Temporal Isolation of Latency-Sensitive Tasks in Real-Time Nested Locking

by

James Robb

Thesis of 60 ECTS credits submitted to the School of Computer Science
at Reykjavík University in partial fulfillment
of the requirements for the degree of
Master of Science (M.Sc.) in Computer Science

December 2020

Examining Committee:

Marcel Kyas, Supervisor
Assistant Professor, Reykjavík University, Iceland

Luca Aceto, Examiner
Professor, Reykjavík University, Iceland

Ýmir Vigfússon, Examiner
Assistant Professor, Emory University, United States

Temporal Isolation of Latency-Sensitive Tasks in Real-Time Nested Locking

James Robb

December 2020

Abstract

Prior work has produced multiprocessor real-time locking protocols that ensure asymptotically optimal bounds on priority inversion, that support fine-grained nesting of critical sections, *or* that are independence-preserving under clustered scheduling. However, while several protocols manage to come with two out of these three desirable features, no protocol to date jointly accomplishes all three. Motivated by this gap in capabilities, this thesis introduces the *Group Independence-Preserving Protocol* (GIPP), the first protocol to guarantee a notion of independence preservation for fine-grained nested locking, support fine-grained nested locking, *and* ensure asymptotically optimal priority-inversion bounds. As a stepping stone, the thesis further presents the *Clustered k-Exclusion Independence-Preserving Protocol* (CKIP), the first asymptotically optimal non-nested independence-preserving k-exclusion lock for clustered scheduling. Both the GIPP and the CKIP rely on allocation inheritance (a.k.a. migratory priority inheritance) as a key mechanism for accomplishing independence preservation.

Preface

This thesis is the result of my research into real-time locking protocols over the last year. It is written to fulfill the graduation requirements of the Master's of Computer Science Program at Reykjavik University (Háskólinn í Reykjavík). The material presented herein is an expanded version of original joint work with Björn Brandenburg [43] that has been published in the proceedings of the 32nd Euromicro Conference on Real-Time Systems (ECRTS 2020). The use of the original work is done with the explicit permission of both parties. In this thesis I will use the first-person singular when presenting the contributions, whereas in the original work the first-person plural is used; this should not be interpreted in any way to deprive the other author of his contributions.

This thesis does not assume that the reader specializes in real-time systems, and thus I present a thorough introduction to the background material required to make its contributions more approachable to a graduate-level computer science audience. The sections of the original work that contribute novel material to the real-time literature are presented largely unchanged in this thesis with the exception of Chapters 5 and 6, where some new contributions are introduced. All other changes to the original work take the form of additional examples and explanations where I felt it would benefit the reader.

Acknowledgements

Conducting the research required to produce this thesis and the corresponding paper [43] has allowed me to grow professionally and personally in ways I could not have predicted beforehand. I am very thankful to have had the opportunity to conduct novel research and contribute to the body of academic literature in computer science.

This thesis would not have been possible without the assistance and guidance of many others. I would like to thank Björn Brandenburg of the Max Planck Institute for Software Systems for the opportunity to study under him, and for all the guidance he provided in learning how to produce impactful research. The unwavering support of the faculty and staff of Reykjavik University through both my undergraduate and graduate studies has been pivotal to my success, and I am very thankful. I would like to thank my thesis advisor Marcel Kyas for all his help in realizing this thesis and guiding me to its completion. My entrance into graduate studies, along with the encouragement and advice to see it through to the end comes from my mentor Ýmir Vigfússon; I could not be more thankful for everything he has done to help me along the way. Finally, I'd like to express my unending gratitude to my wife Katrín Einarsdóttir. None of this would have been possible without her kindness, love, and support through my five years in post-secondary education. I owe all my success to her.

Contents

List of Figures

List of Tables

List of Abbreviations

AI	Allocation Inheritance
AI-RSM	Allocation Inheritance Resource Satisfaction Mechanism
CA-RNLP	CKIP-AI-RSM - RNLP
C-EDF	Clustered Earliest-Deadline First
CKIP	Clustered k-Exclusion Locking Protocol
DM	Deadline Monotonic
EDF	Earliest Deadline First
FIFO	First-In First-Out
FMLP	Flexibile Multiprocessor Locking Protocol
FP	Fixed Priority
G-EDF	Global Earliest-Deadline First
GIPP	Group Independence-Preserving Protocol
GLPK	GNU Linear Programming Kit
G-RM	Global Rate-Monotonic
HAAP	Highly Asymmetric Access Pattern
JLFP	Job-Level Fixed Priority
JRPD	Job-Release Priority Donation
MBWI	Multiprocessor Bandwidth Inheritance Protocol
MrsP	Multiprocessor Resource Sharing Protocol
OMIP	$\mathcal{O}(m)$ Independence-Preserving Protocol
PCP	Priority Ceiling Protocol
P-EDF	Partitioned Earliest-Deadline First
PIP	Priority Inheritance Protocol
RM	Rate Monotonic
RNLP	Real-Time Nested Locking Protocol
R^2DGLP	Replica-Request Donation Global Locking Protocol
RRPD	Replica-Request Priority Donation
RSM	Resource Satisfaction Mechanism
RTOS	Real-Time Operating System
s-aware analysis	Suspension-Aware Analysis
s-oblivious analysis	Suspension-Oblivious Analysis
s-oblivious pi-blocking	Suspension-Oblivious Priority-Inversion Blocking
UAP	Uniform Access Pattern
WCET	Worst-Case Execution Time

List of Symbols

$\mathbb{N}$	The set of natural numbers.
m	Number of processors in a given system.
n	Number of tasks in a given system.
P_i	Denotes the i^{th} processor.
τ	Set of tasks in the system.
τ_k	Set of tasks in cluster C_k.
T_i	Task i.
T_i(x,y)	Task i with $e_i = x$ and $p_i = y$.
J_i	An arbitrary job of T_i.
$J_{i,k}$	The k^{th} job of T_i.
$a_{i,k}$	The arrival (release) time of $J_{i,k}$.
$f_{i,k}$	The completion time of $J_{i,k}$.
e_i	Worst-case execution time of T_i.
p_i	Period of T_i - the minimum arrival separation between jobs.
d_i	The relative deadline of T_i.
u_i	Utilization of T_i.
U	Total utilization of all tasks in the system.
BP(J_i, t)	The base priority of J_i at time t.
EP(J_i, t)	The effective priority of J_i at time t.
HEP(J_i, t)	Predicate that is true when J_i is among the c highest effective-priority tasks in $C(T_i)$ at time t.
B_i	The fixed priority of J_i.
R_i	The maximum response time of T_i.
c	Number of processors in a cluster.
C_k	Cluster k.
$C(T_i)$	T_i's home cluster.
q	Number of shared resources in Γ.
Γ	Set of shared resources in the system.
Γ'	Set of currently-held shared resources in the system.
ℓ_a	Shared resource a.
γ_i	The set of shared resources accessed by T_i.
$\mathcal{R}$	A resource request.
$N_{i,a}$	The maximum number of requests J_i makes for the shared resource ℓ_a.
N_i	The maximum number of shared resource requsts J_i.
$\succ$	Denotes the relation for the partial ordering on the shared resources in Γ.
$L_{i,a}$	Length of T_i's longest outermost critical section that begins with an outermost request for ℓ_a.

$L_i^{\max}$	Length of T_i's longest outermost critical section.
$L^{\max}$	Length of longest outermost critical section among all tasks in τ.
$b_{i,a}$	The maximum s-oblivious pi-blocking any job of T_i incurs due to requests for ℓ_a.
b_i	The maximum s-oblivious pi-blocking any job of T_i incurs.
GQ_a	The OMIP's global FIFO queue for ℓ_a.
$\mathrm{FQ}_{a,k}$	The OMIP's local FIFO queue for ℓ_a in C_k.
$\mathrm{PQ}_{a,k}$	The OMIP's local priority queue for ℓ_a in C_k.
$ts(J_i)$	The timestamp recorded when J_i acquires a token to compete for shared resources.
RQ_a	The RNLP's RSM priority queue for ℓ_a.
$hd(a)$	The job at the head of RQ_a while competing for shared resources under the rules of an RSM.
$[\ell_a]^{ol}$	The set of resources ℓ_a depends on with respect to outer-lock independence preservation.
D_i^{ol}	The set of resources T_i depends on with respect to outer-lock independence preservation.
$\circ$	Symmetric binary relation on shared resources.
$\sim$	The transitive closure of $\circ$.
$g(\ell_a)$	The set of resources ℓ_a is associated with.
D_i	The set of resources T_i is associated with.
G	The set of all resource groups in the system.
g_i	Group i.
r	Number of resource groups (under group independence preservation).
KQ_a	The CKIP's FIFO queue for replicas of ℓ_a.
$sr(J_i, t)$	The set of resources that J_i holds at time t.
$A_{i,a,t}$	The set of jobs that can prevent J_i from acquiring ℓ_a in the AI-RSM at time t.
λ_x	Token for group x.
Λ	Set of the GIPP's replicated group tokens.
θ_x^i	Upper-bound on the maximum number of jobs of T_x that overlap with J_i.
$O_{x,y}$	The y^{th} outermost critical section of T_x.
$L_{x,y}^O$	The length (in the absence of blocking of suspensions) of the y^{th} outermost critical section of T_x.
$S_{x,y}$	The set of shared resouces accessed by $O_{x,y}$.
$O_x(g)$	The set of outermost critical sections of T_x that pertain to resources in group g.
$X_{x,y,v}^R$	The blocking fraction for the RSM blocking J_i incurs due to the v^{th} overlapping instance of $O_{x,y}$.
$X_{x,y,v}^T$	The blocking fraction for the token blocking J_i incurs due to the v^{th} overlapping instance of $O_{x,y}$.
τ_k'	Set of tasks in cluster C_k except T_i (*i.e.*, $\tau_k \setminus \{T_i\}$).
$\phi_{i,g}$	The number of times J_i issues an outermost request for a resource in group g.
$\beta_{k,g}$	The number of tasks in C_k that request a resource in group g.
$W_{i,g}$	Upper-bound on the number of times J_i must wait for a token of group g.

$F_i(s)$	The number of outermost critical sections of T_i which need resources that the RSM may have to withhold due to other jobs holding resources in the set of shared resources s.
$S^i(g)$	The set of all combinations of resources in group g acquired by tasks other than T_i.
$symr(T_i, T_k)$	The request symmetry ratio of T_i and T_k.

Chapter 1

Introduction

A real-time locking protocols aims to arbitrate mutually-exclusive access to shared resources (*e.g.*, network interface cards, shared memory, *etc.*) such that the time required to acquire a shared resource has provably sound upper-bounds. In contrast, traditional or "non-real-time" locking protocols work on the assumption that requests for shared resources will be satisfied "quickly enough", and guarantees are expressed in terms of "fairness" and bounded-overtaking.

From a practical point of view, any effective multiprocessor real-time locking protocol should inarguably avoid some obvious pitfalls by satisfying the following requirements.

REQ1 Non-conflicting accesses to different resources should *not* be needlessly serialized.

REQ2 Tasks should *not* be delayed due to contention for resources they do not access.

REQ3 A real-time locking protocol should *not* make it impossible to provision latency-sensitive tasks that are carefully designed to not require any shared resources (such as critical interrupt handlers with stringent sub-millisecond deadlines).

REQ4 Worst-case blocking should *not* be exponential in the number of processors, number of tasks, nor number of held resources.

It is not difficult to see how a protocol that fails to meet these requirements would result in costly and inefficient over-provisioning. It may thus come as a surprise that *no multiprocessor real-time locking protocol in the published literature satisfies all four properties!*

The reason, however, is all the more understandable: these innocuous-looking requirements translate to well-known real-time locking protocol properties that are difficult to ensure by themselves, let alone *jointly* in a single protocol. In particular, **REQ3** rules out any locking protocol that relies on the non-preemptive execution of critical sections, a trait of virtually all spin-lock protocols [11]. Furthermore, **REQ1** implies that a protocol must support *fine-grained nested locking* [7, 48, 53]—that is, tasks must be able to incrementally lock additional resources while already holding some other shared resources—because the alternative, namely coarse-grained *group locking* [8], serializes even trivially non-conflicting requests for resources in the same group. Fine-grained nested real-time locking, however, is a notoriously difficult problem [7, 11, 48], and easily gives rise to blocking bounds that are exponential in the number of simultaneously acquired resources [11, 26, 48]—it is a fundamental algorithmic challenge to ensure both **REQ1** and **REQ4** in a single protocol. The only known protocol to surmount this challenge is Ward and Anderson's *Real-Time Nested Locking Protocol* (RNLP) [51, 53]. In fact, the RNLP famously solves the problem while ensuring *asymptotically optimal* bounds on *priority inversion blocking* (pi-blocking) [14, 53].

The RNLP, in turn, does not satisfy **REQ2**. Specifically, as is discussed in more detail in Section 2.6.1, the RNLP relies on a *token lock* that regulates contention for shared resources, an ingenious element of the RNLP's design that ensures its asymptotic optimality. However, in its configuration for suspension-based locking (under "suspension-oblivious analysis," see Section 2.4.2), this token lock becomes a global bottleneck that causes tasks to delay each other even if they do not share any resources.

To satisfy **REQ2** and **REQ3**, a locking protocol must temporally isolate tasks from each other when they do not access the same resources, which is known as *independence preservation* [9], a concept I discuss in detail in Section 2.5. The only protocol to date to realize independence preservation for clustered scheduling is the $\mathcal{O}(m)$ *Independence-Preserving Locking Protocol* (OMIP) [9]. However, the OMIP fails to satisfy **REQ1** as it can realize nested locking only through group locks—and if fine-grained locking is permitted under the OMIP, it fails to satisfy Requirement **REQ4** due to its FIFO (First-In First-Out) queuing structure, which gives rise to exponential worst-case blocking [48].

Seemingly, the satisfaction of one of the four requirements comes at the cost of another. Is this a fundamental limitation? Is it perhaps *impossible* to satisfy all four requirements at once? As I show in this thesis, the answer to these questions is *no*—it is in fact possible to combine fine-grained nesting, independence preservation, and asymptotically optimal pi-blocking in a single protocol, which I demonstrate by constructing the first such protocol.

