SETTING PRIORITY POINTS FOR GLOBAL EDF-LIKE SCHEDULING OF SPORADIC TASKS WITH ARBITRARY DEADLINES ON MULTIPROCESSORS

by

Tian Liang, LL.B., M.A.

A thesis submitted to the Graduate Council of
Texas State University in partial fulfillment
of the requirements for the degree of
Master of Science
with a Major in Computer Science
May 2021

Committee Members:

Kecheng Yang, Chair

Xiao Chen

Byron Gao

ACKNOWLEDGEMENTS

I am indebted to many people who helped me have a successful study experience in my Computer Science Master program. First and Foremost is my advisor, Kecheng Yang. Kecheng has always encouraged me and put in a great deal of work himself helping me write and revise. I am also grateful to the remaining members of my committee: Xiao Chen, Byron Gao. In addition to learning from their feedback on my research, I also have learned much in their classes.

Besides this, I have always appreciated that Texas State University (TXST) had accepted me into the Computer Science Master program in 2019 Spring. As I have great interests but little professional experience in this field, it was TXST that gave me the opportunity to start this brand new journey and it was our department that laid me this solid foundation for further development.

I also owe special thanks to my family, especially my mother. It was not easy for a parent to understand why her child would change the arranged routine, but to pursue an uncertain way of the life. Luckily, my mother has always been there to support me through the process of earning a master degree in Computer Science.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

Abbreviation	Description
EDF	Earliest-Deadline-First
EPPF	Earliest-Priority-Point-First
FIFO	First-In-First-Out
GEL	G-EDF-like
G-EDF	Global EDF
G-EPPF	Gloabl EPPF
G-FL	Global Fair Lateness
HRT	Hard Real-Time
LP	Linear Program
PP	Priority Point
SRT	Soft Real-Time
WCET	Worst-Case Execution Time

ABSTRACT

Under the well-studied global earliest-deadline-first (G-EDF) scheduling, the deadline of each job serves for two functionalities simultaneously. First, it specifies the response-time requirement of the job; second, it determines the priority of the job. The concept of the family of G-EDF-like (GEL) schedulers extends G-EDF by separating these two functionalities: the deadline of a job only serves as the first one, and another parameter, called the priority point of a job, serves as the second one.

This thesis studies the problem of GEL scheduling arbitrary-deadline sporadic tasks on multiprocessor platforms where every processor is identical. In particular, the focus is on the priority point setting and schedulability analysis under both preemptive and non-preemptive global earliest-priority-point-first (G-EPPF) scheduling. This work shows that the response times under G-EPPF scheduling can be upper bounded by the relative priority points as well as specification parameters of the tasks and the platform. Thus, the problem of setting proper priority points can be formalized as a linear program (LP), which can be solved efficiently by readily available LP solvers. The schedulability experiments demonstrate the effectiveness of the proposed method and its merits over the state-of-art for the schedulability analysis of arbitrary-deadline tasks.

1. INTRODUCTION

To keep pace with fast development in multi-core chips of CPU architectures, real-time computer systems with new models, algorithms, and analysis techniques are increasingly becoming to be designed and implemented upon multiprocessors and multi-core platforms. In a real-time system, a scheduler is used to schedule tasks among processors which is concerned with the efficient allocation of computational resources and may be available in limited amounts, amongst competing demands in order to optimize specified objectives. Real-time scheduling theory deals with resource allocation in real-time computer systems, which are computer systems with certain computations that have a timing correctness requirement in addition to a functional one – it requires not only to "do the right thing," but also to "do it at the right time."

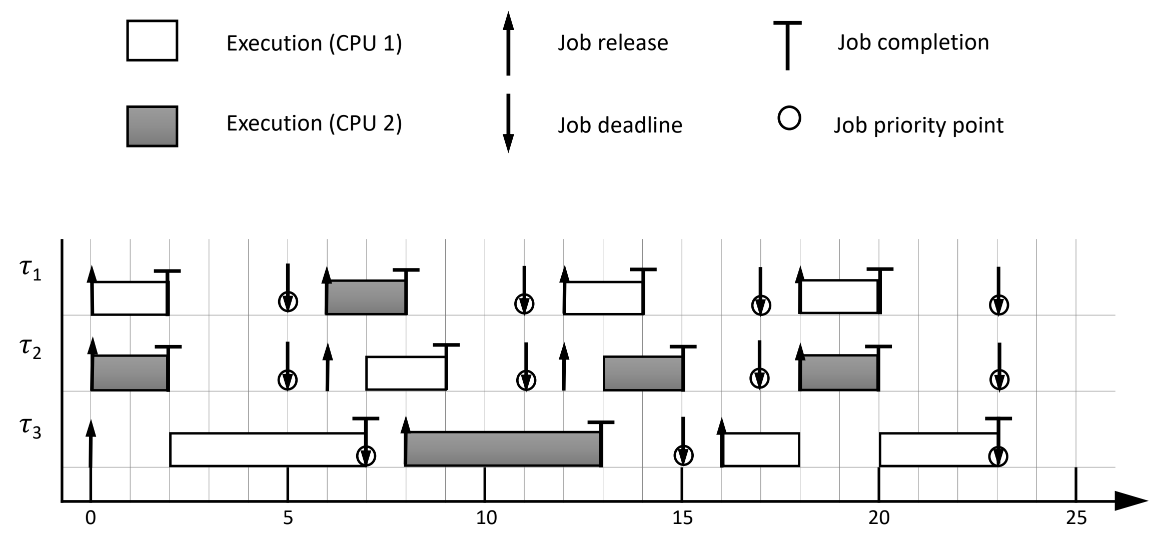
The global earliest-deadline-first (G-EDF) algorithm has been demonstrated to be a good candidate for scheduling real-time tasks on multiprocessor platforms. In fact, the well-studied G-EDF scheduler can be extended to be a more general class of schedulers, called G-EDF-like (GEL) schedulers, which also include another well-known scheduler, global first-in-first-out (FIFO). Significant research results have been obtained concerning the G-EDF scheduling of sporadic task systems. However, most of these results are focus on *constrained-deadline* sporadic task systems, which require the relative deadline of each task to be at most the task's period, whereas the same is not true for *arbitrary-deadline* sporadic task systems, where the relative deadline can be greater than the period. Our focus in this thesis is the GEL scheduling of arbitrary-deadline sporadic tasks on a homogeneous multiprocessor where every processor is identical.

For a GEL scheduler, the deadline works for two functionalities simultaneously: specify the response time of each job and determine the priority of each job. We separate these two roles individually and set the priority of each job by a particular value denote as *priority point (PP)*. Each task is associated with a relative PP, which defines the relative distance between the (absolute) PP and the release for any job of this task. Then a GEL scheduler can also be called a global earliest-priority-point-first (G-EPPF) scheduler, which prioritizes jobs according to their absolute PPs—the earlier the PP, the higher the priority. In particular, the G-EDF scheduler can be viewed as a special case of G-EPPF scheduler where the relative PP is assigned equal to the relative deadline for each task. Similarly, the FIFO scheduler can be viewed as another special case of G-EPPF scheduler where the relative PP is assigned equal to zero for all tasks.

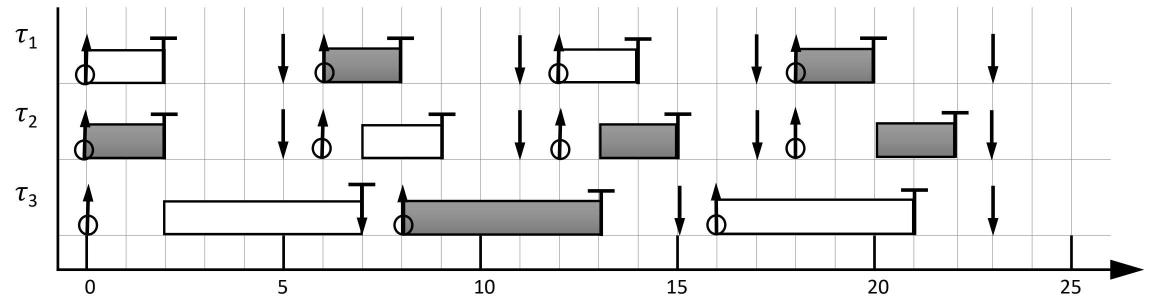
To illustrate, we will explain how PP drives the scheduler with sampled tasks in Figure 1. For each task $\tau_i = (C_i, T_i, D_i, Y_i)$, where C_i is the worst-case execution time (WCET) of task τ_i; T_i is the period, which denotes the minimum inter-arrival separation between any two successive jobs in task τ_i; D_i is the relative deadline of task (τ_i); Y_i is the relative PP of task τ_i.

In Figure 1(a), PP is set equal to the relative deadline of each task, so in G-EPPF scene, the sampled tasks are scheduled in the same way as G-EDF. Similarly, in Figure 1(b), the relative PP is set to be zero which will equal the absolute release time of each job when scheduling, so G-EPPF schedules the sampled tasks the same as FIFO. While in Figure 1(c), PP is different from period and deadline, and G-EPPF schedules the sampled tasks with the earliest PP. If any two jobs at the same priority point are ready to be scheduled, the system will select the job with a smaller task index.

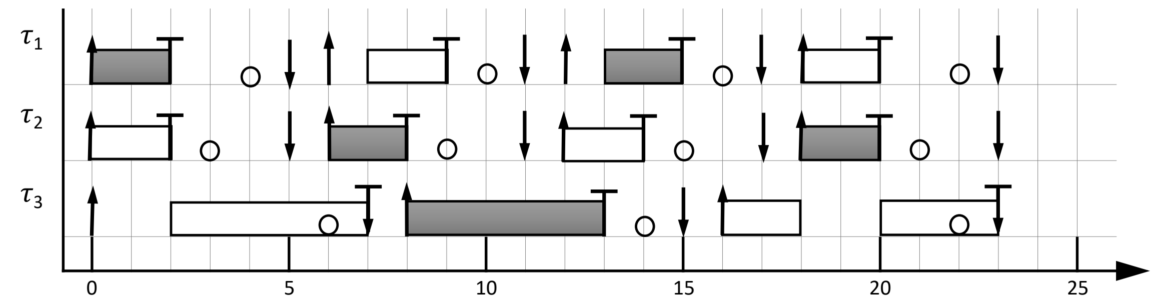
In particular, this thesis would focus on G-EPPF scheduling of sporadic tasks with arbitrary deadlines on identical multiprocessors. We will first prove basic and improved response-time bounds for each task as functions of PP with other system parameters. Then, formulate the schedulability problem as a linear program (LP), and solve it by Gurobi Optimizer. We will explore the G-EPPF for fully preemptive and non-preemptive cases in order to improve the schedulability for the systems.



(a) G-EDF schedule, where $\tau_1 = \tau_2 = (2, 6, 5, 5), \tau_3 = (5, 8, 7, 7)$



(b) FIFO schedule, where $\tau_1 = \tau_2 = (2, 6, 5, 0), \tau_3 = (5, 8, 7, 0)$



(c) G-EPPF schedule, where $\tau_1 = (2, 6, 5, 4), \tau_2 = (2, 6, 5, 3), \tau_3 = (5, 8, 7, 6)$

Figure 1: Comparison of G-EDF, FIFO, G-EPPF scheduling with sampled tasks by priority points.

1.1 Contributions

The main contribution of this thesis is exploring how the conception of priority point (PP) in G-EPPF scheduling may benefit the schedulability for arbitrary-deadline tasks in hard real-time (HRT) systems. We consider both fully preemptive and non-preemptive variants and derive PP-based response-time bounds for each task. Based on these response-time bounds, we form up a linear program (LP) that optimizes the PP setting to improve the schedulability. In the experiments, the Gurobi LP solver is applied to obtain the PPs and the schedulability results. Comparing to the techniques in [1], the experiments demonstrate the merits of the proposed method in this thesis, especially in the case where tasks' relative deadlines are greater than their periods, which is only allowed for arbitrary-deadline tasks. Observations from the experiment results also suggest that the response-time bound improvement techniques in this thesis are more effective for fully preemptive G-EPPF than non-preemptive G-EPPF.

1.2 Organization

In this chapter, we briefly introduce the main method of this thesis, and the rest is organized as following:

- Chapter 2 reviews related work and their contributions to our topic.
- Chapter 3 interprets our model with essential requirements and considered platforms.
- Chapter 4 presents proof-specific preliminaries and definitions for proving our response-time bounds.
- Chapter 5 provides the derivation of basic and improved response-time bounds under fully preemptive G-EPPF.
- Chapter 6 provides the derivation of basic and improved response-time bounds under non-preemptive G-EPPF.
- Chapter 7 implements an experimental evaluation to our models, compares and analyzes the scheduling performance.
- Chapter 8 presents the conclusions.

2. RELATED PRIOR WORK

There are two general approaches for mapping sporadic task sets on multiprocessor platforms. One is global scheduling, that is all generated tasks are scheduled from a single global queue and tasks may migrate among processors. In [2], Goossens and Devillers showed that G-EDF can schedule independent task sets successfully on m processors if the total utilization is at most $m(1 - u_{max}) + u_{max}$. Baruah and Baker in [3] derived a new sufficient G-EDF schedulability test to determine whether a given sporadic task system is guaranteed to be scheduled with all deadlines be met upon an identical multiprocessor platform fully preemptively. In contrast to global scheduling, *partitioned* scheduling is that for each task, a specific processor is assigned statically in advance, and each processor schedules its assigned tasks with a uniprocessor scheduling algorithm. Baruah and Fisher in [4] presented an algorithm for constrained-deadline and arbitrary sporadic tasks on identical multiprocessors platform. Due to the global EDF could make much more utilization of processors than partitioned EDF scheduling, we select G-EDF as our approach.

Devi and Anderson in [5] considered when total utilization of a task system without restriction but may equal the number of processors, the single tardiness for all tasks of G-EDF on multiprocessors could be bounded of all tasks in terms of plus a common added expression to task's WCET in soft real-time (SRT). Based on this, Erickson et al. provided a tighter bound on maximum tardiness through deriving separate tardiness bounds in [6]. In 2007, Leontyev and Anderson [7] derived tardiness bounds for global FIFO scheduling. Later on, they presented a generalized tardiness bound in [8] to a wide variety of global scheduling algorithms like EDF, FIFO, RM, LLF, EDZL, and PD by giving each job a *priority* for SRT scheduling. The scheduler always picked the job with the highest priority to execute. They defined the absolute deadline of a job as its priority for G-EDF and set the prioritization function associated with each released job to specified their priority. For instance, under the preemptive EDF algorithm they defined the priority equal to the absolute deadline, while in least-laxity-first (LLF) algorithm, they associated priority to a time-dependent equation with parameters consist of the deadline, utilization, and a modified execution time. This work provided a general applied bound rather than the tightest. Furthermore in 2011, Leontyev et al. [9] defined the priority of each job as some per-task constant after the job released which is the class of GEL schedulers. Erickson et al. in [10] proposed *global fair lateness(G-FL)* scheduling, where compliant-vector analysis(CVA) is applied to demonstrate that G-FL minimized the maximum lateness bound over all GEL schedulers. To improve the tardiness bounds, they derived response-time bounds via lateness, defined as the difference between deadline and completion time, for arbitrary GEL schedulers since jobs of tasks can be guaranteed to complete by point before their deadlines.

The feasibility condition on uniform multiprocessors was first derived by Funk et al. [11]. In [12, 13, 14], they also investigated the EDF scheduling of the periodic task systems on uniform multiprocessors. Yang and Anderson in [15, 16] extended the tardiness bounds for G-EDF to uniform multiprocessors. They also studied EDF-based semi-partitioned scheduling on uniform multiprocessors in [17, 18]. In an npc-sporadic task system, jobs are not subject to intra-task precedence constraints and such system was first studied by Erickson and Anderson [19]. Yang and Anderson in [20] showed that an optimal G-EDF-based scheduler of npc-sporadic task system on a uniform multiprocessor is possible. They proved that both preemptive and non-preemptive G-EDF algorithms guarantee to bound the response time for any feasible npc-sporadic task system, and they also derived the feasibility conditions. Elliott et al. [21] and Yang et al. [22, 23] later adapted these techniques to the scheduling of directed-acyclic-graph (DAG) based systems.

Other prior work such as [3, 24, 25] had derived important results of G-EDF schedulability analysis for *constrained-deadline* sporadic task systems. While Baruah and Baker [1] and Andersson et al. [26] presented analysis techniques for *arbitrary-deadline* sporadic task system with close schedulability results to the constrained-deadline work. In this thesis, we will explore arbitrary-deadline task sets of HRT sporadic task systems.

To the experimental approaches and analysis techniques, Bini and Buttazzo in 2005 [27] created the UUniFast algorithm to efficiently generate a fixed utilization value with a number of task sets on a uniprocessor. The utilization distribution of UUniFast algorithm is equivalent to uniformly sampling the task utilization value, and then keeping those task sets with the selected total utilization. In 2009, Davis and Burns [28] extended UUniFast method to UUniFast-Discard by simply discard task sets containing invalid values until the task amount satisfied the requirement on multiprocessor platforms.

Baker first designed a practical test for G-EDF schedulability analysis of sporadic task systems in [29, 30] by assuming a task's job missed its deadline, then derived if there exist necessary conditions that all tasks must be satisfied in order to guarantee this deadline miss occur. Negatived these conditions to yield a sufficient schedulability test. Bertogna and Cirinei in [25] developed Baker's analysis technique from uniprocessor scheduling domain to multiprocessor by iteratively computing improved bounds of response time on WCET of jobs. Baruah and Baker in [1] contributed an analysis technique with the load of a task set to evaluate arbitrary sporadic task systems upon preemptive multiprocessor platforms. In our thesis, we will compare the results of our models with Density test and Load test.

3. SYSTEM MODEL

We consider an identical multiprocessor platform with m processors. For each processor $i (1 \leq i \leq m)$, the maximum amount of work that can be done within one time unit of each processor is identical, in the sense that each processor has the same computing capability as every other processor. We normalize the speed of each processor to be 1.0. To our identical multiprocessors model, the accumulative speed of amount i processors is i.

