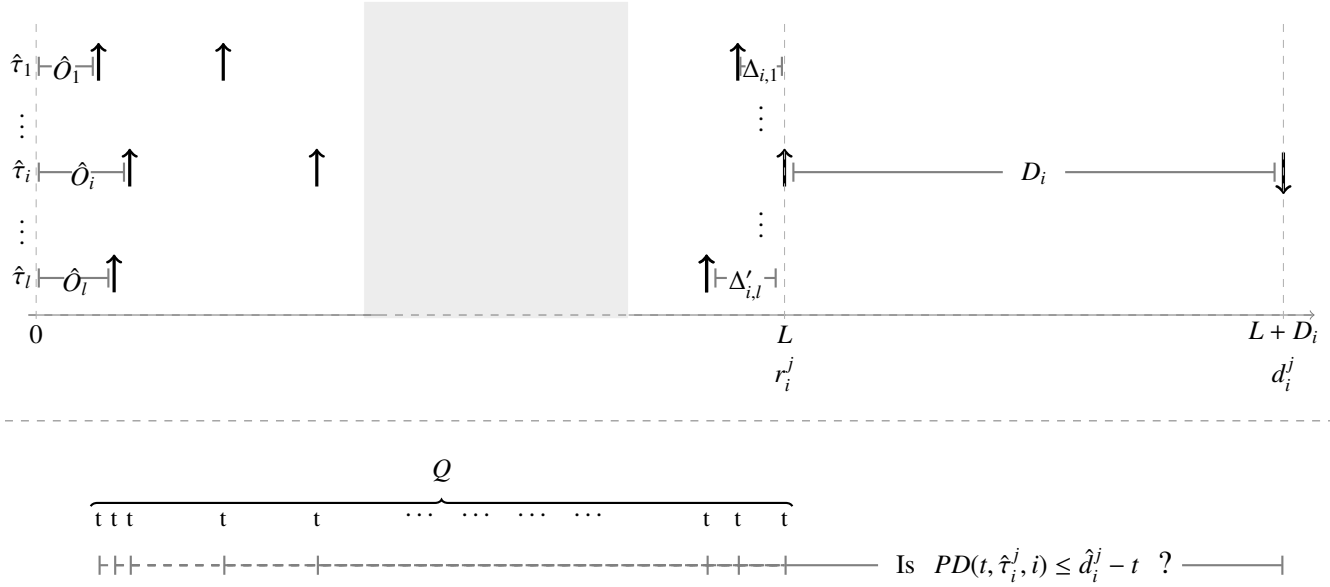


Fig. 3. Illustration of the pseudo task set $\hat{\Gamma}$ and the schedulability test of τ_i with $1 < i < l$. The release time \hat{r}_i^j of job $\hat{\tau}_i^j$ is set to L and its deadline \hat{d}_i^j to $L + D_i$. For all other tasks, the release time of the last job executed before $\hat{\tau}_i^j$ is moved as close to L as possible. For each task release t within $[0; L]$, the schedulability analysis needs to verify that the processor demand does not exceed the available processor time.

The function $\eta_l^{exc}(t_1, t_2)$ denotes the number of job arrivals of task τ_j within the time interval $[t_1; t_2]$, *i.e.*, excluding t_2 and is given as follows:

$$\eta_l^{exc}(t_1, t_2) = \left\lfloor \frac{t_2 - \hat{O}_l}{T_l} \right\rfloor - \left\lfloor \frac{t_1 - \hat{O}_l}{T_l} \right\rfloor \quad (24)$$

Using these two functions, we define the processor demand PD within time interval $[t_1; t_2]$ that can delay the completion of a job of task $\hat{\tau}_i$ released at t_2 :

$$PD(t_1, t_2, i) = \sum_{l \leq i} \eta_l^{inc}(t_1, t_2) \cdot C_l + \sum_{l > i} \eta_l^{exc}(t_1, t_2) \cdot C_l \quad (25)$$

Again, we distinguish between tasks with higher priorities and tasks with lower priorities to correctly account for the tie-breaking policy in case of synchronous job arrivals.

Theorem 3. *A deadline miss is preceded by a busy period, in which the processor demand exceeds the available computation time:*

$$\hat{f}_i^j > \hat{d}_i^j \Leftrightarrow \exists t \in [0; \hat{r}_i^j]: PD(t, \hat{r}_i^j, i) > \hat{d}_i^j - t$$

Proof. We prove both directions separately.

\Leftarrow : We select t so that $PD(t, \hat{r}_i^j, i) > \hat{d}_i^j - t$. By construction, $PD(t, \hat{r}_i^j, i)$ is the computational demand of all jobs released within $[t; \hat{r}_i^j]$ with job $\hat{\tau}_i^j$ being the last to be executed. Hence, the finishing time \hat{f}_i^j of $\hat{\tau}_i^j$ is at least $PD(t, \hat{r}_i^j, i) + t$, *i.e.*, $\hat{f}_i^j \geq PD(t, \hat{r}_i^j, i) + t$ and so, we can conclude that $\hat{f}_i^j > \hat{d}_i^j$.