1.1 Related Work

The *Priority Inheritance Protocol* (PIP) [24, 42, 44] provides independence preservation, but only on uniprocessor or globally-scheduled systems, and the multiprocessor variant [24, 56] does not support nested critical sections. The *Flexible Multiprocessor Locking Protocol* (FMLP) [8] likewise is independence-preserving only under global scheduling, and only supports group locks [8, 56]. The *Multiprocessor Bandwidth Inheritance Protocol* (MBWI) [26, 27] and the *Multiprocessor Resource Sharing Protocol* (MrsP) [19] both allow for fine-grained nested locking. Unfortunately, they are subject to the exponential blow-up in blocking times described by Takada and Sakamura [48]. Several variants of the RNLP [51, 53] have been introduced in recent years to enable reader-writer synchronization [52], to provide contention-sensitive pi-blocking bounds [32], and to reduce implementation overheads in the locking protocol itself by means of a fast path [39] and lock servers [40]. However, none of these variants removes the *algorithmic* bottleneck of a single, shared token lock. For further discussion of the larger area of multiprocessor real-time locking protocols, I refer the interested reader to a recent comprehensive survey [11].

1.2 Contributions

The contributions are as follows. First, I examine what it means to be independence-preserving in the presence of nested locking (Chapter 3), and the ensuing implications on asymptotic pi-blocking bounds (Section 3.1). The main contribution is the *Group Independence-Preserving Protocol* (GIPP), the first asymptotically optimal, independence-preserving, real-time fine-grained nested locking protocol for clustered scheduling under suspension-oblivious analysis (Chapter 4). In other words, the GIPP is the first multiprocessor real-time locking protocol that meets all of the desirable requirements **REQ1**–**REQ4**. To realize the GIPP, I develop and analyze a novel *Clustered k-Exclusion Independence-Preserving Protocol* (CKIP), an asymptotically optimal independence-preserving k-exclusion lock for clustered scheduling (Section 4.1). Lastly, I provide a fine-grained pi-blocking analysis of the GIPP using a state-of-the-art blocking analysis method based on linear programming (Chapter 5), and present an empirical evaluation that shows that the GIPP performs favorably in comparison to both the OMIP and the RNLP across a wide range of workloads (Chapter 6).

Chapter 2

Background

The main contribution of this thesis and the original work [43] is the *Group Independence-Preserving Protocol* (GIPP), the first asymptotically optimal independence-preserving fine-grained nested locking protocol, which is introduced in Chapter 4. The goal of this chapter is to provide the reader with the necessary background on real-time systems and real-time locking to understand the construction of the GIPP and its novel contribution to the literature on real-time locking. The concepts are presented in such an order that the reader benefits from reading the chapter linearly, as most notions build on previously introduced ones.

To begin with I introduce the reader to the *sporadic task model* [5, 38], a widely used model for modeling systems of real-time tasks. I will subsequently use the sporadic task model to introduce the reader to real-time scheduling for uniprocessor systems and multiprocessor systems, and present a brief review on real-time locking as a whole. Finally I will provide a thorough review of the real-time locking protocols relevant to this thesis.

2.1 System Model

In this thesis I use the *sporadic task model* , a relaxation of the *periodic task model* [37], to model systems of hard real-time tasks.

A system is comprised of a set of n tasks $\tau = \{T_1, \ldots, T_n\}$ to be scheduled on m identical processors $P_1, \ldots, P_m$. Each *task* T_i is executed as a series of *jobs*. To build an intuition for this, one can think of a task as an infinite loop where each of its jobs is an iteration of the loop. The k^{th} job of T_i is denoted with $J_{i,k}$ where $k \in \mathbb{N}$. When it is not necessary to refer to a specific job of T_i I will use the common practice of using J_i to refer to an arbitrary job of T_i. Each task is characterized by the following parameters:

- **Arrival Time** $a_{i,k}$ is the time that $J_{i,k}$ first becomes available for execution. Arrival time is also known as *release time* in the absence of *release jitter*. I will use the terms arrival time and release time interchangeably as I do not consider release jitter in this thesis.

- **Completion Time** $f_{i,k}$ is the time that $J_{i,k}$ completes its execution.

- **Worst-Case Execution Time (WCET)** e_i is the upper-bound on the maximum execution time of an arbitrary job of T_i.

- **Period** p_i is the minimum arrival time separation between jobs of T_i.

- **Relative Deadline** d_i is the deadline of J_i relative to its arrival time. For example, if $d_i = 5$ then J_i has five time units to complete its execution relative to its arrival time before a deadline miss occurs.

- **Utilization** $u_i = e_i/p_i$ is the maximum fraction of processor time spent executing T_i.

As the just defined model would suggest, this thesis focuses on the use of *sporadic tasks*. To harness a bit more of an intuition for the term sporadic, consider the following. Real-time tasks can be classified as either *periodic* or *aperiodic*. Jobs of a periodic task are released at a constant rate (*i.e.*, the task's period), whereas jobs of an aperiodic task do not necessarily have a regular rate that they are released at. Coming back to the term sporadic, an aperiodic task where the arrival time between two jobs is *at least* the task's period, but may be more during runtime, is called a sporadic task.

The relation between a task's period and its deadline can also be used to classify tasks. When classifying tasks in this manner there are three ways to do so. A task T_i is an *implicit deadline* task when $p_i = d_i$, a *constrained deadline* task when $p_i \leq d_i$, and a *arbitrary deadline* task otherwise. The deadline types of a task set will determine how the task set is analyzed, scheduled, and what guarantees can be made about it. In this thesis I assume tasks have implicit deadlines, though the derived results do not depend on this assumption. When discussing implicit deadline tasks I will use the notation $T_i(x, y)$ to denote a task T_i with $e_i = x$ and $p_i = d_i = y$.

Deriving WCETs is most often not a trivial endeavor, as the WCET of a task will depend on a number of factors like the underlying system architecture, choice of programming language, and choice of operating system. In practice, WCETs are empirically derived, though tools to aid in determining WCETs are available. For example, static analysis tools like the aiT WCET Analyzer [2] have been developed to compute these values offline. However, the use of such tools are out of the scope of this thesis, and I assume that the WCET of each task is known.

During runtime a job is said to be *pending* from the time it arrives until it completes. While a job is pending, it is in one of two states: a *ready* job can be scheduled (*i.e.*, executing) on a processor, whereas a *suspended* job cannot. I assume that jobs do not self-suspend, and that all suspensions are a result of interactions with a locking protocol; I discuss locking protocols in Section 2.3.

I now provide an example of real-time scheduling using this model in Fig. 2.1, both to provide an intuition for the concepts introduced so far, and to introduce the visual format used throughout this thesis to represent real-time schedules. The example consists of a single task $T_1(3, 5)$ executing on a uniprocessor system (*i.e.*, $m = 1$). The y-axis uses the common notation of denoting the jobs of T_i with J_i, and the x-axis denotes time measured in indivisible units. The following is observed in the schedule:

- 5 jobs of T_1 are released in the time interval $[0, 21)$.

- At times $5, 10, 15, 20$ a double-headed arrow denotes the release time of $J_{1,k}$ and the deadline of $J_{1,k-1}$.

- Each job executes for e_1 time units and completes 2 time units before its deadline.

- P_1 idles in the time interval $[f_{1,k}, a_{1,k+1})$, *i.e.*, the time between the completion of one job and the arrival of the next.

- $J_{1,k}$ is ready between in the time interval $[a_{1,k}, f_{1,k})$.

- T_1's utilization is $u_1 = 3/5 = 0.6$, *i.e.*, T_1 consumes 60% of P_1's capacity.

- None of the jobs suspend; they are all ready during the time intervals they are pending.

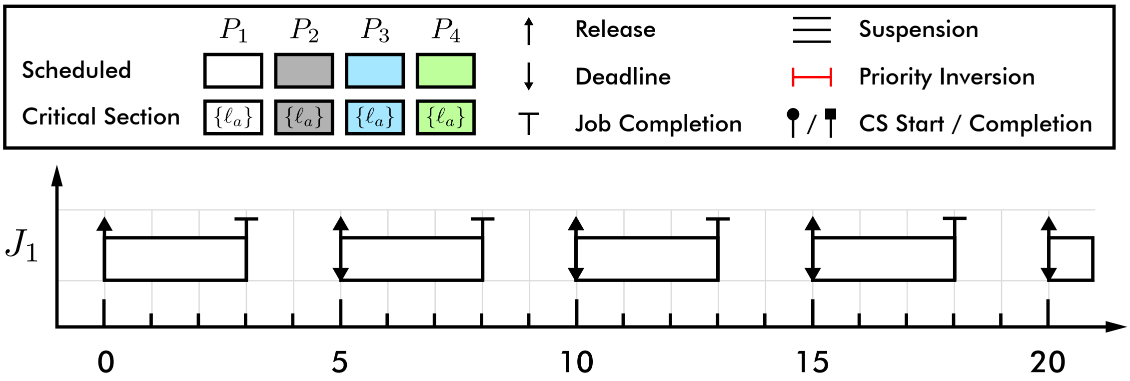


Figure 2.1: A uniprocessor schedule for the task set $\tau = \{T_1\}$ where $p_1 = 5$ and $e_1 = 3$.

Now that I have given a brief overview of the sporadic task model, the necessary background to discuss real-time scheduling has been established. I present a review of real-time scheduling in the following section.

2.2 Real-Time Scheduling

Given a system of m identical processors and n tasks, where $m, n \in \mathbb{N}$, is it possible to schedule the tasks on the processors such that no task misses a deadline? This is one of the primary questions the study of real-time scheduling focuses on. For hard real-time systems, which this thesis focuses on, missing a deadline is equivalent to system failure. The literature on hard real-time scheduling is vast, but in one way or another the goal is to at least answer this question, albeit under a myriad of different assumptions. In this section I present a brief introduction to real-time scheduling. I make the common assumption of an idealized system that has no system overheads (*e.g.*, cache misses, TLB flushes, context switching, *etc.*), and that *at most* one task can be scheduled on a processor at any given time. I use Liu and Layland's notion of a scheduling algorithm [37], but I separate the logic of assigning priorities from the act of using said priorities to assign ready jobs to the processors. The following two definitions reflect this separation of responsibilities.

Definition 2.2.1. A *priority-driven scheduling algorithm* is a set of rules that determine the task(s) to be executed at time t by means of assigning priorities to the tasks in the system.

The priorities assigned to tasks (and therefore their jobs) are assumed to be *unique*, with any ties broken in favor of lower-indexed tasks. A job J_i has both an *effective priority* and a *base priority*. At any point in time t the scheduling algorithm determines J_i's base priority, denoted with $\mathrm{BP}(J_i, t)$. J_i's effective priority, denoted with $\mathrm{EP}(J_i, t)$ may change during its execution due to interactions with a locking protocol. The influence a locking protocol has on a job's effective priority is discussed in Section 2.3, but the concept is introduced here for the sake of consistent notation and terminology. Unless explicitly mentioned otherwise, the reader should assume that $\mathrm{BP}(J_i, t) = \mathrm{EP}(J_i, t)$. Finally, let $\mathrm{HEP}(J_i, t)$ be a predicate that indicates whether J_i is among the m highest effective-priority pending jobs at t.

Definition 2.2.2. A *scheduler* assigns the m highest effective-priority ready jobs at time t to the m processors, *i.e.*, job J_i is scheduled by the scheduler if $\mathrm{HEP}(J_i, t)$.

It is important to make the distinction between a scheduler and a scheduling algorithm; a scheduler *implements* a scheduling algorithm. Scheduling algorithms are high-level algorithmic concepts and do not make assumptions about the capabilities of the underlying system, whereas a scheduler is bounded by the technological capabilities of the system it is developed for. Thus, it can be the case that two schedulers that implement the same scheduling algorithm can vary greatly in design and performance. I assume all schedulers are "perfect" in that they take zero time to execute and are free of any real-world system constraints.

Definition 2.2.3. For a given task set τ and for all t, a *schedule* is an assignment of the tasks in τ to the processor(s) in the system at time t.

Definition 2.2.4. A schedule is *feasible* if all tasks can complete while meeting their constraints (*e.g.*, not missing a deadline).

Definition 2.2.5. A task set τ is *schedulable* if there exists at least one scheduling algorithm that can produce a feasible schedule.

For the sake of simplicity and intuition, I will first focus on the uniprocessor case (*i.e.*, where $m = 1$) before discussing multiprocessor scheduling. In an idealized system, it is intuitively true that a necessary condition for schedulability (*i.e.*, the property of being schedulable) is that the tasks in the system do not require more processor time than is available. The following formalizes this necessary condition.

$$U \leq 1 \tag{2.1}$$

where the total utilization U of a task set is defined as follow:

$$U \triangleq \sum_{i=1}^{n} u_i \tag{2.2}$$

It is evidently true that Eq. (2.1) is a necessary condition for schedulability. Otherwise, it would need to be the case that a processor could schedule at least two tasks at the same time, which contradicts our basic assumption that only one task can be scheduled on a processor a time. However, not all scheduling algorithms can schedule task sets with $U \leq 1$, as we will see shortly.

2.2.1 Fixed-Priority Scheduling

Under *Fixed-Priority* (FP) scheduling the base priorities of tasks are calculated offline and never change during their execution. This is in contrast to scheduling schemes I will discuss later where base priorities can change at runtime. In real-world systems, FP scheduling benefits from simplicity of implementation and low-overheads as the the scheduler never needs to calculate priorities during the execution of the system. In this section I will review the *Rate Monotonic* (RM) scheduling algorithm introduced by Liu and Layland in their seminal paper [37]. They assume the following about a task set τ:

- All tasks are periodic with implicit deadlines.
- Jobs are released at period start.
- Tasks do not self-suspend.
- Tasks are independent (*e.g.*, no resource sharing, *etc.*).
- Tasks have bounded execution time (*i.e.*, WCETs).
- Negligible system overheads.

