Infeasibility Test for Fixed-Priority Scheduling on Multiprocessor Platforms

Hoon Sung Chwa, *Member, IEEE*, and Jinkyu Lee, *Senior Member, IEEE*

***Abstract*—Fixed-priority scheduling (FPS), due to its simplicity to implement, has been one of the most popular scheduling algorithms for real-time embedded systems equipped with multiprocessor platforms. While there have been many studies that find sufficient conditions for a given task set to be feasible (schedulable) by FPS with a proper priority assignment, the other direction (i.e., finding infeasible task sets) has not been studied. In this letter, we address a necessary feasibility condition that judges a given task set to be infeasible under FPS with every priority assignment on multiprocessor platforms. To this end, we derive useful properties for the condition and develop the first infeasibility test for FPS on multiprocessor platforms. Via simulations, we show that the proposed infeasibility test discovers a number of FPS-infeasible task sets which are not proven FPS-infeasible by any existing studies.**

***Index Terms*—Fixed-priority scheduling (FPS), infeasibility test, multiprocessor platforms, real-time embedded systems.**

I. Introduction

FIXED-PRIORITY scheduling (FPS) [1] has been widely used in multiprocessor real-time embedded systems, and its fundamental issue is to determine whether a set of real-time tasks can be guaranteed to always meet their deadlines. We may generally classify the research into two problems: 1) to determine the priority ordering of tasks; and 2) to perform feasibility (schedulability) test to determine whether a task set with the given priority ordering is feasible (i.e., guaranteeing all job deadlines).

Studies for both 1) and 2) have matured for FPS on uniprocessor platforms. Deadline-monotonic scheduling [2], where invocations of tasks (called jobs) are scheduled in nondecreasing order of relative deadlines, has been proven to be an optimal priority assignment (OPA), and its exact feasibility test [3] has been also derived. However, for FPS on multiprocessor platforms, no OPA has been known yet, and all existing feasibility tests [4]–[7] are sufficient but not necessary, i.e., task sets that fail the test may still be feasible.

The goal of this letter is to reduce real-time task sets whose feasibility under FPS is unknown by existing studies. In particular, we aim at developing a *necessary feasibility* test that proves the infeasibility of a given task set under FPS on multiprocessor platforms. Addressing such infeasibility is beneficial to real-time systems as follows. First, in a theoretical aspect, it is meaningful to reduce a region of uncertainty between task sets proven feasible and infeasible by FPS. Second, since each study for the FPS schedulability analysis aims at covering additional FPS-feasible task sets that are not deemed schedulable by any existing studies, this letter not only reduces the candidate task sets potentially FPS-feasible but also helps to develop a new FPS schedulability analysis by providing a guideline of how to make a target task set feasible/infeasible by FPS.

However, it is difficult to develop an infeasibility test for FPS because 1) it has not been known how to derive conditions for a task set to be infeasible under FPS and 2) such a condition should not necessitate an investigation of all priority assignments (because the number of priority assignments increases exponentially as the number of tasks increases linearly, i.e., $n!$ versus n). In this letter, we address 1) and 2) as follows. First, we focus on a given task and derive two lower bounds of execution of its higher-priority tasks within a given interval, which hold regardless of relative priorities among the higher-priority tasks; one lower bound is for the m highest-priority tasks, and the other lower bound is for other tasks, where m is the number of processors. Using the two lower bounds, we derive a condition for a given task to be infeasible assuming a set of its higher-priority tasks is known (but their relative priorities are not known). Applying the OPA technique [8] to the derived condition, we develop an infeasibility test that proves a given task set to be infeasible under FPS with every priority assignment on a multiprocessor platform.

Via simulations, we demonstrate that the proposed infeasibility test finds a number of FPS-infeasible task sets. For instance, if we target implicit-deadline task sets whose task set utilization is no smaller than 99% on a two-processor platform, the proposed infeasibility test reveals that 81.7% of them are FPS-infeasible, which has not been proven by any existing studies.