\Rightarrow : We assume that $\hat{f}_i^j > \hat{d}_i^j$. Let t be the beginning of the busy period prior to the deadline miss. If $PD(t, \hat{r}_i^j, i) \leq \hat{d}_i^j - t$, there is either an idle time prior to \hat{d}_i^j , or the processor is not idle prior to t . Both contradict our assumption that t is the beginning of a busy period. Hence, we can conclude that

$PD(t, \hat{r}_i^j, i) > \hat{d}_i^j - t$, which finishes our proof. \square

Using Theorem 3, we can test for a deadline miss of job $\hat{\tau}_i^j$ as follows:

$$\forall t \in [0; \hat{r}_i^j]: PD(t, \hat{r}_i^j, i) \leq \hat{d}_i^j - t \Rightarrow \hat{f}_i^j \leq \hat{d}_i^j \quad (26)$$

To reduce the number of test, we observe that $PD(t_1, t_2, i)$ only changes at the time of a job release, which means that we only need to validate the schedulability at these points:

$$Q = \{t | \exists l, k: t = k \cdot T_l + \hat{O}_l \wedge t \leq L - D_i\} \quad (27)$$

Hence, we can validate the schedulability of task τ_i as follows:

$$\forall t \in Q: PD(t, \hat{r}_i^j, i) \leq \hat{d}_i^j - t \Rightarrow \hat{f}_i^j \leq \hat{d}_i^j \quad (28)$$

The schedulability analysis for job $\hat{\tau}_i^j$ is illustrated in Figure 3. We note that the schedulability test is sufficient but not necessary. Theorem 2 does not provide an equivalence between the schedulability of Γ and $\hat{\Gamma}$. The schedulability analysis can falsely deem a schedulable task set unschedulable, but not the inverse.

3) *Schedulability Test:* Algorithm 1 shows the complete schedulability test in pseudo-code. Function *computeBusyPeriod* implements Equation (12) and function *computeMinDistance* Equation (18) and (19). In the outer loop (line 4 to 19), the algorithm iterates over all tasks and generates for each task the pseudo task set $\hat{\Gamma}$ (line 7 to 10). In line 12 to 17, the algorithm checks for each point in Q , if the schedulability condition (see Equation (28)) is met. The algorithm terminates either when all tasks are found to be schedulable, or if one unschedulable task is found.

4) *Example:* We illustrate Algorithm 1 using the second task τ_2 of the following example task set Γ :

	C_i	D_i	T_i	O_i
τ_1	1	4	4	0
τ_2	2	4	4	2
τ_3	1	8	9	2

The longest busy period L of Γ is 8. To validate the schedulability of τ_2 , we construct $\hat{\Gamma}$ as follows: The release time \hat{r}_2^j of job of \hat{r}_2^j is set to L and the deadline of that job to $L + D_2$:

$$\hat{r}_2^j = L = 8 \quad \hat{d}_2^j = L + D_2 = 8 + 4 = 12$$

The release times of the jobs \hat{r}_1^k and \hat{r}_3^k are set according to Equation (20):

$$\begin{aligned} \hat{r}_1^k &= \hat{r}_2^j - \Delta_{2,1} = 8 - 2 = 6 \\ \hat{r}_3^k &= \hat{r}_2^j - \Delta'_{2,3} = 8 - 1 = 7 \end{aligned}$$

The set Q is then filled with the release times of jobs of all three tasks within the interval $[0: 8]$:

$$Q = \{0, 2, 4, 6, 7, 8\}$$

For each element in Q , we have to validate Equation (28):

t	$PD(t, \hat{\tau}_2^j, 2)$	\hat{d}_2^j	$\hat{d}_2^j - t$	$PD(t, \hat{\tau}_2^j, 2) \leq \hat{d}_2^j - t?$
0	10	12	12	✓
2	8	12	10	✓
4	7	12	8	✓
6	5	12	6	✓
7	4	12	5	✓
8	2	12	4	✓

Using Theorem 2 and 3, we can conclude that no job of task τ_2 will ever miss deadline when scheduled according to FIFO with the chosen offsets.

Algorithm 1 FIFO Schedulability Test

```

1:  $i = 1$ 
2: isSchedulable = true
3:  $L = \text{computeBusyPeriod}$ 
4: while  $i \leq n \wedge \text{isSchedulable}$  do
5:    $\hat{r}_i^j = L$ 
6:    $\hat{O}_i = r_i^j \bmod T_i$ 
7:   for all  $l$  do
8:      $\hat{d}_{i,l} = \text{computeMinDistance}(i, l)$ 
9:      $\hat{O}_l = r_i^j - \hat{d}_{i,l} \bmod T_i$ 
10:  end for
11:   $Q = \{t | \exists l, k: t = k \cdot T_l + \hat{O}_l \wedge t \leq L\}$ 
12:  for all  $t \in Q$  do
13:    if  $PD(t, \hat{r}_i^j, i) - t > \hat{d}_i^j$  then isSchedulable = false
14:    end if
15:    if  $\neg \text{isSchedulable}$  then break
16:    end if
17:  end for
18:   $i = i + 1$ 
19: end while
20: return isSchedulable

```

C. Offset Optimization

We have so far assumed immutable offsets provided a priori. This assumption may hold, for instance, when precedence constraints are implicitly realized via offset relationships. Often, however, offsets can be considered mutable and offset optimization can provide a means to either improve system performance or even to establish schedulability in the first place. We note that systems with mutable offsets are referred to as *offset free systems* [13].