In our HRT sporadic task system, jobs have deadlines associated with them and it is required all deadlines be met. We consider a task set τ of n sporadic tasks, $\tau = \{\tau_1, \tau_2, \tau_3, ..., \tau_n\}$. The jobs are those repeatedly invocations within a task during the continuous-time. τ can be referred to all jobs generated by this system. In general sporadic task model, each task τ_i is characterized by a 3-tuple parameters $\tau_i = (C_i, T_i, D_i)$, where C_i is the worst-case execution time (WCET) of task τ_i; T_i is the minimum inter-arrival separation between any two successive jobs of task τ_i; D_i is the relative deadline of task τ_i where ($D_i \geq 0$). We let $n \geq m$. If this is not the case, each task is able to be assigned to a separate processor and consequently no job of each τ_i is going to have response time exceeding C_i.

The j^{th} job $(j \geq 1)$ of task τ_i is denoted as $\tau_{i,j}$. We denote its release time as $r_{i,j}$, its absolute deadline as $d_{i,j}$, and its response time as $R_{i,j}$, where $d_{i,j} = r_{i,j} + D_i$. If a job finishes its execution at time $f_{i,j}$, then $R_{i,j} = f_{i,j} - r_{i,j}$. We denote the lateness of $\tau_{i,j}$ as $\Delta_{i,j}$, and $\Delta_{i,j} = f_{i,j} - d_{i,j}$. For task τ_i, we use R_i to refer to the response-time bound and Δ_i refer to the lateness bound. The utilization of a task τ_i is the ratio of its WCET to its period, denoted as $u_i = C_i/T_i$, and the total utilization of entire task system τ is $U_{sum} = \sum_{i=1}^{n} u_i$.

We focus on global scheduling. That is, all released jobs are pushed into a single global queue, then the scheduler will select the first m ready jobs with the highest priorities to execute if at least m jobs are ready, and will schedule all ready jobs if less than m jobs are ready. The priority of each job $\tau_{i,j}$ is determined by its PP, denoted by

$y_{i,j}$, and the earlier the priority point the higher the priority, *i.e.*, job $\tau_{i,j}$ has a higher priority than job $\tau_{k,\ell}$ if $y_{i,j} < y_{k,\ell}$. Each task τ_i is given a *relative priority point*, denoted by Y_i, such that the PP of an arbitrary job $\tau_{i,j}$ of this task is calculated by $y_{i,j} = r_{i,j} + Y_i$.

In this thesis, we assume that jobs of the same task may run simultaneously on different processors if such processors are available. In other words, if a job has not completed its execution and the following job of the same task is released (this is possible, due to arbitrary deadlines), the later job can start its execution without waiting for the earlier one to complete if sufficient processors are available. Therefore, in this thesis, for a system to be feasible, it only requires that

$$U_{sum} \leq m. \tag{3.1}$$

4. PRELIMINARIES

Definition 1. *For any time instant $t > 0$, a job $\tau_{i,j}$ is unreleased if $r_{i,j} > t$, pending if $r_{i,j} \leq t$ and $f_{i,j} < t$, and completed if $f_{i,j} \geq t$.*

Definition 2. *We denote $\mathsf{A}(\mathcal{S}, \tau_{i,j}, t_1, t_2)$ to the accumulative processor capacity allocation to job $\tau_{i,j}$ in an arbitrary schedule $\mathcal{S}$ within the time interval $[t_1, t_2]$. Similarly in task level, denote the arbitrary schedule be $\mathsf{A}(\mathcal{S}, \mathcal{J}, t_1, t_2)$, the $\mathcal{J}$ represents all jobs in a task set.*

$$\mathsf{A}(\mathcal{S}, \mathcal{J}, t_1, t_2) = \sum_{\tau_{i,j} \in \mathcal{J}} \mathsf{A}(\mathcal{S}, \tau_{i,j}, t_1, t_2). \tag{4.1}$$

Ideal schedule. Define $\pi_{\text{IDEAL}} = \{u_1, u_2, ..., u_n\}$ as an ideal multiprocessor for the task set τ with n processors each of which matches the utilization of a task in τ. Let $\mathcal{I}$ be the partitioned schedule for τ on π_{IDEAL} such that each task τ_i execute on a dedicated processor of speed u_i. Then, each job $\tau_{i,j}$ must begin to execute at its release time $r_{i,j}$ and complete its execution within one period T_i. Therefore, we get
$\mathsf{A}(\mathcal{S}, \mathcal{I}, t_1, t_2) \leq u_i \cdot (t_2 - t_1)$ and for an arbitrary job set $\mathcal{J} \subseteq \tau$,

$$\mathsf{A}(\mathcal{I}, \mathcal{J}, t_1, t_2) \leq U_{sum} \cdot (t_2 - t_1). \tag{4.2}$$

The above notion guarantees that a single job will not be paralleled on multiple processors.

Furthermore, we can upper bound the amount of unfinished work for any job of task τ_i at its priority point by L_i, which is defined as follows.

Definition 3. *Let L_i denote the difference between period T_i and priority point Y_i, then we have $L_i = u_i \cdot \max\{0, (T_i - Y_i)\}$, and $L_{sum} = \sum_{i=1}^{n} L_i$. Thus in $\mathcal{I}$, for any time instant t, the amount of incomplete work with priority point at or before t is at most L_{sum}.*

Moreover, based on the function of A, we define two useful functions lag and LAG as follows.

Definition 4. *Let the difference between the total allocation to a job $\tau_{i,j}$ in $\mathcal{I}$ and an actual schedule $\mathcal{S}$ up to time t as*

$$\mathsf{lag}(\tau_{i,j}, t, \mathcal{S}) = \mathsf{A}(\mathcal{I}, \tau_{i,j}, 0, t) - \mathsf{A}(\mathcal{S}, \tau_{i,j}, 0, t), \tag{4.3}$$

and to an arbitrary job set $\mathcal{J}$, the total allocation difference is

$$\mathsf{LAG}(\mathcal{J}, t, \mathcal{S}) = \sum_{\tau_{i,j} \in \mathcal{J}} \mathsf{lag}(\tau_{i,j}, t, \mathcal{S}). \tag{4.4}$$

Based on (4.3) *and Definition 2, given any time interval $[t_1, t_2]$, there is*

$$\mathsf{lag}(\tau_{i,j}, t_2, \mathcal{S}) = \mathsf{lag}(\tau_{i,j}, t_1, \mathcal{S}) + \mathsf{A}(\mathcal{I}, \mathcal{J}, t_1, t_2) - \mathsf{A}(\mathcal{S}, \mathcal{J}, t_1, t_2), \tag{4.5}$$

and by (4.1), (4.3) *and* (4.4)*, there is,*

$$\mathsf{LAG}(\mathcal{J}, t_2, \mathcal{S}) = \mathsf{LAG}(\mathcal{J}, t_1, \mathcal{S}) + \mathsf{A}(\mathcal{I}, \mathcal{J}, t_1, t_2) - \mathsf{A}(\mathcal{S}, \mathcal{J}, t_1, t_2). \tag{4.6}$$

Lemma 1. *If a job $\tau_{i,j}$ is unreleased or complete at time t, then $\mathsf{lag}(\tau_{i,j}, t, \mathcal{S}) \le 0$; if $\tau_{i,j}$ is pending at t, then $0 < \mathsf{lag}(\tau_{i,j}, t, \mathcal{S}) \le C_i$.*

Proof. Follows from Definition 1 and 4. □

Job of interest. To derive response-time bounds, we consider an arbitrary job $\tau_{k,\ell}$ in τ, and upper bound its response time. Let t_y be the PP of $\tau_{k,\ell}$, *i.e.*, $t_y = y_{k,\ell}$.

Definition 5. *To our HRT system, denote Ψ to the job set of all jobs with priority points at or before t_y. We call jobs in Ψ as competing jobs for $\tau_{k,\ell}$.*

Definition 6. *For any time instant t, if all m processors are executing jobs belong to Ψ, then t is a busy instant for Ψ; else t is a non-busy instant for Ψ. For any time interval $[t_1, t_2]$, if every time instant is a busy instant for Ψ, then $[t_1, t_2]$ is a busy interval for Ψ; if not, non-busy interval.*

Lemma 2. *If $[t_1, t_2]$ is a busy interval for Ψ in $\mathcal{S}$, then $\mathsf{LAG}(\Psi, t_1, \mathcal{S}) \geq \mathsf{LAG}(\Psi, t_2, \mathcal{S})$.*

Proof. By (4.6), $\mathsf{LAG}(\Psi, t_2, \mathcal{S}) = \mathsf{LAG}(\Psi, t_1, \mathcal{S}) + \mathsf{A}(\mathcal{I}, \mathcal{J}, t_1, t_2) - \mathsf{A}(\mathcal{S}, \mathcal{J}, t_1, t_2)$. Because $[t_1, t_2]$ is a busy interval for Ψ in $\mathcal{S}$, we have $\mathsf{A}(\mathcal{S}, \mathcal{J}, t_1, t_2) = m \cdot (t_2 - t_1)$. Then, by (3.1) and (4.2), we have

$$\mathsf{A}(\mathcal{I}, \mathcal{J}, t_1, t_2) - \mathsf{A}(\mathcal{S}, \mathcal{J}, t_1, t_2) \leq U_{sum} \cdot (t_2 - t_1) - m \cdot (t_2 - t_1) \leq 0$$

Therefore, $\mathsf{LAG}(\Psi, t_2, \mathcal{S}) \leq \mathsf{LAG}(\Psi, t_1, \mathcal{S})$, and the lemma follows. □

Definition 7. *For any time instant t, denote t^+ to the time instant $(t + \epsilon)$ and t^- to the time instant $(t - \epsilon)$, where $\epsilon \to 0^+$.*

Definition 8. *The competing work for a job $\tau_{i,j}$ at time t is defined by the summation of the remaining work at time t of all jobs (pending or to be released) with a priority higher than or equal to $\tau_{i,j}$ (including the work of $\tau_{i,j}$ itself). That is, the competing work for our job of interest, $\tau_{k,\ell}$, at time t is simply the remaining work of jobs in Ψ at time t (including those to be released after time t).*

5. RESPONSE-TIME BOUNDS UNDER FULLY PREEMPTIVE G-EPPF

In this chapter, we study the *fully preemptive* global earliest-priority-point-first (G-EPPF) scheduler, under which the priority of a job is determined by its priority point (PP). In addition, any preemption and migration are allowed to enable that: at any time instant, if there are at most m pending jobs, then all of them are scheduled; if there are more than m pending jobs, then the m such jobs with the earliest PP are scheduled.

We aim at deriving response-time bounds for *all* jobs in the system under G-EPPF scheduling. In particular, we focus on an arbitrary job $\tau_{k,\ell}$ and derive a response-time bound for it. Because job $\tau_{k,\ell}$ is an arbitrary one in the system, the derived bound is applicable to any job. Moreover, in this chapter, we use $\mathcal{S}$ to denote the fully preemptive G-EPPF schedule specifically.

5.1 A Basic Response-Time Bound

We first derive a response-time bound here in Section 5.1. This bound is based on a classic real-time schedulability analysis technique that studies the *busy* time intervals. Later in Section 5.2, we will present techniques to improve such bound.

Lemma 3. *For any non-busy instant t which is at or before t_y,*
$\mathsf{LAG}(\Psi, t, \mathcal{S}) \leq (m-1) \cdot C_{max}$.

Proof. We consider three subsets of Ψ: Ψ_1 is the set of all jobs in Ψ that are *unreleased* at time t, Ψ_2 is the set of all jobs in Ψ that are *pending* at time t, and Ψ_3 is the set of all jobs in Ψ that are *complete* at time t. It is clear that Ψ_1, Ψ_2, and Ψ_3 are disjoint and

$\Psi_1 \cup \Psi_2 \cup \Psi_3 = \Psi$. Therefore,

$$\begin{aligned}
\mathsf{LAG}(\Psi, t, \mathcal{S}) &= \mathsf{LAG}(\Psi_1, t, \mathcal{S}) + \mathsf{LAG}(\Psi_2, t, \mathcal{S}) + \mathsf{LAG}(\Psi_3, t, \mathcal{S}) \\
&= \sum_{\tau_{i,j} \in \Psi_1} \mathsf{lag}(\tau_{i,j}, t, \mathcal{S}) + \sum_{\tau_{i,j} \in \Psi_2} \mathsf{lag}(\tau_{i,j}, t, \mathcal{S}) + \sum_{\tau_{i,j} \in \Psi_3} \mathsf{lag}(\tau_{i,j}, t, \mathcal{S}) \\
&\leq \sum_{\tau_{i,j} \in \Psi_1} 0 + \sum_{\tau_{i,j} \in \Psi_2} C_i + \sum_{\tau_{i,j} \in \Psi_3} 0 \\
&\leq |\Psi_2| \cdot C_{max} \\
&\leq (m-1) \cdot C_{max}.
\end{aligned}$$

Note that, the first "$\leq$" is by Lemma 1.The last "$\leq$" is because under the fully preemptive G-EPPF scheduling, there can be at most $(m-1)$ pending jobs at a *non-busy* time instant t; otherwise, m jobs would be scheduled and time t and it would be a busy time instant. □

Lemma 4. *At or after time t_y, once job $\tau_{k,\ell}$ starts executing, it must continuously execute until it completes.*

Proof. Because the relative priority point $Y_i \geq 0, \forall i$, no job with PP earlier than t_y can be released at or after time t_y, *i.e.*, no job that is released at or after t_y has a higher priority than $\tau_{k,\ell}$ and can preempt $\tau_{k,\ell}$. Therefore, once $\tau_{k,\ell}$ executes at or after t_y, its execution will be continuous until its completion. □

Lemma 5. *In $\mathcal{S}$, the amount of competing work for $\tau_{k,\ell}$ at t_y is at most $L_{sum} + (m-1) \cdot C_{max}$.*

Proof. According to Definition 3 and 4, the competing work in $\mathcal{S}$ pending at t_y is at most $L_{sum} + \mathsf{LAG}(\Psi, t, \mathcal{S})$. Let t' be the latest non-busy instant at or before t_y; if no such time instant exists, we let $t' = 0$. Then, by Lemma 2, we have $\mathsf{LAG}(\Psi, t', \mathcal{S}) \geq \mathsf{LAG}(\Psi, t_y, \mathcal{S})$, and by Lemma 3 we have $\mathsf{LAG}(\Psi, t', \mathcal{S}) \leq (m-1) \cdot C_{max}$. Based on these, we can derive that $\mathsf{LAG}(\Psi, t_y, \mathcal{S}) \leq (m-1) \cdot C_{max}$, and the lemma follows. □

Lemma 6. *We denote the competing work for $\tau_{k,\ell}$ at t_y by W. Then, the job $\tau_{k,\ell}$ will complete execution no later than time*

$$t_y + \frac{W - C_k}{m} + C_k. \tag{5.1}$$

Proof. If $\tau_{k,\ell}$ is complete at or before t_y, then the lemma trivially holds. Therefore, in the rest of this proof, we focus on the case where $\tau_{k,\ell}$ is not complete at or before t_y. Denote δ as the amount of work in $\tau_{k,\ell}$ that has been completed by t_y, and denote $e_{k,\ell}$ as the real execution requirement of $\tau_{k,\ell}$. Then the remaining execution work of $\tau_{k,\ell}$ at t_y is $e_{k,\ell} - \delta$. If $\tau_{k,\ell}$ does not execute within time period $[t_y, t_y + \frac{W-(e_{k,\ell}-\delta)}{m})$, we can conclude this time period must be a busy interval for Ψ. In this case, the competing work that is completed within this busy interval will be $W - (e_{k,\ell} - \delta)$, cause within a busy interval all processors executing competing work with a total speed m, and the remaining competing work at $t_y + \frac{W-(e_{k,\ell}-\delta)}{m}$ is $e_{k,\ell} - \delta$, which must be totally belong to $\tau_{k,\ell}$. As a result, $\tau_{k,\ell}$ will execute at time $t_y + \frac{W-(e_{k,\ell}-\delta)}{m}$. Conversely, if $\tau_{k,\ell}$ does not complete at or before t_y, then the latest time $\tau_{k,\ell}$ starts execution after t_y will be $t_y + \frac{W-(e_{k,\ell}-\delta)}{m}$. Recall Lemma 4, $\tau_{k,\ell}$ will not be preempted once it executes after t_y, and the execution speed is 1.0, $\tau_{k,\ell}$ will complete within(at) $e_{k,\ell} - \delta$ time units. With all above, we can conclude $\tau_{k,\ell}$ will complete by

$$\begin{aligned}
&t_y + \frac{W - (e_{k,\ell} - \delta)}{m} + (e_{k,\ell} - \delta) \\
=&\{\text{rearranging}\} \\
&t_y + \frac{W}{m} - \frac{e_{k,\ell}}{m} + e_{k,\ell} + (\frac{\delta}{m} - \delta) \\
\leq&\{\text{because } \delta \geq 0 \text{ and } m \geq 1, (\frac{\delta}{m} - \delta) \leq 0\} \\
&t_y + \frac{W}{m} - \frac{e_{k,\ell}}{m} + e_{k,\ell} \\
\leq&\{\text{because } e_{k,\ell} \leq C_k \text{ and } m \geq 1\} \\
&t_y + \frac{W}{m} - \frac{C_k}{m} + C_k \\
=&t_y + \frac{W - C_k}{m} + C_k,
\end{aligned}$$

and the lemma follows. □

Theorem 1. *Under fully preemptive* G-EPPF *scheduling on m identical unit-speed processors, the response time of an arbitrary job $\tau_{k,\ell}$ in τ is at most*

$$Y_k + \frac{1}{m} \cdot L_{sum} + \frac{(m-1)}{m} \cdot C_{max} + \frac{(m-1)}{m} \cdot C_k.$$

Proof. The theorem follows directly from Lemma 5 and Lemma 6. □

5.2 An Improved Response-Time Bound

In this section, we improve the response-time bound for fully preemptive G-EPPF in Theorem 1 mainly by the following two observations:

- We have focused on busy intervals. However, in fact, intervals that the system LAG does not increase would be sufficient.

- We have investigated the execution job $\tau_{k,\ell}$ of our interest after its priority point t_y. Such investigation could be extended to as early as $\tau_{k,\ell}$'s release time $r_{k,\ell}$.

To begin with, we introduce a new integer Λ, which is defined as:

$$\Lambda = \lceil U_{sum} \rceil . \tag{5.2}$$

Lemma 7. *At any time instant t where at most* $(\Lambda - 1)$ *processors are busy, it must hold that* $\mathsf{LAG}(\Psi, t, \mathcal{S}) \leq (\Lambda - 1) \cdot C_{max}$.