II. System Model, Assumptions, and Notations

This letter studies the feasibility problem on a multiprocessor platform with m (≥ 2) identical processors with the same speed. We consider a sporadic task model [1], [9] composed of n tasks, in which a task $\tau_i \in \tau$ is characterized by (T_i, C_i, D_i) : T_i is the *minimum* interjob separation, C_i is the

Manuscript received August 19, 2021; accepted September 13, 2021. Date of publication September 15, 2021; date of current version May 19, 2022. This work was supported in part by the National Research Foundation of Korea (NRF) under Grant 2017M3A9G8084463, Grant 2021R1A2B5B02001758, and Grant 2020R1F1A1076058 funded by the Korea Government (MSIT); in part by the Institute of Information & Communications Technology Planning & Evaluation (IITP, Resilient Cyber-Physical Systems Research) under Grant 2014-3-00065 funded by the Korea Government (MSIT); and in part by the DGIST R&D Program of MSIT under Grant 20-CoE-IT-01. This manuscript was recommended for publication by P. Roop. *(Corresponding author: Jinkyu Lee.)*

Hoon Sung Chwa is with the Department of Information and Communication Engineering, Daegu Gyeongbuk Institute of Science and Technology, Daegu 42988, Republic of Korea (e-mail: chwahs@dgist.ac.kr).

Jinkyu Lee is with the Department of Computer Science and Engineering, Sungkyunkwan University, Suwon 16419, Republic of Korea (e-mail: jinkyu.lee@skku.edu).

Digital Object Identifier 10.1109/LES.2021.3112671

worst-case execution time, and D_i is the relative deadline. Such a task τ_i is assumed to generate a potentially infinite sequence of jobs, each of which is separated from its predecessor by *at least* T_i time units and is required to complete C_i units of work within D_i time units from its release. We target implicit- and constrained-deadline task sets, in which every task $\tau_i \in \tau$ satisfies $D_i = T_i$ and $D_i \leq T_i$, respectively. We assume quantum-based time, and without loss of generality, a time unit describes a quantum length of 1; all task parameters are specified by the multiples of the quantum length. A single job is assumed to be unable to execute in parallel upon more than one processor. Since any task set with $n \leq m$ is trivially schedulable, we only consider $n > m$.

We consider FPS, under which each task is assigned a unique priority and all jobs released by the task inherit this priority; under FPS, the system chooses to execute at most m highest-priority jobs in each time slot. We assume that jobs are independent and preemptible, and migrate from one processor to another (i.e., global scheduling); also, we assume that the preemption/migration costs are already included in the worst-case execution time. We define FPS-feasibility and FPS-infeasibility as follows.

Definition 1: A task set τ is said to be *FPS-feasible* if there exists *at least one* priority assignment for FPS that satisfies all the jobs of $\tau_i \in \tau$ meet their deadlines for *all* possible legitimate job arrival sequences. Also, a task set τ is said to be *FPS-infeasible* if there exists *no* such priority assignment.

III. Development of Infeasibility Test for FPS

Different from existing feasibility tests for FPS [4]–[7], we aim at developing an infeasibility test for FPS, which proves that a task set is FPS-infeasible (i.e., not feasible under FPS with every priority assignment) on a multiprocessor platform. To this end, we focus on the synchronous periodic release pattern, implying that the first job invoked by each task in the target task set is released at the same time and the following jobs are released in a strictly periodic manner.[1] We then derive a condition for the first job of each task to be infeasible under FPS with every priority assignment. We would like to emphasize that deriving the condition should not necessitate the investigation of all possible priority assignments; this is because, such investigation is intractable for the large number of tasks (due to $n!$ priority assignments with n tasks).

We first consider the following situations. For a given task τ_k, we assume to know a set of its higher-priority tasks [denoted by $\tau^{\mathsf{HI}}(\tau_k)$], but we do not assume to know the relative priorities among tasks in $\tau^{\mathsf{HI}}(\tau_k)$. We now check whether the first job of τ_k triggers its deadline miss when there is no deadline miss of jobs of tasks in $\tau^{\mathsf{HI}}(\tau_k)$. To this end, we need to calculate $E_i(\ell)$ for $\tau_i \in \tau^{\mathsf{HI}}(\tau_k)$, which is the amount of execution of jobs of τ_i in an interval of length ℓ when the first job of τ_i is released at the beginning of the interval and the following jobs are released in a strictly periodic manner. The following lemma calculates a lower bound of $E_i(\ell)$.

