CHALMERS | GÖTEBORG UNIVERSITY

A New Fixed-Priority Assignment Algorithm for Global Multiprocessor Scheduling

RISAT MAHMUD PATHAN
JAN JONSSON

Technical Report No. 2012:10
ISSN 1652-926X

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY/
UNIVERSITY OF GOTHENBURG

A New Fixed-Priority Assignment Algorithm for Global Multiprocessor Scheduling

Risat Mahmud Pathan and Jan Jonsson
Chalmers University of Technology, Sweden
{risat, janjo}@chalmers.se

***Abstract*—Global fixed-priority scheduling of constrained-deadline sporadic tasks systems is important not only for CPU scheduling but also in other domains, for example, scheduling real-time flows in WirelessHART networks designed for industrial process control and monitoring. In this paper, we propose a novel priority assignment scheme for scheduling such task systems on multiprocessors and demonstrate, using proof and simulation, that the scheme is superior to prior schemes.**

I. Introduction

We address the following problem in this paper: Given a collection of n constrained-deadline sporadic tasks, is it possible to meet all the task deadlines when the tasks are executed on m unit-capacity processors using global fixed-priority scheduling?

Single chip multiprocessors are viewed as serious contenders for many hard real-time systems in order to to meet the growing demand of computing power. Many hard real-time applications, e.g., control and monitoring, are often modeled as a collection of recurrent real-time tasks where the instances or jobs of each recurrent task have *hard* deadlines. The fixed-priority (FP) scheduling policy to meet the deadlines of application tasks is the preferred scheduling approach in industry due to its flexibility, ease of debugging and predictability. Considering the trend towards adopting chip multiprocessors for many safety-critical and real-time applications, the study of FP scheduling algorithm and its analysis on multiprocessors is important.

Real-time task scheduling on multiprocessors is primarily based on either the *global* or the *partitioned* approach. In global scheduling, a task is allowed to execute on any processor even when it is resumed after preemption. In partitioned scheduling, each task is assigned to a fixed processor on which the task is allowed to run. A recent survey on various global and partitioned scheduling methods can be found in [12]. We focus on global FP scheduling in this paper. The global FP scheduling of hard real-time systems can be thought of as a 2-phase process: the *assignment of priorities* to the tasks, followed by *schedulability analysis* to determine whether all deadlines will be met under the priority assignment. The major conceptual contribution in this paper is a novel insight that allows us to bound the amount of interference that higher-priority tasks may have on the lower-priority tasks. Based on this insight, we propose a novel priority-assignment algorithm and a corresponding schedulability test, the Fixed-Priority Test (FPT), for global FP scheduling.

Context of this research. The adoption of global scheduler in actual multicore systems is becoming more likely as various mechanisms (e.g., inter-core prefetching [16], locked-cache [20]) are being proposed to reduce migration overhead in the design of recent multicore technology. The analysis of global FP scheduling has also been applied to the end-to-end delay analysis and priority assignment of the periodic real-time flow scheduling on multiple communication channels of WirelessHART networks [18], [19]. WirelessHART is an open wireless sensor-actuator network standard specifically designed for industrial process control to avoid severe economic loss or environmental threats, reduce production inefficiency, enhance equipment monitoring and maintenance [3]. Improvement of global FP schedulability analysis and the priority assignment policy would result in less pessimistic end-to-end delay calculation and would enhance the schedulability of the real-time flows transmitted over multiple communication channels in WirelessHART networks; and consequently, better control and monitoring of industrial processes can be attained.

To achieve economic advantage by hosting multiple avionics functions on a single processor, aviation industry is contemplating integrated modular avionics (IMA) [1]. Similarly, the growing complexity and increased safety requirements in automotive systems have led to the development of AUTOSAR framework focusing composability of components [2]. Version 4.0 of AUTOSAR provides the specification for multicore OS architecture. When multiple functions/components are integrated on the same multicore chip, the tasks/runnables of each function/component can be globally FP scheduled on a (dedicated) subset of the processing cores. This scheduling approach requires no explicit task/runnable assignment algorithm, and more importantly, the temporal behavior of each function can be restricted only to its dedicated cores. Such restriction is necessary and beneficial for function/component upgrade, modification and incremental certification.

In many real-time systems, e.g., avionics, spacecraft and automotive, it is important to efficiently use the processing resources due to size, weight and power constraints. Reducing the resource requirement (e.g., number of processors) of such systems would significantly cut costs for mass production of, for example, cars, trucks or aircrafts. However, if the pessimism in schedulability analysis for such systems is large, then a relatively-higher number of processors is required to meet all the deadlines of the tasks. Our endeavor in this paper is to reduce such pessimism by proposing better schedulability tests for global FP scheduling.

Since the optimal priority assignment for global FP scheduling on a multiprocessor system (at present time) is unknown, the quality (e.g., minimum number of processors required) of many previously proposed global FP schedulability tests depends on the actual priority ordering of the tasks. Therefore, *determining a good priority ordering is as important as deriving a good schedulability test*. The FPT test proposed in this paper is based on a novel fixed-priority assignment policy.

We make the following contributions in this paper. First, we propose a new criterion for reducing the pessimism in estimating the interference inflicted by higher-priority tasks on a lower-priority task. Second, we propose a novel priority-assignment algorithm for global FP scheduling based on this criterion. We prove that if all the tasks are successfully assigned priorities using our proposed priority-assignment policy, then all deadlines of the tasks are met (the FPT test). Third, we prove that the FPT test strictly dominates the HPDALC test [17] (the best global FP schedulability test at present time). Fourth, we empirically show that the schedulability of the FPT test is noticeably better than that of the the HPDALC test. One of the major findings of our empirical study is that *task-set cardinality*, not considered in many previous works, has a significant impact on the global FP schedulability of a task set.

The rest of the paper is organized as follows. Section II presents system models and related work. Building blocks upon which the FPT test is built are presented in Section III. Our new priority-assignment policy and the FPT test are presented in Section IV. Simulation results are presented in Section V. Finally, we conclude the paper in Section VI.

II. Model and Related Work

Task Model. We consider a set of n independent, sporadic tasks in $\Gamma = \{\tau_1, \tau_2, \ldots \tau_n\}$ to be scheduled on m identical processors. Each sporadic task τ_i is characterized by a triple (C_i, D_i, T_i) where: C_i represents the worst-case execution time (WCET); D_i is the relative deadline, which is the length of the scheduling window of each job; T_i is a lower bound on the separation between release times of the jobs of the task. Each job of task τ_i requires at most C_i units of execution time between its release time and deadline which is D_i time units after its release time. The utilization and density of task τ_i are C_i/T_i and C_i/D_i, respectively.

In this paper, all time values (e.g, WCET, deadline, and interval length) are assumed to be positive integer. This is a reasonable assumption since all the events in the system happen only at clock ticks. The cost of different kinds of overhead, for example, context switch, preemption and migration, are assumed to be included in the WCET of each task. Although we do not address such issues in this paper, one can rely on experimental studies (similar to [11]) to measure these overhead costs considering the application, operating system and the target hardware platform.

Related Works. FP schedulability tests can broadly be categorized in two flavors: *utilization-bound tests* and *iterative tests*. Several utilization-based tests exist for global FP scheduling [5], [7], [4], [17]. The idea behind such tests is to compare the total utilization of a taskset against a given bound; if the total utilization does not exceed the bound, then the tasks are guaranteed to meet their deadlines. These tests are very time efficient (linear in the number of tasks) but are not exact (if the total utilization exceeds the utilization bound, the task set may or may not meet all deadlines).