Proof. Similar to Lemma 3, we consider three subsets of Ψ: Ψ_1, Ψ_2, and Ψ_3 — the sets of all jobs in Ψ that are *unreleased*, *pending*, and *complete*, respectively, at time t. Therefore,

$$\begin{aligned}\mathsf{LAG}(\Psi, t, \mathcal{S}) &= \mathsf{LAG}(\Psi_1, t, \mathcal{S}) + \mathsf{LAG}(\Psi_2, t, \mathcal{S}) + \mathsf{LAG}(\Psi_3, t, \mathcal{S}) \\ &= \sum_{\tau_{i,j} \in \Psi_1} \mathsf{lag}(\tau_{i,j}, t, \mathcal{S}) + \sum_{\tau_{i,j} \in \Psi_2} \mathsf{lag}(\tau_{i,j}, t, \mathcal{S}) + \sum_{\tau_{i,j} \in \Psi_3} \mathsf{lag}(\tau_{i,j}, t, \mathcal{S}) \\ &\leq \sum_{\tau_{i,j} \in \Psi_1} 0 + \sum_{\tau_{i,j} \in \Psi_2} C_i + \sum_{\tau_{i,j} \in \Psi_3} 0 \\ &\leq |\Psi_2| \cdot C_{max} \\ &\leq (\Lambda - 1) \cdot C_{max}.\end{aligned}$$

The last "$\leq$" is because only $(\Lambda - 1) = (\lceil U_{sum} \rceil - 1) < U_{sum} \leq m$ processor being busy implies that all pending jobs are scheduled, by the fully preemptive G-EPPF scheduler. Thus, the lemma follows. □

Lemma 8. *For any time instant t,* $\mathsf{LAG}(\Psi, t, \mathcal{S}) \leq (\Lambda - 1) \cdot C_{max}$.

Proof. For any time instant t, we let t_0 denote the latest time instant at or before time t such that at most $(\Lambda - 1)$ processors are busy. Then, it directly implies that at least Λ processors are busy at any time instant in $(t_0, t]$, and this means

$$\mathsf{A}(\mathcal{S}, \Psi, t_0, t) \geq \Lambda \cdot (t - t_0). \tag{5.3}$$

Also, by Lemma 7, $\mathsf{LAG}(\Psi, t_0, \mathcal{S}) \leq (\Lambda - 1) \cdot C_{max}$. Therefore,

$$\begin{aligned}
\mathsf{LAG}(\Psi, t, \mathcal{S}) =& \{\text{by (4.6)}\} \\
& \mathsf{LAG}(\Psi, t_0, \mathcal{S}) + \mathsf{A}(\mathcal{I}, \Psi, t_0, t) - \mathsf{A}(\mathcal{S}, \Psi, t_0, t) \\
\leq & (\Lambda - 1) \cdot C_{max} + \mathsf{A}(\mathcal{I}, \Psi, t_0, t) - \mathsf{A}(\mathcal{S}, \Psi, t_0, t) \\
\leq & \{\text{by (4.2)}\} \\
& (\Lambda - 1) \cdot C_{max} + U_{sum} \cdot (t - t_0) - \mathsf{A}(\mathcal{S}, \Psi, t_0, t) \\
\leq & \{\text{by (5.3)}\} \\
& (\Lambda - 1) \cdot C_{max} + U_{sum} \cdot (t - t_0) - \Lambda \cdot (t - t_0) \\
\leq & \{\text{because } \Lambda = \lceil U_{sum} \rceil \geq U_{sum}\} \\
& (\Lambda - 1) \cdot C_{max},
\end{aligned}$$

and the lemma follows. □

Lemma 9. *Suppose the competing work for $\tau_{k,\ell}$ at $r_{k,\ell}$ is W. Then the response time of $\tau_{k,\ell}$ is upper bounded by*

$$\frac{W - C_k}{m} + C_k.$$

Proof. Recall that we denote the release time and finish time of $\tau_{k,\ell}$ by $r_{k,\ell}$ and $f_{k,\ell}$, respectively. Because the fully preemptive G-EPPF scheduler always actively schedule pending jobs on available processors for execution, any time instant within $[r_{k,\ell}, f_{k,\ell}]$ must be either busy on competing workload or executing $\tau_{k,\ell}$. Therefore, competing workload other than $\tau_{k,\ell}$ itself could prevent $\tau_{k,\ell}$ from being executed for at most $\frac{W-C_k}{m}$ time units and $\tau_{k,\ell}$ could execute for at most C_k time units. Thus, $\tau_{k,\ell}$ must be complete within $(\frac{W-C_k}{m} + C_k)$ time units after $r_{k,\ell}$. The lemma follows. □

Lemma 10. *The competing work for $\tau_{k,\ell}$ at $r_{k,\ell}$ is at most*

$$U_{sum} \cdot Y_k + L_{sum} + (\Lambda - 1) \cdot C_{max}.$$

Proof. By the definition of the idea schedule and by Definition 3, the total workload in the system with a priority higher than or equal to $\tau_{k,\ell}$ is at most $L_{sum} + \mathsf{A}(\mathcal{I}, \Psi, 0, t_y)$. Meanwhile, by Definition 2 and the definition of Ψ, $\mathsf{A}(\mathcal{S}, \Psi, 0, r_{k,\ell})$ denotes the amount of such workload that has been done by time $r_{k,\ell}$ in the fully preemptive G-EPPF schedule $\mathcal{S}$. Therefore, the competing work for $\tau_{k,\ell}$ at $r_{k,\ell}$ is at most

$$\begin{aligned}
& L_{sum} + \mathsf{A}(\mathcal{I}, \Psi, 0, t_y) - \mathsf{A}(\mathcal{S}, \Psi, 0, r_{k,\ell}) \\
=& L_{sum} + \mathsf{A}(\mathcal{I}, \Psi, 0, r_{k,\ell}) + \mathsf{A}(\mathcal{I}, \Psi, r_{k,\ell}, t_y) - \mathsf{A}(\mathcal{S}, \Psi, 0, r_{k,\ell}) \\
=& \mathsf{A}(\mathcal{I}, \Psi, r_{k,\ell}, t_y) + L_{sum} + (\mathsf{A}(\mathcal{I}, \Psi, 0, r_{k,\ell}) - \mathsf{A}(\mathcal{S}, \Psi, 0, r_{k,\ell})) \\
\leq& \{\text{by (4.2) and by Definition 4}\} \\
& U_{sum} \cdot (t_y - r_{k,\ell}) + L_{sum} + \mathsf{LAG}(\Psi, r_{k,\ell}, \mathcal{S}) \\
=& \{\text{by its definition, } t_y = r_{k,\ell} + Y_k\} \\
& U_{sum} \cdot Y_k + L_{sum} + \mathsf{LAG}(\Psi, r_{k,\ell}, \mathcal{S}) \\
\leq& \{\text{by Lemma 8}\} \\
& U_{sum} \cdot Y_k + L_{sum} + (\Lambda - 1) \cdot C_{max}.
\end{aligned}$$

Thus, the lemma follows. □

Theorem 2. *Under fully preemptive G-EPPF scheduling on m identical unit-speed processors, the response time of an arbitrary job $\tau_{k,\ell}$ in τ is at most*

$$\frac{U_{sum}}{m} \cdot Y_k + \frac{1}{m} \cdot L_{sum} + \frac{\Lambda - 1}{m} \cdot C_{max} + \frac{m-1}{m} \cdot C_k, \tag{5.4}$$

where $\Lambda = \lceil U_{sum} \rceil$.

Proof. This theorem directly follows from Lemma 9 and Lemma 10. □

6. RESPONSE-TIME BOUNDS UNDER NON-PREEMPTIVE G-EPPF

Non-preemptive G-EPPF is similar to fully preemptive G-EPPF in the sense that when multiple jobs are eligible to be selected to execute on available processors, the ones with earlier priority points are selected; however, under non-preemptive G-EPPF scheduling, once a job starts execution, it will continuously execute to completion without being preempted or migrated. In this section, we let $\mathcal{S}$ denote the non-preemptive G-EPPF schedule specifically.

Definition 9. *Given a non-busy time instant t for Ψ, if every pending job in Ψ is currently executing, then t is a* non-blocking non-busy instant *for Ψ; otherwise if some pending job in Ψ is blocked by jobs that are not in Ψ, then t is a* blocking non-busy instant *for Ψ. In a time interval $[t_1, t_2]$, if every time instant is a blocking non-busy instant for Ψ, then $[t_1, t_2]$ is a blocking non-busy interval for Ψ.*

Definition 10. *For any time instant t, we denote $\mathcal{B}(t)$ to be the set of jobs which are not in Ψ but executing in schedule $\mathcal{S}$. Denote $B(t)$ as the* blocking work *to those incomplete work of jobs in $\mathcal{B}(t)$.*

Because $\mathcal{B}(t)$ is after all a set of jobs that are being executed, it must hold that $|\mathcal{B}(t)| \leq m$ for all t, and therefore $B(t) \leq m \cdot C_{max}$ for all t.

6.1 A Basic Response-Time Bound

In this section, we will derive a response-time bound for non-preemptive G-EPPF. Similar to the analysis for fully preemptive G-EPPF, this response-time bound is based on the analysis of busy and non-busy time intervals at or before time t_y and the execution after time t_y. By contrast, for non-preemptive G-EPPF, we further consider two cases: blocking and non-blocking for a non-busy time instant.

Lemma 11. *For any non-blocking non-busy time instant t,*
$\mathsf{LAG}(\Psi, t, \mathcal{S}) + B(t) \leq m \cdot C_{max}$.

Proof. Let $b = |\mathcal{B}(t)|$ $(0 \le b \le m)$. Then $B(t) \le b \cdot C_{max}$, and by the definition of a non-blocking non-busy time instant, the number of pending jobs in Ψ is at most $m - b$. Then, by decomposing Ψ into three disjointed subsets: $\Psi_1, \Psi_2, and \Psi_3$ that consist of jobs that are $unreleased, pending, complete$ respectively, we have

$$
\begin{aligned}
&\mathsf{LAG}(\Psi, t, \mathcal{S}) + B(t) \\
=&\mathsf{LAG}(\Psi_1, t, \mathcal{S}) + \mathsf{LAG}(\Psi_2, t, \mathcal{S}) + \mathsf{LAG}(\Psi_3, t, \mathcal{S}) + B(t) \\
=&\sum_{\tau_{i,j} \in \Psi_1} \mathsf{lag}(\tau_{i,j}, t, \mathcal{S}) + \sum_{\tau_{i,j} \in \Psi_2} \mathsf{lag}(\tau_{i,j}, t, \mathcal{S}) + \sum_{\tau_{i,j} \in \Psi_3} \mathsf{lag}(\tau_{i,j}, t, \mathcal{S}) + B(t) \\
\le& \sum_{\tau_{i,j} \in \Psi_1} 0 + \sum_{\tau_{i,j} \in \Psi_2} C_i + \sum_{\tau_{i,j} \in \Psi_3} 0 + B(t) \\
\le&|\Psi_2| \cdot C_{max} + B(t) \\
\le&(m - b) \cdot C_{max} + B(t) \\
\le&(m - b) \cdot C_{max} + b \cdot C_{max} \\
=&m \cdot C_{max},
\end{aligned}
$$

and the lemma follows. □

Lemma 12. *If* $[t_1, t_2]$ *is a busy interval for* Ψ *in* $\mathcal{S}$*, then*
$\mathrm{LAG}(\Psi, t_1, \mathcal{S}) + B(t_1) \ge \mathrm{LAG}(\Psi, t_2, \mathcal{S}) + B(t_2)$.

Proof. From Lemma 2, if $t_1, t_2 \in [t_1, t_2]$ is a busy interval for Ψ in $\mathcal{S}$, then $\mathsf{LAG}(\Psi, t_1, \mathcal{S}) \ge \mathsf{LAG}(\Psi, t_2, \mathcal{S})$. And $\mathcal{B}(t_1) = \mathcal{B}(t_2) = \emptyset$, therefore $B(t_1) = B(t_2) = 0$. We can conclude that $\mathsf{LAG}(\Psi, t_1, \mathcal{S}) + B(t_1) \ge \mathsf{LAG}(\Psi, t_2, \mathcal{S}) + B(t_2)$. □

Lemma 13. *If* $[t_1, t_2]$ *is a blocking non-busy interval for* Ψ *in* $\mathcal{S}$*, then any blocking job(i.e., any job that is not in* Ψ *but executing at some time instant t in* $[t_1, t_2]$*), must execute continuously in* $[t_1^-, t_2]$.

Proof. Since $[t_1, t_2]$ is a blocking non-busy interval for Ψ, for any time instant within$[t_1, t_2]$ there is at least one job in Ψ pending but not executing. Meanwhile, since any job in Ψ has an earlier deadline or higher priority than any job not in Ψ, there is no job not in Ψ or not executing at $[t_1^-]$ can execute in $[t_1, t_2]$. $\square$

Lemma 14. *If $[t_1, t_2]$ is a blocking non-busy interval for Ψ in $\mathcal{S}$, then* $\mathsf{LAG}(\Psi, t_1, \mathcal{S}) + B(t_1) \geq \mathsf{LAG}(\Psi, t_2, \mathcal{S}) + B(t_2)$.

Proof. We let $[t, t^{'}]$ denote a sub-interval of $[t_1, t_2]$ such that $|\mathcal{B}(t)| = |\mathcal{B}(t^{'})|$. By Lemma 13, the blocking jobs at every time instant in $[t, t^{'}]$ are exactly the jobs in $\mathcal{B}(t)$. Let p denote the number of processors on which blocking jobs execute in $[t, t^{'}]$. Then,

$$B(t^{'}) = B(t) - p \cdot (t^{'} - t).$$

Since $[t, t^{'}] \subseteq [t_1, t_2]$ is a blocking non-busy interval, the other $(m - p)$ processors must execute jobs in Ψ; otherwise, it would be a non-blocking non-busy interval. Therefore,

$$\begin{aligned} &\mathsf{LAG}(\Psi, t^{'}, \mathcal{S}) \\ &= \mathsf{LAG}(\Psi, t, \mathcal{S}) + \mathsf{A}(\mathcal{I}, \Psi, t, t^{'}) - \mathsf{A}(\mathcal{S}, \Psi, t, t^{'}) \\ &\leq \mathsf{LAG}(\Psi, t, \mathcal{S}) + U_{sum} \cdot (t^{'} - t) - (m - p) \cdot (t^{'} - t). \end{aligned}$$

Thus,

$$\begin{aligned} &\mathsf{LAG}(\Psi, t^{'}, \mathcal{S}) + B(t^{'}) \\ &\leq \mathsf{LAG}(\Psi, t, \mathcal{S}) + U_{sum} \cdot (t^{'} - t) - (m - p) \cdot (t^{'} - t) + B(t) - p \cdot (t^{'} - t) \\ &= \mathsf{LAG}(\Psi, t, \mathcal{S}) + B(t) + U_{sum} \cdot (t^{'} - t) - m \cdot (t^{'} - t) \\ &\leq \{\text{since } U_{sum} \leq m\} \\ &\quad \mathsf{LAG}(\Psi, t, \mathcal{S}) + B(t) \end{aligned}$$

Then, for every such sub-interval $[t, t^{'}] \subseteq [t_1, t_2]$, we have

$$\mathsf{LAG}(\Psi, t, \mathcal{S}) + B(t) \geq \mathsf{LAG}(\Psi, t^{'}, \mathcal{S}) + B(t^{'}).$$

Thus, the lemma follows. $\square$

Lemma 15. *In $\mathcal{S}$, the competing work for $\tau_{k,\ell}$ plus the blocking work at t_y is at most $L_{sum} + m \cdot C_{max}$.*

Proof. Similarly to Lemma 5, the competing work pending at t_y is at most $L_{sum} + \mathsf{LAG}(\Psi, t_y, \mathcal{S})$. The blocking work at t_y is $B(t_y)$, therefore the competing work for $\tau_{i,j}$ plus the blocking work at t_y is at most $L_{sum} + \mathsf{LAG}(\Psi, t_y, \mathcal{S}) + B(t_y)$. Let $t^{'}$ denote the latest non-blocking non-busy instant at or before t_y (time 0 if no such time instant exists). Then by Lemma 11, we could have

$$\mathsf{LAG}(\Psi, t^{'}, \mathcal{S}) + B(t^{'}) \leq m \cdot C_{max} \tag{6.1}$$

Besides, by the definition of $t^{'}$, time interval $(t', t_y]$ consists of busy intervals and/or blocking non-busy intervals. Therefore, by Lemma 12 and Lemma 14,

$$\mathsf{LAG}(\Psi, t^{'}, \mathcal{S}) + B(t^{'}) \geq \mathsf{LAG}(\Psi, t_y, \mathcal{S}) + B(t_y) \tag{6.2}$$

Thus, $\mathsf{LAG}(\Psi, t_y, \mathcal{S}) + B(t_y) \leq m \cdot C_{max}$ and the lemma follows. $\square$

Lemma 16. *Letting W denote the competing work plus the blocking work for $\tau_{k,\ell}$ at t_y, the job of interest $\tau_{k,\ell}$ will complete execution by*

$$t_y + \frac{W - C_k}{m} + C_k.$$

Proof. Under non-preemptive G-EPPF, if $\tau_{k,\ell}$ starts its execution at or before time t_y, then it will continuously execute and complete by time $t_y + C_k$. We then focus on the case where $\tau_{k,\ell}$ does not start its execution at or before time t_y. We let $e_{k,\ell}$ denote the actual execution time of $\tau_{k,\ell}$ — by the definition of C_k, $e_{k,\ell} \leq C_k$. Then, the competing work for $\tau_{k,\ell}$ from other jobs plus the blocking work at time t_y is $(W - e_{k,\ell})$, which can prevent $\tau_{k,\ell}$ from starting its execution for at most $\frac{W-e_{k,\ell}}{m}$ time units. That is, $\tau_{k,\ell}$ must start its execution by time $t_y + \frac{W-e_{k,\ell}}{m}$ and therefore complete by time $t_y + \frac{W-e_{k,\ell}}{m} + e_{k,\ell}$. Because $e_{k,\ell} \leq C_k$ and $m \geq 1$, it is clear that

$$t_y + \frac{W - e_{k,\ell}}{m} + e_{k,\ell} \leq t_y + \frac{W - C_k}{m} + C_k,$$

and the lemma follows. □

Theorem 3. *Under non-preemptive* G-EPPF *scheduling on m identical unit-speed processors, the response time of an arbitrary job $\tau_{k,\ell}$ in τ is at most*

$$Y_k + \frac{1}{m} \cdot L_{sum} + C_{max} + \frac{m-1}{m} \cdot C_k.$$

Proof. The theorem follows directly from Lemmas 15 and 16. □

6.2 An Improved Response-Time Bound

Following the LAG-based analysis framework, the analysis resulting in Theorem 3 in Section 6.1 focused on the competing and blocking work at time t_y. However, under non-preemptive G-EPPF, once a job starts to execute, it continuously executes until completes. Therefore, it could especially benefit to further analyze the competing and blocking work since the release of the job of interest, *i.e.*, time $r_{k,\ell}$. Based on this observation, we derive the following improved response-time bound for non-preemptive G-EPPF.