[1]By Definition 1, a task set is said to be FPS-infeasible, if there exists a job arrival sequence under which no priority assignment of the task set makes all the jobs meet their deadlines. In this section, we choose the synchronous periodic release pattern, which has been widely used for deriving tight necessary feasibility conditions, e.g., [10].

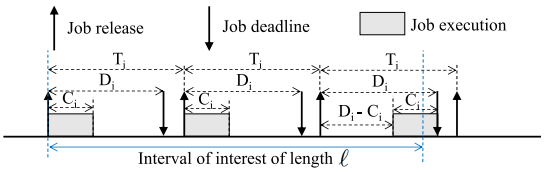

Fig. 1. Release and execution pattern to calculate $W_i(\ell)$.

Lemma 1: $E_i(\ell)$ for τ_i is at least as much as $W_i(\ell)$, which is calculated as follows:

$$W_i(\ell) = \left\lfloor \frac{\ell}{T_i} \right\rfloor \cdot C_i + \max\left(0, \min\left(C_i, \ell - \left\lfloor \frac{\ell}{T_i} \right\rfloor \cdot T_i - (D_i - C_i)\right)\right). \quad (1)$$

Proof: The release and execution pattern for $W_i(\ell)$ is depicted in Fig. 1, and calculated as follows. There are $\lfloor(\ell/T_i)\rfloor$ jobs of τ_i in the interval of length ℓ, whose deadlines are within the interval, e.g., the first and second jobs in Fig. 1. Then, each of the $\lfloor(\ell/T_i)\rfloor$ jobs should execute for C_i time units within the interval; otherwise, it misses its deadline. For a job whose deadline is later than the interval (if any), e.g., the third job in Fig. 1, we focus on the interval between the job's release and the end of the interval of length ℓ and consider three cases: the interval length (i.e., $\ell - \lfloor(\ell/T_i)\rfloor \cdot T_i$) is 1) no larger than $(D_i - C_i)$; 2) larger than D_i; and 3) otherwise. For 1)–3), the job's minimum execution within the interval to avoid its deadline miss is 0, C_i, and $(\ell - \lfloor(\ell/T_i)\rfloor \cdot T_i - (D_i - C_i))$ times units, respectively. The max term in (1) unifies the execution requirements for 1)–3). Therefore, $W_i(\ell)$ is a lower bound for $E_i(\ell)$, which proves the lemma. ■

If we use the fact that on an m-processor platform, every job of the m highest-priority tasks starts its execution as soon as it is released and is not preempted by other jobs, the following lemma derives a tighter lower bound for $E_i(\ell)$.

Lemma 2: If τ_i is one of the m highest-priority tasks, $E_i(\ell)$ for τ_i is at least as much as $W'_i(\ell)$, which is calculated as follows:

$$W'_i(\ell) = \left\lfloor \frac{\ell}{T_i} \right\rfloor \cdot C_i + \min\left(C_i, \ell - \left\lfloor \frac{\ell}{T_i} \right\rfloor \cdot T_i\right). \quad (2)$$

Proof: The release and execution pattern for $W'_i(\ell)$ can be depicted in a figure similar to Fig. 1, where the third job is executed as early as possible. The proof for the $\lfloor(\ell/T_i)\rfloor \cdot C_i$ part is the same as Lemma 1. Since every job of the m highest-priority tasks starts its execution as soon as it is released and is not preempted by other jobs, a job whose deadline is later than the interval (if any) executes for $\min(C_i, \ell - \lfloor(\ell/T_i)\rfloor \cdot T_i)$ within the interval of length ℓ. This proves the lemma. ■

Note that it is easily observed that $W'_i(\ell) \geq W_i(\ell)$ for every τ_i and $\ell \geq 0$.