In recent years, several offset optimization techniques have been developed, such as for instance [13, 15, 20]. The techniques were typically tailored towards a particular scheduling policy (FPPS with offsets [13, 15]) or towards a dedicated application domain (e.g., automotive runnables [20] or avionics networks [19]) and are thus not immediately applicable to FIFO scheduling. Yet, the evaluation in [15] indicates that randomization provides sub-optimal yet satisfactory results. With randomization we refer to a completely random distribution of all task offsets. The feasibility of such a random offset assignment will be validated until either a fixed number of assignments has been unsuccessfully validated, in which case the task set is considered unschedulable, or until a feasible offset assignment has been found. As future work, we intend to evaluate the existing offset optimization techniques towards their usability for FIFO scheduling, and/or to develop an offset optimization specifically tailored towards it. Both research topics, however, are out-of-scope of this paper.

We note that offset optimization does not violate event order determinism: exactly one pre-defined event order is permissible at runtime, only which of the permissible event orders will be selected is decided statically at design-time.

IV. PERFORMANCE OF FIFO SCHEDULING

In this section, we evaluate the behavior of FIFO scheduling with respect to

- its ability to lead to feasible schedules compared to other scheduling policies,
- the precision and analysis time of the schedulability test presented in Section III, and
- the predictability of FIFO scheduling.

We acknowledge that the performance of scheduling policies is highly sensitive to the choice of parameters. This is in particular true for non-preemptive policies. To achieve transparency and to ease the reproduction of the results, the source code of the programs used in our experiments, including the schedulability test, is available online². This source code enables the reproduction of the experiments presented in this section, as well as evaluation for different parameter settings.

A. Experimental Setup

To evaluate the performance of FIFO scheduling in terms of schedulable task sets, we have randomly generated 10000 task sets for each task set utilization from 0.025 to 0.975 in steps of 0.025. We have found that the dominant parameters in case of

²<https://github.com/SebastianAltmeyer/FIFO-Scheduling-Analysis>

FIFO are (i) the period range, (ii) the type of periods and (iii) the granularity. The period range, or to be precise, the difference between smallest and largest period is a common performance indicator for non-preemptive scheduling policies [25]. With granularity of a task set, we refer to the greatest common divisor of all periods and offsets. We have conducted three sets of experiments, one for each of the following period types:

- Random: task periods T_i were randomly chosen in the range [1000: 100.000]ms,
- Loosely-harmonic: the task periods T_i were set to $x \cdot 1000$ ms with x randomly chosen in the range [1: 100],
- Harmonic: task periods T_i were set to $2^x \cdot 1000$ ms with x randomly chosen in the range [0: 7].

Random-period and harmonic-period tasks are on the opposite sides of the spectrum with respect to period randomness, while loosely-harmonic tasks are representative of the common situation in practice where tasks are a multiple of a time quantity larger than the intrinsic granularity of time, such as 5 or 10ms.

The remaining parameters were set as follows:

- the granularity was set to 100ms,
- the task set size was 10,
- task utilizations U_i were generated using UUnifast[7],
- task execution times C_i were set $C_i = T_i \cdot U_i$,
- deadlines were implicit, *i.e.* $D_i = T_i$,
- task offsets were randomly chosen in the range $[0: T_i]$.

The schedulability of each task set has been assessed using the following approaches:

- Fixed-priority pre-emptive scheduling FPPS without offsets (red line),
- Fixed-priority non-preemptive scheduling FPNS without offsets (green line, filled circle),
- FIFO with strictly period task releases, resp. with initial random offset (pink line, empty square) and optimized offsets (dark blue line, triangle),
- FIFO with aperiodic releases (light blue line, filled square).

Optimized offsets means here that we have tried up to 1000 random offset assignments for each task set before concluding that the task was unschedulable.

In addition to the analytical schedulability tests presented in Section III, we have performed simulation to twice the hyperperiod to evaluate the feasibility of FIFO with offsets (black line, empty circle) for harmonic and loosely-harmonic periods. For random periods, simulation is computationally infeasible due to the length of the hyperperiod.

It should be noted that when it can be performed, simulation provides an exact schedulability test for FIFO scheduling with offsets and thus can be used to evaluate the accuracy of the schedulability test in Section III. Simulation is however not usable with FPNS with offsets which is not sustainable in the execution times and thus can exhibit scheduling anomalies. We have not yet found in the literature a feasibility test for FPNS with offsets for the exact same system model (*i.e.*, not using the transaction model that is for instance used in the CAN network schedulability analysis [27]).

B. Schedulability under FIFO

The number of task sets that are schedulable with the different policies under study out of 10000 randomly generated task sets are shown in Figure 4 for random periods, in Figure 5 for loosely-harmonic periods and in Figure 6 for harmonic periods.