Lemma 17. *For any time instant t at or before t_y,* $\mathsf{LAG}(\Psi, t, \mathcal{S}) + B(t) \leq m \cdot C_{max}$.

Proof. Let t' denote the latest non-blocking non-busy instant at or before t_y (or time 0 if no such non-blocking non-busy instant exists). According to Lemma 11, we have

$$\mathsf{LAG}(\Psi, t', \mathcal{S}) + B(t') \leq m \cdot C_{max}$$

Moreover, because t' is the latest such time instant at or before t_y, time interval $(t', t_y]$ must consist of busy intervals and/or blocking non-busy intervals. Therefore, by Lemma 12 and Lemma 14,

$$\mathsf{LAG}(\Psi, t', \mathcal{S}) + B(t') \geq \mathsf{LAG}(\Psi, t_y, \mathcal{S}) + B(t_y)$$

Thus, $\mathsf{LAG}(\Psi, t_y, \mathcal{S}) + B(t_y) \leq m \cdot C_{max}$ and the lemma follows. □

Lemma 18. *Letting W denote the competing work plus the blocking work for $\tau_{k,\ell}$ at $r_{k,\ell}$, the job of interest $\tau_{k,\ell}$ will complete execution by*

$$r_{k,\ell} + \frac{W - C_k}{m} + C_k.$$

Proof. We let $e_{k,\ell}$ denote the actual execution time of $\tau_{k,\ell}$ — by the definition of C_k, $e_{k,\ell} \leq C_k$. Then, the competing work for $\tau_{k,\ell}$ from other jobs plus the blocking work at time $r_{k,\ell}$ is $(W - e_{k,\ell})$, which can prevent $\tau_{k,\ell}$ from starting its execution for at most $\frac{W-e_{k,\ell}}{m}$ time units. (Recall that, by Definition 8, competing work includes work that is not released yet but to be released.) That is, $\tau_{k,\ell}$ must start its execution by time $r_{k,\ell} + \frac{W-e_{k,\ell}}{m}$ and therefore complete by time $r_{k,\ell} + \frac{W-e_{k,\ell}}{m} + e_{k,\ell}$. Because $e_{k,\ell} \leq C_k$ and $m \geq 1$, it is clear that

$$r_{k,\ell} + \frac{W - e_{k,\ell}}{m} + e_{k,\ell} \leq t_y + \frac{W - C_k}{m} + C_k,$$

and the lemma follows. □

Lemma 19. *The competing work plus the blocking work for $\tau_{k,\ell}$ at $r_{k,\ell}$ is at most*

$U_{sum} \cdot Y_k + L_{sum} + m \cdot C_{max}$.

Proof. By Definitions 3, 4, 5 and 10, the competing work plus the blocking work for $\tau_{k,\ell}$ at $r_{k,\ell}$ is

$$
\begin{aligned}
& L_{sum} + \mathsf{A}(\mathcal{I}, \Psi, 0, t_y) - \mathsf{A}(\mathcal{S}, \Psi, 0, r_{k,\ell}) + B(r_{k,\ell}) \\
=& \{\text{by Definition 2}\} \\
& L_{sum} + \mathsf{A}(\mathcal{I}, \Psi, 0, r_{k,\ell}) + \mathsf{A}(\mathcal{I}, \Psi, r_{k,\ell}, t_y) - \mathsf{A}(\mathcal{S}, \Psi, 0, r_{k,\ell}) + B(r_{k,\ell}) \\
=& \{\text{by Definition 4}\} \\
& L_{sum} + \mathsf{LAG}(\Psi, r_{k,\ell}, \mathcal{S}) + \mathsf{A}(\mathcal{I}, \Psi, r_{k,\ell}, t_y) + B(r_{k,\ell}) \\
\leq& \{\text{by (4.2)}\} \\
& U_{sum} \cdot (t_y - r_{k,\ell}) + L_{sum} + \mathsf{LAG}(\Psi, r_{k,\ell}, \mathcal{S}) + B(r_{k,\ell}) \\
\leq& \{\text{because } t_y = r_{k,\ell} + Y_k \text{ and by Lemma 17}\} \\
& U_{sum} \cdot Y_k + L_{sum} + m \cdot C_{max}.
\end{aligned}
$$

The lemma follows. □

Theorem 4. *Under non-preemptive G-EPPF scheduling on m identical unit-speed processors, the response time of an arbitrary job $\tau_{k,\ell}$ in τ is at most*

$$\frac{U_{sum}}{m} \cdot Y_k + \frac{1}{m} \cdot L_{sum} + C_{max} + \frac{m-1}{m} \cdot C_k.$$

Proof. This theorem directly follows from Lemma 18 and Lemma 19. □

7. EVALUATION

7.1 Linear Programming Based Schedulability Tests

In a task set, each competing task $\tau_k = (C_k, T_k, D_k)$ is given two extra variables: Y_k and L_k respectively represent the relative PP and the difference between relative period and relative PP. The goal is to solve the values of $\{Y_k\}$ and $\{L_k\}$ under the constraints we have derived in Chapter 5 and Chapter 6 to guarantee the absolute deadlines to be met in each task set, so to minimize the sum of $\{L_k\}$.

The time complexity of each Linear Program(LP) depends on the number of variables and the number of constraints. As the definitions of $\{Y_k\}$ and $\{L_k\}$, there are $2n$ variables of each task set while the other parameters are constants. To each LP, $3n$ linear constraints are set to help bound the response time.

The formulations are shown below:

LP1:

$$\text{minimize} \qquad \sum_{k=1}^{n} L_k \tag{7.1}$$

subject to

$$L_k \geq 0, \qquad \forall k : 1 \leq k \leq n, \tag{7.2}$$

$$L_k \geq (T_k - Y_k) \cdot U_k, \qquad \forall k : 1 \leq k \leq n, \tag{7.3}$$

$$Y_k + \frac{1}{m} \cdot L_{sum} + \frac{m-1}{m} \cdot C_{max} + \frac{m-1}{m} \cdot C_k \leq D_k, \quad \forall k : 1 \leq k \leq n, \tag{7.4}$$

The objective function (7.1) minimizes the sum of difference between each absolute period and the corresponding PP. We set constraint (7.2) and constraint (7.3) by Definition 3 that $L_i = u_i \cdot \max\{0, (T_i - Y_i)\}$. Constraint (7.4) set the basic response-time bound of the arbitrary-deadline jobs for fully preemptive variant derived in Theorem 1.

LP2:

$$\text{minimize} \qquad \sum_{k=1}^{n} L_k \tag{7.5}$$

subject to

$$L_k \geq 0, \qquad \forall k : 1 \leq k \leq n, \tag{7.6}$$

$$L_k \geq (T_k - Y_k) \cdot U_k, \qquad \forall k : 1 \leq k \leq n, \tag{7.7}$$

$$\frac{U_{sum}}{m} \cdot Y_k + \frac{1}{m} \cdot L_{sum} + \frac{\Lambda - 1}{m} \cdot C_{max} + \frac{m-1}{m} \cdot C_k \leq D_k, \quad \forall k : 1 \leq k \leq n, \tag{7.8}$$

The objective function (7.5) and constraints (7.6) and (7.7) are similarly. Constraint (7.8) set the improved response-time bound of the arbitrary-deadline jobs for fully preemptive variant derived in Theorem 2. Note that $\Lambda = \lceil U_{sum} \rceil$ as defined in (5.2) of Chapter 5.2.

LP3:

$$\text{minimize} \qquad \sum_{k=1}^{n} L_k \tag{7.9}$$

subject to

$$L_k \geq 0, \qquad \forall k : 1 \leq k \leq n, \tag{7.10}$$

$$L_k \geq (T_k - Y_k) \cdot U_k, \qquad \forall k : 1 \leq k \leq n, \tag{7.11}$$

$$Y_k + \frac{1}{m} \cdot L_{sum} + C_{max} + \frac{m-1}{m} \cdot C_k \leq D_k, \qquad \forall k : 1 \leq k \leq n, \tag{7.12}$$

The objective function (7.9) and constraints (7.10) and (7.11) are similarly. Constraint (7.12) set the basic response-time bound of the arbitrary-deadline jobs for the non-preemptive variant derived in Theorem 3.

LP4:

$$\text{minimize} \qquad \sum_{k=1}^{n} L_k \tag{7.13}$$

subject to

$$L_k \geq 0, \qquad \forall k : 1 \leq k \leq n, \tag{7.14}$$

$$L_k \geq (T_k - Y_k) \cdot U_k, \qquad \forall k : 1 \leq k \leq n, \tag{7.15}$$

$$\frac{U_{sum}}{m} \cdot Y_k + \frac{1}{m} \cdot L_{sum} + C_{max} + \frac{m-1}{m} \cdot C_k \leq D_k, \quad \forall k : 1 \leq k \leq n, \tag{7.16}$$

The objective function (7.13) and constraints (7.14) and (7.15) are similarly. Constraint (7.16) set the improved response-time bound of arbitrary-deadline tasks for the non-preemptive variant derived in Theorem 4.

7.2 Task Sets Generation

To evaluate our derivations, we randomly generate task sets and calculate the response-time bound for each task in our models. Instead of generating WCET of each task directly, it is more common to generate task utilization and then calculate the WCET with $C_i = U_i \times T_i$. The reason for this is because the total utilization is a common covariate in schedulability test experiments and it is often related to the number of processors m. We conduct the experiments separately on 16, 8, and 4 identical multiprocessor platforms, where the total utilization $U_{sum} \leq m$. In each platform, we increase the total utilization step by step from an initialized u greater than zero. 1000 task sets are generated within one step and for each task set 50 tasks are generated. We implement the UUniFast-Discard algorithm, suggested by Davis and Burns in [28], to generate utilization of each task, that keeps all tasks utilization uniformly random and the sum utilization equals to a sampled value for multiprocessor systems.

We take the period of each task randomly from {200, 400, 500, 600}, and the least common multiple(lcm) of this list is 6000. We also compare this period setting with {200, 300, 400, 500, 600, 700, 800, 900, 1000} and the uniform period {500}, the lcm for the former is 252000 and the latter is 500 itself. As our models are arbitrary-deadline tasks, we set the relative deadline by multiply a coefficient from list {0.3, 0.5, 0.7, 1.0, 1.5, 2.0, 3.0, 5.0} to period. Then we could compute WCET with the product of utilization and period for each task.

7.3 Schedulability Experiments

7.3.1 G-EPPF v.s. Prior Work on G-EDF

We examine whether a task set is schedulable under HRT sporadic task multiprocessor platforms and calculate the scheduled ratio of all task sets under a total utilization. We employ Density test and Load test as the comparative models. And then examine both basic and improved response-time bounds for each task set of our fully preemptive variant. The four lines represent Density test, Load test, basic, and improved response-time bounds respectively in the plots. With the LP we have formalized in Section 7.1, the optimal solution of the LP will provide the proper values of $\{Y_k\}$ and $\{L_k\}$ of Theorem 1 and Theorem 2. Here we implement [31] (version 9.1.0), a powerful mathematical optimization solver, to find if there exists an optimal solution for each task set, and if there does, calculate the corresponding values.

Due to the space limitation, all experiment results under various settings are reported in Chapter 8 attached at the end of this thesis. In this section, we will only illustrate several representative plots for a better explanation.

As Figure 2 depicts, on a 16 processors platform when the relative deadline is 2.0 times as large as the period, the lower total utilization, the better schedulability ratio for all four schedulers. Specifically, the schedulability of Density test and Load test drop down sharply before half of the accumulative total utilization. While both fully preemptive

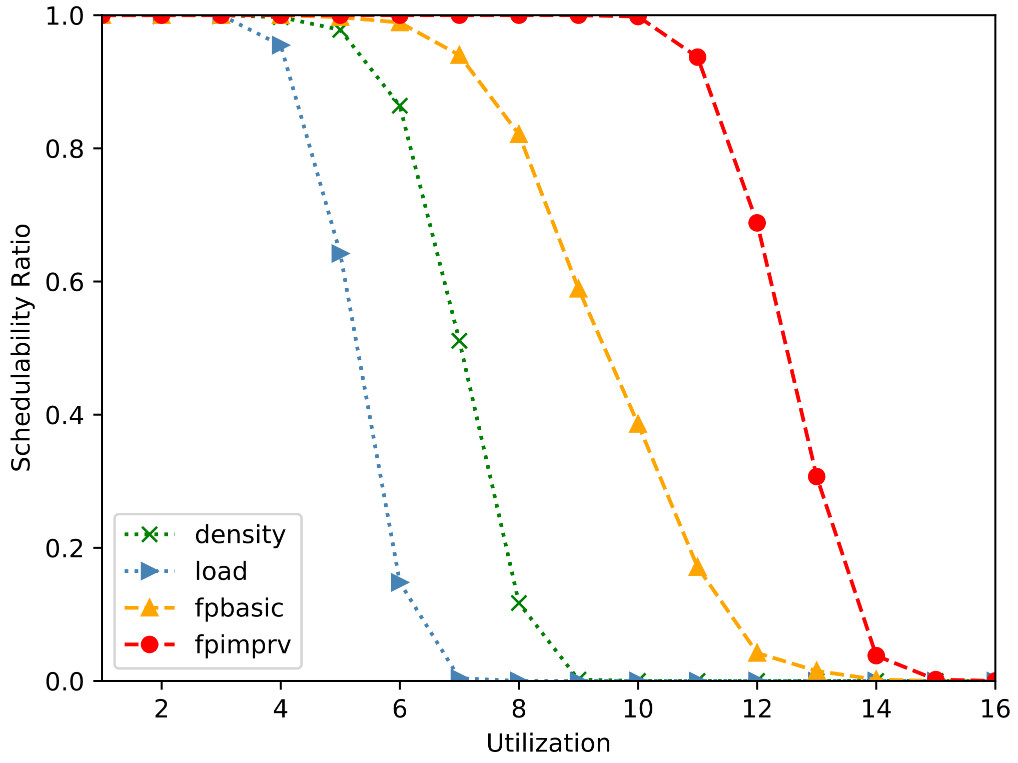


Figure 2: $m = 16$, 1000 task sets per given total utilization, 50 tasks per set, deadline = 2.0 $\times$ period.

variant basic and improved lines start to decline after they exceed half. The basic model line declines relatively slowly just after half point, while the improved model line almost starts to drop after 70% of the total. Overall, our fully preemptive variant has better schedulability than the prior work, and the improvement, especially for improved response-time bound scheduler, is more significant. Similar results can be obtained on 8 and 4 processor systems with relative deadlines larger than the period.

In terms of processor amount, in Figure 2 and Figure 3, we keep all parameters the same except comparing the impact of processors amount. The precise ratios are displayed in Table 1. Under the same total utilization, it is simple to get a higher schedulability ratio with more processors. However, compared to the limited increasing schedulability ratios of Density test and Load test, fully preemptive response-time bounds take more advantages to the growth of processors number.

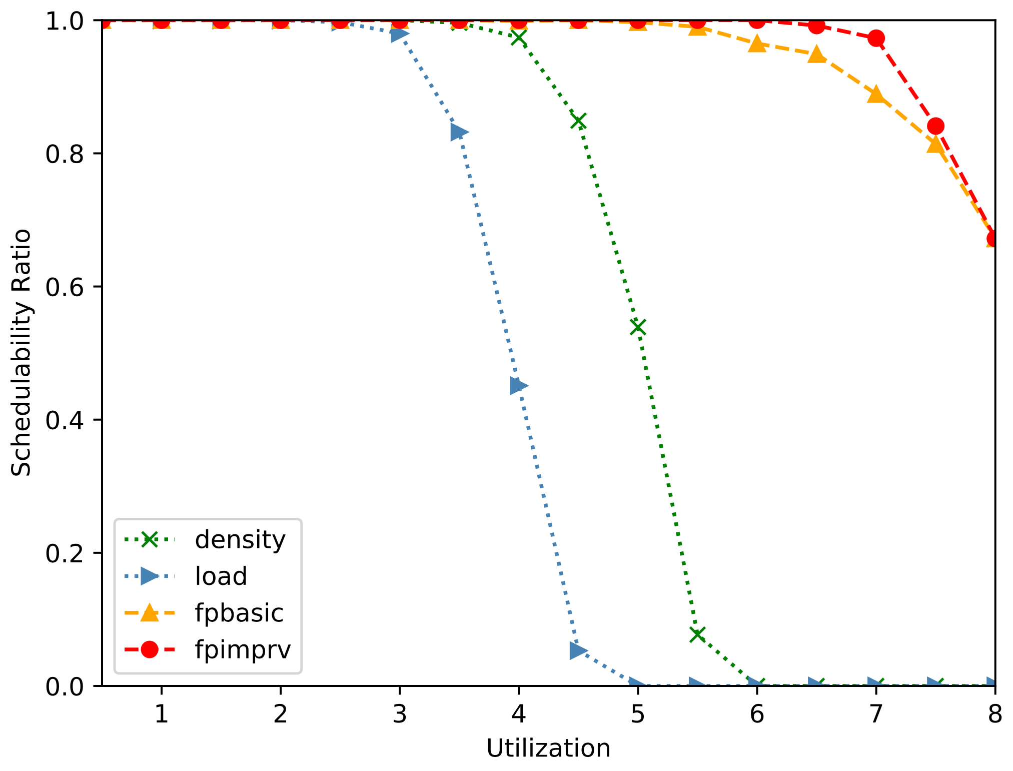


Figure 3: $m = 8$, 1000 task sets per given total utilization, 50 tasks per set, deadline = 2.0 $\times$ period.