The basic idea of iterative schedulability test is that *one* condition is tested for each lower-priority task. Several iterative tests exist for global FP scheduling of constrained-deadline sporadic tasks [7], [9], [10], [15], [13], [17]. Empirical investigations in [7], [10], [13], [17] show that such tests are highly efficient in determining the schedulability of task sets having a total utilization beyond the state-of-the-art utilization bound. The FPT test proposed in this paper is of the iterative type.

Based on Baker's seminal work in [7], several works [10], [9], [15] have proposed iterative schedulability tests for constrained-deadline sporadic task systems. In [10], [9], [15], the work done by a job of a higher-priority task τ_i is considered as "carry-in" work within the scheduling window of a lower-priority task τ_k if a job of task τ_i is released before the beginning of the window and executes (partially or fully) within the window. If a higher-priority task is considered to constitute carry-in work, then its worst-case interference on the lower-priority task is higher than that of its non-carry-in counterpart [15], [13].

Many global FP schedulability analyses of a lower-priority task τ_k considers all the higher-priority tasks to have carry-in work [10], [9]. Baruah's global EDF schedulability analysis in [8] limits the number of higher-priority tasks considered to have carry-in work to $(m-1)$, where m is the number of processors. The test proposed by Guan et al. [15], here called the RTA-LC test ("LC" stands for "limited carry"), employs the same carry-in task limitation as the analysis in [8] to improve the response-time analysis proposed in [9] for global FP scheduling of constrained-deadline sporadic tasks. Recently, inspired by the works in [9], [8], [15], Davis and Burns [13] proposed a test (the DA-LC test) that also considers $(m-1)$ tasks having carry-in work to improve the schedulability analysis in [10] for global FP scheduling of constrained-deadline sporadic tasks.

The RTA-LC test dominates the DA-LC test for any given fixed-priority ordering of the constrained-deadline tasks [13]. However, Davis and Burns [13] also addressed the problem of finding an effective priority assignment by combining Audsley's optimal priority assignment (OPA) algorithm [6] with the DA-LC test. It is empirically shown that OPA combined with DA-LC is better than an RTA-LC test that uses some other (for example, deadline-monotonic) heuristic priority assignment [13].

Pathan and Jonsson recently proposed a new priority-assignment policy and a corresponding schedulability test, called the HPDALC test [17]. The HPDALC test uses a mechanism, called the hybrid-priority assignment (HPA) policy (to be presented shortly), to improve the priority-assignment policy for the combination of OPA algorithm and the DA-LC test proposed by Davis and Burns in [13]. The HPDALC test considers $(m-1-m')$ carry-in tasks for the schedulability analysis of each lower-priority task where $0 \leq m' < m$. The HPDALC test is the state-of-the-art iterative test for global FP multiprocessor scheduling. We will show that our proposed FPT schedulability test strictly dominates the HPDALC test.

III. Building Blocks of the FPT Test

The priority-assignment policy and the corresponding FPT test proposed in this paper are based on three different building blocks: (i) the DA-LC test [13] (ii) the basic idea of the OPA algorithm [13], and (iii) the HPA policy [17]. In this section, we present these three building blocks upon which the FPT test in Section IV is built.

A. The DA-LC Test

Given a set Γ of constrained-deadline sporadic tasks and a priority assignment on the tasks, the DA-LC test [13] determines the schedulability of each task $\tau_k \in \Gamma$ based on two factors: (i) the upper bound on interference due to the higher-priority tasks on any job of task τ_k, and (ii) the WCET of task τ_k. If the sum of these two factors does not exceed D_k, then the deadline of any job of τ_k is met.

In the remainder of the paper, we call the higher priority task τ_i a "carry-in task"(CI) if it is considered to have carry-in work within the scheduling window of a lower priority task τ_k; otherwise, τ_i is called a "non-carry-in task"(NC).

In order to understand the DA-LC test, we need to know how the **workload**, **interfering workload (IW)**, **total interfering workload (TIW)**, and **interference** within the scheduling window of any job of task τ_k are calculated in [13]. Note that the length of the scheduling window of any job of task τ_k is D_k. Assume that ζ is the set of all the higher-priority tasks of task τ_k.

Workload: The workload of task τ_i within the scheduling window of length D_k is the cumulative length of intervals during which task τ_i executes in that window. We denote by $W_{i,k}^{CI}$ and $W_{i,k}^{NC}$ the upper bounds on the workload of task $\tau_i \in \zeta$ within any interval of length D_k whenever τ_i is a CI task and NC task, respectively. The values of $W_{i,k}^{CI}$ and $W_{i,k}^{NC}$ are given as follows [13]:

$$W_{i,k}^{CI} = N_i^k \cdot C_i + min(C_i, D_k + D_i - C_i - N_i^k \cdot T_i) \quad (1)$$

$$W_{i,k}^{NC} = \lfloor D_k/T_i \rfloor \cdot C_i + min(C_i, D_k - \lfloor D_k/T_i \rfloor \cdot T_i) \quad (2)$$

where $N_i^k = \lfloor (D_k + D_i - C_i)/T_i \rfloor$ in Eq. (1).

Interfering Workload (IW): The IW of task $\tau_i \in \zeta$ is the cumulative length of the intervals during which jobs of task τ_i execute and a job of task τ_k is ready but not executing within its scheduling window of length D_k. We denote by $I_{i,k}^{CI}$ and $I_{i,k}^{NC}$ the upper bounds on the IW of task $\tau_i \in \zeta$ on any job of task τ_k whenever τ_i is a CI task and NC task, respectively. The values of $I_{i,k}^{CI}$ and $I_{i,k}^{NC}$ are given as follows [13]:

$$I_{i,k}^{CI} = min(W_{i,k}^{CI}, D_k - C_k + 1) \quad (3)$$

$$I_{i,k}^{NC} = min(W_{i,k}^{NC}, D_k - C_k + 1) \quad (4)$$

The difference between the carry-in and non-carry-in IW of task τ_i is denoted by $I_{i,k}^{DIFF}$ and is given as:

$$I_{i,k}^{DIFF} = I_{i,k}^{CI} - I_{i,k}^{NC} \quad (5)$$

Total Interfering Workload (TIW): The TIW is the sum of IW of all the tasks in ζ. We denote by $I_k(\zeta)$ the upper bound on TIW due to all the higher-priority tasks in set ζ. The value of $I_k(\zeta)$ is calculated as follows [13]:

$$I_k(\zeta) = \sum_{\tau_i \in \zeta} I_{i,k}^{NC} + \sum_{\tau_i \in Max(\zeta, m-1)} I_{i,k}^{DIFF} \quad (6)$$

where $Max(\zeta, m-1)$ is the set of $(m-1)$ tasks from set ζ that have the largest values of $I_{i,k}^{DIFF}$.

Interference: The interference on a job of task τ_k is the cumulative length of the intervals during which the job of task τ_k within its scheduling window is ready but not executing. Because interference is an integer and all the m processors are busy executing tasks from ζ while task τ_k is interfered, the upper bound on interference due to the tasks in ζ on any job of task τ_k is $\lfloor \frac{I_k(\zeta)}{m} \rfloor$. The DA-LC test for each lower-priority task $\tau_k \in \Gamma$ is given as follows:

$$D_k \geq C_k + \left\lfloor \frac{I_k(\zeta)}{m} \right\rfloor \quad (7)$$

where ζ is the set of higher priority tasks of task τ_k and m is the number of processors.