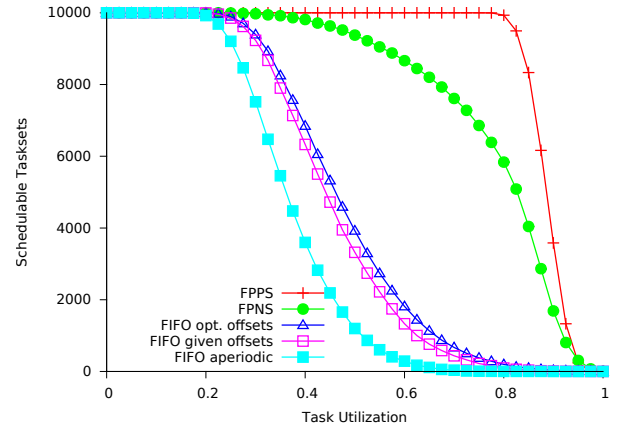


Fig. 4. Evaluation of the base configuration, random periods.

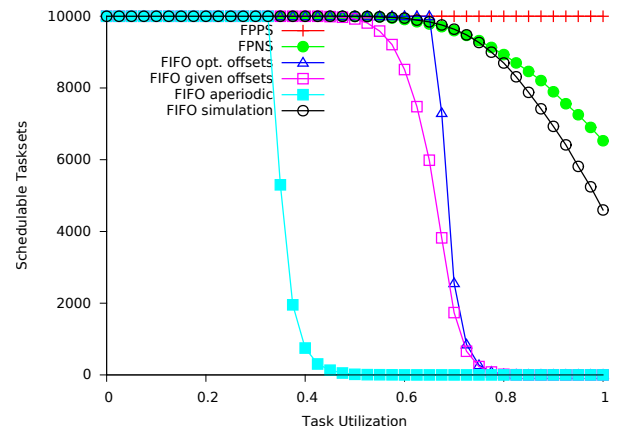


Fig. 5. Evaluation of the base configuration, loosely-harmonic periods.

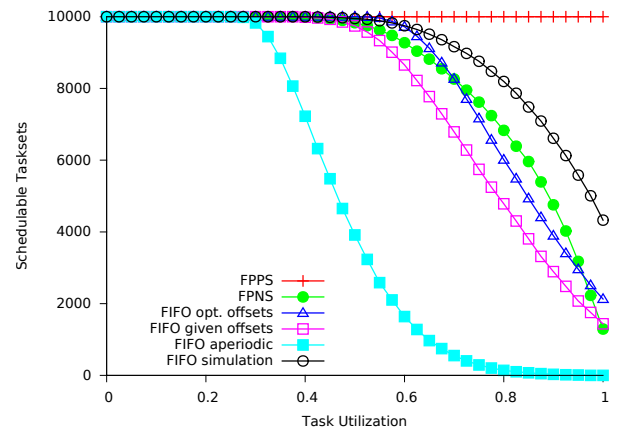


Fig. 6. Evaluation of the base configuration, harmonic periods.

As expected, FIFO with aperiodic task releases performs poorly and is only able to schedule task sets at a low processor utilization, irrespective of period types. In contrast, the performance of the schedulability test for FIFO with strictly period task releases (with and without offset optimization) strongly depends on the type of periods. For random periods, FIFO with offsets performs only slightly better than aperiodic FIFO scheduling. Due to the high variability of the periods, the minimal distances between two job releases of different tasks decreases and the analysis has to assume unfavorable release-scenarios for the pseudo tasks sets $\hat{\Gamma}$. For loosely-harmonic and harmonic periods, adding offsets to FIFO scheduling significantly increases the number of schedulable task sets. Especially for harmonic periods, FIFO with offsets can profit from the very regular periods and is able to compete with FPNS (without offsets). Furthermore, we observe that the regularity of the periods also improves the precision of the analysis; in case of harmonic (Figure 6) and loosely-harmonic periods (Figure 5), the analysis is able to deem a similar number of tasks as the, more much more computationally intensive, simulation, and only fails to compete at high processor load. For instance, as can be seen in Figure 5, results with simulation and analysis remain identical up to 0.65 load.

C. Weighted schedulability measure

To further evaluate the sensitivity of the scheduling policies in terms of (i) number of tasks, (ii) period factor, and (iii) granularity, we use the weighted schedulability measure [6]. The weighted schedulability measure condenses an otherwise three-dimensional graph to two dimensions. It takes the average of the schedulability ratio weighted by the utilization U . Let $S(\Gamma, p)$ be the result of the schedulability test for task set Γ and parameter p . $S(\Gamma, p) = 1$ indicates that the task set is schedulable, otherwise $S(\Gamma, p) = 0$. The weighted measure is defined as follows:

$$W(p) = \frac{\sum_{\Gamma} U \cdot S(\Gamma, p)}{\sum_{\Gamma} U}, \quad (29)$$

The weighted schedulability measure (for 1000 task sets per utilization) for a varying number of tasks is shown here for random task sets in Figure 7, for loosely-harmonic task sets in Figure 8, and for harmonic task sets in Figure 9. FPNS profits from an increasing number of tasks. Non-preemptive scheduling is highly dependent on the difference between shortest and longest task, which tends to decrease as the number of task increase. FIFO, even though also non-preemptive, can not profit from an increasing number of tasks. The positive effect of the decrease in the difference between the tasks' execution times is diminished by the higher pessimism of the schedulability analysis; Indeed, the higher the number of tasks, the more pessimistic the schedulability test due to cumulated conservative assumptions. Consequently, the weighted measure for FIFO scheduling (in all cases) remains largely constant starting from a task set size of 8 onwards. It is also noteworthy that the offset optimization scheme by randomization is only