Table 1: Schedulability Ratio Comparison with $m = 16$ and $m = 8$

Utilization	**m**	**density**	**load**	**fpbasic**	**fpimprv**
4.0	16	99.7	95.5	100.0	100.0
4.0	8	97.4	45.1	99.9	100.0
6.0	16	86.4	14.8	98.9	100.0
6.0	8	0.0	0.0	96.5	100.0
8.0	16	11.7	0.0	82.1	100.0
8.0	8	0.0	0.0	67.2	67.2

In terms of task number, in Figure 4, we allocate 100 tasks to each task set and keep all other parameters remaining the same. We can conclude that increases the number of tasks do improve the schedulability of all schedulers. Compare to Density test and Load test, the fully preemptive variant gets more significant improvement, even more than 20% schedulability ratio at the full utilization with the maximum computing capability.

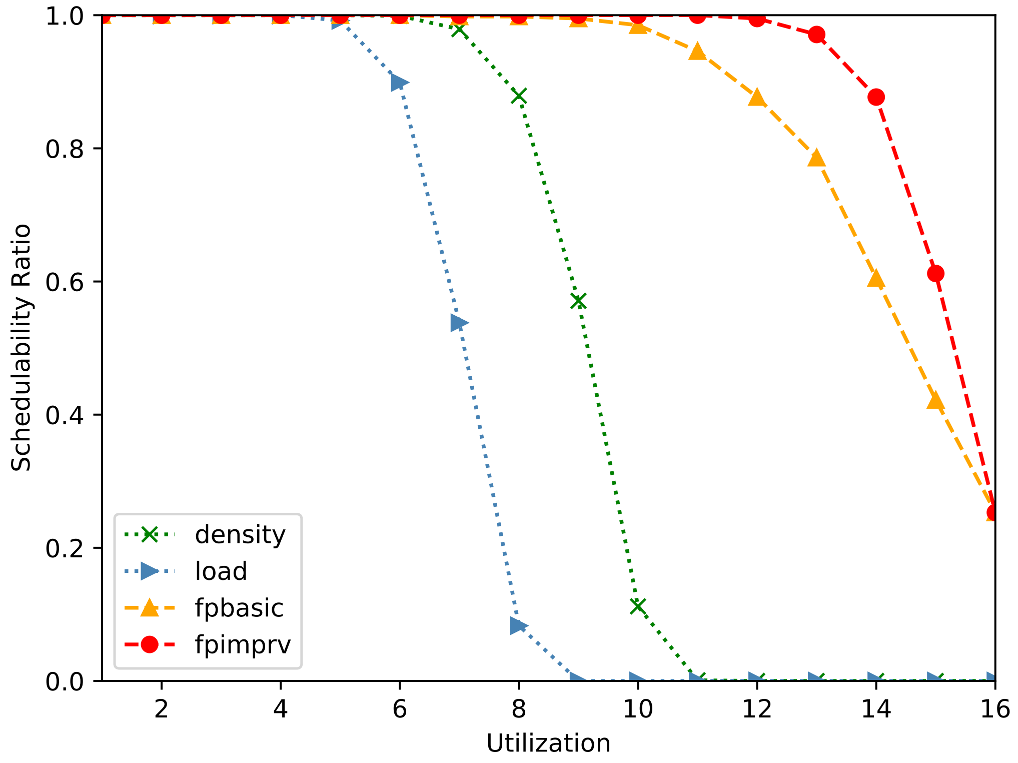


Figure 4: $m = 16$, 1000 task sets per given total utilization, 100 tasks per set, deadline = $2.0 \times$ period.

To all these four schedulability test models, in order to examine various arbitrary-deadline setting, we first test task sets with all deadlines have a fixed coefficient to its periods. Figure A.1 to Figure A.4 show the results under constrained-deadline conditions. Compare to Density test and Load test, fully preemptive basic response-time bound performs poorly whereas the improved bounds keep the best schedule performance out of the four. When the deadline getting larger than the period, the basic bounds gradually surpass the Load test then the Density test, while the improved bounds enlarge greatly as shown in Figure A.5 and Figure A.6. Fully preemptive variant could even come up to 100% scheduled if the relative deadline is large enough, this is shown in Figure A.7 to Figure A.8. Similar results can be obtained with 8 and 4 processors platforms, as shown from Figure A.13 to Figure A.25. Secondly, we break the coefficient list randomly and mix the deadline arbitrarily in each task set. Figure A.9, Figure A.19

and Figure A.26 illustrate this hybrid condition of deadline. The improved bound has a better schedulability than prior work for arbitrary-deadline tasks.

In term of period values, Figure A.10 shows the results from uniform period {500}, and Figure A.11 shows the results from period list {200, 300, 400, 500, 600, 700, 800, 900, 1000}. The smaller the difference between the minimum and the maximum period, the better the scheduling performance. Though the large range of the period list has a negative impact on the whole, the improved response-time bound keeps better schedulability than prior work.

7.3.2 Fully Preemptive v.s. Non-preemptive G-EPPF

As illustrated in Figure 5, the basic and improved bounds of fully preemptive surpass the corresponding lines of non-preemptive variant, which indicates that the former could schedule better than the latter under the same conditions. In particular, the improved bound of fully preemptive variant offers a more significant improvement.

From the point of conventional view, the non-preemptive scheduling will have a substantial loss compared to the fully preemptive case. While in our experiment, the non-preemptive bounds perform relatively close to the fully preemptive basic bound, which indicates that our analysis techniques can hold effectiveness in scheduling. Similar results with different parameters can be found in Figure A.27 to Figure A.31.

According to various requirements, there are several ways to set the parameters of our bounds. Firstly, as the number of processors m is a parameter in our models, increase m would reduce the response-time bounds and consequently improve the performance of scheduling. Secondly, if we increase the task number n in each task set, the allocated utilization u_i to each task would getting smaller under a fixed total utilization. Due to $u_i = C_i/T_i$, the C_{max} would probably also getting smaller if keep T_i the same, thus we could obtain a better schedulability. Thirdly, the period list with a narrow range would have better schedulability under the same conditions.

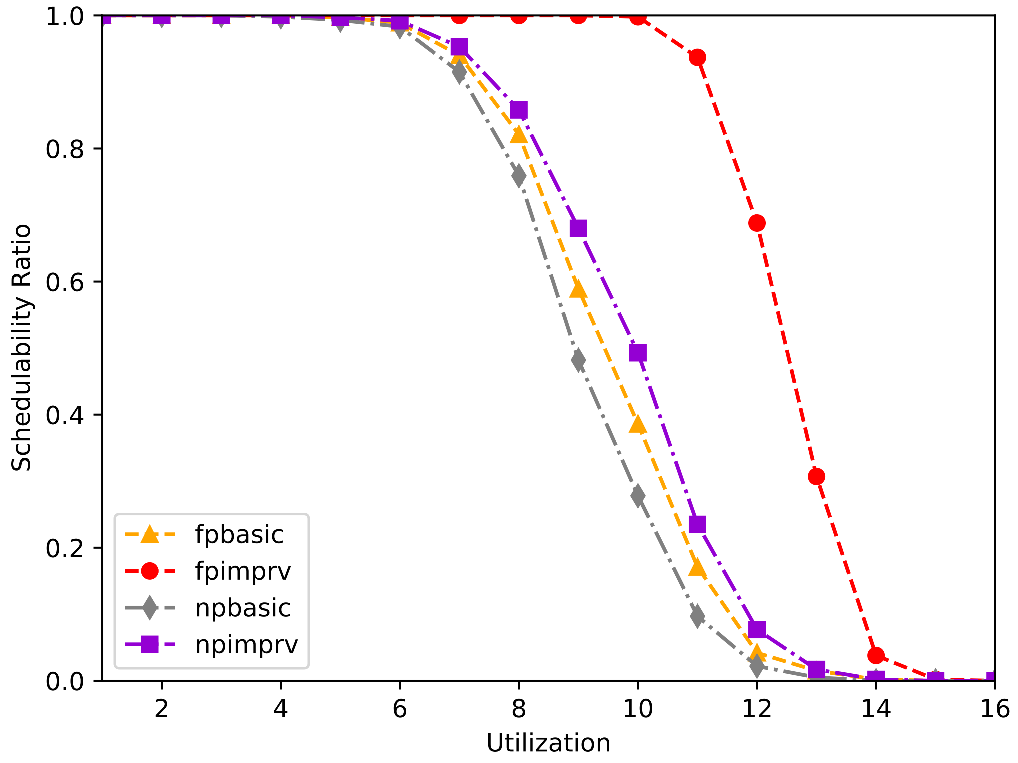


Figure 5: Comparison of fully preemptive and non-preemptive cases: $m = 16$, 1000 task sets per given total utilization, 50 tasks per set, deadline = 2.0 $\times$ period.

8. CONCLUSION

In this thesis, we explore that the G-EPPF schedulers can be driven by the priority point of a task. Chapter 5 and Chapter 6 derived the basic and improved response-time bounds of our G-EPPF scheduling algorithm, under both fully preemptive and non-preemptive variants, for scheduling arbitrary-deadline tasks of HRT sporadic task system upon multiprocessor platforms. We address the problem of setting PP for a set of tasks to meet their specified deadlines by solving a LP. In the LP, the number of variables is linear to $2n$ and the number of LP constraints is linear to $3n$. Thus, the task set is schedulable if the LP has a feasible solution, and the priority point setting is directly obtained by the the feasible solution of the LP.

In arbitrary-deadline task systems, our basic bound of fully preemptive variant performs relatively pessimistic under constrained-deadline conditions, but shows better schedulability than Density test and Load test for tasks with longer deadlines relative to their periods. As for the improved response-time bound of fully preemptive scheduling, it outperforms both Density test and Load test in most of the cases, especially offers more significant improvement when tasks' relative deadlines are larger than their periods.

In contrast to Density test and Load test, which are not applicable to non-preemptive cases, our non-preemptive bounds have close results to the basic bound of fully preemptive cases, which indicates that our analysis techniques are applicable and effective to non-preemptive schedulers.

APPENDIX SECTION

APPENDIX 1: Schedulability Results for G-EPPF v.s. G-EDF

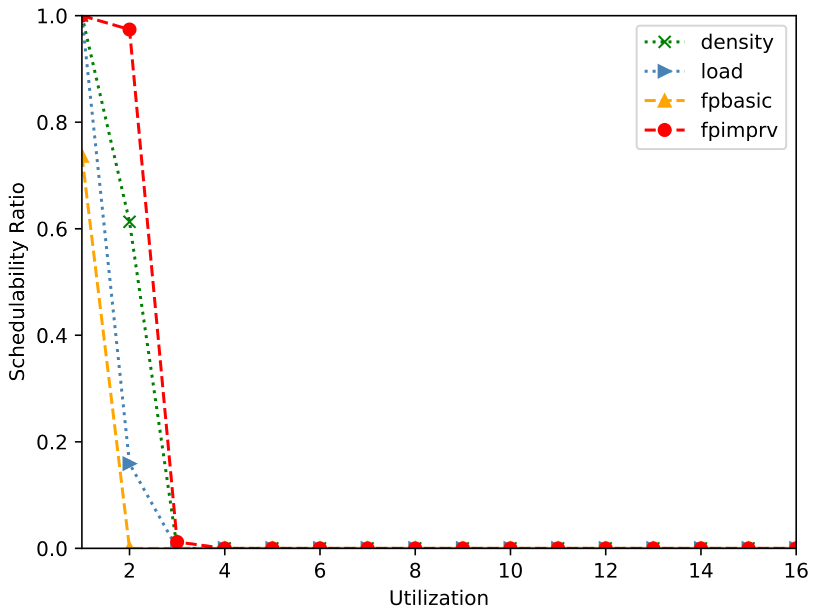


Figure A.1: $m = 16$, 1000 task sets per given total utilization, 50 tasks per set, deadline = 0.3 $\times$ period.

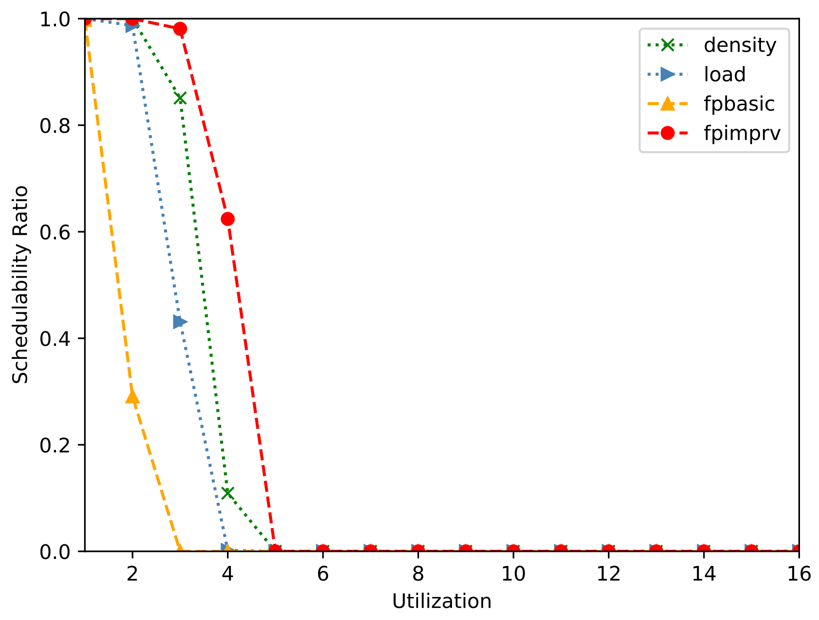


Figure A.2: $m = 16$, 1000 task sets per given total utilization, 50 tasks per set, deadline = 0.5 $\times$ period.

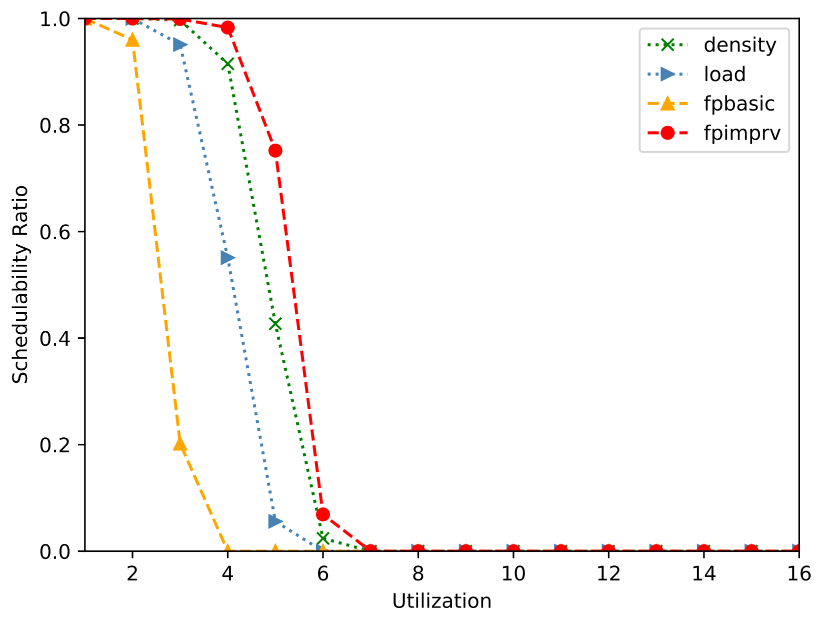


Figure A.3: $m = 16$, 1000 task sets per given total utilization, 50 tasks per set, deadline = 0.7 $\times$ period.

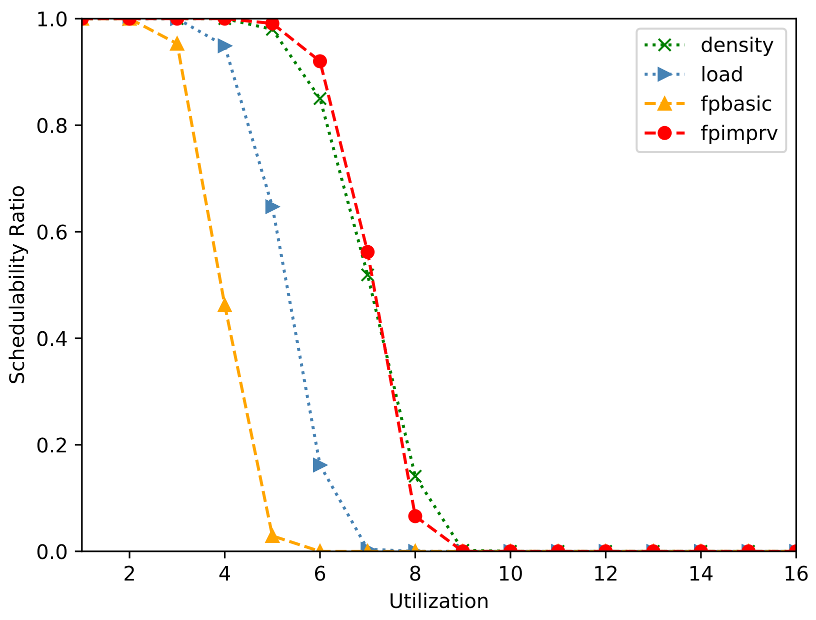


Figure A.4: $m = 16$, 1000 task sets per given total utilization, 50 tasks per set, deadline = 1.0 $\times$ period.

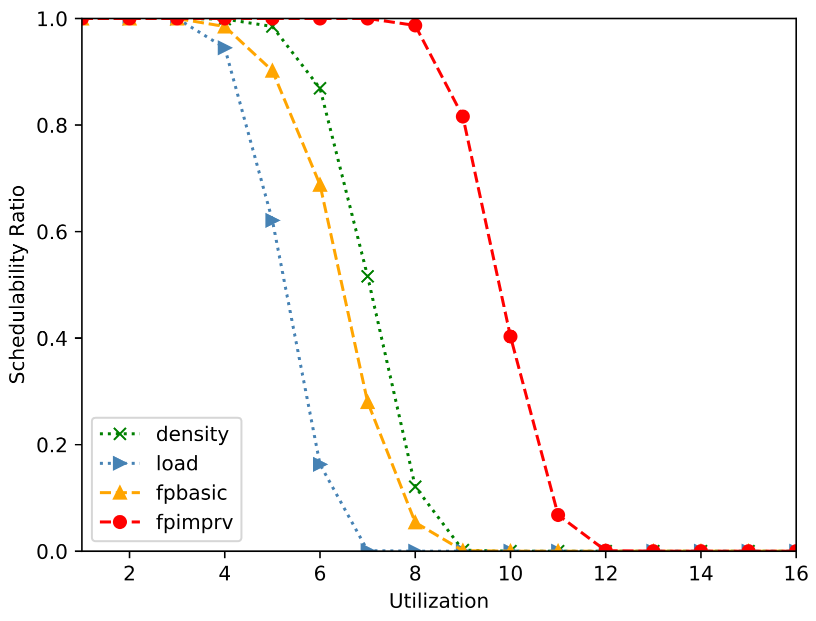


Figure A.5: $m = 16$, 1000 task sets per given total utilization, 50 tasks per set, deadline = 1.5 $\times$ period.