Under RM scheduling tasks are assigned base priorities offline that do not change. This means that each job of a given task T_i executes with the same base priority. For this reason I use B_i as a short-hand to denote the base priority of a J_i at any point in time. The base priority of a task is inversely proportional to its period. Thus $B_i > B_j \iff p_i < p_j$. For tasks with equal periods, consistent tie-breaking is realized by assigning the task with the lower-index the higher-priority. Fig. 2.2 depicts a RM schedule of the task set $\tau^{\mathrm{RM}} = \{T_1(1,4), T_2(2,5), T_3(1,8)\}$ on $m = 1$ processors. It follows from the definition of τ^{RM} that $B_1 > B_2 > B_3$. As tasks are independent (and therefore do not interact with a locking protocol) their base priority and effective priority never differ.

RM scheduling is an example of a scheduling algorithm where the total utilization check in Eq. (2.1) is necessary, but not sufficient, *i.e.*, a task set with $U \leq 1$ is not guaranteed to be schedulable under RM scheduling; this is important to be aware of given the prevalence of RM scheduling in real-time systems. Liu and Layland introduced the following sufficient condition for the schedulability of periodic implicit-deadline tasks on a uniprocessor system [37].

Theorem 2.2.1 ([37, Theorem 4])**.** A task set that conforms to Liu and Layland's task set assumptions is schedulable under RM scheduling on a single processor if $U \leq n(2^{1/n} - 1)$.

As $n \to \infty$, the bound $n(2^{1/n} - 1)$ approaches $\ln(2)$ (which is approximately 0.69). This leaves a considerable gap between the sufficient condition for RM schedulability, and the necessary condition of $U \leq 1$. An exact schedulability test is required when $\ln(2) < U \leq 1$.

Joseph and Pandya introduced a *necessary and sufficient* schedulability test [33] for constrained deadline task sets (and therefore implicit deadline task sets as well) scheduled under RM scheduling on a uniprocessor system. To perform the test the *maximum response time* of each $T_i \in \tau$ must be calculated. The maximum response

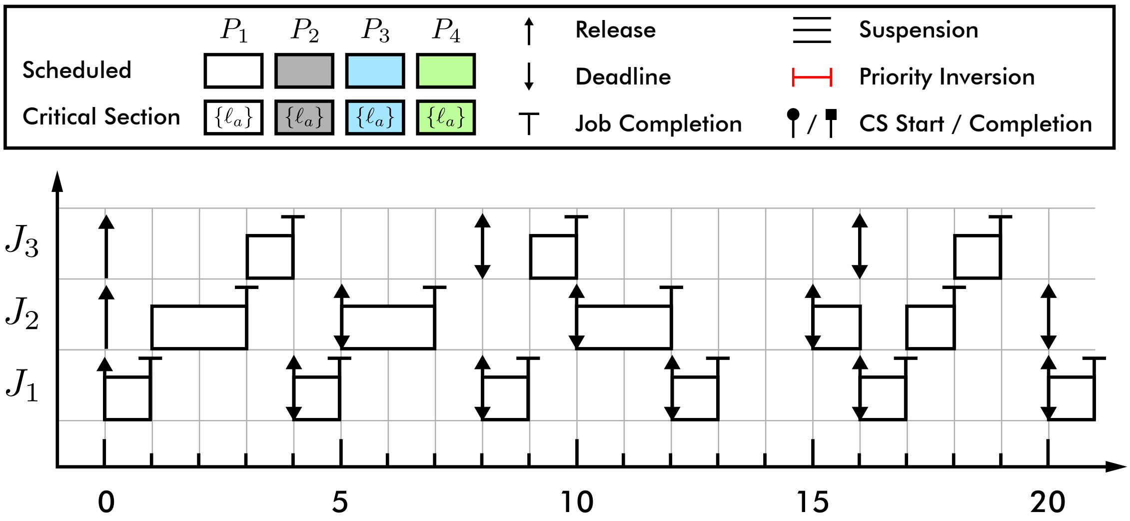


Figure 2.2: A uniprocessor RM schedule of the task set τ^{RM}. At any time t the ready job (if any) with the highest-effective priority is scheduled on P_1.

time of T_i, denoted with R_i, is an upper-bound on the maximum interval in time between the release of a job $J_{i,k}$ and the completion of $J_{i,k}$. Under this test τ is schedulable iff $\forall_{1 \leq i \leq n}\ R_i \leq d_i$. This intuitively holds as a task set that misses no deadlines is clearly schedulable. Unlike the schedulability test in Theorem 2.2.1, this schedulability test does not have a closed-form calculation as the maximum response times of the tasks are calculated via fixed-point iteration. Let $\text{hp}(i) = \{T_j \mid B_j > B_i\}$. The maximum response time of each task is then calculated as follows.

$$R_i^{k+1} = e_i + \sum_{\forall j \in \text{hp}(i)} \left\lceil \frac{R_i^k}{p_j} \right\rceil \cdot e_j \tag{2.3}$$

To provide some intuition for what is happening here, T_i's worst-case response time R_i is increased by the maximum possible time that higher-priority tasks can delay the execution of T_i. This propagation starts with the highest-priority task, as its execution is never delayed by any other tasks. Once the response time values have stabilized (*i.e.*, they no longer change with further iterations), then the worst-case response time for each tasks is known, and the schedulability test can be applied.

As a final note on RM scheduling, it is *optimal* in the sense that if task set that satisfies Liu and Layland's assumptions can be scheduled under FP scheduling, then it can be scheduled under RM scheduling [37].

2.2.2 Job-Level Fixed-Priority Scheduling

Unlike FP scheduling, *Job-Level Fixed-Priority* scheduling (JLFP) scheduling allows for the reassignment of priorities between job boundaries, that is, different jobs of the same task may have different base priorities, but the base priority of a job remains constant from the time it arrives until the time it finishes. JLFP scheduling algorithms can also be classified as *dynamic* scheduling algorithms as scheduling decisions are based on dynamic parameters that can change during runtime. FP scheduling does fall under the class of JLFP schedulers as the base priority of a job never changes, but in this section I will focus on a scheduling algorithm were base priorities do change between job boundaries. Specifically, I will provide a brief introduction to *Earliest Deadline First* (EDF) scheduling.

Under EDF scheduling, the job with the earliest (*i.e.*, nearest) deadline is scheduled at any point in time. Each job $J_{i,k}$ is assigned its base priority upon arrival at $a_{i,k}$, and remains constant until $f_{i,k}$. EDF is then dynamic in the sense that a job's base priority depends on the deadlines of other tasks at runtime. Fig. 2.3 depicts an EDF schedule of the task set $\tau^{\mathrm{EDF}} = \{T_1(1,4), T_2(3,5), T_3(1,8)\}$ on $m = 1$ processors.

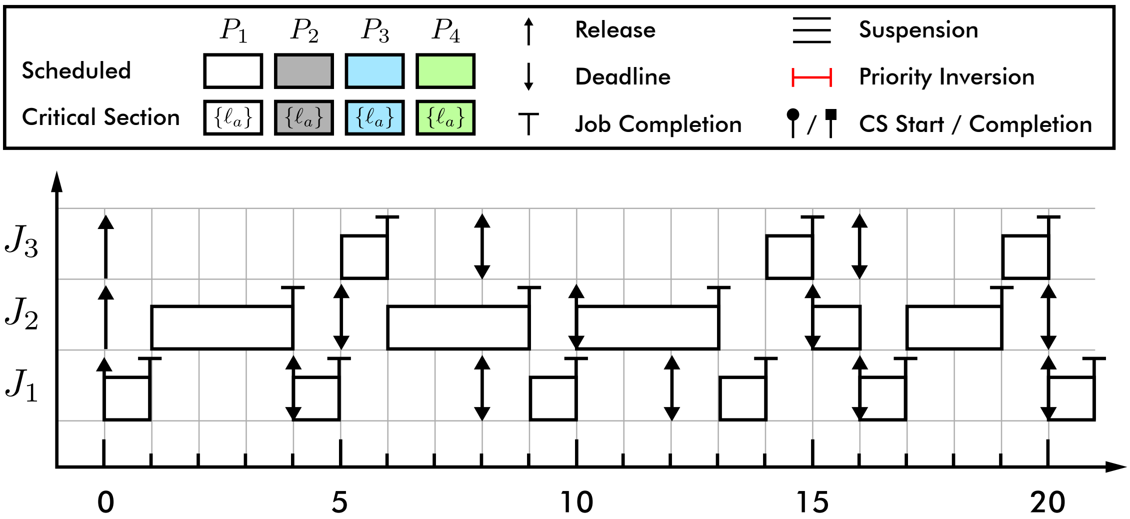


Figure 2.3: A uniprocessor EDF schedule of the task set τ^{EDF}. At any time t the ready job (if any) with the nearest deadline is scheduled on P_1.

On uniprocessor systems, EDF scheduling is optimal with respect to processor utilization. That is, Eq. (2.1) is both a necessary and sufficient condition.

Theorem 2.2.2 ([37, Theorem 7])**.** A sporadic task set that conforms to Liu and Layland's task sets assumptions is schedulable under EDF scheduling on a single processor iff $U \leq 1$.

EDF scheduling benefits from the simple schedulability test in Theorem 2.2.2 and optimality with respect to processor utilization. However there are two drawbacks to consider with real-world systems in mind. Unlike FP scheduling, an EDF scheduler calculates priorities at runtime and can thus incur larger scheduling overheads than a FP scheduler. EDF scheduling also performs poorly when jobs overrun their deadlines as the execution of newly arriving jobs can be pushed further and further out. In contrast, under FP scheduling higher-priority jobs will never be delayed due to the execution of lower-priority jobs (when tasks are independent of each other), but at the possible consequence of completely starving lower-priority jobs of processor time when jobs overrun their deadlines. Which of these scheduling algorithms is "better" will depend on the specific application.

This concludes my review on uniprocessor real-time scheduling. The next section builds upon the concepts introduced so far and gives a brief introduction to real-time scheduling in multiprocessor systems.

2.2.3 Multiprocessor Real-Time Scheduling

Multiprocessors (*e.g.*, multi-core processors) are ubiquitous today due to availability, cost-efficiency, increasing demand for processing power [1], and the thermal barriers that arise as clock speeds increase [50]. In this section I provide a brief introduction to multiprocessor real-time scheduling. The first place to start is with extending the two scheduling algorithms discussed so far (RM and EDF) to multiprocessor scheduling. I will use the terms *Global EDF* (G-EDF) and *Global RM* (G-RM) when applying EDF and RM scheduling to $m > 1$ processors. Under G-EDF and G-RM scheduling, the m highest-priority ready jobs are scheduled on the m processors at any given time; this is exactly what happens in the uniprocessor case where $m = 1$.

G-EDF and G-RM do not necessarily need to be applied to every processor in the system. In this thesis I assume the m processors are grouped into disjoint subsets of size c called *clusters*. For the sake of simplicity I assume that $m = c \cdot k$ where $k \in \mathbb{N}$; thus, there are $\frac{m}{c}$ clusters. I denote the k^{th} cluster with C_k. Each task T_i is assigned to a single cluster offline called its *home cluster*, which is denoted with $C(T_i)$. Clusters function independently from each other with respect to scheduling; that is, each cluster "behaves" as if it comprised the entire set of processors. For this reason, it important to note that from this point on the predicate $\text{HEP}(J_i, t)$ now reflects whether J_i is among the c highest-effective priority jobs at time t in $C(T_i)$; an obvious generalization on the predicate's original formulation. There are three ways to classify multiprocessor scheduling based on the value of c. The following outlines the three cases and their associated terminology.

- **Global Scheduling** All processors belong to a single cluster (*i.e.*, $c = m$), and the m highest-priority (as determined by the scheduling algorithm) ready jobs are scheduled on the m processors at any point in time.

- **Partitioned Scheduling** Each cluster contains a single processor (*i.e.*, $c = 1$). This means that each cluster can run a uniprocessor scheduling algorithm, as clusters are scheduled independently of each other.

- **Clustered Scheduling** This is the general case where $1 \leq c \leq m$. The tasks in each cluster are scheduled with a global scheduling algorithm applied to the processors in that cluster.

From this point on in this thesis I will focus on clustered scheduling as it is the general case; anything demonstrated for clustered scheduling also applies to global and partitioned scheduling. I primarily assume the use of *Clustered EDF* (C-EDF) scheduling in this thesis; under C-EDF each cluster is scheduled independently with a G-EDF scheduler.

Before proceeding further, one might ask themselves what splitting processors up into clusters accomplishes. In a globally scheduled system a single *run queue* (*i.e.*, queue of jobs to be scheduled) is maintained; this single run queue could become a bottle neck depending on the task set, hardware, and underlying architecture. However, multiprocessor systems come with drawbacks, which I will discuss next.

Intuitively one might think that introducing additional processors to a system would always make the goal of scheduling task sets easier as there would be more processor capacity to do so. For example, under uniprocessor EDF an independent task set with $U \leq 1$ is schedulable, so one might assume an independent task set with $U \leq 4$

would be schedulable under G-EDF scheduling on a system with $m = 4$ processors. This is unfortunately not true. Consider a task set $\tau = \{T_1(3,6), T_2(3,6), T_3(5,7)\}$ with $U \approx 1.71$ scheduled on $m = 2$ processors under G-EDF scheduling; a visual example of this is depicted in Fig. 2.4. J_1 and J_2 are scheduled first as they both have an earlier deadline than J_3. Once J_1 and J_2 complete it is not possible to schedule J_3 such that it does not miss its deadline. This clearly demonstrates the non-optimality of G-EDF w.r.t. to processor utilization, as τ is not schedulable despite $U \leq 2$.