B. The OPA Algorithm

Audsley's OPA algorithm, originally proposed for uniprocessors in [6], is extended by Davis and Burns for priority assignment in global FP multiprocessor scheduling [13]. The OPA algorithm given in Figure 1 assigns fixed priorities to the tasks in set A to be scheduled on $\hat{m}$ processors based on some global FP schedulability test S.

Algorithm OPA(Taskset A, number of processors $\hat{m}$, Test S)

```
1. for each priority level PL, lowest first
2.   for each priority-unassigned task $\tau \in A$
3.     If $\tau$ is schedulable on $\hat{m}$ processors at priority level PL
4.       according to schedulability test $S$ with all other priority-
5.       unassigned tasks assumed to have higher priorities, Then
6.         assign $\tau$ to priority PL
7.         break (continue outer loop)
8.   return "unschedulable"
9. return "schedulable"
```

Figure 1. Audsley's OPA algorithm for multiprocessors [13].

The OPA algorithm assigns priority to each task in set A starting from the lowest-priority level. In order to be used, the FP schedulability test S has to be OPA-compatible [13] which requires that *the relative priority ordering of the higher priority tasks must be irrelevant to* S. It is proved in [13] that the `DA-LC` test is OPA-compatible.

If the function call OPA(Γ, m, `DA-LC`) returns "schedulable", then all deadlines of the tasks in Γ are met on m processors according to the priorities assigned by the OPA algorithm in Figure 1. Whether or not a (priority-unassigned) task, say task τ, can be assigned the particular priority level `PL` is determined by applying the `DA-LC` test to task τ assuming higher priorities for all other (priority-unassigned) tasks. It is proved in [13] that the combination of OPA and `DA-LC` is the optimal fixed-priority assignment policy when using the `DA-LC` test.

C. Hybrid Priority Assignment (HPA) Policy

Pathan and Jonsson recently observed in [17] that, if not all the higher-priority tasks and all the processors are included when applying the `DA-LC` test to a lower-priority task, the pessimism in the estimation of interference due to the higher-priority tasks may be reduced. Based on this finding, a new priority-assignment policy and a corresponding schedulability test, the `HPDALC` test, is proposed in [17].

The priority assignment for the `HPDALC` test is based on the HPA policy. *The basic idea of the* HPA *policy is to keep some tasks and processors "separate" from the schedulability analysis of a lower priority task.* The priority-assignment policy of the `HPDALC` test works as follows [17]: (i) a total of m' *highest-density* tasks are given the highest fixed priority, and (ii) the priority ordering of the remaining $(n-m')$ lowest-density tasks are determined based on the combination of the OPA algorithm in Figure 1 and the `DA-LC` test using $(m-m')$ processors, $0 \leq m' < m$. A task set Γ passes the `HPDALC` test, if and only if, all the tasks are assigned priority using this scheme.

Notice that the `HPDALC` test "separates" a total of m' highest-density tasks, here referred to as "**separated tasks**", and "separates" a total of m' processors, here referred to as "**separated processors**", from the schedulability analysis of the remaining $(n-m')$ lowest-density tasks. The separated tasks and separated processors are not considered while evaluating the `DA-LC` test for a lower-priority task. Therefore, *the number of `CI` tasks when applying the `DA-LC` test to each of the $(n-m')$ lower-priority tasks is limited to* $(m-1-m')$. At present time the `HPDALC` test is the state-of-the-art iterative test for global FP multiprocessor scheduling.

In this paper, we have developed a new criterion to determine the set of tasks that are separated when analyzing the schedulability of a lower-priority task. Our proposed criterion is special in the sense that it is ***not*** based on "highest density" and separates ***different*** set of tasks for each lower priority tasks. The "separation" of tasks and processors has nothing to do with partitioned multiprocessor scheduling — *the separation only exists as a means for reducing the pessimism of interference due to the higher-priority tasks on a lower-priority task.*

IV. Priority Assignment and The `FPT` Test

In this section, we present our proposed priority-assignment algorithm and the corresponding `FPT` test. First, we present an overview of our proposed priority-assignment policy in subsection IV-A. Then, in subsection IV-B, we present a new and elegant criterion for finding the set of separated tasks for a lower-priority task. Finally, the details of our novel priority-assignment policy, based on the new separation criterion, is proposed in subsection IV-C.

A. Overview of `FPT` Test

Our proposed priority-assignment policy assigns priorities to the tasks starting[1] from lowest-priority level `PL=1` to the highest priority level `PL=n`. At each priority level `PL`, all tasks that are not yet assigned any priority are called the *priority-unassigned tasks*. Our objective is to assign priority to one of the priority-unassigned tasks at each priority level.

Each of the priority-unassigned tasks when selected as a candidate for priority assignment is called the ***target*** task. Given a target task at priority level `PL`, we *temporarily* separate m' processors and separate m' tasks from the set of other priority-unassigned tasks where $0 \leq m' < m$. Unlike the `HPDALC` test, the m' separated tasks are *not* assigned any priority when separated, and more importantly, the criterion for selecting the separated tasks is *not* based on the "highest density". We have proposed a new criterion for selecting the tasks for separation for each target task at each priority level (the criterion will be presented in subsection IV-B).

After separating m' tasks for a particular target task at priority level `PL`, we check (using the `DA-LC` test) whether or not the target task can be assigned priority level `PL`. The separated tasks and separated processors are not considered while evaluating the `DA-LC` test for the target task. If the target task passes the `DA-LC` test at priority level `PL`, then the task is assigned priority level `PL`. If the target task does not pass the `DA-LC` test at priority level `PL`, then another priority-unassigned task is selected as the target for priority assignment at priority level `PL`. If no priority-unassigned task can be assigned priority level `PL`, the priority assignment *fails*. If all tasks are assigned priorities, then the priority assignment *succeeds*.

When a target task can not be assigned priority level `PL`, the corresponding separated tasks and separated processors are no more considered "separated". These tasks along with other priority-unassigned tasks are considered as candidates for selecting the next target task at priority level `PL`. Similarly, if a target task is assigned priority level `PL`, then the

[1]In this paper, we assume without loss of generality that a task having priority level 1 (n) has the lowest (highest) fixed priority.

corresponding separated tasks and separated processors are no more considered "separated". And, these tasks are also considered as candidates for target tasks at next priority level. Thus, the separated tasks and separated processors for each target task are temporary in the sense that *priority assignment for each new target task always starts with all m processors and all the priority-unassigned tasks*.