Using Lemmas 1 and 2, we can derive a condition for a task to be infeasible under FPS assuming its higher-priority set is known (but their relative priorities are not known), recorded in the following lemma.

Lemma 3: Suppose that τ_k's higher-priority tasks [denoted by $\tau^{\mathsf{HI}}(\tau_k)$] are known. Then, τ_k is infeasible under FPS on an m-processor platform, if the following condition holds for at least one α where $1 \leq \alpha \leq C_k$:

$$m \cdot (D_k - C_k + \alpha) < \alpha + \mathtt{Diff}(\tau_k) + \sum_{\tau_i \in \tau^{\mathsf{HI}}(\tau_k)} W_i(D_k - C_k + \alpha) \quad (3)$$

Algorithm 1 FPS-Infeasibility Test on an m-Processor Platform

```
1: for every τ_k ∈ τ do
2:    P_k ← 0 (i.e., the priority for τ_k is not assigned.)
3: end for
4: for Priority p = n, n − 1, n − 2, ... 1 do
5:    for every τ_k ∈ τ|P_k = 0 do
6:       if Eq. (3) does not hold for every 1 ≤ α ≤ C_k then
7:          P_k ← p and break
8:       end if
9:    end for
10:   if there exists no τ_i ∈ τ satisfying P_i = p then
11:      Return "Infeasible"
12:   end if
13: end for
14: Return "No decision"
```

where $\texttt{Diff}(\tau_k)$ is calculated using $\texttt{Diff}_i(\tau_k)$ as follows. For every task $\tau_i \in \tau^{\mathsf{HI}}(\tau_k)$, we compute the value of $\texttt{Diff}_i(\tau_k)$ as $W_i'(D_k - C_k + \alpha) - W_i(D_k - C_k + \alpha)$; for every task $\tau_i \notin \tau^{\mathsf{HI}}(\tau_k)$, we set $\texttt{Diff}_i(\tau_k) = 0$. Then, $\texttt{Diff}(\tau_k)$ is set to the summation of the m smallest values among all $\texttt{Diff}_i(\tau_k)$.

Proof: By Lemma 1, τ_i should execute for at least $W_i(D_k - C_k + \alpha)$ time units within the interval of length $(D_k - C_k + \alpha)$. Also, by Lemma 2, τ_i should execute for at least $W_i'(D_k - C_k + \alpha)$ time units within the interval of length $(D_k - C_k + \alpha)$, if τ_i belongs to the m highest-priority tasks.

Therefore, regardless of relative priorities among tasks in $\tau^{\mathsf{HI}}(\tau_k)$, $\texttt{Diff}(\tau_k) + \sum_{\tau_i \in \mathsf{HI}(\tau_k)} W_i(D_k - C_k + \alpha)$ is a lower bound of the amount of execution of jobs of tasks in $\tau^{\mathsf{HI}}(\tau_k)$ within the interval of length $(D_k - C_k + \alpha)$.

Also, within the interval of length $(D_k - C_k + \alpha)$, the job of τ_k should execute for at least α time units; otherwise, it will miss the deadline.

Hence, the right-hand side of (3) is a lower bound of the amount of execution of the job of τ_k and jobs of tasks in $\tau^{\mathsf{HI}}(\tau_k)$ within the interval of length $(D_k - C_k + \alpha)$, in order to avoid any job deadline miss. Since the amount of total execution within the interval of length $(D_k - C_k + \alpha)$ on an m-processor platform is at most $m \cdot (D_k - C_k + \alpha)$, the lemma holds. ■

Using Lemma 3, we can develop an FPS-infeasibility test on an m-processor platform, presented in Algorithm 1. Since Lemma 3 calculates the minimum execution of higher-priority tasks [using $\texttt{Diff}(\tau_k)$] that does not depend on the relative priorities of the higher-priority tasks, we can apply the OPA technique [8].