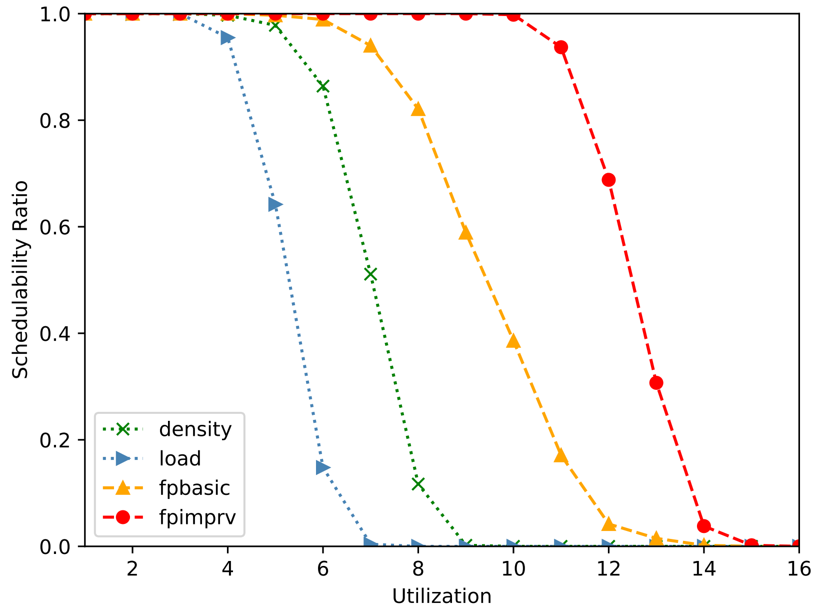


Figure A.6: $m = 16$, 1000 task sets per given total utilization, 50 tasks per set, deadline = 2.0 $\times$ period.

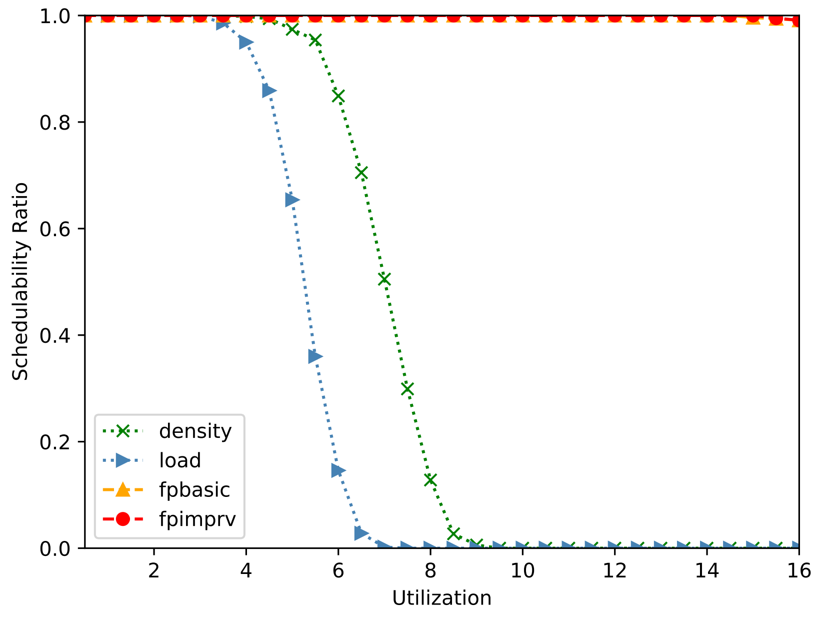


Figure A.7: $m = 16$, 1000 task sets per given total utilization, 50 tasks per set, deadline = 3.0 $\times$ period.

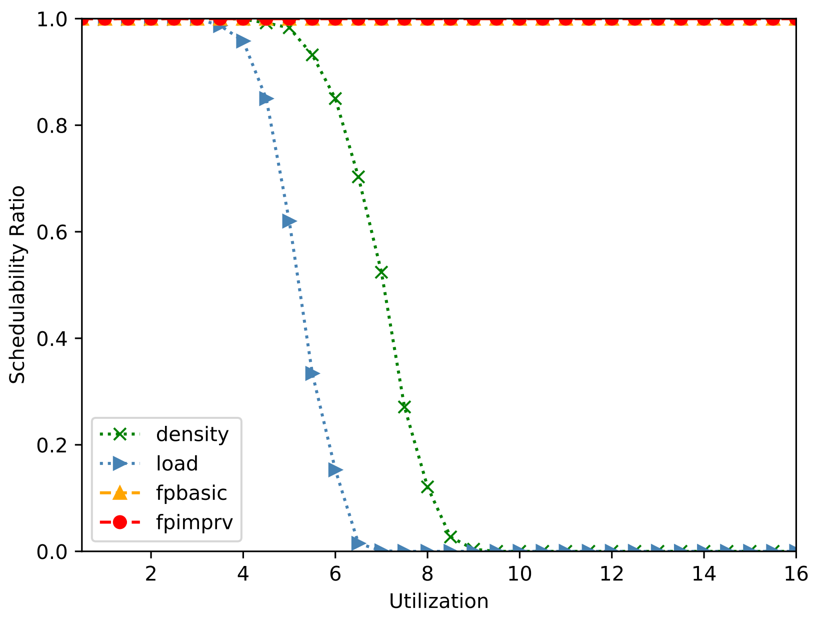


Figure A.8: $m = 16$, 1000 task sets per given total utilization, 50 tasks per set, deadline = 5.0 $\times$ period.

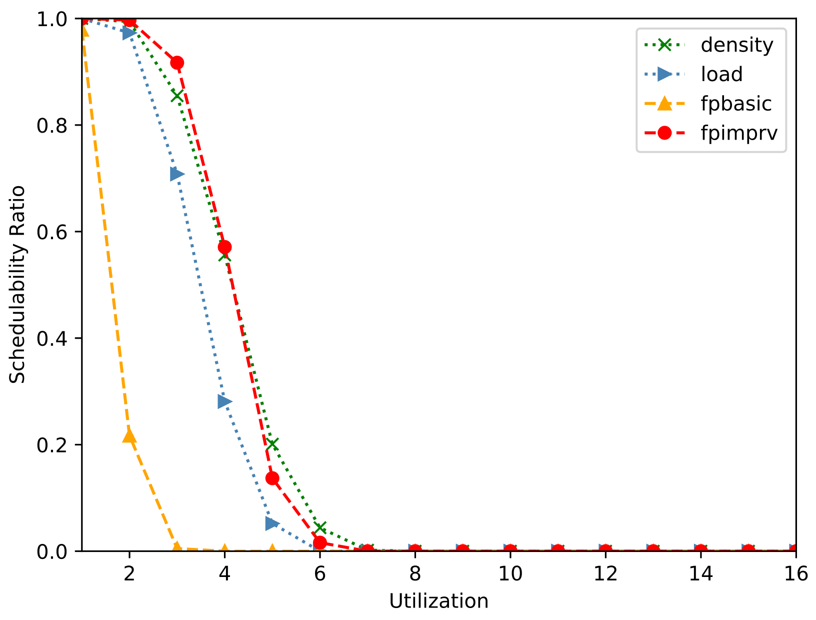


Figure A.9: $m = 16$, 1000 task sets per given total utilization, 50 tasks per set, deadline = random $\times$ period.

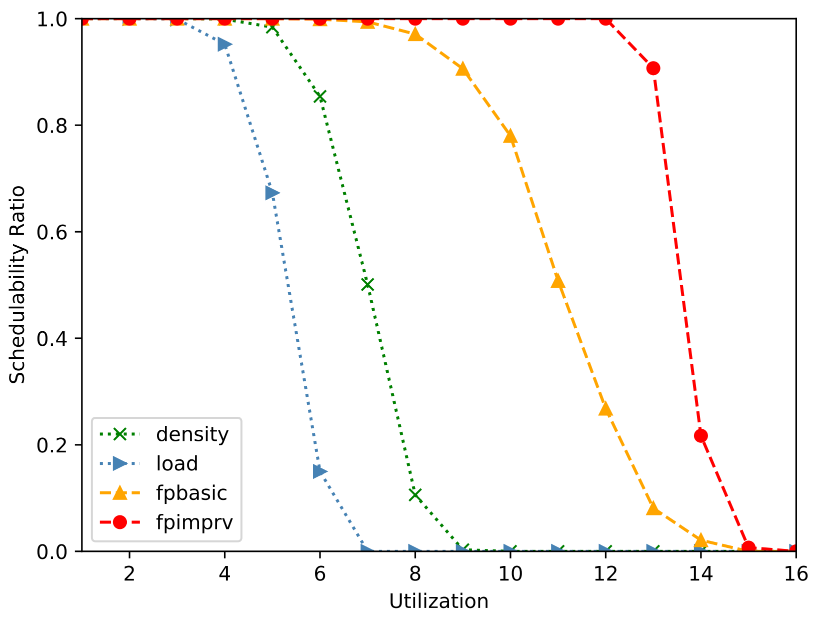


Figure A.10: $m = 16$, 1000 task sets per given total utilization, 50 tasks per set, deadline = 2.0 $\times$ period, each period = 500.

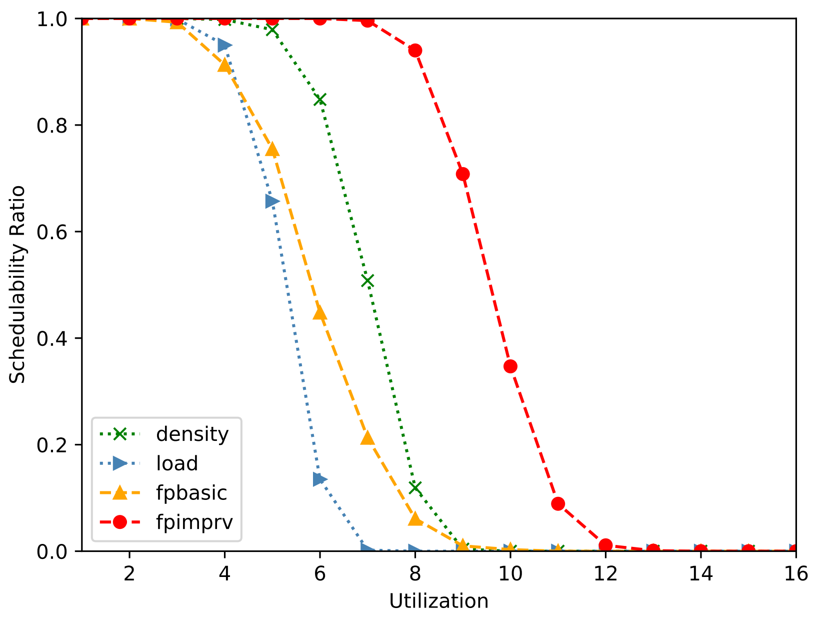


Figure A.11: $m = 16$, 1000 task sets per given total utilization, 50 tasks per set, deadline = 2.0 $\times$ period, period selected from {200, 300, 400, 500, 600, 700, 800, 900, 1000}.

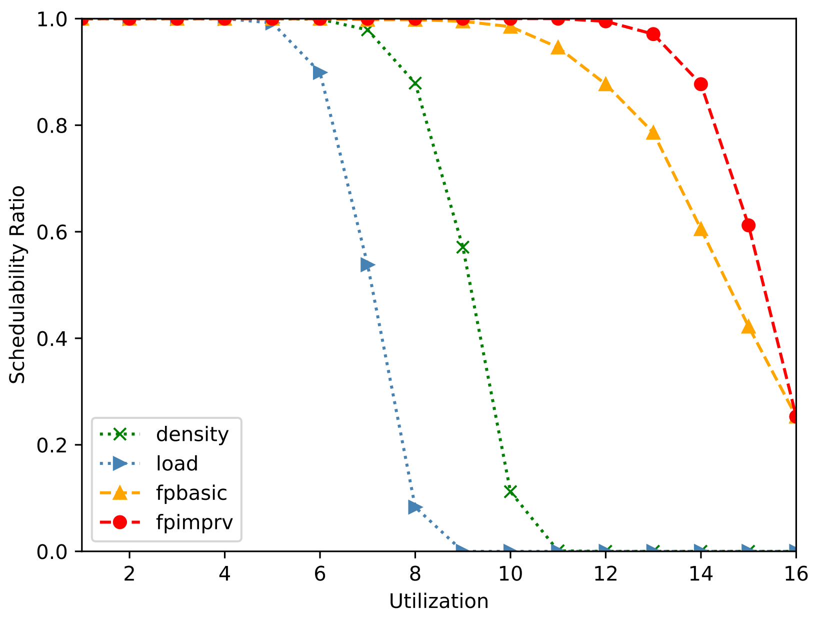


Figure A.12: $m = 16$, 1000 task sets per given total utilization, 100 tasks per set, deadline = 2.0 $\times$ period.

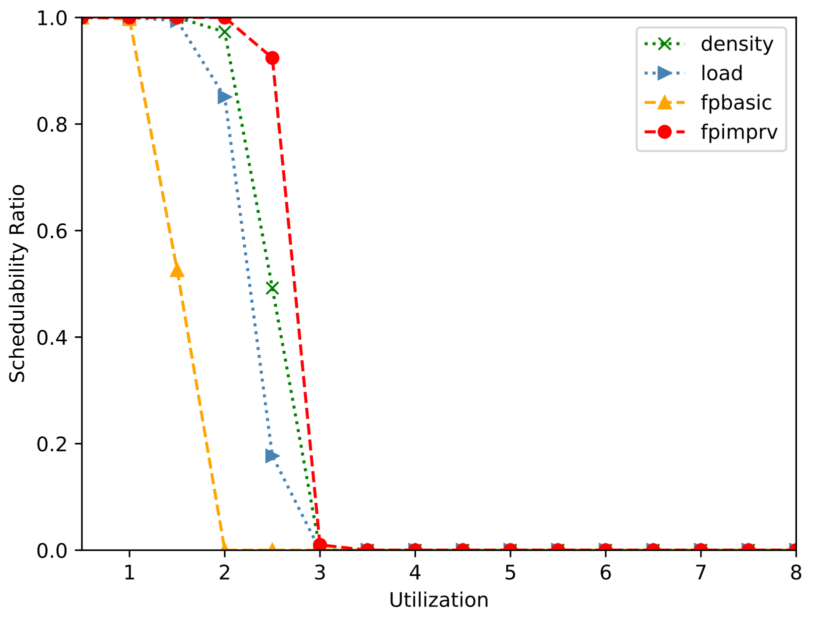


Figure A.13: $m = 8$, 1000 task sets per given total utilization, 50 tasks per set, deadline = 0.5 $\times$ period.

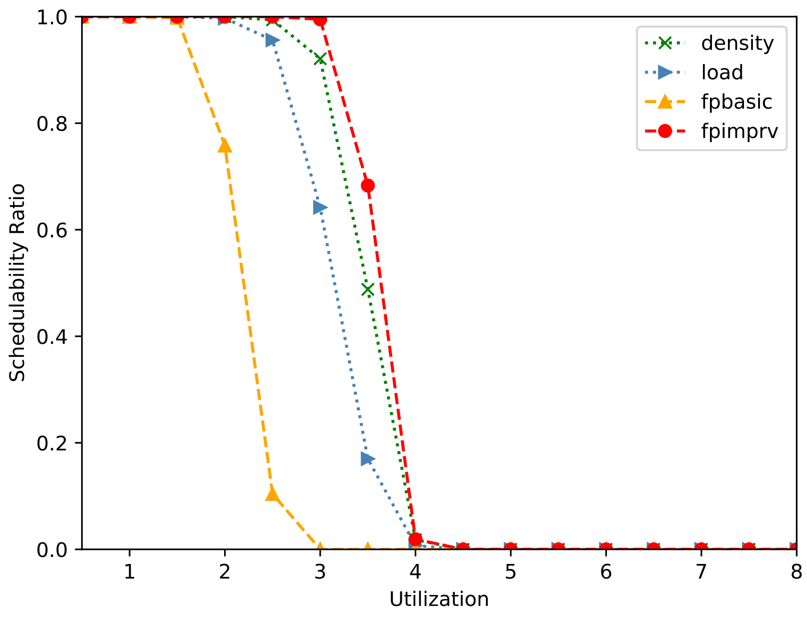


Figure A.14: $m = 8$, 1000 task sets per given total utilization, 50 tasks per set, deadline = 0.7 $\times$ period.

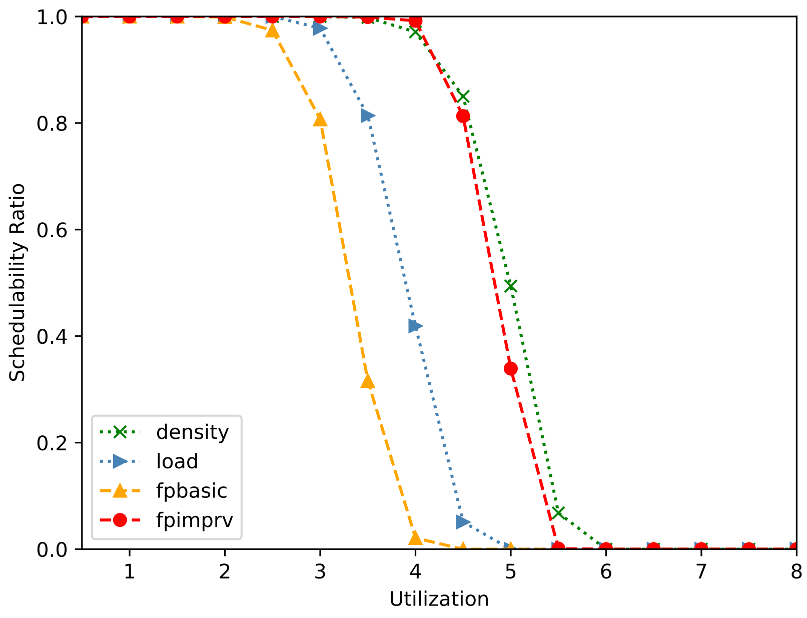


Figure A.15: $m = 8$, 1000 task sets per given total utilization, 50 tasks per set, deadline = 1.0 $\times$ period.