B. New Criterion for Separation

In this subsection, we propose an elegant criterion for separating the tasks for each target task τ_k. Remember that HPDALC test separates m' *highest-density* tasks from Γ and then applies the combination of OPA and DA-LC test to the remaining $(n-m')$ lowest density tasks using $(m-m')$ processors for some m', $0 \le m' < m$. However, by studying the details of the HPDALC test, we find a very interesting fact: *it is not necessarily the pessimism of the interference estimation of the highest-density tasks that may cause some lower priority task τ_k to fail the DA-LC test*. To see why, consider the following example:

Example 1: Consider four tasks in $\Gamma = \{\tau_1, \ldots \tau_4\}$ to be scheduled on $m = 3$ processors using global FP scheduling. The parameters (C_i, D_i, T_i) of the four tasks are as follows: $(26, 51, 54)$, $(11, 14, 25)$, $(32, 33, 37)$, and $(19, 25, 29)$. The densities of the tasks are $C_1/D_1 = 0.509$, $C_2/D_2 = 0.785$, $C_3/D_3 = 0.967$, and $C_4/D_4 = 0.760$.
The taskset Γ does not pass the HPDALC test. In particular, none of the tasks in Γ can be assigned the lowest priority level PL=1 by separating m' highest-density tasks for any $m' = 0, 1, 2$.
However, there exits a valid priority assignment for Γ. We separate the two tasks $\{\tau_3, \tau_4\}$ and also separate $m' = 2$ processors. The other two tasks $\{\tau_1, \tau_2\}$ are schedulable on $(m - m') = 1$ processor by assigning priority levels PL=1 and PL=2 to tasks τ_1 and τ_2, respectively. Then, the two separated tasks τ_3 and τ_4 are assigned priority levels PL=3 and PL=4, respectively. These two highest priority tasks τ_3 and τ_4 are trivially schedulable since we have $m = 3$ processors; and these two highest priority tasks uses at most two processors at any time. Consequently, the entire taskset is global FP schedulable. ***Note that the two separated tasks τ_3 and τ_4 are not the two highest density tasks.*** $\square$

The lesson learned from this example is that "separation" based on the HPA policy is effective; however, the best criterion to separate the tasks from the schedulability analysis of the lower priority tasks is not necessarily based on "highest density". The crucial observations we make is that the (*constant*) set of m' highest-density tasks may not be the *best* set of separated tasks for the schedulability of the lower-priority tasks. As will be evident now *our proposed criterion separates different sets of tasks for each possible target task at each priority level*.

Proposed Separation Criterion. Consider a target task τ_k at priority level PL such that ξ is the set of all higher-priority tasks of τ_k. Assume that task τ_k does not pass the DA-LC test when considering all the tasks from ξ and all the m processors in the DA-LC test. So, according to Eq. (7), the upper bound on interference, i.e., $\lfloor \frac{I_k(\xi)}{m} \rfloor$, that task τ_k suffers due to the tasks in ξ is greater than $(D_k - C_k)$.

Now, separating m' tasks from set ξ and separating m' processors may able task τ_k to pass the DA-LC test. Our objective is to separate those m' tasks from ξ such that the interference $\lfloor \frac{I_k(\xi)}{m} \rfloor$ is *maximally* reduced. We also separate m' processors. If SEP is the set of m' separated tasks selected from set ξ, then the value of (new) interference on any job of task τ_k (after separation) is $\lfloor \frac{I_k(\xi - \mathtt{SEP})}{m-m'} \rfloor$ where $I_k(\xi - \mathtt{SEP})$ considers $(m-1-m')$ carry-in tasks from set $(\xi - \mathtt{SEP})$. In other words, the problem we are trying to solve is the following: *What is the best way to separate m' tasks from set ξ (i.e., finding set SEP) such that the value of $I_k(\xi)$ is maximally reduced for $m' > 0$?*

Note that when task τ_k fails to pass the DA-LC test before separation of any task from ξ, the value of $I_k(\xi)$ depends on $(m-1)$ carry-in tasks from set ξ. We denote by cis and ncs, respectively, the sets of CI tasks and NC tasks from set ξ such that $\xi = (\mathtt{cis} \cup \mathtt{ncs})$. According to Eq. (6), we have $\mathtt{cis} = Max(\xi, m-1)$, and then obviously $\mathtt{ncs} = (\xi - \mathtt{cis})$. Separating each of the m' tasks from ξ is equivalent to separating the task either from cis or ncs.

We first develop the criterion for separating exactly *one* task from ξ, particularly, separating one task either from set cis or ncs. Then, based on the criterion of separating one task, the criteria for separating subsequent tasks is presented.

(Separation of one task) When $m' = 1$, we separate either one CI-task or one NC-task that needs to be selected either from set cis or ncs, respectively. Remember that we also separate $m' = 1$ processor. Thus, the number of CI tasks after separation is at most $(m-1-m') = (m-2)$ when applying the DA-LC test to task τ_k considering the non-separated tasks from ξ using $(m-m')$ processors.

When separating a CI-task τ_i where $\tau_i \in \mathtt{cis} \subseteq \xi$, the value of $I_k(\xi)$ is reduced by $I^{CI}_{i,k}$ (i.e., the carry-in IW of task τ_i) according to Eq. (6). In order to maximally reduce $I_k(\xi)$ by separating exactly one CI task from cis, the best criterion is to select the task from cis that has the *largest* value of carry-in IW. The largest value of carry-in IW of any task in cis is $\max\limits_{\tau_i \in \mathtt{cis}} I^{CI}_{i,k}$.

Separating a NC-task τ_j where $\tau_j \in \mathtt{ncs} \subseteq \xi$ has *two* effects. First, separating the NC task τ_j from ncs reduces the value of $I_k(\xi)$ by $I^{NC}_{j,k}$ (i.e., the non carry-in IW of τ_j). Second, one of the CI tasks from cis becomes a new NC task since, after separation, we have at most $(m-2)$ carry-in tasks. The CI task from cis that becomes a NC task is the one with the *minimum* value of the difference between its carry-in and non carry-in IW among all the tasks in cis. This is because, after separation, the Max function in Eq. (6) considers $(m-2)$ carry-in tasks that have the largest values of the difference between the carry-in and non carry-in IW. Thus, separating a NC-task τ_j from ncs reduces the value of $I_k(\xi)$ by $(I^{NC}_{j,k} + \min\limits_{\tau_d \in \mathtt{cis}} I^{DIFF}_{d,k})$ where $\min\limits_{\tau_d \in \mathtt{cis}} I^{DIFF}_{d,k}$ is the *minimum* value of the difference between the carry-in and non carry-in IW for any task in cis.

Note that the value of $\min_{\tau_d \in \texttt{cis}} I^{DIFF}_{d,k}$ is completely *independent* of the NC task τ_j that is selected for separation from ncs. Thus, in order to maximally reduce $I_k(\xi)$ by separating exactly one NC task from ncs, the best criterion is to select the NC task from ncs that has the *largest* value of non carry-in IW. The largest value of non carry-in IW of any task in ncs is $\max_{\tau_j \in \texttt{ncs}} I^{CI}_{j,k}$. Whether to separate a CI task or a NC task, when $m' = 1$, is determined as follows.

Criterion For Separating One Task: The task $\tau_a \in \texttt{cis}$ that satisfies $I^{CI}_{a,k} = \max_{\tau_i \in \texttt{cis}} I^{CI}_{i,k}$ is selected for separation if

$$\max_{\tau_i \in \texttt{cis}} I^{CI}_{i,k} > \max_{\tau_j \in \texttt{ncs}} I^{NC}_{j,k} + \min_{\tau_d \in \texttt{cis}} I^{DIFF}_{d,k} \qquad (8)$$

otherwise, task $\tau_b \in \texttt{ncs}$ satisfying $I^{NC}_{b,k} = \max_{\tau_j \in \texttt{ncs}} I^{NC}_{j,k}$ is selected for separation.

(Separation of more than one task) If $m' > 1$, we first separate one task from set $\xi = (\texttt{cis} \cup \texttt{ncs})$ using the criterion in Eq. (8). Then, this separated task, say task τ_s, is *removed* from either cis or ncs depending on whether $\tau_s \in \texttt{cis}$ or $\tau_s \in \texttt{ncs}$, respectively. Now separating the next task is the same as separating one task from the updated set $(\texttt{cis} \cup \texttt{ncs}) = \xi - \{\tau_s\}$ using Eq. (8).