In lines 1–3, we set P_k for every $\tau_k \in \tau$ to 0 (which means the priority for τ_k is not assigned). In lines 4–13, we try to assign each priority to a task from the lowest priority ($p = n$) to the highest priority ($p = 1$). For given priority, lines 5–9 find a task whose priority is unassigned and which is not proven infeasible by Lemma 3. Once line 6 finds such a task, we continue repeating the process for the next priority. Otherwise (i.e., if there is no such a task checked by line 10), return "Infeasible" (by line 11); in this case, all tasks whose priorities are unassigned are proven infeasible by Lemma 3. Since no task can be assigned to the given priority, the task set is infeasible under FPS with every priority assignment. On the other hand, if every priority is assigned, the algorithm concludes "No decision" in line 14, which means that it is impossible to prove FPS-infeasibility. The following theorem records the correctness of Algorithm 1.

Theorem 1: If Algorithm 1 returns "Infeasible" for a task set, the task set is infeasible under FPS with every priority assignment on an m-processor platform.

Proof: Suppose that Algorithm 1 returns "Infeasible" for the task set but the task set is actually feasible under FPS with a priority assignment; let $\{P_k^*\}$ denote the priority assignment. Then, there should exist a priority $p = x$ in lines 4–13, such that $p = x$ yields "Infeasible" in lines 10 and 11; then, priorities $p = n, n-1, \ldots, x+1$ are assigned, and priorities $p = x, x-1, \ldots, 1$ are not assigned. Let $\{P_k'\}$ denote the priority assignment by Algorithm 1. We show the contradiction for the following two cases: 1) a set of tasks τ_k that satisfy $x+1 \le P_k^* \le n$ is the same as a set of tasks τ_k that satisfy $x+1 \le P_k' \le n$ and 2) otherwise.

Case (i): Lines 4–13 check whether priority x can be assigned to every task whose priority is not assigned, but Lemma 3 (i.e., line 6) concludes that every task whose priority is not assigned cannot be feasible with priority x. This contradicts that $\{P_k^*\}$ is a priority assignment that makes the task set FPS-feasible in the supposition.

Case (ii): In this case, there should exist τ_k that does not satisfy $x+1 \le P_k^* \le n$, but satisfies $x+1 \le P_k' \le n$. Then, Lemma 3 (i.e., line 6) should conclude that τ_k can be feasible with priority P_k'. However, this is impossible, because Lemma 3 (i.e., line 6) concludes that every task whose priority is not assigned cannot be feasible with priority x for case (i), and $\tau^{\mathsf{HI}}(\tau_k)$ for case (ii) subsumes $\tau^{\mathsf{HI}}(\tau_k)$ for case (i). Therefore, τ_k cannot be feasible with priority $x+1 \le P_k' \le n$. This contradicts the qualification of case (ii).

Both cases contradict, which proves the theorem. ■

IV. Evaluation

In this section, we evaluate the performance of Algorithm 1 in finding FPS-infeasible task sets.

Task Set Generation: To generate task sets to be tested, we consider three parameters: 1) five distributions for task utilization C_i/T_i (bimodal distributions with 0.1, 0.3, 0.5, 0.7, and 0.9); 2) the number of processors ($m = 2$, 4, and 8); and 3) implicit- or constrained-deadline task set ($D_i = T_i$ or $D_i \le T_i$). For given combination of 1)–3), we randomly generate 10 000 task sets according to a popular task set generation method in [11], all of which are not proven infeasible by existing infeasibility tests for implicit- or constrained-deadline task sets [10]; in total, we generate $10\,000 \times 5 \times 3 \times 2 = 300\,000$ task sets.

Evaluation Settings: Since it is known that a task set tends to be FPS-feasible if its task set utilization (i.e., $\sum_{\tau_i \in \tau} C_i/T_i$) is low compared to m, we present three different groups of task sets, whose task set utilization is no smaller than $0.90 \cdot m$, $0.95 \cdot m$, and $0.99 \cdot m$, respectively, and plot the ratio of task sets proven FPS-infeasible by Algorithm 1 for each group in Fig. 2. In this section, we present the evaluation results for implicit-deadline task sets; those for constrained-deadline task sets exhibit a similar trend.