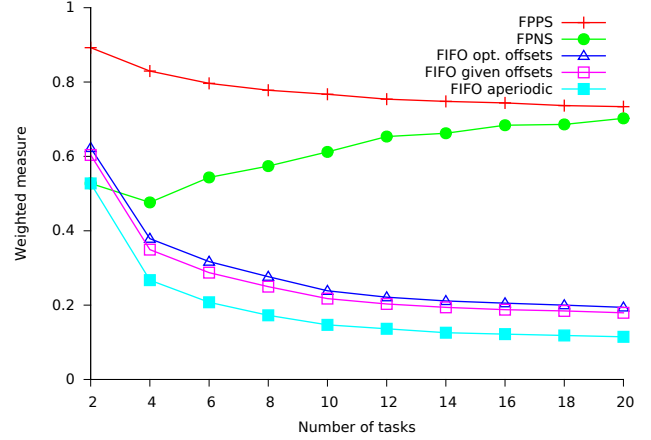


Fig. 7. Weighted schedulability measure, varying number of tasks, random periods.

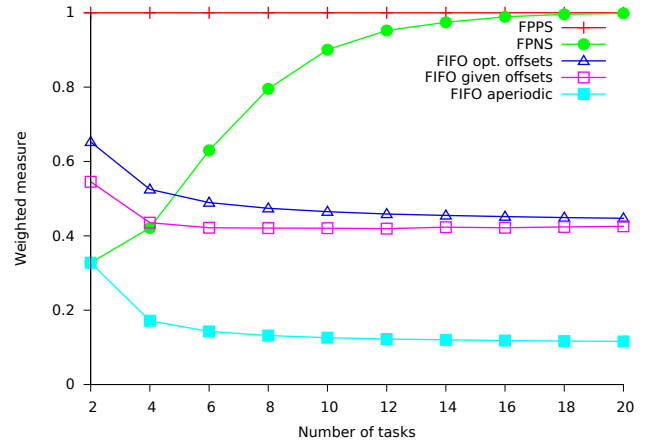


Fig. 8. Weighted schedulability measure, varying number of tasks, loosely-harmonic periods.

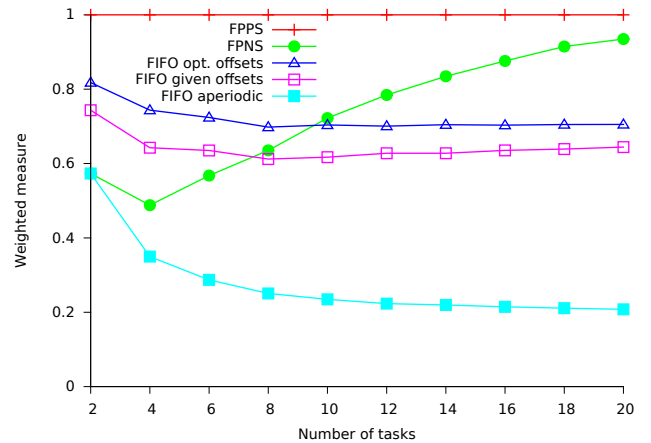


Fig. 9. Weighted schedulability measure, varying number of tasks, harmonic periods.

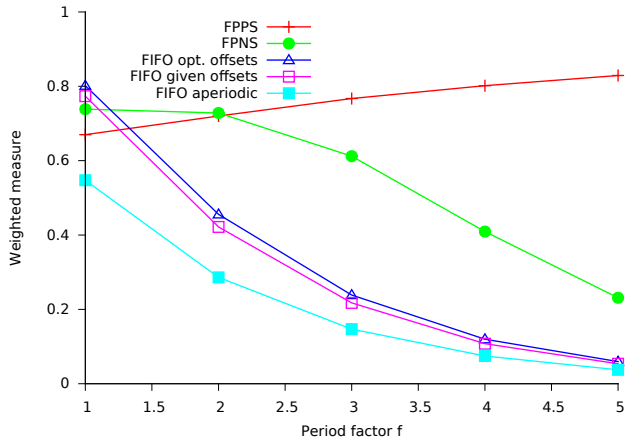


Fig. 10. Weighted schedulability measure, varying period factor ($T_i \in [100: 100 * 10^f]$), random periods.

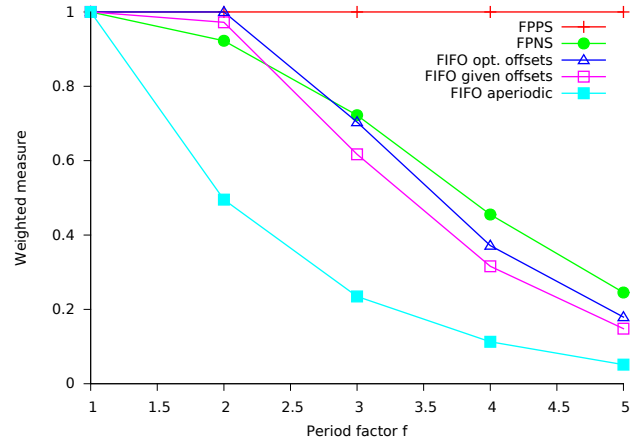


Fig. 12. Weighted schedulability measure, varying period factor ($T_i = 2^x \cdot 1000$, $x \in [0: f]$), harmonic periods.