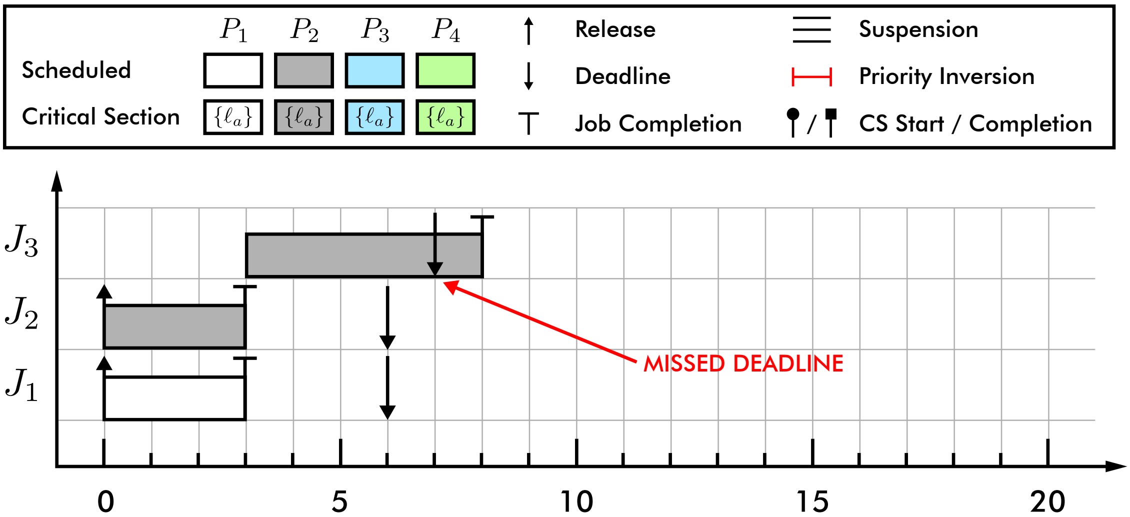


Figure 2.4: A sporadic task set $\tau = \{T_1(3,6), T_2(3,6), T_3(5,7)\}$ with $U \approx 1.71$ scheduled on $m = 2$ processors under G-EDF scheduling. J_3 misses a deadline despite that the total processor capacity is greater than the task set's total utilization.

The example in Fig. 2.4 can be taken further. Dhall and Liu demonstrate that it is possible to construct a task set with total utilization arbitrarily close to 1 that is unschedulable under both G-EDF and G-RM scheduling, irrespective of the number of processors in the system [23].

I review one last consideration one needs to make when scheduling a task set on a multiprocessor system. As mentioned earlier, tasks are statically assigned to clusters offline, but how should these assignments be determined? In the general case this reduces to a bin-packing problem, which is known to be NP-Hard [34]. In practice different heuristics like *First-Fit Decreasing* and *Worst-Fit Decreasing* are used, but a thorough discussion of assigning tasks to clusters is out of the scope of this thesis.

To summarize, multiprocessor systems can offer benefits such as fault tolerance, isolation, and additional processing power. These benefits come with scheduling tradeoffs. Typical uniprocessor scheduling algorithms like EDF and RM lose their respective notions of optimality. Optimal scheduling algorithms do exist for multiprocessor systems, such as PD^2 [47], but they come with the consequence of relatively high complexity and non-trivial overheads [13, 17]. For a more comprehensive review of both uniprocessor and multiprocessor real-time scheduling, I refer the reader to Sha et al.'s (slightly dated) review [45] on the key developments in the history of real-time scheduling, and Davis and Burns's survey [22] on hard real-time scheduling for multiprocessor systems.

2.3 Real-Time Locking

In this section I will provide an overview of real-time locking and the challenges it introduces to real-time scheduling. This section will also review the necessary literature required to realize the main contributions of this thesis.

Any non-trivial system will require tasks to share resources that require mutually-exclusive access. For example, tasks may require mutually-exclusive access to a network interface controller or a shared region of memory. It then follows that tasks will now have their executions delayed as they compete for these shared resources. In order to determine if a task set is still schedulable after accounting for these delays, we must be able to derive a provably sound upper-bound on the maximum time a task waits to obtain mutually-exclusive access to a resource. A *locking protocol* arbitrates requests from tasks for shared resources such that no shared resource is held at the same time by two different tasks. A *lock* is used to protect a shared resource; to hold a shared resource implicitly means to hold the lock that protects the shared resource.

Naturally, when a task requires mutually-exclusive access to a shared resource, it must *wait* until the lock protecting the resource is no longer held by another task. The two most common ways waiting is realized are through *busy-waiting* and *suspension*.

busy-waiting A task T_i that *busy-waits* does not yield processor time to other tasks, but instead loops (or "spins") until some condition that denotes that T_i has acquired the lock is satisfied.

suspension A task T_i that *suspends* explicitly yields the processor to the next task to be scheduled (as determined by the scheduling algorithm). Once T_i acquires the lock, it is no longer suspended and becomes ready for execution.

Locks that are realized through busy-waiting are referred to as *spin locks*, and are commonly seen in real-world systems. One example is in the AUTOSAR real-time operating system standard (RTOS) [4], which mandates the use of spin locks to arbitrate access to shared resources in multiprocessor systems. Spin locks are attractive as they are both easy to analyze and implement; implementation requires very basic hardware and operating system facilities.

Spin locks can be grouped into two broad categories: *preemptive* and *non-preemptive*. Preemptive spin locks allow a task to be preempted by higher-priority tasks while spinning, whereas non-preemptive spin locks do not allow this. Each flavor has its advantages and disadvantages. For example, with non-preemptive spin locks the system benefits from fewer context switches, but at the risk of delaying higher-priority tasks for "too long", whereas preemptive spin locks yield to higher-priority tasks, but can have more system overheads as they will context switch where non-preemptive spin locks would not. The use of busy-waiting precludes the objectives of this thesis; the reason for this will be discussed in Section 2.5. Thus, I will not discuss spin locks any further in this thesis. For the interested reader, Brandenburg provides a thorough overview on spin locks (and real-time locking as a whole) [11].

I focus on the use of binary semaphores—which wait by suspending—to realize locking. I assume the reader is familiar with the use of semaphores and the atomic operations used to modify their values. In the remainder of this section I present the following:

- The notation and terminology used to model shared resources in this thesis.

- The priority inversion problem, resource-related blocking, and their implications on schedulability.
- Progress mechanisms and their properties.
- Nested real-time locking.
- An in-depth review on the three real-time protocols used to build the main contribution of this thesis, the GIPP.

2.3.1 Shared Resource Model

Tasks compete for a set of q serially-reusable shared resources $\Gamma = \{\ell_1, \ldots, \ell_q\}$. Each task T_i accesses a possibly empty subset $\gamma_i \subseteq \Gamma$ of the shared resources in the system.

I say a job J_i *requires* a shared resource ℓ_a at the first instant access to ℓ_a is required for the continued execution of J_i. Once J_i requires ℓ_a, it *issues* a request $\mathcal{R}$ to the locking protocol to *acquire* ℓ_a. $\mathcal{R}$ is said to be *unsatisfied* from the time it is issued until J_i acquires ℓ_a, at which point $\mathcal{R}$ is said to be *satisfied.* $\mathcal{R}$ becomes *complete* when J_i *releases* ℓ_a, which is also when J_i no longer requires ℓ_a. Finally, $\mathcal{R}$ is said to be *incomplete* from the time it is issued until it is completed. Fig. 2.5 provides a visual timeline from when J_i requires ℓ_a to when the associated request is complete. While J_i waits to acquire ℓ_a, it is said to make *progress* if (one of) the job(s) that prevent(s) J_i from acquiring ℓ_a is scheduled. Any method employed by a locking protocol to ensure that a job makes progress is called a *progress mechanism*. I review progress mechanisms in more detail in Section 2.4.3.

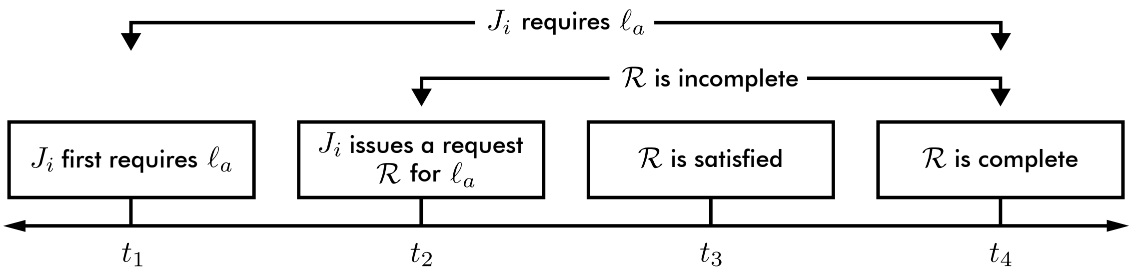


Figure 2.5: J_i requires ℓ_a in the time interval $[t_1, t_4)$ and issues a request $\mathcal{R}$ for ℓ_a at time t_2. $\mathcal{R}$ is incomplete the time interval $[t_2, t_4)$ and unsatisfied in the time interval $[t_2, t_3)$.

If J_i issues a request $\mathcal{R}$ for a shared resource ℓ_a while holding no other shared resources, then $\mathcal{R}$ is said to be an *outermost request.* Conversely, if J_i issues $\mathcal{R}$ for a shared resource ℓ_b while holding ℓ_a, then $\mathcal{R}$ is said to be a *nested request.* Requests do not need to be properly nested; it is possible for J_i to acquire ℓ_a, and then ℓ_b via a nested request, but release ℓ_a before releasing ℓ_b, as seen in Fig. 2.6. I use $N_{i,a}$ to denote the maximum number of (outer and nested) requests J_i issues for ℓ_a, and the total number of resource requests J_i issues is $N_i = \sum_{\ell_a \in \gamma_i} N_{i,a}$.

A strict (irreflexive) partial ordering $\succ$ on Γ is derived from the behavior of the tasks in τ. Let $\ell_a \succ \ell_b$ iff there exists a task that issues a request for ℓ_b while holding ℓ_a. It then follows that workloads where $\ell_a \succ \ell_b$ and $\ell_b \succ \ell_a$ both hold are not permitted.

Consider Fig. 2.6. Let t_1 be the time that J_i issues an outermost request for a shared resource ℓ_a, and t_4 be the next point in time where J_i holds no shared resources. I call the part of J_i's execution in the time interval $[t_1, t_4)$ an *outermost critical section.* I define the *length of a critical section* to be the time required to execute the critical section in the absence of any blocking or suspensions, and use $L_{i,a}$ to denote the length of J_i's longest outermost critical section that begins with an outermost request for ℓ_a. Consequently, it may be the case that $t_4 - t_1 \geq L_{i,a}$, as J_i could experience scheduling or blocking delays during the execution of its outermost critical section. The longest critical section length in J_i's execution is denoted with $L_i^{\max} = \max_a L_{i,a}$ and the longest critical section length among all tasks is $L^{\max} = \max_i L_i^{\max}$.

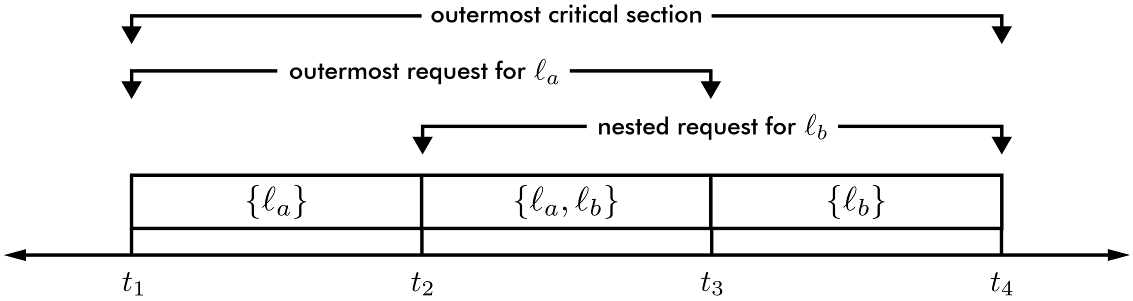


Figure 2.6: An outermost critical section that spans from time t_1 until time t_4. It begins with an outermost request for ℓ_a, which completes at time t_3. The nested request for ℓ_b begins at time t_2 and completes at time t_4. All requests are satisfied immediately in this example.

Up until now all the schedulability tests I have discussed make the assumption that tasks are independent of each other. When shared resources are introduced this is no longer true, and tasks incur blocking due to contention for shared resources. In the following section I discuss how this blocking is measured and defined in real-time locking and scheduling.

2.4 Priority Inversion Blocking

Intuitively, a *priority inversion* is said to occur when the execution of a higher-priority job is delayed due to the execution of a lower-priority job [42, 44]. A typical example of this occurs when a lower-priority job holds a shared resource that a higher-priority job requires, and so the higher-priority job's execution is delayed until the resource is released. I refer to this type of blocking as *priority inversion blocking* (pi-blocking).

Consider the uniprocessor EDF schedule depicted in Fig. 2.7. When J_1 arrives at $t = 5$ it becomes the highest-priority job as its deadline is earlier than J_2's. Even though J_1 is the highest-priority job, it is not scheduled in the time interval $[6, 9)$ as the resource it requires is held by J_2, and thus a priority-inversion occurs.

A common and primary goal in the design of all (sensible) real-time locking protocols is to minimize the pi-blocking that tasks incur while still being able to provide provably sound upper-bounds on said pi-blocking. Without these bounds, it would not be possible to adapt the schedulability tests reviewed in Section 2.2 to account for pi-blocking, nor develop new tests. The following section reviews one of the most notable uniprocessor real-time locking protocols. I choose to return to the uniprocessor case initially to provide both background and intuition before discussing pi-blocking in multiprocessor real-time locking.

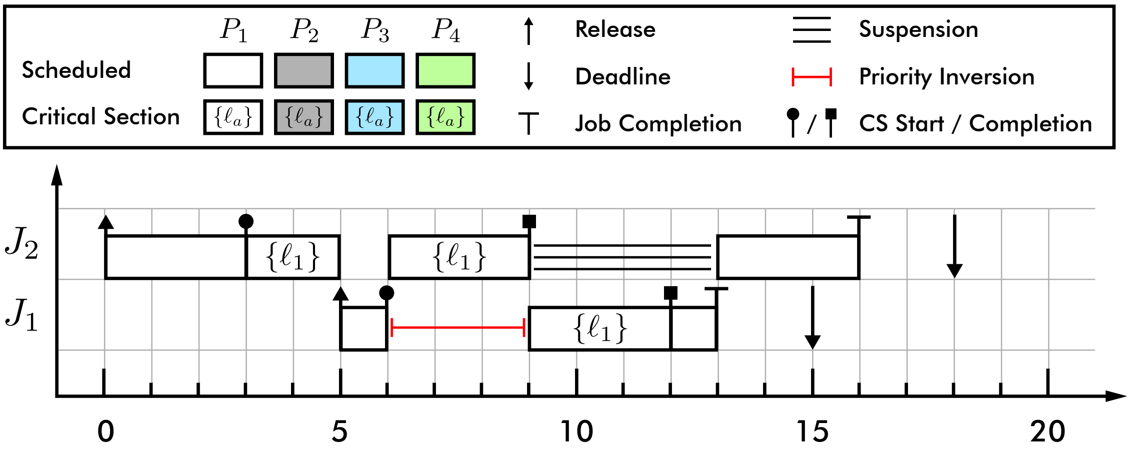


Figure 2.7: A uniprocessor EDF schedule of two jobs. J_1 issues a request at $t = 5$ for the shared resource ℓ_1 which is already held by J_2. Thus, J_1's execution is delayed and a priority inversion occurs in the time interval $[6, 9)$. J_2's request for ℓ_1 is completed at $t = 9$ and is then preempted by J_1 which can now acquire ℓ_1. J_1 continues its execution until $t = 13$, at which point the processor is yielded to J_2.