The pseudocode for selecting the m' tasks from set ξ for separation is given in Figure 2. The algorithm Select(ξ, m', τ_k) returns m' separated tasks selected from set ξ considering the target task τ_k.

Algorithm Select(ξ, m', τ_k)

```
1.  cis = $Max(\xi, m-1)$
2.  ncs = $\xi$ - cis
3.  For $g = 1$ to $m'$        // each iteration separates one task
4.      Find the task $\tau_a \in$ cis where $I^{CI}_{a,k} = \max_{\tau_i \in cis} I^{CI}_{i,k}$
5.      Find the task $\tau_b \in$ ncs where $I^{CI}_{b,k} = \max_{\tau_j \in ncs} I^{CI}_{j,k}$
6.      Find the task $\tau_c \in$ cis where $I^{DIFF}_{c,k} = \max_{\tau_d \in cis} I^{DIFF}_{d,k}$
7.      If $(I^{CI}_{a,k} > I^{NC}_{b,k} + I^{DIFF}_{c,k})$ Then
8.          cis = cis $- \{\tau_a\}$
9.      Else
10.         cis = cis $- \{\tau_c\}$
11.         ncs = (ncs $\cup \{\tau_c\}) - \{\tau_b\}$
12.     End If
13. End For
14. Return $\xi -$ (ncs $\cup$ cis)
```

Figure 2. Algorithm for selecting the tasks for separation

We determine the set of CI tasks and NC tasks from set ξ in line 1–2 of Figure 2 where set $Max(\xi, m-1)$ is defined in Eq. (6). Each iteration of the loop in line 3–13 selects one task from $(\texttt{cis} \cup \texttt{ncs})$ for separation. The CI task $\tau_a \in \texttt{cis}$ having the *largest* carry-in IW is determined in line 4. The NC task $\tau_b \in \texttt{ncs}$ having the *largest* non carry-in IW is determined in line 5. The CI task $\tau_c \in \texttt{cis}$ having the *smallest* value of the difference between its carry-in and non carry-in IW is determined in line 6.

The condition in line 7 (based on the criterion in Eq. (8)) determines whether separation of the CI task τ_a or separation of the NC task τ_b would maximally reduce the value of $I_k(\xi)$. If CI task τ_a is separated, i.e., condition in line 7 is true, then τ_a is removed from set cis in line 8. If NC task τ_b is separated, i.e., condition in line 7 is false, then the CI task τ_c determined in line 6 becomes a NC task, and thus task τ_c is first removed from set cis in line 10. Then, task τ_c is included in set ncs, and finally, the NC task τ_b is removed from set ncs in line 11. Separation of the subsequent task in next iteration uses these updated sets of CI and NC tasks. When the for loop exits, the set of total m' separated tasks in $\xi - (\texttt{cis} \cup \texttt{ncs})$ is returned in line 14.

Now we will show in Lemma 1 that our proposed separation criterion is *better* than the separation criterion that is based on the "highest-density".

Lemma 1. *If task τ_k passes the DA-LC test by separating m' highest-density tasks from set ξ of higher priority tasks, then τ_k also passes the DA-LC test by separating the tasks returned by algorithm Select(ξ, m', τ_k) from set ξ, where DA-LC test in both cases after separation uses $(m - m')$ processors and the non-separated tasks from set ξ.*

Proof: Let $\text{SEP}_{density}$ is the set of m' highest density from set ξ and $H_{density} = (\xi - \text{SEP}_{density})$. Let SEP_{our} is the set of m' tasks returned by Select(ξ, m', τ_k) and $H_{our} = (\xi - \text{SEP}_{our})$. If task τ_k passes the DA-LC test by separating the tasks in $\text{SEP}_{density}$ from ξ, then according to DA-LC test in Eq. (7), we must have $\lfloor \frac{I_k(H_{density})}{m-m'} \rfloor \leq (D_k - C_k)$.

Since our separation criterion maximally reduces the value of $I_k(\xi)$ by separating m' tasks from set ξ, we have $I_k(H_{our}) \leq I_k(H_{density})$. Consequently, $\lfloor \frac{I_k(H_{our})}{m-m'} \rfloor \leq (D_k - C_k)$ which implies that τ_k also passes the DA-LC test when m' tasks are separated using algorithm Select(ξ, m', τ_k) from set ξ. ■

The two tasks (i.e., τ_3 and τ_4), separation of which makes the taskset in Example 1 schedulable, can be determined using our separation criterion; but can not be determined using the "highest-density" based separation criterion. Thus, our proposed separation criterion is *better*. We now present the details of our priority assignment policy for global FP scheduling based on this new separation criterion.

C. New Fixed-Priority Assignment Algorithm

The development of our priority assignment algorithm takes the advantage of the HPA policy, applies the DA-LC test to each target task and uses the basic idea of OPA algorithm. The priority assignment to the tasks in Γ starts from the lowest priority level $\text{PL} = 1$ ends at the highest priority level $\text{PL} = n$. The pseudocode of the priority assignment policy of the FPT test is presented in Figure 3.

Initially, all tasks are considered as potential target tasks for priority assignment at the lowest priority level PL=1. All the tasks in set Γ are stored in variable Γ_U (set of priority-unassigned tasks) in line 1. Each iteration of the loop in line 2–22 represents one priority level starting from the lowest priority level PL=1 to the highest priority level PL=n.

At each priority level PL, the loop in line 3–20 considers one-by-one priority-unassigned task in Γ_U until one such task is assigned the priority level PL. During each iteration of the loop in line 3–20, a new task $\tau_k \in \Gamma_U$ is selected as a target task in line 3. The set of other priority-unassigned tasks $\xi = (\Gamma_U - \{\tau_k\})$ is determined in line 4. If the target task τ_k is eventually assigned the priority level PL, then the tasks in ξ will have higher priorities than task τ_k.

For a given target task τ_k, we (temporarily) separate m' tasks from set ξ and we also separate m' processors. During each iteration (using the variable $m' = 0, \ldots (m-1)$) of the loop in line 5–19 a total of m' tasks from set ξ are separated in line 6 by calling algorithm Select(ξ, m', τ_k). The other non-separated, priority-unassigned tasks are stored in set H in line 6 where $H = (\xi -$ Select(ξ, m', τ_k)). Notice that the separated tasks for each target task may be *different*. Next the DA-LC test is applied in line 7 to determine if the target task τ_k can be assigned priority level PL by assuming the higher priorities of the tasks in set H. In such case, the DA-LC test uses $(m - m')$ processors.

If the DA-LC test in line 7 is satisfied, then task τ_k is assigned priority level PL in line 8 and removed from the set of priority-unassigned tasks in line 9. If the current priority level PL is equal to $(n-m)$, i.e., condition in line 10 is true, then there are exactly m (priority-unassigned) tasks in Γ_U after τ_k is removed from Γ_U in line 9. And, each of these m priority-unassigned tasks in Γ_U is assigned one unique priority level between PL=(n-m+1) and PL=n in line 12–13 (note that these are the m highest priority tasks and are always schedulable). At this point, all tasks are assigned priorities and the algorithm returns "schedulable" in line 14. If the current priority level PL is less than $(n-m)$, i.e., the condition in line 10 is false, then the priority assignment for next priority level starts (jumping from line 16 to line 2).