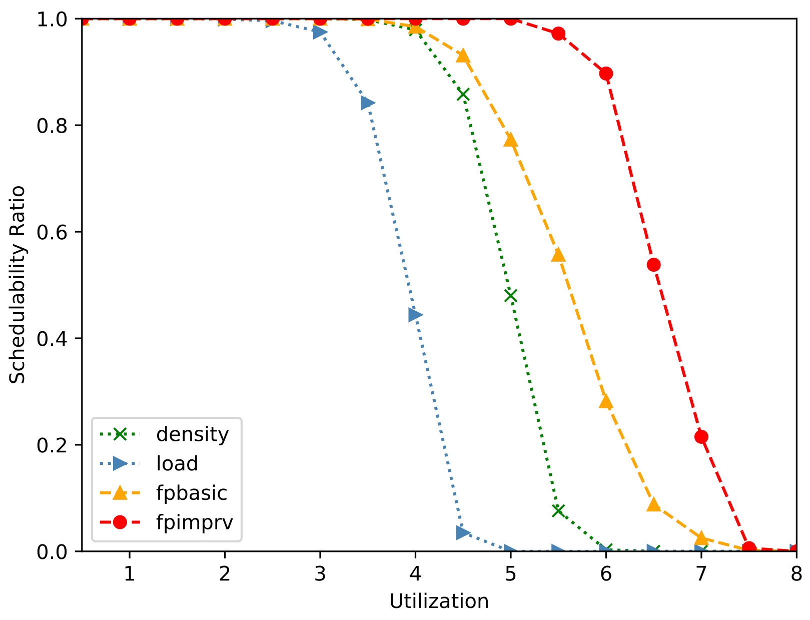


Figure A.16: $m = 8$, 1000 task sets per given total utilization, 50 tasks per set, deadline = 1.5 $\times$ period.

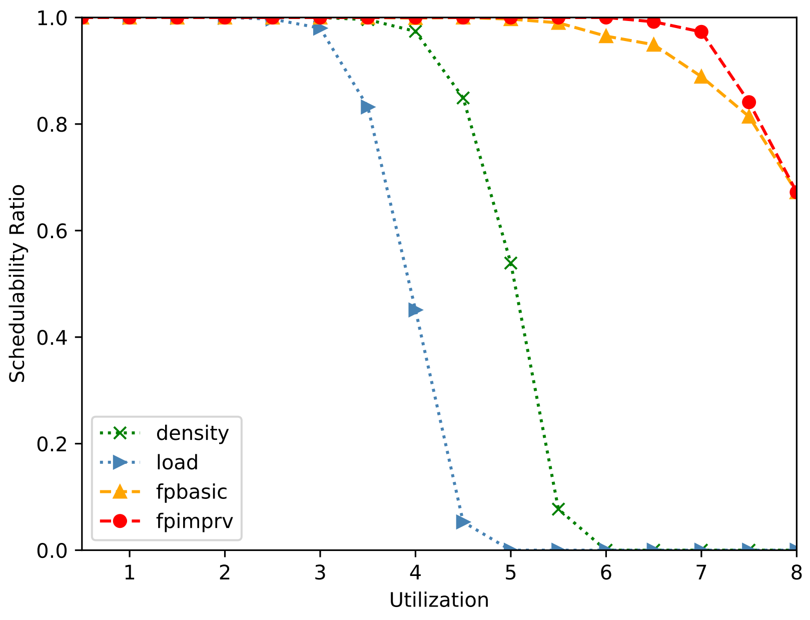


Figure A.17: $m = 8$, 1000 task sets per given total utilization, 50 tasks per set, deadline = 2.0 $\times$ period.

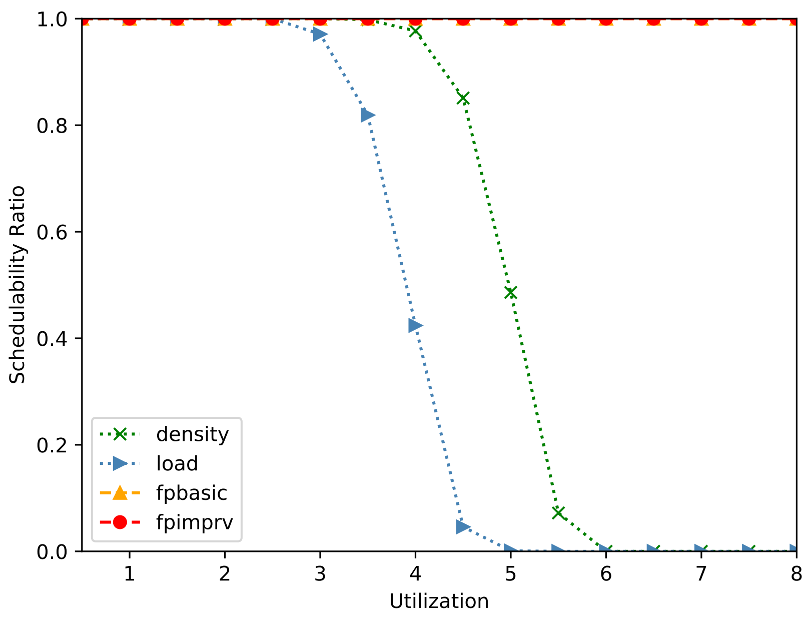


Figure A.18: $m = 8$, 1000 task sets per given total utilization, 50 tasks per set, deadline = 3.0 $\times$ period.

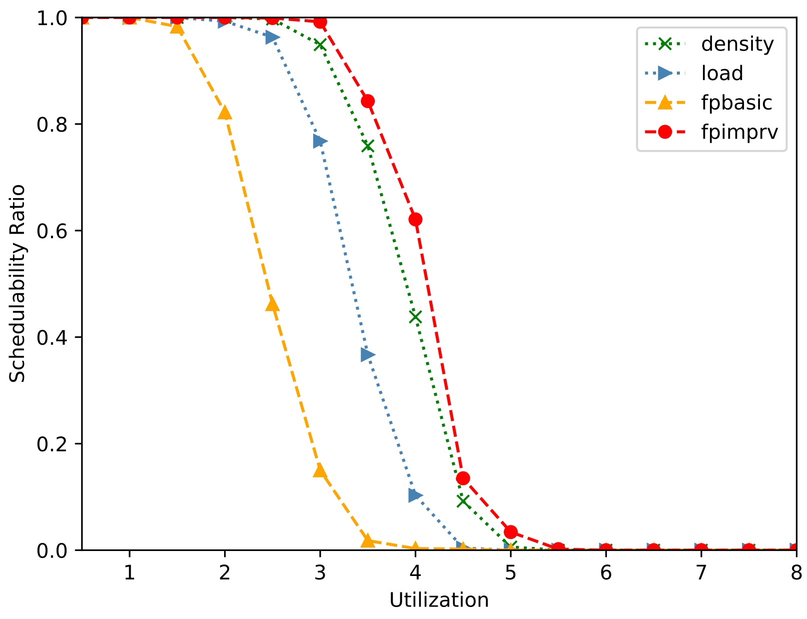


Figure A.19: $m = 8$, 1000 task sets per given total utilization, 50 tasks per set, deadline = random $\times$ period.

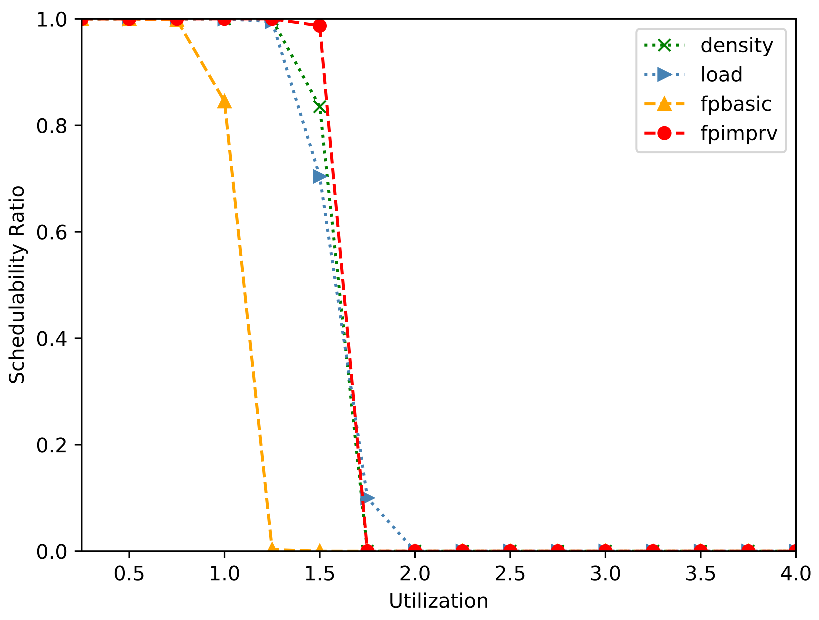


Figure A.20: $m = 4$, 1000 task sets per given total utilization, 50 tasks per set, deadline = 0.5 $\times$ period.

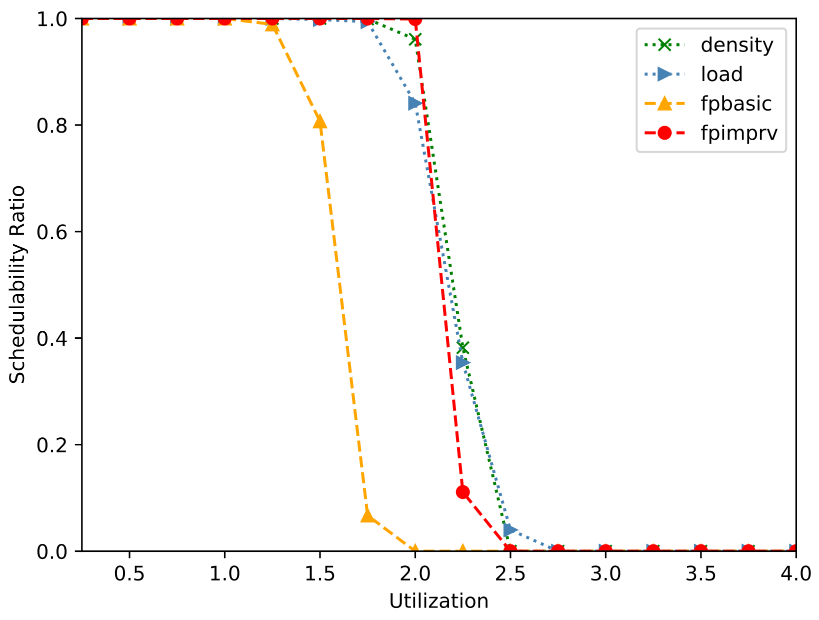


Figure A.21: $m = 4$, 1000 task sets per given total utilization, 50 tasks per set, deadline = 0.7 $\times$ period.

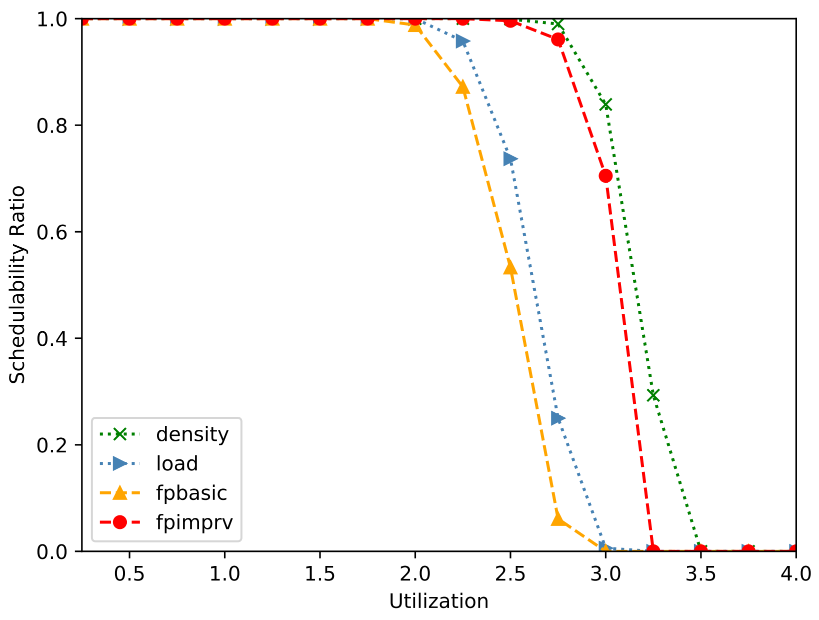


Figure A.22: $m = 4$, 1000 task sets per given total utilization, 50 tasks per set, deadline = 1.0 $\times$ period.

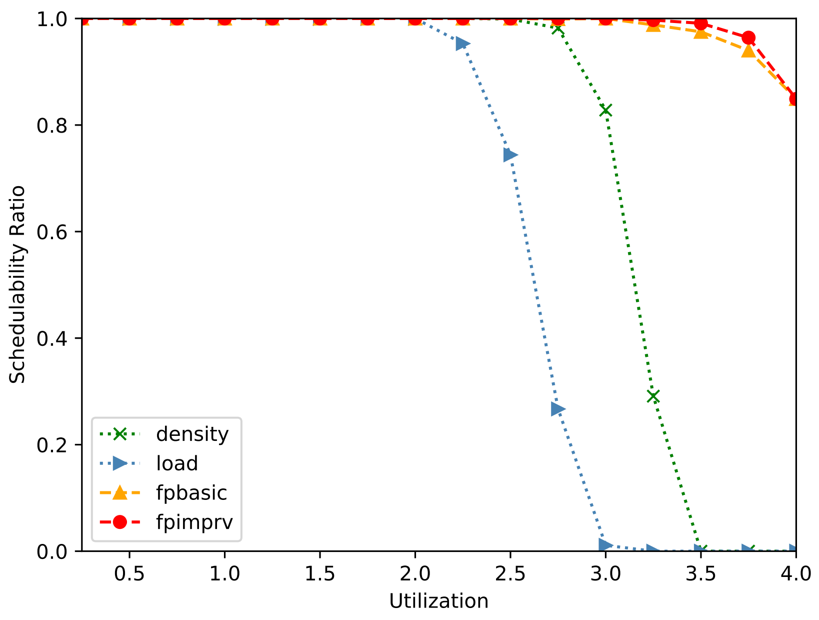


Figure A.23: $m = 4$, 1000 task sets per given total utilization, 50 tasks per set, deadline = 1.5 $\times$ period.

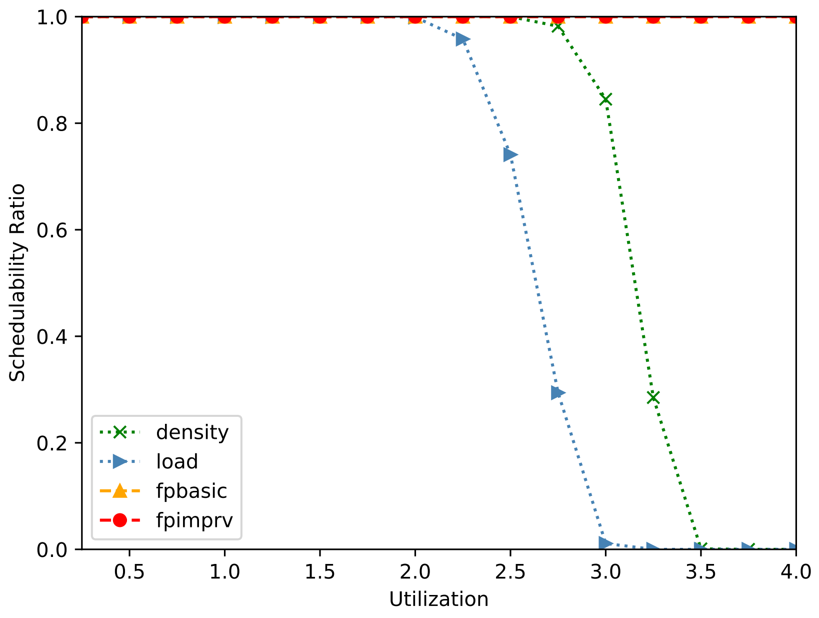


Figure A.24: $m = 4$, 1000 task sets per given total utilization, 50 tasks per set, deadline = 2.0 $\times$ period.

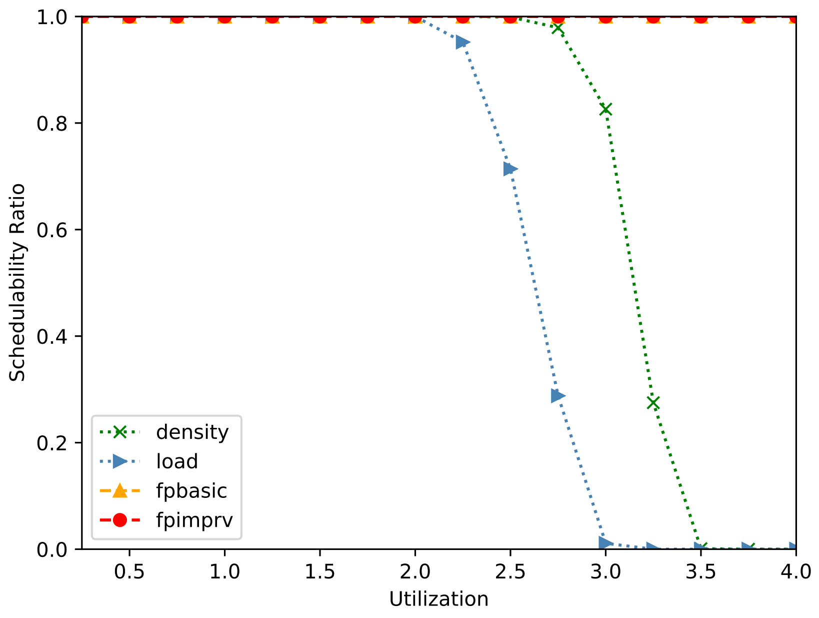


Figure A.25: $m = 4$, 1000 task sets per given total utilization, 50 tasks per set, deadline = 3.0 $\times$ period.