2.4.1 The Priority Inheritance Protocol

The *Priority Inheritance Protocol* (PIP) [42, 44] is one of the most notable (and earliest) real-time locking protocols. The PIP can be applied to globally scheduled systems, but only the uniprocessor case is presented here. In the following rules for the PIP let J_i be a job that requires a shared resource ℓ_a at time t.

D1 If ℓ_a is not held by another job at t then J_i acquires it.

D2 If ℓ_a is held by another job J_k at t then J_k *inherits* J_i's effective-priority, *i.e.*, J_k executes with J_i's effective priority.

D3 J_i suspends until ℓ_a is no longer held by another job and J_i has sufficient priority to be scheduled again.

D4 If J_k is preempted while executing with J_i's effective-priority by a job J_d that also requires ℓ_a, then J_k inherits J_d's effective-priority. Rule **D3** then applies to J_d.

D5 Once J_k releases ℓ_a it assumes its former effective-priority.

D6 J_i acquires ℓ_a once it is no longer held by another job, and J_i has sufficient priority to be scheduled again.

Rules **D4** and **D6** are worth further discussion. In particular **D4** shows that priority inheritance is transitive; even as higher-priority jobs are released J_k still makes progress. Rule **D6** takes this transitive property into consideration and ensures that J_d (if it exists) will be the next job to acquire ℓ_a. This is important as J_d (if it exists) is by definition the highest-priority job waiting to acquire ℓ_a, and therefore the job that will incur pi-blocking as it waits to acquire ℓ_a.

Fig. 2.7 depicts a uniprocessor RM schedule of three implicit-deadline tasks that compete for a shared resource ℓ_1. The transitive effect of priority inheritance is observed at time $t = 8$ when J_3's effective priority is raised from J_2's effective priority to J_1's effective priority. By Rule **D6** J_1 acquires ℓ_1 at $t = 10$ despite J_2's request for ℓ_1 is older than J_1's request for ℓ_1.

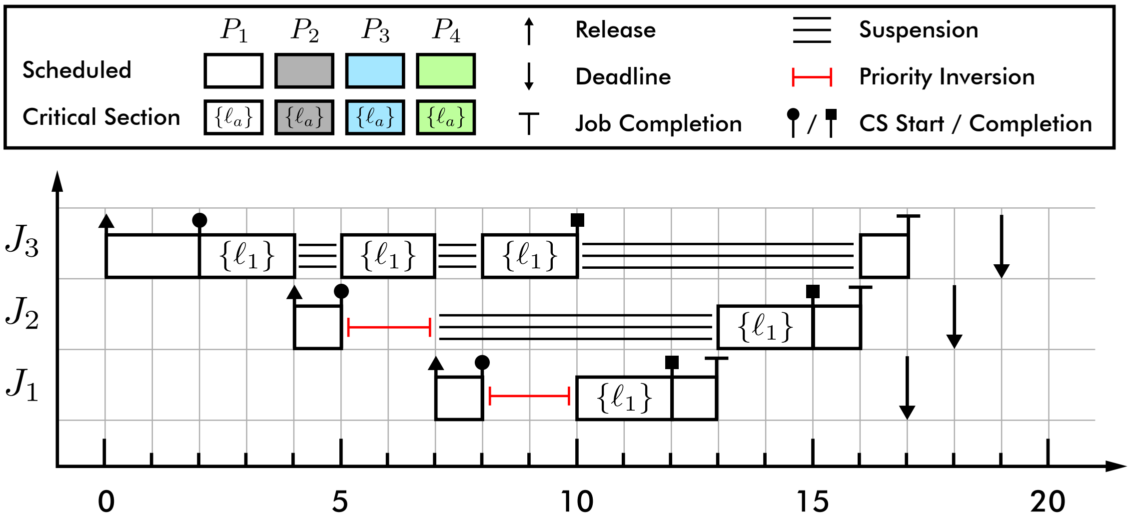


Figure 2.8: A uniprocessor RM schedule of three implicit-deadline jobs that compete for the shared resource ℓ_1 using the PIP.

An appropriate question to ask now is how do the schedulability tests described in Section 2.2.1 and Section 2.2.2 apply when pi-blocking is present—like the schedule shown in Fig. 2.8. The answer is that they no longer directly apply, as the assumption that tasks are independent no longer holds. They must be modified to account for the blocking that tasks now incur. Under the PIP a job J_i can be pi-blocked for at most one critical section of every lower-priority job [42, 44]; let b_i denote this bound. This bound could be reduced by analyzing which resources each task accesses, but such a fine-grained analysis is not relevant to a basic introduction to the PIP. With the blocking bound b_i established, the FP schedulability test from Section 2.2.1 can be adapted as follows [20]:

$$\forall_{1 \le i \le n} \sum_{\forall j \in \mathrm{hp}(i)} \left\lceil \frac{e_j}{p_j} \right\rceil + \left\lceil \frac{e_i + b_i}{p_i} \right\rceil \le i \cdot (2^{1/i} - 1) \tag{2.4}$$

What is intuitively happening here is that the sufficient schedulability condition for RM scheduling in Theorem 2.2.1 is being checked for each task after inflating the task's WCET by its worst-case blocking term. If the condition holds for each task, then the task set is schedulable. In the rest of this thesis the primary focus with regards to schedulability analysis is calculating the blocking terms. The test shown in Eq. (2.4) is simply to aid the reader in developing an understanding for how blocking effects schedulability. The next section examines how to account for priority-inversion blocking in multiprocessor environments.

2.4.2 Analysis Methods for Priority Inversion Blocking

The concept of pi-blocking on uniprocessor systems is quite clear; if the highest-priority job is not executing, then a priority inversion occurs. The idea is the same for multiprocessor systems but now one needs to decide how self-suspensions are accounted for when defining priority inversion. I examine two methods here, as well as their implications on asymptotic upper-bounds on worst-case pi-blocking. The two methods are *suspension-oblivious analysis* (s-oblivious analysis) [14] and *suspension-aware analysis* (s-aware analysis) [14].

Under s-oblivious analysis, a task is assumed to never self-suspend (even though it may due to interactions with a locking protocol), and any self-suspensions are treated as execution time. Conversely, under s-aware analysis, self-suspensions are explicitly accounted for. The definitions for priority inversion under these two analysis methods [14] are as follows, though stated in terms of clustered scheduling as seen in other work [9].

Definition 2.4.1. J_i incurs an *s-oblivious priority inversion* at time t iff J_i is not scheduled and its priority is among the top c priorities of **pending** jobs in cluster $C(T_i)$, *i.e.*, if $\mathrm{HEP}(J_i, t)$.

Definition 2.4.2. J_i incurs an *s-aware priority inversion* at time t iff J_i is not scheduled and its priority is among the top c priorities of **ready** jobs in cluster $C(T_i)$, *i.e.*, if $\mathrm{HEP}(J_i, t)$.

These two methods of analysis yield different lower-bounds on the pi-blocking incurred by a job due to requests for shared resources, which are $\Omega(m)$ and $\Omega(n)$, respectively [14]. For example, the worst-case pi-blocking a job incurs per resource request is upper-bounded by $m \cdot K$, where K is a constant that is typically the length of the longest critical section among all the tasks (*i.e.*, $L^{\max}$). Importantly, these bounds remain the same (*i.e.*, expressed in terms of m instead of c) under clustered scheduling as resources can be shared among tasks in different clusters.

In the rest of this thesis I use $b_{i,a}$ to denote the maximum s-oblivious pi-blocking incurred by J_i due to requests by any task for a shared resource ℓ_a, and b_i to denote J_i's cumulative s-oblivious pi-blocking. The following theorem introduced by Brandenburg and Anderson proves the fundamental lower-bound on worst-case s-oblivious pi-blocking due to resource sharing in multiprocessor systems; I provide intuition for the result after stating the theorem. Before presenting the theorem, I define the parameterized task set used during its proof [14].

- Let $\tau^{seq}(n)$ denote a task set of n identical tasks that share a single resource ℓ_1.
- $\forall_{1 \le i \le n} e_i = 1, p_i = 2n, N_{i,1} = 1, L_{i,1} = 1$.
- The number of tasks and processors are such that $n \ge m \ge 2$.

Theorem 2.4.1 ([14, Lemma 1])**.** There exists an arrival sequence for $\tau^{seq}(n)$ such that, under s-oblivious analysis, $\max_{T_i \in \tau^{seq}(n)}\{b_i\} = \Omega(m)$.

I refer the reader to the original work for the proof. However, to build an intuition for why this is the case, consider Fig. 2.9. Each job is identical, and each of them requires access to ℓ_1. Regardless of the method one chooses to arbitrate access to ℓ_1, one of the jobs will necessarily need to wait for $m-1$ other requests for ℓ_1 to complete, and by Definition 2.4.1 the job incurs s-oblivious pi-blocking while doing so. In this example J_4 is the job that waits for $m-1$ requests for ℓ_1 to complete, and thus incurs $\Omega(m)$ s-oblivious pi-blocking.

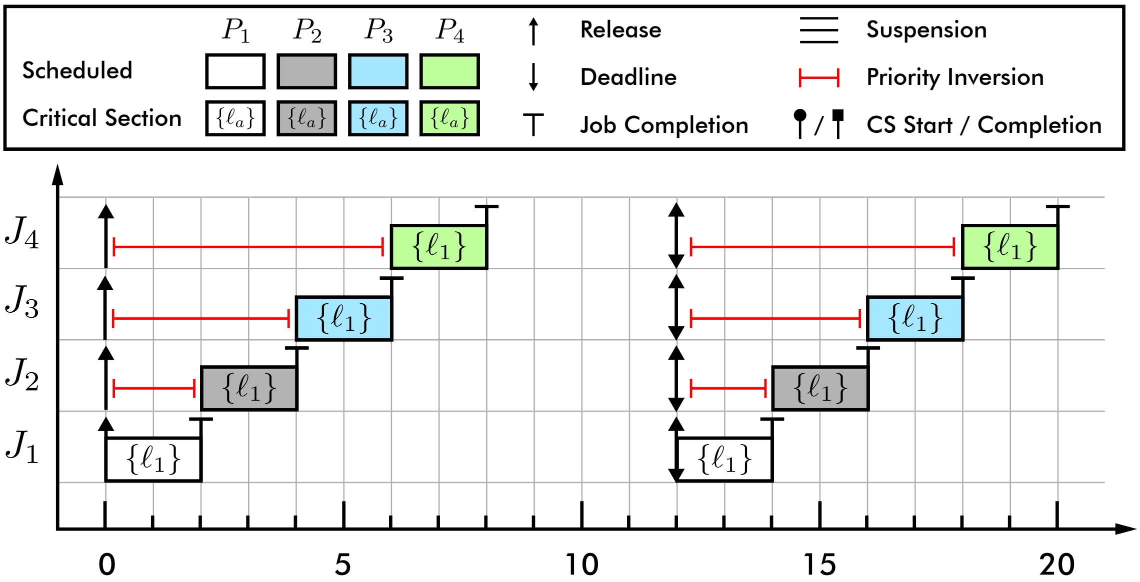


Figure 2.9: A G-EDF schedule (adapted from [14, Figure 2]) of $n = 4$ tasks scheduled on $m = 4$ processors to demonstrate the fundamental lower-bound of $\Omega(m)$ worst-case s-oblivious pi-blocking due to resource sharing in multiprocessor systems.

In the case of s-aware analysis there is an analogous theorem in the original work [14, Lemma 10] that proves the fundamental lower-bound of $\Omega(n)$. A more in-depth review of s-aware analysis is out of the scope of this thesis as I focus exclusively on s-oblivious analysis from this point on. The interested reader should reference the original work.

One of the important properties of the locking protocols discussed and developed in the remainder of this thesis is *optimality*. To conclude this section I will define what it means for a locking protocol to be (asymptotically) optimal with respect to s-oblivious pi-blocking, as well as the notion of *bounded blocking*.

Definition 2.4.3. A locking protocol is *asymptotically optimal* under s-oblivious analysis if the worst-case s-oblivious pi-blocking a job incurs per request is upper-bounded by $\mathcal{O}(m)$.

By Theorem 2.4.1 the worst-case s-oblivious pi-blocking a job incurs per request is lower-bounded by $\Omega(m)$, *i.e.*, it is not possible to realize a lower bound on worst-case s-oblivious pi-blocking in the general case. Thus, if a locking protocols bounds the worst-case s-oblivious pi-blocking a job incurs per request by $\mathcal{O}(m)$, it achieves a tight asymptotic bound $\Theta(m)$ on worst-case s-oblivious pi-blocking. Stated differently, such a protocol is optimal with respect to s-oblivious pi-blocking as its worst-case upper-bound is no worse than the fundamental worst-case lower-bound (within a constant factor).

When deriving asymptotic bounds on b_i, I consider $L^{\max}$ (the maximum critical section length among all tasks) to be a constant (*i.e.*, not a function of m nor n), whereas each e_i is *not* considered to be a constant. Historically, pi-blocking is considered to be *bounded* only if no b_i depends on any e_i [14, 41, 42, 44]; I make this assumption as well. Thus, any locking protocol that provides s-oblivious pi-blocking bounds in terms of some e_i is *not* asymptotically optimal. From a more pragmatic perspective, this assumption is made as WCETs can be orders of magnitude larger than critical sections (*e.g.*, [49]).