If the DA-LC test for task τ_k in line 7 is never satisfied for any m', $0 \le m' < m$, then the for loop in line 5–19 exits; and the loop in line 3–20 begins by selecting another new target task. If no new task can be selected as a target task at line 3, then the for loop in line 3–20 exits. Since at this stage there is *no* task that is assigned the current priority level PL, the algorithm returns "unschedulable" in line 21.

Notice that if a target task can not be assigned priority level PL, the corresponding separated processors and separated tasks are *no* more considered "separated". And, these tasks along with other priority-unassigned tasks are considered as candidates for selecting the next target task at the current priority level. Similarly, if a target task is assigned priority level PL, the separated tasks along with other priority-unassigned tasks are considered as candidates for selecting the target tasks at next priority level. In other words, *the priority assignment for each new target task starts with all the priority-unassigned tasks, i.e., set Γ_U, and all the m processors*. It is not difficult to see that the time complexity of algorithm FPT is polynomial.

```
Algorithm FPT($\Gamma$ , $m$)
1.  $\Gamma_U = \Gamma$
2.  For PL $= 1$ to $(n-m)$
3.      For each $\tau_k \in \Gamma_U$      //a new task is selected as target task
4.          $\xi = \Gamma_U - \{\tau_k\}$
5.          For $m' = 0$ to $(m-1)$   //$m'$ tasks from $\xi$ will be separated
6.              H $= \xi -$ Select$(\xi, m', \tau_k)$
7.              If ($\lfloor \frac{I_k(H)}{m-m'} \rfloor + C_k \le D_k$) Then
8.                  Task $\tau_k$ is assigned priority level PL
9.                  $\Gamma_U = \Gamma_U - \{\tau_k\}$
10.                 If (PL$= n-m$) Then
11.                         //there are $m$ tasks left in $\Gamma_U$
12.                     Each task in $\Gamma_U$ is assigned one unique
13.                       priority level between $(n-m+1)$ to $n$
14.                     Return "Schedulable"
15.                 Else
16.                         Break and Go to next priority level (line 2)
17.                 End If
18.             End If
19.         End For
20.     End For
21.     Return "Unschedulable"
22. End For
```

Figure 3. The FPT test

Correctness of FPT: We now prove the correctness of the priority assignment policy of the FPT test in Theorem 1.

Theorem 1. *If algorithm FPT in Figure 3 returns "schedulable", then all the tasks in set Γ meet deadlines using global* FP *scheduling on m processors according to the priorities assigned by FPT.*

Proof: If algorithm FPT in Figure 3 returns "schedulable", then each of the tasks in Γ is assigned a unique priority level between 1 to n. We prove that each task that is assigned a priority level using algorithm FPT meets all the deadlines.

If a task τ_k is assigned any priority level PL between $(n-m+1)$ and n in line 12–13 of Figure 3, then task τ_k is one of the m highest-priority tasks. Since we have m processors, each task assigned any priority level between $(n-m+1)$ and n meets all its deadlines.

Now consider a task τ_k that is assigned priority level PL where PL $< (n-m+1)$. We show that task τ_k meets all the deadlines. Since PL $< (n-m+1)$, task τ_k is assigned priority in line 8 of the FPT algorithm in Figure 3. This implies that the condition in line 7 is true, and we have:

$$\left\lfloor \frac{I_k(H)}{m-m'} \right\rfloor + C_k \le D_k \tag{9}$$

where $H = \xi - \texttt{Select}(\xi, m', \tau_k)$ and the set ξ (determined in line 4) is the set of all tasks that are assigned higher priorities than that of task τ_k.

Since Eq. (9) holds, the maximum interference that any job of task τ_k suffers due to the higher priority tasks in H is $\lfloor \frac{I_k(H)}{m-m'} \rfloor$. Eq. (9) holds, if and only if, $I_k(H) \le [(m-m') \cdot (D_k - C_k + 1) - 1]$. Therefore, *the upper bound on the TIW due to the tasks in H within the scheduling window of any job of task τ_k is* $[(m-m') \cdot (D_k - C_k + 1) - 1]$.

Notice that after task τ_k is assigned priority level PL, its corresponding separated tasks are considered as target tasks at next priority level, and hence assigned higher priority levels. Thus, task τ_k suffers interference not only from the tasks in set H but also from the "separated" tasks returned by `Select`(ξ, m', τ_k). The upper bound on IW due to each of the tasks in `Select`(ξ, m', τ_k) is $(D_k - C_k + 1)$ according to Eq. (3) and Eq. (4). Thus, the TIW due to all the m' tasks in `Select`(ξ, m', τ_k) is at most $[m' \cdot (D_k - C_k + 1)]$. Consequently, the TIW due to all the higher priority tasks in $\xi = \mathsf{H} \cup$ `Select`(ξ, m', τ_k) on any job of task τ_k is:

$$[(m - m') \cdot (D_k - C_k + 1) - 1] + [m' \cdot (D_k - C_k + 1)]$$
$$= m \cdot (D_k - C_k) + (m - 1)$$

Because interference is an integer and all the m processors are simultaneously busy executing the tasks in ξ when task τ_k is interfered, the interference that any job of task τ_k suffers is at most $\lfloor \frac{m(D_k - C_k) + (m-1)}{m} \rfloor = (D_k - C_k)$. Consequently, any job of task τ_k meets its deadline. ■

Domination of FPT: Now we will prove in Theorem 2 that the FPT test dominates the state-of-the-art HPDALC test. Remember that HPDALC test assigns the highest fixed priority to the m' highest-density tasks and the remaining $(n-m')$ lowest-density tasks are assigned priorities based on the OPA and DA-LC test, for some m' where $0 \le m' < m$.

Theorem 2. *If taskset Γ is schedulable using the HPDALC test, then Γ is also schedulable using the FPT test, and not conversely.*

Proof: Assume a contradiction that taskset Γ does not pass the FPT test but passes the HPDALC test. Note that FPT test *can not* fail to assign priorities between priority levels $(n - m + 1)$ and n because the FPT algorithm in Figure 3 assigns these m highest priority levels in line 11–12 and returns "schedulable" in line 13.

Therefore, the FPT test can fail to assign priority only at some priority level between 1 and $(n-m)$. Let the FPT test *first* fails to assign priority at some priority level PL, where $1 \le \mathsf{PL} \le (n - m)$. Thus, when FPT test fails at priority level PL, there are total (PL-1) priority-assigned tasks and there are total $(n - \mathsf{PL} + 1)$ priority-unassigned tasks. Consequently, the *minimum* number of priority-unassigned tasks when FPT fails is $(m + 1)$ since $1 \le \mathsf{PL} \le (n - m)$. We denote F as the set of all priority-unassigned tasks when FPT fails and we have $|\mathsf{F}| \ge (m + 1)$.

Since Γ passes the HPDALC test, there are $(n-m')$ lowest-density tasks that are successfully assigned priorities using the combination of OPA and the DA-LC test for some m', $0 \le m' < m$. In other words, each of the $(n - m')$ lowest-density tasks passes the DA-LC test (because algorithm OPA in Figure 1 and the DA-LC test is applied). We denote P as the set of these $(n - m')$ lowest-density tasks and we have $|\mathsf{P}| \ge (n - m + 1)$ since $0 \le m' < m$.