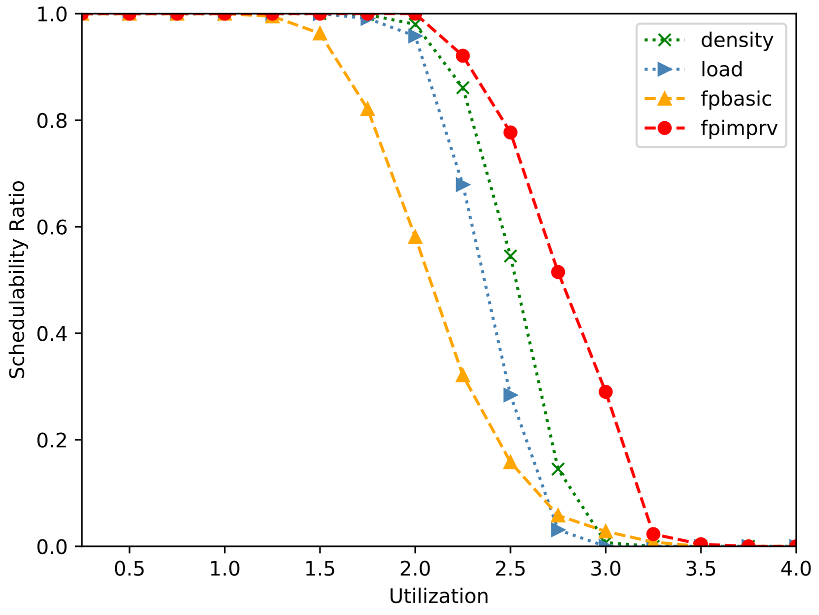


Figure A.26: $m = 4$, 1000 task sets per given total utilization, 50 tasks per set, deadline = random $\times$ period.

APPENDIX 2: Schedulability Results for Fully Preemptive G-EPPF v.s. Non-Preemptive G-EPPF

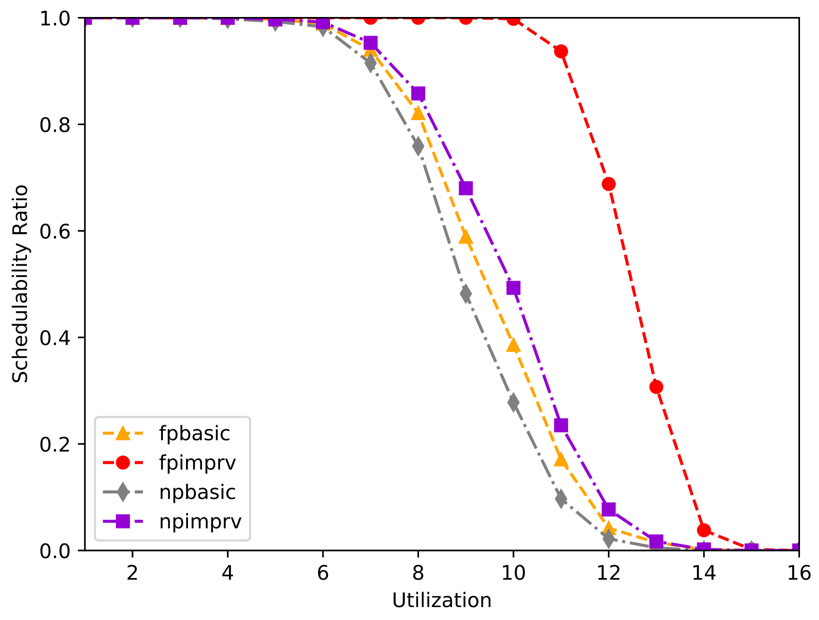


Figure A.27: Comparison of fully preemptive and non-preemptive cases: $m = 16$, 1000 task sets per given total utilization, 50 tasks per set, deadline = 2.0 $\times$ period.

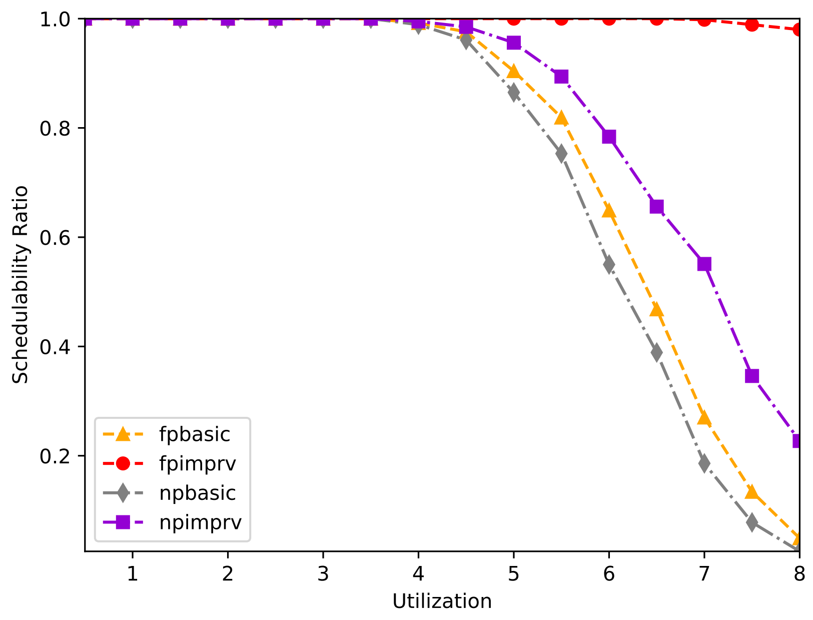


Figure A.28: Comparison of fully preemptive and non-preemptive cases: $m = 8$, 1000 task sets per given total utilization, 50 tasks per set, deadline = 1.5 $\times$ period.

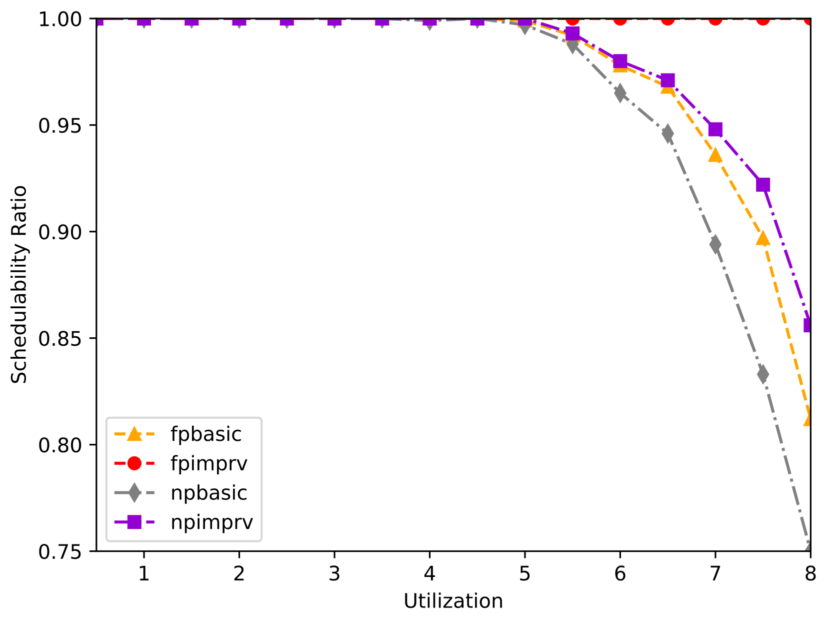


Figure A.29: Comparison of fully preemptive and non-preemptive cases: $m = 8$, 1000 task sets per given total utilization, 50 tasks per set, deadline = 2.0 $\times$ period.

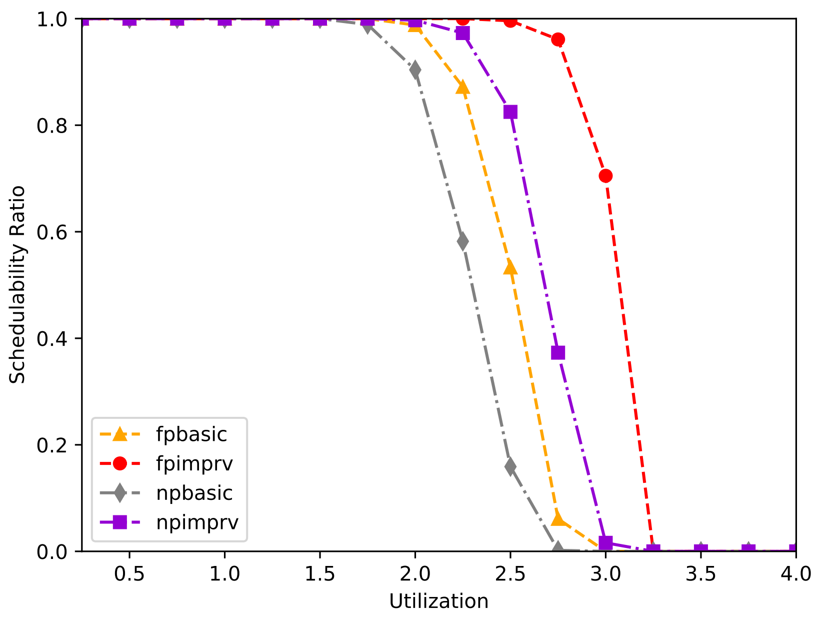


Figure A.30: Comparison of fully preemptive and non-preemptive cases: $m = 4$, 1000 task sets per given total utilization, 50 tasks per set, deadline = 1.0 $\times$ period.

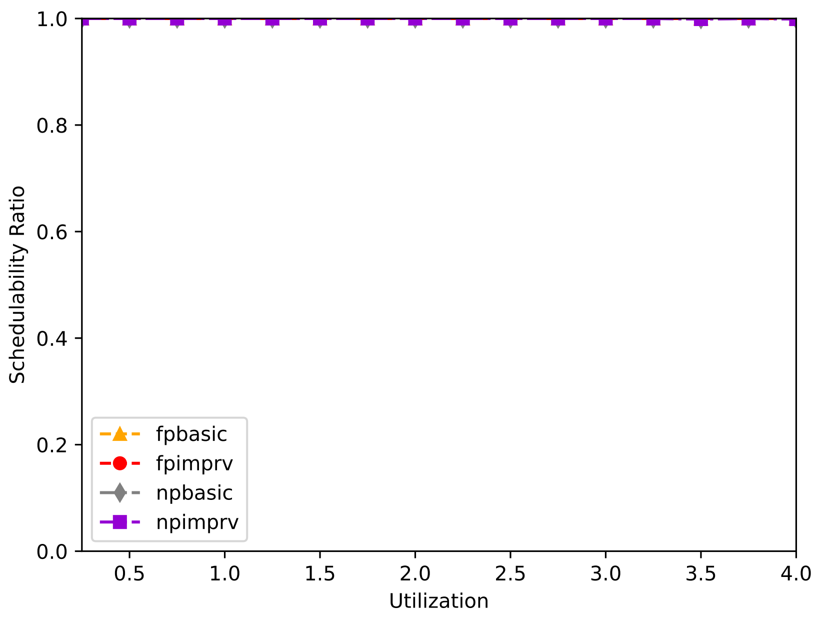


Figure A.31: Comparison of fully preemptive and non-preemptive cases: $m = 4$, 1000 task sets per given total utilization, 50 tasks per set, deadline = 2.0 $\times$ period.

REFERENCES

[1] S. Baruah and T. Baker, “Global edf schedulability analysis of arbitrary sporadic task systems,” in *2008 Euromicro Conference on Real-Time Systems*, pp. 3–12, IEEE, 2008.

[2] J. Goossens and R. Devillers, “Feasibility intervals for the deadline driven scheduler with arbitrary deadlines,” in *Proceedings Sixth International Conference on Real-Time Computing Systems and Applications. RTCSA’99 (Cat. No. PR00306)*, pp. 54–61, IEEE, 1999.

[3] S. Baruah and T. Baker, “Schedulability analysis of global edf,” *Real-Time Systems*, vol. 38, no. 3, pp. 223–235, 2008.

[4] S. K. Baruah and N. W. Fisher, “The partitioned dynamic-priority scheduling of sporadic task systems,” *Real-Time Systems*, vol. 36, no. 3, pp. 199–226, 2007.

[5] U. C. Devi and J. H. Anderson, “Tardiness bounds under global edf scheduling on a multiprocessor,” *Real-Time Systems*, vol. 38, no. 2, pp. 133–189, 2008.

[6] J. Erickson, U. Devi, and S. Baruah, “Improved tardiness bounds for global edf,” in *2010 22nd Euromicro Conference on Real-Time Systems*, pp. 14–23, IEEE, 2010.

[7] H. Leontyev and J. H. Anderson, “Tardiness bounds for fifo scheduling on multiprocessors,” in *19th Euromicro Conference on Real-Time Systems (ECRTS’07)*, pp. 71–71, IEEE, 2007.

[8] H. Leontyev and J. H. Anderson, “Generalized tardiness bounds for global multiprocessor scheduling,” *Real-Time Systems*, vol. 44, no. 1-3, pp. 26–71, 2010.

[9] H. Leontyev, S. Chakraborty, and J. H. Anderson, “Multiprocessor extensions to real-time calculus,” *Real-Time Systems*, vol. 47, no. 6, p. 562, 2011.

[10] J. P. Erickson, J. H. Anderson, and B. C. Ward, “Fair lateness scheduling: Reducing maximum lateness in g-edf-like scheduling,” *Real-Time Systems*, vol. 50, no. 1, pp. 5–47, 2014.

[11] S. Funk, J. Goossens, and S. Baruah, “On-line scheduling on uniform multiprocessors,” in *Proceedings 22nd IEEE Real-Time Systems Symposium (RTSS 2001)(Cat. No. 01PR1420)*, pp. 183–192, IEEE, 2001.

[12] S. Funk and S. Baruah, “Characteristics of edf schedulability on uniform multiprocessors,” in *15th Euromicro Conference on Real-Time Systems, 2003. Proceedings.*, pp. 211–218, IEEE, 2003.

[13] S. Funk and S. Baruah, “Restricting EDF migration on uniform heteroteneous multiprocessors,” *Technique et Science Informatiques*, vol. 24, no. 8, pp. 917–938, 2005.

[14] S. Funk and S. Baruah, “Task assignment on uniform heterogeneous multiprocessors,” in *17th Euromicro Conference on Real-Time Systems (ECRTS’05)*, pp. 219–226, IEEE, 2005.

[15] K. Yang and J. H. Anderson, “On the soft real-time optimality of global edf on multiprocessors: From identical to uniform heterogeneous,” in *2015 IEEE 21st International Conference on Embedded and Real-Time Computing Systems and Applications*, pp. 1–10, IEEE, 2015.

[16] K. Yang and J. H. Anderson, “On the soft real-time optimality of global edf on uniform multiprocessors,” in *2017 IEEE Real-Time Systems Symposium (RTSS)*, pp. 319–330, IEEE, 2017.

[17] K. Yang and J. H. Anderson, “Soft real-time semi-partitioned scheduling with restricted migrations on uniform heterogeneous multiprocessors,” in *Proceedings of the 22nd International Conference on Real-Time Networks and Systems*, pp. 215–224, 2014.

[18] K. Yang and J. H. Anderson, “An optimal semi-partitioned scheduler for uniform heterogeneous multiprocessors,” in *2015 27th Euromicro Conference on Real-Time Systems*, pp. 199–210, IEEE, 2015.

[19] J. P. Erickson and J. H. Anderson, “Response time bounds for g-edf without intra-task precedence constraints,” in *International Conference On Principles Of Distributed Systems*, pp. 128–142, Springer, 2011.

[20] K. Yang and J. H. Anderson, “Optimal gedf-based schedulers that allow intra-task parallelism on heterogeneous multiprocessors,” in *2014 IEEE 12th Symposium on Embedded Systems for Real-time Multimedia (ESTIMedia)*, pp. 30–39, IEEE, 2014.

[21] G. A. Elliott, K. Yang, and J. H. Anderson, “Supporting real-time computer vision workloads using openvx on multicore+ gpu platforms,” in *2015 IEEE Real-Time Systems Symposium*, pp. 273–284, IEEE, 2015.

[22] K. Yang, G. A. Elliott, and J. H. Anderson, “Analysis for supporting real-time computer vision workloads using openvx on multicore+ gpu platforms,” in *Proceedings of the 23rd International Conference on Real Time and Networks Systems*, pp. 77–86, 2015.

[23] K. Yang, M. Yang, and J. H. Anderson, “Reducing response-time bounds for dag-based task systems on heterogeneous multicore platforms,” in *Proceedings of the 24th International Conference on Real-Time Networks and Systems*, pp. 349–358, 2016.

[24] S. Baruah, “Techniques for multiprocessor global schedulability analysis,” in *28th IEEE International Real-Time Systems Symposium (RTSS 2007)*, pp. 119–128, IEEE, 2007.

[25] M. Bertogna and M. Cirinei, “Response-time analysis for globally scheduled symmetric multiprocessor platforms,” in *28th IEEE International Real-Time Systems Symposium (RTSS 2007)*, pp. 149–160, IEEE, 2007.

[26] B. Andersson, K. Bletsas, and S. Baruah, “Scheduling arbitrary-deadline sporadic task systems on multiprocessors,” in *2008 Real-Time Systems Symposium*, pp. 385–394, IEEE, 2008.

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

[28] R. I. Davis and A. Burns, “Priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems,” in *2009 30th IEEE Real-Time Systems Symposium*, pp. 398–409, IEEE, 2009.

[29] T. P. Baker, “Multiprocessor edf and deadline monotonic schedulability analysis,” in *RTSS 2003. 24th IEEE Real-Time Systems Symposium, 2003*, pp. 120–129, IEEE, 2003.

[30] T. P. Baker, “An analysis of edf schedulability on a multiprocessor,” *IEEE transactions on parallel and distributed systems*, vol. 16, no. 8, pp. 760–768, 2005.

[31] Gurobi, “Gurobi optimizer.” Website, 2021. `https://www.gurobi.com/`.