Considering no test has been developed to prove FPS-infeasibility, we apply the following simulation method for checking the effectiveness of Algorithm 1 in finding FPS-infeasible task sets. Focusing on the synchronous periodic release pattern, we simulate the task set by FPS with a given priority assignment until 100 000 time units. We can judge

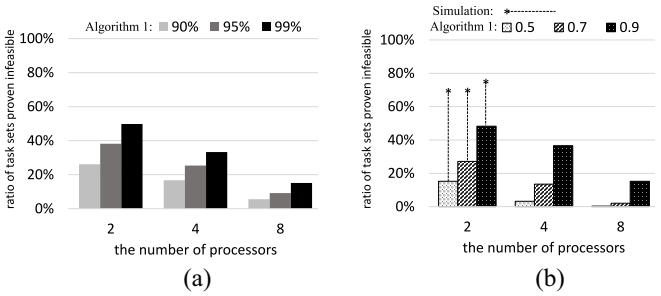

Fig. 2. Ratio of implicit-deadline task sets proven FPS-infeasible by Algorithm 1 [and the simulation method only for $m = 2$ in (b)]. (a) Different task set utilization groups. (b) Different bimodal distributions with at least 90% task set utilization.

that the task set is FPS-infeasible if the simulation with every priority assignment yields at least one job deadline miss. Due to necessitating the simulation for every priority assignment, the simulation method can be applied only for task sets each of whose number of tasks is small; for example, even a task set with $n = 10$ yields $10! \cdot 0.01$ s ≈ 10 h (if it takes 0.01 s to simulate a task set with a priority assignment). Therefore, we apply the simulation method to task sets only for $m = 2$ with bimodal parameters of 0.5, 0.7, and 0.9, each of whose number of tasks is small.

Performance in Finding FPS-Infeasible Task Sets: We have the following four observations. First, as task set utilization becomes closer to m, the ratio of task sets proven FPS-infeasible increases. For example, for the case of $m = 2$ shown in Fig. 2(a), 26.1%, 38.2%, and 49.8% task sets are proven FPS-infeasible if the task set utilization is no smaller than 90%, 95%, and 99%, respectively. This demonstrates that Algorithm 1 properly captures the property of FPS-infeasible task sets: the higher task set utilization, the more FPS-infeasible task sets.

Second, as the number of tasks in each task set gets smaller (which is equivalent to employing a larger bimodal parameter), the ratio of task sets proven FPS-infeasible increases. This is shown in Fig. 2(b), in which we do not plot for bimodal parameters with 0.1 and 0.3 due to their low ratio. For the case of $m = 2$ with task set utilization no smaller than 90%, Algorithm 1 proves 0.6%, 5.2%, 15.2%, 27.1%, and 48.3% task sets to be FPS-infeasible for bimodal distributions with parameters 0.1, 0.3, 0.5, 0.7, and 0.9, respectively. Likewise, the case of $m = 2$ with task set utilization no smaller than 99%, Algorithm 1 proves 2.4%, 6.9%, 30.5%, 57.5%, and 81.7%, respectively. This is due to the pessimism in calculating a lower bound of higher-priority execution; the larger number of tasks, the more pessimism. This commonly happens in interference-based analysis techniques [4]–[7].

Third, as the number of processors gets larger, the ratio of task sets proven FPS-infeasible decreases (shown in Fig. 2). This also comes from the pessimism in calculating a lower bound of higher-priority execution; as the number of processors gets larger, the number of tasks also gets larger, which yields more pessimism.

Finally, Algorithm 1 finds many FPS-infeasible task sets that are covered by the simulation method. For example, Algorithm 1 exhibits $48.3\%/77.9\% = 62.0\%$, $27.1\%/68.3\% = 40.0\%$ and $13.5\%/68.7\% = 19.7\%$ of the simulation method's performance, respectively, in the case of task sets with the bimodal parameter of 0.9, 0.7, and 0.5 for $m = 2$ as shown in Fig. 2(b). We would like to emphasize that Algorithm 1 can be applied to general task sets (even with large n), but the simulation method can be applied only to task sets with small n.