Employing an asymptotically optimal locking protocol to arbitrate access to resources allows one to "easily" incorporate provably sound blocking bounds into a schedulability analysis. For example, the protocol I will discuss in Section 2.5.1 has a worst-case upper-bound of $(2m-1) \cdot L^{\max}$ s-oblivious pi-blocking per resource request — if a job J_i issues $N_{i,a}$ requests for a shared resource ℓ_a, one can safely inflate e_i by $N_{i,a} \cdot (2m-1) \cdot L^{\max}$ to account for the blocking J_i incurs due its requests for ℓ_a. While this is a provably safer upper-bound on the s-oblivious pi-blocking J_i incurs, it is clearly overly pessimistic if only one other job requests ℓ_a. These bounds can be tightened with a *fine-grained blocking analysis*. One of the main contributions of this thesis is such an analysis; it is presented in Chapter 5 for the locking protocol constructed in Section 4.3.

2.4.3 Progress Mechanisms

In section Section 2.3.1 I briefly stated that a progress mechanism is "any method employed by a locking protocol to ensure that a job makes progress". With this in mind, I define *progress* as follows.

Definition 2.4.4. Let J_i be a job that incurs s-oblivious pi-blocking at time t (*i.e.*, $\mathrm{HEP}(J_i, t)$ is true) while it waits to acquire a shared resource ℓ_a. J_i makes *progress* if (one of) the job(s) that prevent(s) J_i from acquiring ℓ_a is scheduled.

The motivation for ensuring a job makes progress is to be able to bound the s-oblivious pi-blocking it incurs. In Section 2.4.1 priority inheritance was introduced as a progress mechanism; it ensured that if at any time a job was pi-blocked waiting on a resource, then the resource-holder was scheduled.

The choice of progress mechanism can vary greatly based on the locking protocol used, the goals of the system, the behavior of a given task set, and the underlying system being used. As a simple example, one of the simplest progress mechanisms is *priority boosting*. With priority boosting a job's effective priority is set higher than effective priority that any other job can have and in-effect runs non-preemptively.

The progress mechanisms used in this thesis are introduced as needed — justification for the use of a particular progress mechanism is not discussed unless it is specifically relevant to the contributions of this work. A thorough discussion of progress mechanisms is out of the scope of this thesis; for a more in-depth review of progress mechanisms, I refer the reader to Brandenburg's review on real-time locking protocols [11].

2.5 Independence Preservation

The high-level idea of independence preservation is that tasks are isolated from "unrelated" critical sections. This can be easily pictured for locking protocols that do not permit nested locking; if a task never requests a shared resource ℓ_a, it incurs no pi-blocking as a result of requests by other tasks for ℓ_a.

Prior work introduced the notion of independence preservation [9] among tasks in a system that compete for shared resources where no nested requests are made. For clarity, and to build upon it later, we restate the definition here.

Definition 2.5.1. Let $b_{i,a}$ denote the pi-blocking incurred by J_i due to requests by any task for a shared resource ℓ_a. Under s-oblivious analysis, a locking protocol is *non-nested independence-preserving* iff $N_{i,a} = 0$ implies $b_{i,a} = 0$.

The motivation for independence preservation becomes clear when considering *latency-sensitive* tasks, defined as follows.

Definition 2.5.2. A task T_i is said to be *latency-sensitive* if its *slack*, the difference between its relative deadline and WCET, is less than the length of the longest critical section of any other task T_j, *i.e.*, $d_i - e_i < L_j^{\max}$.

Workloads with latency-sensitive tasks present an interesting scheduling issue. If critical sections are permitted to execute non-preemptively then a latency-sensitive task T_i necessarily misses its deadline if it is not scheduled due to a lower-priority task T_j executing non-preemptively for $L_j^{\max}$ time units. Thus, independence preservation precludes the use of progress mechanisms like priority boosting that have tasks execute non-preemptively either explicitly or in-effect.

2.5.1 The O(m) Independence Preserving Locking Protocol

The *$\mathcal{O}(m)$ Independence-Preserving Protocol* (OMIP) is an asymptotically optimal (w.r.t. s-oblivious pi-blocking) independence-preserving real-time locking protocol [9] for clustered JLFP scheduling. I review this protocol as it is the first independence-preserving real-time locking protocol designed for clustered scheduling. I first define the *allocation inheritance* (AI) progress mechanism [29, 30, 31] (sometimes referred to as *migratory priority inheritance* [9, 16]), and then restate the rules and structure of the OMIP [9] here in full. This serves as a review on independence-preserving locking protocols, and to provide the background necessary to discuss some fundamental limits with locking protocols that are realized with FIFO queues.

Definition 2.5.3. Let J_i be a job that holds a shared resource ℓ_a, and W_i be the set of jobs across all clusters waiting to acquire ℓ_a. Under *allocation inheritance* (AI), if J_i is not scheduled and there exists a job $J_k \in W_i \cup \{J_i\}$ that has sufficient priority to be scheduled in $C(T_k)$, then J_i migrates to $C(T_k)$ (if necessary) and runs with J_k's priority. While J_i executes in $C(T_k)$ with J_k's priority, J_k is called an *allocation donor*. Once J_i releases ℓ_a, it migrates back to $C(T_i)$ (if necessary) and resumes execution when it has sufficient priority. Finally, J_i's allocation donor (if any) ceases to be an allocation donor when J_i releases ℓ_a.

Structure Each shared resource ℓ_a is protected by a global FIFO queue GQ_a of maximum length $\frac{m}{c}$. The job at the head of GQ_a holds ℓ_a. Access to GQ_a is resolved on a per-cluster basis and per-resource basis: in each cluster C_k, there exist another two queues for each ℓ_a; a bounded-length FIFO queue $\mathrm{FQ}_{a,k}$ of maximum length c that feeds into GQ_a, and a priority queue $\mathrm{PQ}_{a,k}$ that feeds into $\mathrm{FQ}_{a,k}$. Requests for each ℓ_a

are satisfied as follows. Let J_i denote a job of a task assigned to C_k. Conceptually, J_i first feeds into its local $\mathrm{PQ}_{a,k}$ and then advances through the queues until it becomes the head of GQ_a. A visual example of this queuing structure is depicted in Fig. 2.10. The rules of the OMIP are as follows.

R1 When J_i issues a request for ℓ_a and $\mathrm{FQ}_{a,k}$ is empty, then J_i is enqueued in both GQ_a and $\mathrm{FQ}_{a,k}$. Otherwise, if there are fewer than c jobs queued in $\mathrm{FQ}_{a,k}$, then J_i is enqueued only in $\mathrm{FQ}_{a,k}$. Finally, if there are already c jobs queued in $\mathrm{FQ}_{a,k}$, then J_i is enqueued in $\mathrm{PQ}_{a,k}$.

R2 J_i's request for ℓ_a is satisfied when it becomes the head of GQ_a. J_i is suspended while it waits (if necessary).

R3 While J_i holds ℓ_a, it benefits from AI (w.r.t. any job waiting to acquire ℓ_a).

R4 When J_i releases ℓ_a, it is dequeued from both GQ_a and $\mathrm{FQ}_{a,k}$. If $\mathrm{PQ}_{a,k}$ is non-empty, then the head of $\mathrm{PQ}_{a,k}$ is transferred to $\mathrm{FQ}_{a,k}$. Further, if $\mathrm{FQ}_{a,k}$ is non-empty, then the new head of $\mathrm{FQ}_{a,k}$ is enqueued in GQ_a. The new head of GQ_a, if any, is resumed.

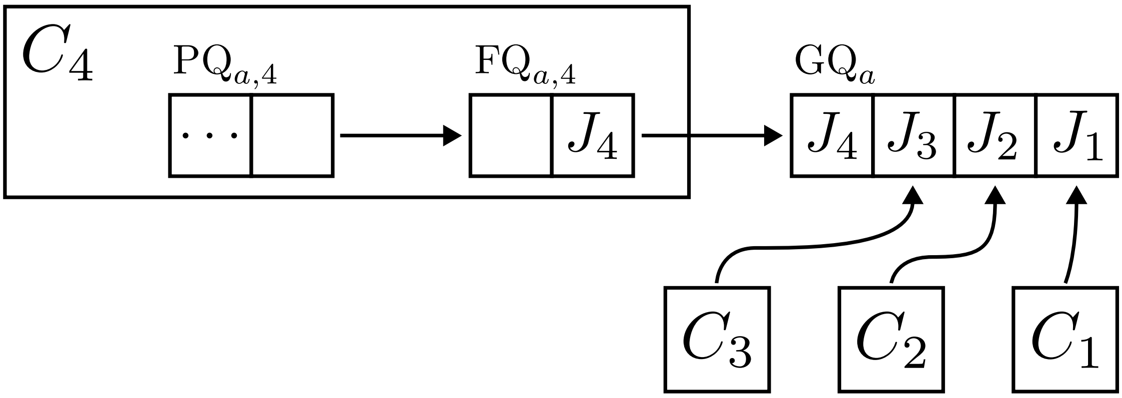


Figure 2.10: A visualization of the OMIP queuing structure for a system with four clusters. J_4 is at the head of $\mathrm{FQ}_{a,4}$ and the tail of GQ_a. Before J_4 acquires ℓ_a (*i.e.*, by becoming the head of both $\mathrm{FQ}_{a,4}$ and GQ_a) it must first wait for one job from each of the clusters C_1, C_2, C_3 to complete their requests for ℓ_a.

I refer the reader to the original work [9] for the full proofs of optimality and independence-preservation. The OMIP solves the problem of providing an independence-preserving locking protocol for clustered JLFP systems, but *only* for non-nested locking. That is, the OMIP does not allow a job to hold more than one resource simultaneously unless group locks are utilized. I discuss group locks and nested locking in the following sections.

2.6 Nested Locking

The way nested locking protocols realize nested locking can be divided into two broad categories: *coarse-grained* locking and *fine-grained* locking. The *Flexible Multiprocessor Locking Protocol* (FMLP) [8] is an example of a real-time locking protocol that employs coarse-grained nested locking. Under the FMLP, resources are split into groups, and each group has a corresponding *group lock*. A job that holds a group lock has mutually-exclusive access to all the resources in the corresponding group. The simplicity of coarse-grained locking comes at the expense of reduced parallelism. If a job J_i accesses only a single resource in a group, it must still acquire the corresponding group lock, which precludes other jobs from accessing the otherwise free resources in the group. For example, consider two shared resources ℓ_a and ℓ_b that are held simultaneously by some job, and so they will be protected by the same group lock. When J_i issues a request ℓ_a, it acquires the group lock, and no other job can access ℓ_b until J_i releases the group lock.

In contrast to coarse-grained locking, fine-grained locking allows shared resources to be acquired incrementally. For example, the *Real-Time Nested Locking Protocol* (RNLP) [53] allows jobs to issue nested requests for resources as they are needed, which provides more opportunities for parallelism when compared to simple group locks. However, the increased potential for parallelism comes at the cost of more complicated protocol rules and data structures when compared to group locks, as group locks can be realized with simpler non-nested locking protocols.

One might consider extending the OMIP to realize the goal of this thesis. The structure of the OMIP, in particular its reliance on FIFO queues, precludes it from realizing asymptotically optimal fine-grained nested locking. Let d be the *maximum nesting level* [48] of a task set τ. The value of d is the maximum number of locks held at any on time by a task $T_i \in \tau$. For example, consider a job J_i and three shared resources ℓ_a, ℓ_b, and ℓ_c. If J_i issues a request for ℓ_b while holding ℓ_a, and then a request for ℓ_c while holding ℓ_a and ℓ_b, then $d = 3$. Takada and Sakamura show that the use of FIFO queues to realize nested locking results in $\mathcal{O}(m^d)$ worst-case pi-blocking [48].

Takada and Sakamura's upper-bound is easily conceptualized. Consider the following scenario under the rules and structure of the OMIP where fine-grained nested resource requests are permitted.

- Assume two FIFO queues FQ_a and FQ_b used to arbitrate access to ℓ_a and ℓ_b, respectively. In order for a job to hold ℓ_a or ℓ_b the job must be the head of the respective FIFO queue.

- Assume J_i is at the head of FQ_a, J_k at the tail of FQ_a, and there are $m-2$ jobs in between them in FQ_a.

- In the worst-case J_i incurred $\mathcal{O}(m)$ s-oblivious pi-blocking while it waited to acquire ℓ_a.

- While J_i holds ℓ_a it issues a request for ℓ_b, and again incurs $\mathcal{O}(m)$ s-oblivious pi-blocking blocking in the worst-case, as there may already be $m-1$ jobs in FQ_b when J_i issues a request for ℓ_b. The maximum nesting level is then $d = 2$ in this scenario.

- Assume now that the $m-2$ jobs between J_i and J_k in FQ_a will also issue a nested request for ℓ_b after acquiring ℓ_a. Each of them incurs $\mathcal{O}(m)$ s-oblivious pi-blocking in the worst-case while waiting to acquire ℓ_b.

- J_k then waits for $m-1$ (*i.e.*, $\mathcal{O}(m)$) requests for ℓ_a to complete before acquiring ℓ_a, and in the worst-case J_k will incur $\mathcal{O}(m)$ s-oblivious pi-blocking while it waits for each those requests to complete. Thus J_k incurs $\mathcal{O}(m) \cdot \mathcal{O}(m) = \mathcal{O}(m^2)$ s-oblivious pi-blocking in the worst-case.

This example helps to demonstrate that the structure of an asymptotically optimal nested locking protocol cannot rely (solely) on FIFO queues to arbitrate access to shared resources. In Section 2.6.1 I review a fined-grained nested locking protocol that addresses this problem, and is pivotal in realizing the results of this thesis.

2.6.1 The Real-Time Nested Locking Protocol

The *Real-Time Nested Locking Protocol* (RNLP) [53] presented a breakthrough in real-time nested locking, as it is the first, and up until now, the only asymptotically optimal fine-grained nested locking protocol for multiprocessor systems; it can be applied under clustered (and therefore global/partitioned) scheduling. Actually, the RNLP is a "meta protocol" in the sense that it defines the properties that a *token lock*, and a *request satisfaction mechanism* (RSM) must obey to realize an optimal fine-grained nested locking protocol. The token lock restricts the number of jobs that can hold resources at any given time, whereas the RSM determines when resource requests among the token holders are satisfied; one key component of how a RSM determines this is by its associated progress mechanism(s). The behavior of a particular *instantiation* (*i.e.*, a token lock/RSM combination) of the RNLP is largely determined by the progress mechanisms that the token lock and RSM employ. Ward and Anderson provide a number of instantiations and their corresponding upper-bounds on worst-case s-oblivious pi-blocking in the original work [53, Table 2]. The life cycle of a request under the RNLP is depicted in Fig. 2.11.