Because $|\mathsf{F}| + |\mathsf{P}| \ge (m + 1) + (n - m + 1) = n + 2$ and $|\Gamma| = n$, there are at least two tasks that are common to both sets F and P. Let τ_x be a common task both in F and P such that no other such common task in P is assigned a priority *lower* than that of task τ_x in the HPDALC test. Therefore, each of the tasks in $(\mathsf{F} - \{\tau_x\})$ is assigned higher priority than that of task τ_x in the HPDALC test. In other words, $(\mathsf{F} - \{\tau_x\}) \subseteq \phi$ where ϕ is the set of tasks that are assigned higher priorities than task τ_x in the HPDALC test.

Since $\tau_x \in \mathsf{P}$, task τ_x passes DA-LC when assigning priority using the HPDALC test. Note that set ϕ includes the m' highest-density tasks that are separated and assigned the highest fixed-priority in HPDALC test. If task τ_x passes the DA-LC test, where m' highest-density tasks from set ϕ are separated, then according to Lemma 1, task τ_x must pass the DA-LC test by separating m' tasks using algorithm `Select(`ϕ, m', τ_x`)` from set ϕ. Consequently, task τ_x must pass the DA-LC test by separating m' or lower number of tasks from set $(F - \{\tau_x\})$ using the `Select` algorithm since $(F - \{\tau_x\}) \subseteq \phi$. Therefore, the FPT test that uses the `Select` algorithm for separation of the tasks can not fail to assign priority to task τ_x at priority level PL if Γ passes the HPDALC test. Therefore, any taskset that passes the HPDALC test also passes the FPT test.

The taskset in Example 1 passes the FPT test but not the HPDALC test. **Therefore, FPT test strictly dominates the state-of-the-art HPDALC test.** ■

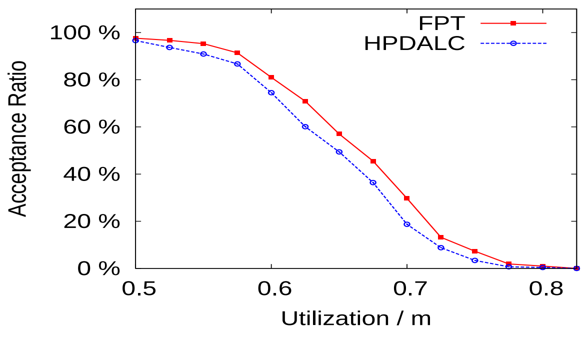
V. Experimental Results

As shown in Theorem 2, the FPT test dominates the state-of-the-art HPDALC test. The question is how much better the FPT test is in comparison to the HPDALC test. To answer this question, we conducted simulation experiments using randomly-generated task sets.

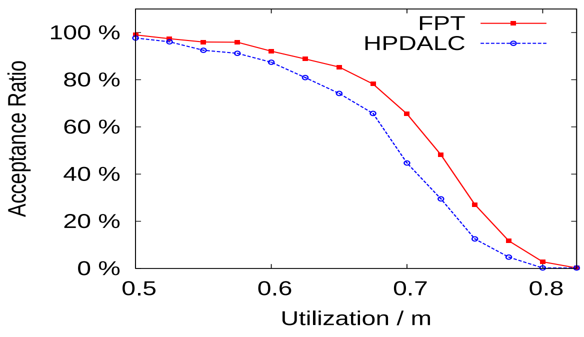
We use the well-known metric, called *acceptance ratio*, to compare the FPT test with the HPDALC test. The acceptance ratio of a schedulability test is the percentage of the randomly generated tasksets that pass the test at a given utilization level. One of the major findings of our experimental evaluation is that the *taskset size* (often not considered in many simulation studies) is one of the most important parameters to determine the schedulability of a taskset using global FP scheduling. Before we present our results, we present the taskset generation algorithm.

Taskset Generation Algorithm. To generate random tasksets, we used the `UUnifast-Discard` algorithm [13] proposed by Davis and Burns. The `UUnifast-Discard` algorithm with two parameters (n, U) generates n utilization values for n tasks with total utilization equal to U.

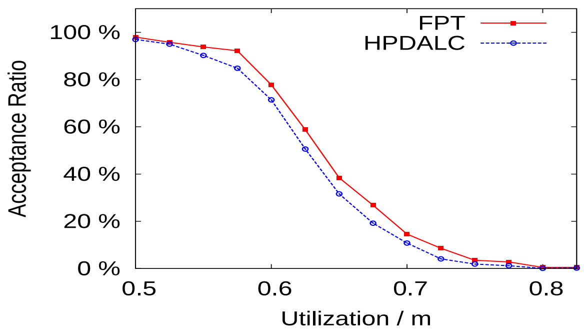
Once a set of n utilizations $\{u_1, u_2, \ldots u_n\}$ of a taskset is generated, the minimum inter-arrival time T_i of each task τ_i is generated from the uniform random distribution within the range $[3ms, 500ms]$. The inter-arrival time of the tasks in many practical real-time systems (e.g., robotics and control applications) often belong to this interval. For example, the periods of the tasks of some avionics application as used



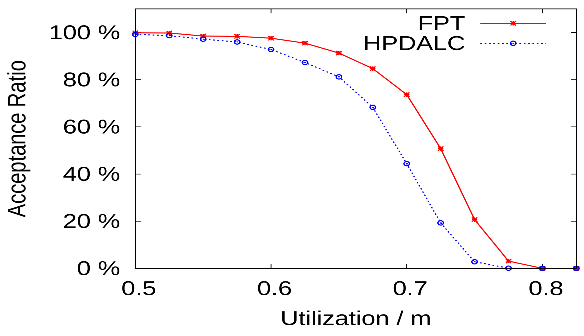
(a) $m = 4$, $n = 20$



(b) $m = 4$, $n = 40$



(c) $m = 6$, $n = 20$



(d) $m = 6$, $n = 80$

Figure 4. Acceptance ratio of the FPT and the HPDALC tests

by Vestal in his work in [21] ranges from $25ms$ to $200ms$. Finally, the WCET of task τ_i is set to $C_i = u_i \cdot T_i$ and the relative deadline D_i of task τ_i is generated from the uniform random distribution within the range $[C_i, T_i]$.

Given the applicability of global FP scheduling in a wide ranges of actual real-time systems, evaluating the performance of the FPT test using randomly generated task sets is reasonable as long as the effect of one parameter to generate the random task sets is not confounded with another parameter. It has been proved that the UUnifast-Discard algorithm generates an unbiased taskset in the sense that the utilizations of the n tasks of a task set are uniformly distributed (Theorem 12 in [13]). Another reason for us to use the UUnifast-Discard algorithm is to have the ability to control the parameter n when generating a taskset at some utilization level U.

Each of our experiments is characterized by a pair (m, n) where m is the number of processors and n is the task set size. We considered 40 different utilization levels $\{0.025m, 0.5m, \ldots 0.975m, m\}$ for each experiment (m, n). We generate 1000 task sets at each of these 40 utilization levels using the UUnifast-Discard algorithm with parameters n and U, where U is the utilization level. Each of the 1000 task sets generated at a particular utilization level, say U, has cardinality n and total utilization equal to U.

Result Analysis. We conducted 15 experiments with $m \in \{2, 4, 6\}$ and $n \in \{10, 20, 40, 60, 80\}$ to compare the acceptance ratios of FPT and HPDALC tests. We present the acceptance ratios of the experiments with parameters $(m = 4, n = 20)$, $(m = 4, n = 40)$, $(m = 6, n = 20)$ and $(m = 6, n = 80)$ in Figure 4(a)–4(d), respectively (the results of the other experiments follow similar trends).