Time Complexity: The time complexity for Algorithm 1 can be calculated as follows. To test Lemma 3 for τ_k (i.e., line 6 in Algorithm 1), we need $O(n)$ calculation for given α, and we have C_k options of α for τ_k; therefore, Lemma 3 for τ_k requires $O(n \cdot \max_{\tau_i \in \tau} C_i)$ computations. Also, lines 4–13 in Algorithm 1 try to test Lemma 3 at most $n + (n-1) + \cdots + 1 = O(n^2)$ times. Therefore, the total time complexity for Algorithm 1 is $O(n^3 \cdot \max_{\tau_i \in \tau} C_i)$.

If a lower time complexity is required, we can only apply $\alpha = 1$ and $\alpha = C_k$ for Lemma 3. This modification decreases the total time complexity of Algorithm 1 into $O(n^3)$, but it slightly drops the performance in finding FPS-infeasible task sets. For example, for the case of $m = 2$ and implicit-deadline task sets, 22.4%, 33.8%, and 47.9% task sets are proven infeasible if the task set utilization is larger than 90%, 95%, and 99%, respectively, when we only apply $\alpha = 1$ and $\alpha = C_k$ to Lemma 3 (which are 26.1%, 38.2%, and 49.8%, respectively when we apply all alphas).

V. Conclusion

In this letter, we addressed the multiprocessor feasibility problem with a special focus on FPS and discovered a number of FPS-infeasible task sets. In the future, we would like to make the proposed FPS-infeasibility test tighter, reducing the gap between the coverage of existing FPS-feasibility tests and that of the proposed FPS-infeasibility test.

References

[1] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *J. ACM*, vol. 20, no. 1, pp. 46–61, 1973.
[2] J. Y.-T. Leung and J. Whitehead, "On the complexity of fixed-priority scheduling of periodic real-time tasks," *Perform. Eval.*, vol. 2, pp. 237–250, Dec. 1982.
[3] M. Joseph and P. Pandya, "Finding response times in a real-time system," *Comput. J.*, vol. 29, no. 5, pp. 390–395, 1986.
[4] M. Bertogna and M. Cirinei, "Response-time analysis for globally scheduled symmetric multiprocessor platforms," in *Proc. IEEE Real Time Syst. Symp. (RTSS)*, Tucson, AZ, USA, 2007, pp. 149–160.
[5] N. Guan, M. Sitgge, W. Yi, and G. Yu, "New response time bounds for fixed priority multiprocessor scheduling," in *Proc. 30th IEEE Real Time Syst. Symp. (RTSS)*, Washington, DC, USA, 2009, pp. 387–397.
[6] N. Guan, M. Han, C. Gu, Q. Deng, and W. Yi, "Bounding carry-in interference to improve fixed-priority global multiprocessor scheduling analysis," in *Proc. IEEE Int. Conf. Embedded Real Time Comput. Syst. Appl.*, Hong Kong, 2015, pp. 11–20.
[7] Q. Zhou, G. Li, and J. Li, "Improved carry-in workload estimation for global multiprocessor scheduling," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 9, pp. 2527–2538, Sep. 2017.
[8] N. Audsley, "Optimal priority assignment and feasibility of static priority tasks with arbitrary start times," Dept. Comput. Sci., Univ. York, York, U.K., Rep. YCS164, 1991.
[9] A. Mok, "Fundamental design problems of distributed systems for the hard-real-time environment," Ph.D. dissertation, Dept. Elect. Eng. Comput. Sci., Massachusetts Inst. Technol., Cambridge, MA, USA, 1983.
[10] T. P. Baker and M. Cirinei, "A necessary and sometimes sufficient condition for the feasibility of sets of sporadic hard-deadline tasks," in *Proc. IEEE Real Time Syst. Symp. (RTSS)*, Rio de Janeiro, Brazil, 2006, pp. 178–190.
[11] T. P. Baker, "Comparison of empirical success rates of global vs. partitioned fixed-priority EDF scheduling for hard real-time," Dept. Comput. Sci., Florida State Univ., Tallahassee, FL, USA, Rep. TR–050601, 2005.