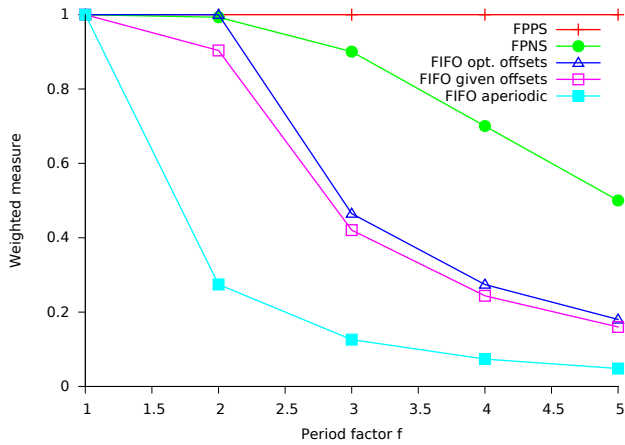


Fig. 11. Weighted schedulability measure, varying period factor ($T_i = x \cdot 1000$, $x \in [1: 10^f]$), loosely-harmonic periods.

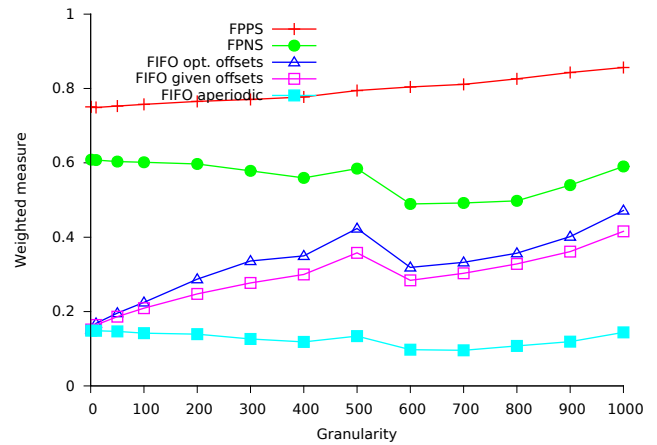


Fig. 13. Weighted schedulability measure, varying granularity, random periods.

marginally efficient (e.g., 5% more schedulable tasks sets than without optimization with 14 tasks).

The evaluation of varying period ranges for loosely-harmonic task sets is shown in Figure 10 to 12. All approaches except for FPPS are similarly affected by an increase in the period ranges due to the long task problem [25].

The evaluation of varying time granularities, defined as the greatest common divisor of all periods and offsets is given in Figure 13 for random periods. Only the results for random periods may profit from an increases in the granularity. For harmonic and loosely-harmonic period, the periods are already regular and changing the offsets has only a limited effect. We observe that the performance of all scheduling policies improve when the granularity increases, with a small peak at a granularity of 500ms. The minimal period is set to 1000ms, which means that setting the granularity to 500 leads to more regular periods than for instance a granularity of 600. In fact, by increasing the granularity, we gradually move from random periods to loosely-harmonic periods. The weighted measures for harmonic and loosely-harmonic periods remain largely

constant (see Figure 14 and 15).

D. Predictability concerns

FIFO with strictly periodic task releases exhibits a unique execution sequence, or event order, which strongly eases verification and validation. To further evaluate the predictability concerns, we have derived the number of distinct event order for two of the dominant non-preemptive scheduling policies, EDF_{np} and FPNS, both with offsets and strictly period task releases. The task activation pattern is thus the same for all three policies and event orders can only differ in the order of task executions, but not task release. Also, we do not record the timing of an event, but only the order of events. For each task set utilization, we have generated 100 task sets and performed 1000 simulations to twice the hyperperiod per task set and scheduling policy. To vary the execution times of the tasks, we have randomly selected a value $C'_i \in [C_i/2: C_i]$ with a uniform distribution. The results are shown in Figure 16 for loosely-harmonic periods and in Figure 17 for harmonic periods. As already discussed by Buttazzo in [8], FPNS shows a higher variability than EDF_{np} , despite the static priorities of FPNS.

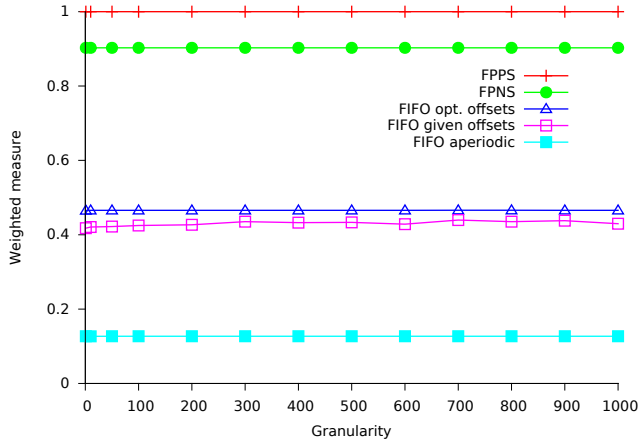


Fig. 14. Weighted schedulability measure, varying granularity, loosely-harmonic periods.