Each graph in Figure 4(a)–4(d) presents the acceptance ratio for both tests. The x-axis is the system utilization U/m for utilization level U and the y-axis represents the acceptance ratio. The acceptance ratios of both HPDALC and FPT tests are around 100% at relatively low utilization level (e.g., $U \leq 0.5m$) and 0% at very high utilization level (e.g., $U > 0.8m$). We plot the acceptance ratio in Figure 4(a)–4(d) for the utilization levels between $0.5m$ and $0.8m$.

Observation 1. The acceptance ratios for *both* FPT test and HPDALC test are higher when the task set size increases for a given m. Notice that the acceptance ratio for both HPDALC and FPT tests in Figure 4(b) and Figure 4(d) are relatively “healthier” than that of in Figure 4(a) and Figure 4(c), respectively.

A taskset with smaller cardinality having total utilization U has relatively higher number of high-utilization tasks in comparison to that of a taskset with larger cardinality having total utilization U. With higher number of high-utilization tasks, the global FP scheduling suffers from so called “Dhall’s effect” [14], and consequently, relatively smaller number of (low cardinality) tasksets passes both HPDALC and FPT tests in Figure 4(a) and Figure 4(c).

Observation 2. The *improvement* in acceptance ratio of the FPT test over the HPDALC test is noticeable at higher utilization level (e.g., $0.55m \leq U \leq 0.75m$) in all the four cases in Figure 4(a)–4(d). Both priority assignment policy and schedulability test play very important roles in determining the global FP schedulability of a task set at *large* utilization levels. The improvement in acceptance ratio of the FPT test at higher utilization levels is due to our improved priority assignment policy. For example, the acceptance ratio of the FPT test is around 30% higher than that of HPDALC test at utilization level $0.7m$ in Figure 4(d).

Observation 3. The FPT test outperforms the HPDALC test for tasksets with relatively larger cardinality for a given number of processors. The *difference* between the acceptance ratios of the FPT test and HPDALC test in Figure 4(b) and Figure 4(d) is considerably larger than that of in Figure 4(a) and Figure 4(c), respectively.

The improvement of the FPT test over the HPDALC test increases when *both* task set cardinality and number of processors increases. The *difference* between the acceptance ratios of the FPT test and the HPDALC test at higher utilization level in Figure 4(d) is considerably larger than that of in Figure 4(a).

When the number of tasks in a task set at a particular utilization level is relatively larger, there are relatively fewer tasks with large density. And, separating tasks based on the "highest-density" criterion of the HPDALC test is not that effective. Our proposed (improved) separation criterion for FPT test reduces the pessimism of interference on a lower priority tasks to a larger extent in comparison to the criterion of separating "highest-density" tasks of the HPDALC test. Thus, with increasing number of task set size, the FPT test performs significantly better than the HPDALC test.

Summary. The task set size (i.e., parameter n) in addition to parameters m and U has significant impact on global FP scheduling. The FPT test outperforms the HPDALC test for large n (i.e., where there are fewer large-density tasks in a task set). However, the performance of the FPT test with relatively smaller n is also significant, for example, the acceptance ratio of the FPT test is around 5% to 8% higher than that of the HPDALC test between utilization level $0.55m$ and $0.7m$ in Figure 4(c). Any improvement in acceptance ratio of the FPT test implies relatively lower demand on total processing capacity which in turn could significantly cut the cost of mass production of actual systems with relatively fewer number of processors.

VI. Conclusion

In this paper, we have proposed a priority assignment algorithm for global FP scheduling. The priority assignment of the FPT test dominates similar tests proposed earlier and also performs well in simulation. The priority assignment algorithm uses an elegant criterion that reduces the pessimism of the analysis of global FP scheduling. Such reduced pessimism of the FPT test will require relatively fewer number of processors for resource-constrained hard real-time systems.

Acknowledgement

The authors would like to thank Professor Sanjoy Baruah for his valuable comments and suggestions.

References

[1] ARINC Incorporated. ARINC specification 651: Design guidance for integrated modular avionics, November 1997.

[2] AUTOSAR, Automotive Open System Architecture, www.autosar.org.

[3] WirelessHART Specification, www.hartcomm.org, 2007.

[4] B. Andersson. Global Static-Priority Preemptive Multiprocessor Scheduling with Utilization Bound 38%. In *Proc.of OPODIS*, pages 73–88, 2008.

[5] B. Andersson, S. Baruah, and J. Jonsson. Static-Priority Scheduling on Multiprocessors. In *Proc. of RTSS*, 2001.

[6] N. C. Audsley. On Priority Assignment in Fixed Priority Scheduling. *Info. Proc. Letters*, 79(1):39–44, 2001.

[7] T. P. Baker. An Analysis of Fixed-Priority Schedulability on a Multiprocessor. *Real-Time Systems*, 32(1-2):49–71, 2006.

[8] S. Baruah. Techniques for multiprocessor global schedulability analysis. In *Proc of RTSS*, pages 119–128, 2007.

[9] M. Bertogna and M. Cirinei. Response-Time Analysis for Globally Scheduled Symmetric Multiprocessor Platforms. In *Proc. of RTSS*, pages 149–160, 2007.

[10] M. Bertogna, M. Cirinei, and G. Lipari. Schedulability Analysis of Global Scheduling Algorithms on Multiprocessor Platforms. *IEEE Transactions on Parallel and Distributed Systems*, 20(4):553–566, 2009.

[11] B. Brandenburg, J. Calandrino, and J. Anderson. On the Scalability of Real-Time Scheduling Algorithms on Multicore Platforms: A Case Study. *Proc. of RTSS*, 2008.

[12] R. Davis and A. Burns. A Survey of Hard Real-Time Scheduling for Multiprocessor Systems. *Accepted for publication in the ACM Computing Surveys*, 2011. http://www-users.cs.york.ac.uk/~robdavis/papers/MPSurveyv5.0.pdf.

[13] R. Davis and A. Burns. Improved priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems. *Real-Time Systems*, 47:1–40, 2011.

[14] S. K. Dhall and C. L. Liu. On a Real-Time Scheduling Problem. *Operations Research*, 26(1):127–140, 1978.

[15] N. Guan, M. Stigge, W. Yi, and G. Yu. New Response Time Bounds for Fixed Priority Multiprocessor Scheduling. *Proc. of RTSS*, pages 387–397, 2009.

[16] M. Kamruzzaman, S. Swanson, and D. M. Tullsen. Inter-core prefetching for multicore processors using migrating helper threads. In *Proc. of ASPLOS*, pages 393–404, 2011.

[17] R. M. Pathan and J. Jonsson. Improved Schedulability Tests for Global Fixed-Priority Scheduling. *Proc. of ECRTS*, 2011.

[18] A. Saifullah, Y. Xu, C. Lu, and Y. Chen. End-to-End Delay Analysis for Fixed Priority Scheduling in WirelessHART Networks. In *Proc. of RTAS*, pages 13 –22, 2011.

[19] A. Saifullah, Y. Xu, C. Lu, and Y. Chen. Priority Assignment for Real-time Flows in WirelessHART Networks. In *Proc. of ECRTS*, pages 33–44, 2011.

[20] A. Sarkar, F. Mueller, and H. Ramaprasad. Predictable task migration for locked caches in multi-core systems. In *Proc. of LCTES*, pages 131–140, 2011.

[21] S. Vestal. Preemptive Scheduling of Multi-criticality Systems with Varying Degrees of Execution Time Assurance. In *Proc. of RTSS*, pages 239 –243, 2007.