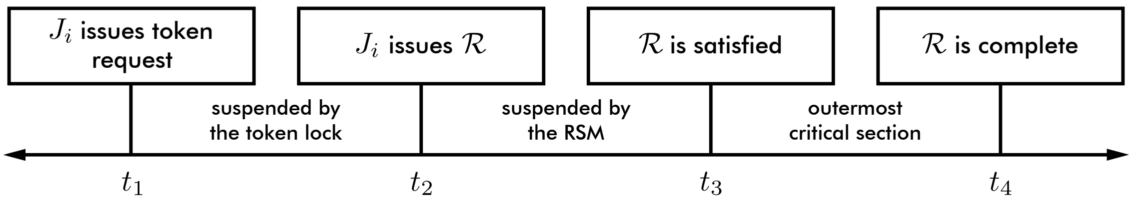


Figure 2.11: The life cycle of a request for a shared resource under the RNLP (adapted from [53]).

The RNLP is a key component in realizing the GIPP. Therefore, I will restate the following from the original work on the RNLP [53] in this section: **(i)** the necessary properties every token lock and RSM must satisfy to realize a valid instantiation of the RNLP, **(ii)** the structure and rules of the RNLP, and **(iii)** the theorem that states the worst-case s-oblivious pi-blocking per outermost request for any valid RSM, which is used in Section 4.3 to prove the optimality of the GIPP.

T1 There are at most m token-holding jobs at any time, of which there are no more than c from each cluster (and thus m across all clusters).

T2 If a job is pi-blocked waiting for a token, then it makes progress.

R1 If a job is pi-blocked by the RSM, then the job makes progress.

Every token lock must satisfy Properties **T1** and **T2**, and every RSM must satisfy Property **R1**.

Structure Jobs first compete for one of m identical tokens under the rules of the token lock. Once a job acquires a token, it can then compete for shared resources under the rules of the RSM. All RSMs share a number of common traits. For each shared resource ℓ_a there is a resource queue RQ_a of length at most m. A timestamp of token acquisition is stored for each job J_i, and denoted $ts(J_i)$. Note that $ts(J_i)$ is actually a function of time as it is updated upon each token acquisition, but the structure and rules focus on a single request, so the time parameter is omitted for simplicity of notation. Each RQ_a is priority-ordered by increasing timestamp. In the absence of any nested locking, the ordering is the same as FIFO ordering. This priority ordering allows a job that issues a nested request to "cut in line" to where it would have been at the time of its outermost request. The job at the head of $\mathrm{RQ}_a$is denoted with $hd(a)$. A visualization of this queuing structure is depicted in Fig. 2.12. The following rules are common to *all* RSMs.

Q1 When J_i acquires a token at time t, its timestamp is recorded: $ts(J_i) := t$. A total order is assumed on all such timestamps.

Q2 All jobs in RQ_a are waiting with the possible exception of $hd(a)$.

Q3 A job J_i acquires resource ℓ_b when it is the head of the RQ_b, *i.e.*, $J_i = hd(b)$, and there is no resource ℓ_a such that $\ell_a \succ \ell_b \wedge ts(hd(a)) < ts(J_i)$.

Q4 When a job J_i issues a request for resource ℓ_a it is enqueued in RQ_a in increasing timestamp order.

Q5 When a job releases resource ℓ_a it is dequeued from RQ_a and the new head of RQ_a (if any) can gain access to ℓ_a, subject to Rule **Q3**.

Q6 When J_i completes its outermost critical section, it releases its token.

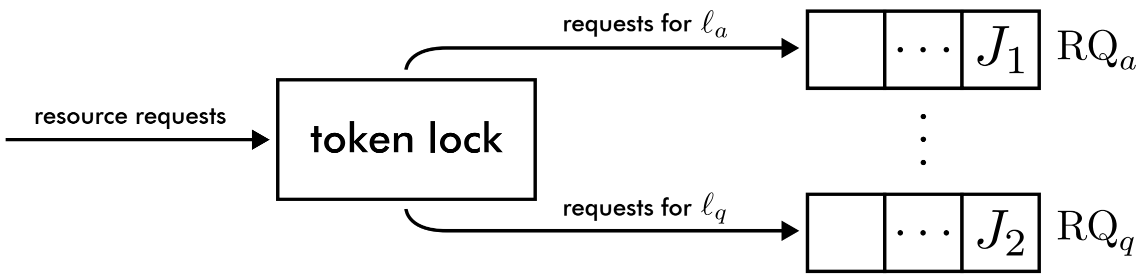


Figure 2.12: Visualization of the RNLP's queuing structure. When a job first requires a resource, it first issues a request for a token under the rules of a chosen token lock. Upon acquiring a token, the job competes for resources under the rules of the RSM.

Theorem 2.6.1 ([53, Theorem 1] Paraphrased)**.** The worst-case s-oblivious pi-blocking in the time interval $[t_2, t_4)$ (see Fig. 2.11) for any RSM is $(m-1)\cdot L^{\max}$.

The key component of the RNLP that allows for asymptotically optimal fine-grained nested locking is the timestamp ordering of requests in the RSM; it prevents the *transitive blocking chain* problem. Consider Fig. 2.13. J_2 issues a request for ℓ_2 at time $t = 2$, but J_2 is suspended by Rule **Q3** as $\ell_1 \succ \ell_2 \wedge ts(hd(1)) < ts(J_2)$ where

$hd(1) = J_1$. At $t = 10$ the just mentioned condition is no longer true and J_2 is able to acquire ℓ_2. However J_3 must still wait at $t = 10$ as $\ell_2 \succ \ell_3 \wedge ts(hd(2)) < ts(J_3)$ where $ts(hd(2)) = J_2$. Notice that J_4 is able to acquire ℓ_1 at $t = 10$, but this does not block J_2's request. While it is the case that $\ell_1 \succ \ell_2$, the condition that $ts(hd(1)) < ts(J_2)$ does not hold where $ts(hd(1)) = J_4$. Without the timestamp condition imposed by Rule **Q3** it could have been the case that J_2 acquired ℓ_2 before J_1. Likewise, J_3 could have acquired ℓ_3 before J_2. Should a pattern like this continue, it would form a long transitive blocking chain, and J_1 would need to wait for the critical sections of all the jobs in said chain to complete before acquiring ℓ_2. Thus, J_i's worst-case s-oblivious pi-blocking would be bounded by the length of such a chain instead of the number of processors.

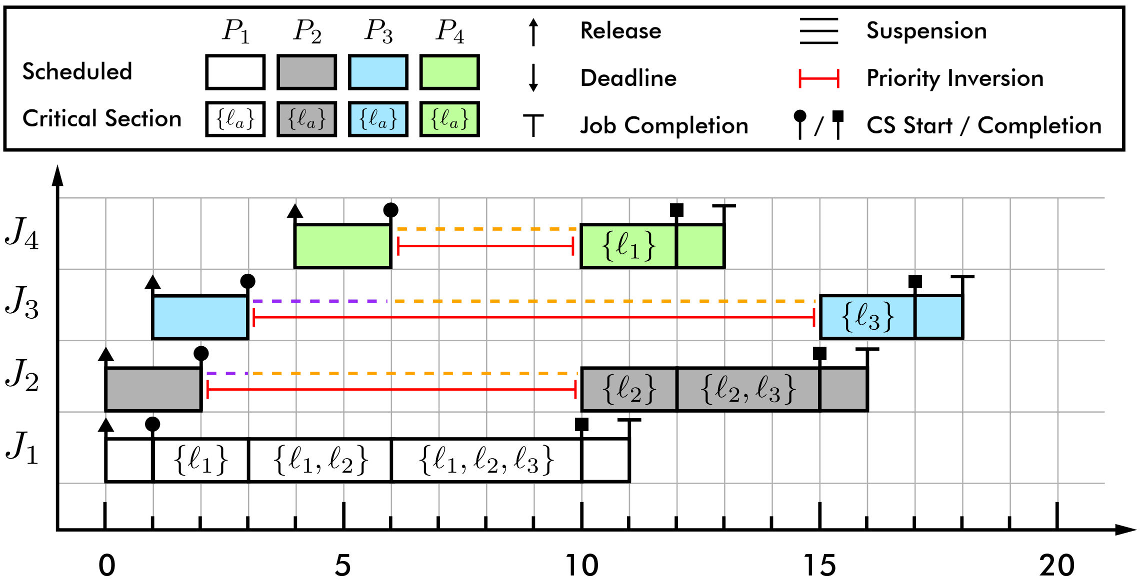


Figure 2.13: An example (adapted from [53, Figure 4]) of $n = 4$ tasks that compete for $q = 3$ shared resources under the RNLP. J_2 and J_3 are suspended by Rule **Q3** in the time intervals $[2, 3)$ and $[3, 6)$, respectively. Rule **Q2** accounts for the remainder of the time that J_2, J_3, and J_4 are suspended. A dashed purple line and a dashed orange line mark the time intervals that jobs are blocked by Rule **Q2** and Rule **Q3**, respectively.

The following section introduces a progress mechanism that will be generalized to clustered scheduling in order to realize the GIPP.

2.6.2 The Replica-Request Donation Global Locking Protocol

The *Replica-Request Donation Global Locking Protocol* R^2DGLP [54] is a k-exclusion lock designed for globally-scheduled systems that is both non-nested independence-preserving and asymptotically optimal under s-oblivious analysis. In short, a k-exclusion lock arbitrates access to k identical shared resources; tokens, for example. The R^2DGLP employs a progress mechanism called *Replica-Request Priority Donation* (RRPD) [54].

RRPD is a modification of the earlier *Job-Release Priority Donation* (JRPD) progress mechanism [15]; I refer the reader to the original work for full details on JRPD. The key difference between RRPD and JRPD is that under RRPD a job J_i donates its priority (*i.e.*, another job inherits J_i's priority) upon requesting a resource, whereas J_i would donate its priority upon release under JRPD. This allows the R^{2}DGLP to realize non-nested independence preservation. JRPD cannot be used to realize non-nested independence preservation, as J_i may donate its priority to a job J_d despite $\gamma_i \cap \gamma_d = \emptyset$ (*i.e.*, J_i and J_d do not access a common resource). Lastly, RRPD relies on the ability to compare priorities among all jobs and therefore only applies to globally-scheduled systems, as analytically speaking, numeric priority values are incomparable across clusters. However, this thesis adapts RRPD for use in clustered scheduling as follows.

I reason about RRPD on a per-cluster basis, *i.e.*, each cluster is treated as an entirely independent globally-scheduled system with respect to RRPD. Therefore, I adapt all rules and notation for clustered scheduling when discussing RRPD. The term *replica* is used when discussing a shared resource under RRPD; each shared resource is assumed to be part of a set of $k \geq 1$ identical replicas. For clarity, $k = 1$ in a system where resources are not replicated. I restate the rules of RRPD [54] here as they are necessary for the GIPP. These rules are stated in the context of a single cluster, and therefore all jobs are implicitly assigned to the same cluster.

In the following rules for the RRPD let J_i be a job that first requires a replica of ℓ_a at time t_1. Let t_2 be the time that J_i issues the corresponding request, t_3 be the time the request is satisfied, and t_4 be the time the request is completed; reference Fig. 2.14 for a visual representation. Finally let J_d be a job that requires ℓ_a and becomes J_i's priority donor at t_x. If necessary, J_d may suspend until it may issue its request for ℓ_a.

D1 J_i may issue a request for a replica of ℓ_a only if it is among the c highest effective-priority jobs that currently require a replica of ℓ_a (including jobs with an incomplete request for a replica of ℓ_a). If necessary, J_i suspends until it may issue its replica request.

D2 J_d becomes J_i's priority donor at time t_x if **(i)** J_d has one of the c highest base-priorities among jobs that currently require a replica of ℓ_a, **(ii)** J_i is the lowest effective-priority job with an incomplete request for a replica of ℓ_a at time t_x, and **(iii)** there are c jobs with an incomplete request for a replica of ℓ_a .

D3 J_i assumes the priority of J_d (if any) during $[t_2, t_4)$. J_d is considered to have no effective priority while it is a donor.

D4 If a job J_d donating its priority to J_i is displaced from the set of the c highest base-priority jobs that require a replica of ℓ_a by a job J_h, then J_h becomes J_i's priority donor and J_d ceases to be a priority donor. (By Rule **D3**, J_i thus assumes J_h's priority.)

D5 A priority donor is suspended throughout the duration of its donation.

D6 J_d ceases to be a priority donor as soon as either **(i)** J_i completes its critical section (*i.e.*, at time t_4), or **(ii)** J_d is relieved by Rule **D4**.

As stated earlier, the R^{2}DGLP is realized through the use of RRPD as a progress mechanism. However, the use of RRPD by itself is not sufficient to guarantee that jobs make progress [54], and so it must be paired with another progress mechanism. In the case of the R^{2}DGLP, the additional progress mechanism used is priority inheritance, but any progress mechanism that satisfies the following property suffices [54].

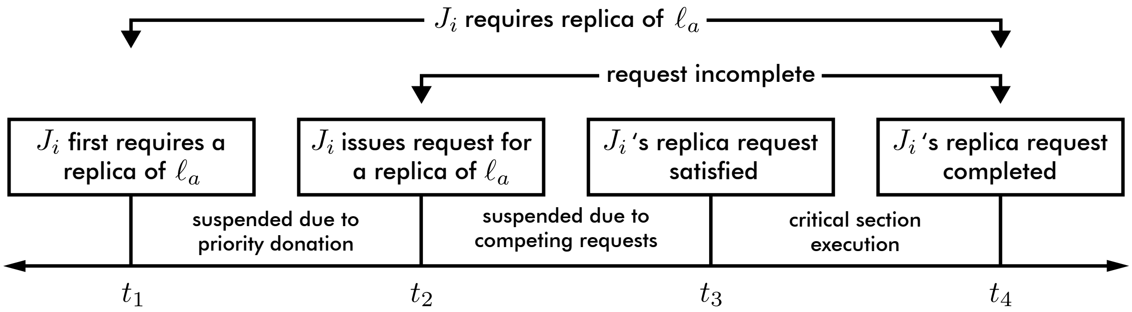


Figure 2.14: Life cycle of a request under RRPD for a replica of a shared resource ℓ_a (adapted from [54]).