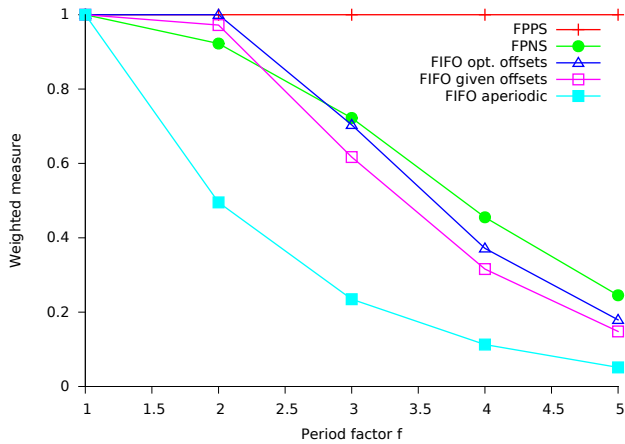


Fig. 15. Weighted schedulability measure, varying granularity, harmonic periods.

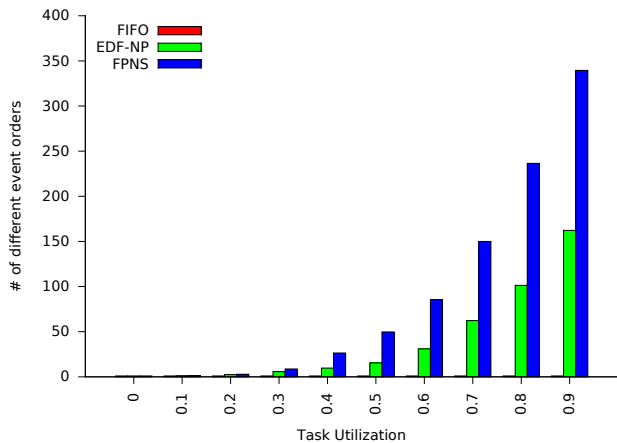


Fig. 16. Number of different event orders, harmonic periods

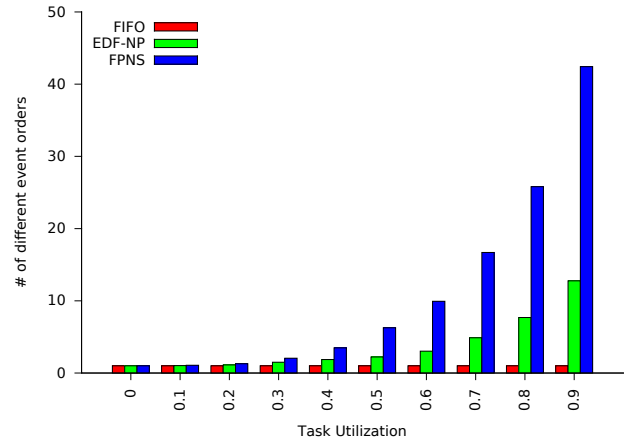


Fig. 17. Number of different event orders, loosely-harmonic periods

Also, quite surprisingly, harmonic periods lead to an order of magnitude higher number of event orders, both for EDF_{np} and FPNS, compared to loosely-harmonic periods. In case of loosely-harmonic periods, task releases are stretched over a longer period of time compared to harmonic period, which reduces the freedom to re-order task executions.

V. CONCLUSION & FUTURE WORK

In this paper, we provided a schedulability test for FIFO with and without offsets and made the case that FIFO scheduling, with strictly periodic tasks and offsets, is a competitive scheduling policy when predictability and simplicity matter. FIFO is non-preemptive by construction and provides no means to account for task priorities. Consequently, FIFO is in general not able to schedule a competitive number of task sets at high processor utilizations as for instance FPNS, let alone pre-emptive scheduling policies.

Adding offsets and enforcing strictly periodic task releases provides two significant advantages to FIFO: (i) The performance issues are mitigated and a higher number of task sets are schedulable, even at high utilization rates, and especially for task sets with harmonic or loosely-harmonic periods and (ii) a unique execution order, defined by the order of job arrivals is enforced which greatly simplifies validation and testing. We have shown that FIFO with offsets is unique in the second property amongst all work-conserving algorithms. It is thus a good fit for our CPAL design flow which aims at automating system synthesis and hiding away from the designer the complexity of the underlying runtime environments, lowering thus the barriers to designing and modeling provably-safe real-time systems. In future works, we plan to consider and evaluate the scheduling overheads of FIFO, but also of the other scheduling policies, and design an offset optimization strategy tailored to FIFO.

ACKNOWLEDGMENT

This research is supported by FNR (Fonds National de la Recherche), the Luxembourg National Research Fund (AFR Grant n°10053122).

REFERENCES

- [1] S. Altmeyer and N. Navet. Towards a declarative modeling and execution framework for real-time systems. In *proceedings of the 1st Workshop on Declarative Programming for Real-Time and Cyber-Physical Systems (DPRTCPS '15)*, 2015.
- [2] S. Altmeyer, S. M. Sundharam, and N. Navet. The case for FIFO real-time scheduling. Technical report, University of Luxembourg, 2016.
- [3] S. Anssi, S. Kuntz, S. Grard, and F. Terrier. On the gap between schedulability tests and automotive task model. *Journal of Systems Architecture*, 59(6):341–350, May 2013.
- [4] S. Baruah and A. Burns. Sustainable scheduling analysis. In *proceedings of the 27th IEEE International Real-Time Systems Symposium (RTSS '06)*, pages 159–168, 2006.
- [5] S. K. Baruah. The non-preemptive scheduling of periodic tasks upon multiprocessors. *Real-Time Systems*, 32(1):9–20, February 2006.
- [6] A. Bastoni, B. Brandenburg, and J. Anderson. Cache-related preemption and migration delays: Empirical approximation and impact on schedulability. In *proceedings of the International Workshop on Operating Systems Platforms for Embedded Real-Time Applications (OSPERT '10)*, pages 33–44, 2010.
- [7] E. Bini and G. C. Buttazzo. Measuring the performance of schedulability tests. *Real-Time Systems*, 30(1):129–154, May 2005.
- [8] G. C. Buttazzo. Rate monotonic vs. EDF: Judgment day. *Real-Time Systems*, 29(1):5–26, January 2005.
- [9] R. I. Davis, A. Thekkilakattil, O. Gettings, R. Dobrin, and S. Punnekkat. Quantifying the exact sub-optimality of non-preemptive scheduling. In *proceedings of the Real-Time Systems Symposium, (RTSS '15)*, pages 96–106, 2015.
- [10] M. Di Natale, C. Dong, and H. Zeng. Reality check: the need for benchmarking in RTS and CPS. In *proceedings of the 4th Real-Time Scheduling Open Problems Seminar (RTSOPS '13)*, pages 18–19, 2013.
- [11] L. George and P. Minet. A FIFO worst case analysis for a hard real-time distributed problem with consistency constraints. In *proceedings of the 17th International Conference on Distributed Computing Systems (ICDCS '97)*, pages 441–448, 1997.
- [12] L. George, N. Rivierre, and M. Spuri. Preemptive and non-preemptive real-time uni-processor scheduling. Technical Report 2966, Institut National de Recherche et Informatique et en Automatique (INRIA), France, 1996.
- [13] J. Goossens. Scheduling of offset free systems. *Real-Time Systems*, 24(2):239–258, March 2003.
- [14] R. L. Graham. Bounds on multiprocessing timing anomalies. *SIAM Journal on Applied Mathematics*, 17(2):416–429, March 1969.
- [15] M. Grenier, J. Goossens, and N. Navet. Near-Optimal Fixed Priority Preemptive Scheduling of Offset Free Systems. In *proceedings of 14th International Conference on Real-Time and Networks Systems (RTNS '06)*, pages 35–42, 2006.
- [16] J. C. P. Gutiérrez and M. G. Harbour. Offset-based response time analysis of distributed systems scheduled under EDF. In *proceedings of the 15th Euromicro Conference on Real-Time Systems (ECRTS '03)*, pages 3–12, 2003.
- [17] R. Henia and R. Ernst. Improved offset-analysis using multiple timing-references. In *proceedings of the Conference on Design, Automation and Test in Europe (DATE '06)*, pages 450–455, 2006.
- [18] H. Leontyev and J. H. Anderson. Tardiness bounds for FIFO scheduling on multiprocessors. In *proceedings of the 19th Euromicro Conference on Real-Time Systems (ECRTS '07)*, pages 71–82, 2007.
- [19] X. Li, J.-L. Scharbag, F. Ridouard, and C. Fraboul. Existing offset assignments are near optimal for an industrial afdx network. *SIGBED Review*, 8(4):49–54, December 2011.
- [20] A. Monot, N. Navet, B. Bavoux, and F. Simonot-Lion. Multisource software on multicore automotive ecus; combining runnable sequencing with task scheduling. *IEEE Transactions on Industrial Electronics*, 59(10):3934–3942, October 2012.
- [21] M. Nasri and G. Fohler. Non-work-conserving scheduling of non-preemptive hard real-time tasks based on fixed priorities. In *proceedings of the 23rd International Conference on Real Time and Networks Systems (RTNS '15)*, pages 309–318, 2015.
- [22] N. Navet, L. Fejoz, L. Havet, and S. Altmeyer. Lean model-driven development through model-interpretation: the CPAL design flow. In *proceedings of the Embedded Real-Time Software and Systems (ERTSS '16)*, pages 207–216, 2016.
- [23] R. Pellizzoni and G. Lipari. Feasibility analysis of real-time periodic tasks with offsets. *Real-Time Systems*, 30(1-2):105–128, May 2005.
- [24] I. Ripoll, A. Crespo, and A. K. Mok. Improvement in feasibility testing for real-time tasks. *Real-Time Systems*, 11(1):19–39, July 1996.
- [25] M. Short. The case for non-preemptive, deadline-driven scheduling in real-time embedded systems. In *proceedings of the World Congress on Engineering (WCE '10)*, pages 399–404, 2010.
- [26] K. Tindell. Adding time-offsets to schedulability analysis. Technical report, University of York, 1994.
- [27] P. M. Yomsi, D. Bertrand, N. Navet, and R. I. Davis. Controller Area Network (CAN): response time analysis with offsets. In *proceedings of 9th IEEE International Workshop on Factory Communication Systems (WFCS '12)*, pages 43–52, 2012.