P1 A job J_i with an incomplete replica request makes progress (*i.e.*, either J_i is scheduled itself or the replica-holding job that J_i is waiting for is scheduled) if J_i has sufficient priority to be scheduled in $C(T_i)$.

The following three lemmas will complete the necessary background on RRPD I need to introduce for this thesis. The lemmas are required when constructing the main contribution of this thesis.

Lemma 2.6.1 ([54, Lemma 2])**.** There are at most m jobs with an incomplete request for a replica of a shared resource ℓ_a at any time.

Lemma 2.6.2 ([54, Lemma 4])**.** Under RRPD, if a job J_i that requires a replica of ℓ_a is pi-blocked waiting for a replica of ℓ_a it either has an incomplete request for a replica of ℓ_a or it is a priority donor.

Lemma 2.6.3 ([54, Lemma 5])**.** A priority donor J_d can be pi-blocked during priority donation for at most the maximum duration of time that a job can be pi-blocked with an incomplete request for a replica of ℓ_a (refer to timeline in Fig. 2.14), plus one critical section.

To conclude, the key concept of this section is the RRPD progress mechanism. The R^{2}DGLP will be discussed in more detail in Section 4.1 where its structure and rules are used to build a token lock to be paired with the RNLP.

2.7 Summary

In this chapter I have presented the reader with a brief introduction to real-time scheduling and real-time locking. The major points touched on, along with their corresponding sections, are as follows:

- Section 2.1—The sporadic task model was introduced to model systems of real-time tasks.
- Section 2.2—Uniprocessor scheduling algorithms and their schedulability tests. This section also reviewed the difference between FP and JLFP scheduling.
- Section 2.2.3—Multiprocessor scheduling and its properties.

- Section 2.3—An introduction to real-time locking and common lock types.
- Section 2.3.1—The model used for shared resources in this thesis.
- Section 2.4 and Section 2.4.2—The priority inversion problem was introduced as well as two methods for analyzing priority inversion in multiprocessor environments.
- Section 2.5 and Section 2.5.1—Independence preservation and a review of the OMIP, the first non-nested independence-preserving locking protocol for clustered scheduling.
- Section 2.6—An overview of coarse-grained vs fine-grained nested locking, and the necessary background on the protocols and progress mechanisms used to realize the GIPP.

To summarize, I will outline where the protocols discussed so far fall short in terms of realizing the main contribution of this thesis, that is, providing asymptotically optimal fine-grained nested real-time locking with independence preservation.

The use of group locks comes at the loss of parallelism that fine-grained nested locking can offer. If shared resources that will be held at the same time are protected by a group lock, then both the OMIP and R^{2}DGLP can realize asymptotically optimal coarse-grained nested locking through the use of group locks under clustered and global scheduling, respectively. Additionally, if we treat the resources protected by the group lock as a single shared resource, then the two protocols retain the property of being non-nested independence-preserving.

Even in the absence of nested locking, non-nested independence preservation is not a property that is trivially realized with the RNLP. Fundamentally, the use of a token lock that arbitrates access to m tokens (and thus restricts the number of resource-holding jobs to m) precludes non-nested independence preservation. Consider a system with m tokens, each held by a distinct job that needs access to a shared resource ℓ_a. If a job J_i that only accesses ℓ_b issues a request for a token, it must wait for one to become available, which means the pi-blocking it incurs due to requests for ℓ_a is non-zero (*i.e.*, $N_{i,a} = 0$ but $b_{i,a} > 0$).

With the exception of the work this thesis is based on [43], I am not aware of any fine-grained nested locking protocol that is non-nested independence-preserving nor any work that has considered what it means to be independence-preserving in the context of fine-grained nested locking.

In this thesis I extend the notion of independence preservation to nested locking. To this end I create both an asymptotically optimal k-exclusion lock, and an asymptotically optimal independence-preserving fine-grained nested locking protocol for use under clustered scheduling; the CKIP and the GIPP, respectively. Finally, a fine-grained (*i.e.*, non-asymptotic) analysis of the GIPP is introduced and subsequently used to conduct simulated schedulability experiments. The fine-grained analysis also doubles as an analysis for a particular instantiation of the RNLP; to the best of my knowledge, this is the first time a fine-grained analysis has been applied to an instantiation of the RNLP.

Chapter 3

Nested Independence Preservation

The notion of independence preservation introduced in Definition 2.5.1 does not directly apply to nested locking, and there exists more than one way to generalize the notion in a conceptually analogous way, depending on when exactly resources involved in nesting are considered to be "related" (*i.e.*, when they are considered "non-independent"). I consider two possible definitions in the following that I consider to be the most natural way of expressing the idea.

Consider the P-EDF (C-EDF where $c = 1$) schedule of three jobs depicted in Fig. 3.1. For the purpose of this example I assume no particular progress mechanism, and simply have a job suspend until the resource it requires becomes available. Intuitively, some relationship exists between resources ℓ_1 and ℓ_2, since J_2 holds both of them at the same time. How this relationship is defined will determine the properties of a nested independence-preserving locking protocol.

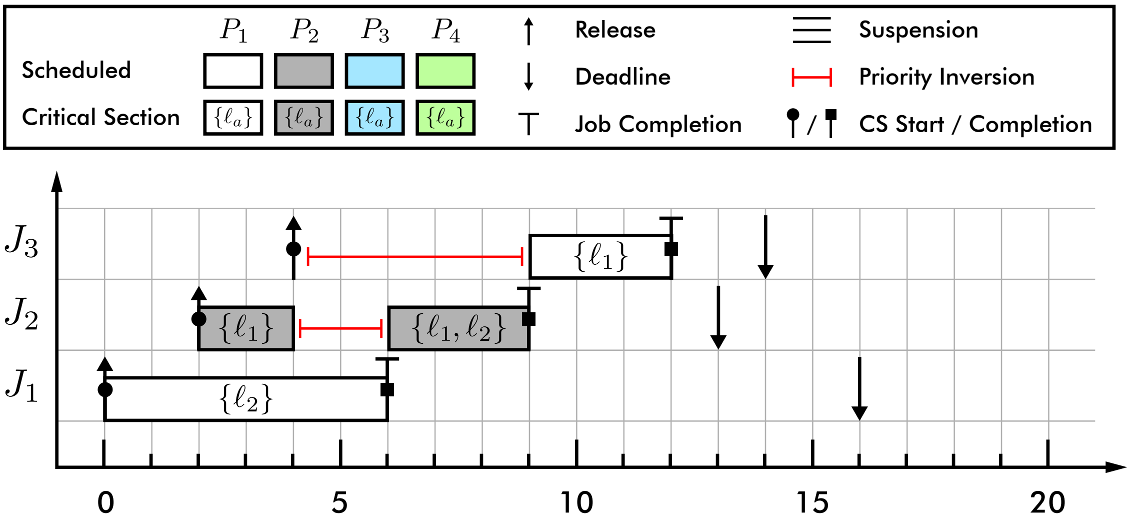


Figure 3.1: P-EDF schedule of three jobs that access two shared resources. J_3 incurs s-oblivious pi-blocking in the time interval $[4, 9)$ as it waits to acquire ℓ_1. During the time interval $[4, 6)$, J_3 is transitively blocked by J_1 as J_1 holds ℓ_2, which J_2 must acquire before it can release ℓ_1.

3.1 Outer-Lock Independence Preservation

The core idea behind *outer-lock independence preservation* is that there is a relation, which I call *dependence*, between a shared resource ℓ_a and shared resources acquired via nested requests (with respect to an outer request for ℓ_a). The following two definitions formalize this idea.

Definition 3.1.1. For a shared resource $\ell_a \in \Gamma$ the set $[\ell_a]^{ol} = \{\ell_x \mid \ell_a \succ \ell_x\} \cup \{\ell_a\}$ is the set of resources ℓ_a *depends* on.

Definition 3.1.2. For a task T_i, the set $D_i^{ol} = \bigcup_{\ell_a \in \gamma_i} [\ell_a]^{ol}$ is the set of shared resources T_i depends on.

Dependence among shared resources can be thought of as a directed acyclic graph (DAG). The vertices represent shared resources, and for any two vertices v_a and v_b, which respectively represent ℓ_a and ℓ_b, there exists an arc between them only if $\ell_a \succ \ell_b$. Then, for any shared resource ℓ_x, it depends on itself, and the shared resources represented by the vertices of which there exists a directed path to. For example, in Fig. 3.2, ℓ_a depends on the set of resources $\{\ell_a, \ell_b, \ell_c, \ell_d\}$, but ℓ_d depends only on itself.

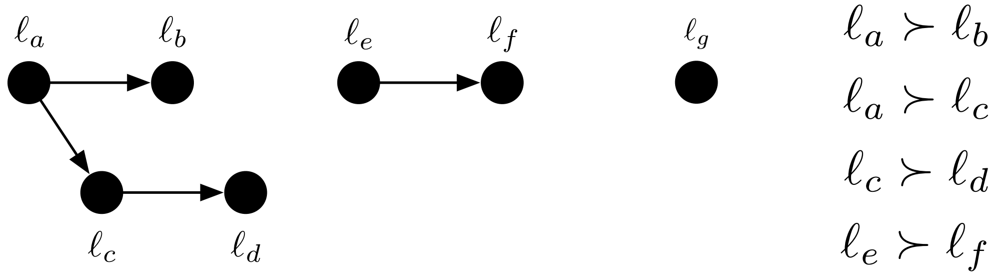


Figure 3.2: Visualization of dependence among shared resources as a directed acyclic graph. The partial ordering used to construct the graph is depicted on the right. Dependence is defined to be both transitive and reflexive (but not symmetric). For example, ℓ_a depends on $\{\ell_b, \ell_c, \ell_d\}$, but ℓ_d only depends on itself.

Based on the precise notion of dependence just introduced, I define outer-lock independence preservation as follows.

Definition 3.1.3. Let $b_{i,a}$ denote the pi-blocking incurred by J_i due to requests by any task for a shared resource ℓ_a. Under s-oblivious analysis, a locking protocol is *outer-lock independence-preserving* iff $\ell_a \notin D_i^{ol}$ implies $b_{i,a} = 0$.

Outer-lock independence preservation as a notion for nested independence preservation has a fundamental impact on the pi-blocking incurred by a job under s-oblivious analysis. In fact, it turns out that for a large class of locking protocols (that arguably includes all possible "reasonable" locking protocols), the per-request pi-blocking bound is necessarily non-optimal (w.r.t. s-oblivious analysis) under RM, *deadline monotonic* (DM) [36], and EDF scheduling. The proof of the non-optimality of outer-lock independence-preserving locking protocols requires the following seemingly obvious property.

Definition 3.1.4. Let $\Gamma' \subseteq \Gamma$ denote the set of shared resources currently held by all tasks in the system. A locking protocol is *non-procrastinating* if any request for a shared resource (by one of the c highest-priority pending jobs in each cluster) is satisfied immediately if $|\Gamma'| = 0$.

I am not aware of *any* real-time locking protocol in the literature that does not satisfy a request $\mathcal{R}$ for a shared resource by one of the c highest-priority pending jobs when $|\Gamma'| = 0$. If non-clairvoyance is assumed, and that tasks are sporadic (*i.e.*, we cannot predict future job arrivals), then delaying the satisfaction of $\mathcal{R}$ is tantamount to willingly wasting CPU time; this is in direct contradiction to one of the most important goals of an effective real-time locking protocol. Non-procrastination is also a fairly weak property as it does not impose restrictions on how to arbitrate access to resources once contention is present.

The following parameterized task set is used in my proof of non-optimality for outer-lock independence preservation.

Definition 3.1.5. Let $\tau^{ol}(n) = \{T_1, \ldots, T_n\}$ be a task set of n tasks that share n resources $\{\ell_1, \ldots, \ell_n\}$, where $n \geq m \geq 2$, with the following properties:

- **(i)** $\ell_1 \succ \ell_2 \succ \ldots \succ \ell_{n-1} \succ \ell_n$,
- **(ii)** $\forall_{1 \leq i \leq n}\ e_i = 4$,
- **(iii)** $\forall_{1 \leq i \leq n}\ p_i = d_i = e_i \cdot n \cdot i$,
- **(iv)** $\forall_{1 \leq i < n}$ jobs of T_i require $\{\ell_i\}$ during the first two units of their execution, and then $\{\ell_i, \ell_{i+1}\}$ during the last two units of their execution,
- **(v)** jobs of T_n require $\{\ell_n\}$ throughout the four units of their execution.

Theorem 3.1.1. There exists an arrival sequence of the task set $\tau^{ol}(n)$ such that $\max_{T_i \in \tau^{ol}(n)} b_i = \Omega(n)$ under s-oblivious analysis for any suspension-based incremental locking protocol that is non-procrastinating and outer-lock independence-preserving, when scheduled under RM, DM, or EDF scheduling (with respect to each cluster).

Proof. Let $a_{i,j}$ denote the first arrival of J_i. Consider the arrival sequence of $\tau^{ol}(n)$ where $a_{i,1} = i-2$ for $2 \leq i \leq n$ and $a_{1,1} = n-1$. An example of $\tau^{ol}(4)$ with this arrival sequence is depicted in Fig. 3.3. At time $t = 0$, J_2 requests and acquires ℓ_2, as we the use of a non-procrastinating locking protocol is assumed. At time $t = 1$, a request for ℓ_3 is made by J_3. If J_3 does not acquire ℓ_3 (and is therefore not scheduled) at $t = 1$, then the blocking that results from delaying J_3's request would result in a violation of outer-lock independence preservation as we would then have $b_{3,2} > 0$ despite $\ell_2 \notin D_3^{ol}$. The same argument analogously applies to all jobs released up until $t = n-1$ when J_1 arrives and issues a request $\mathcal{R}_1$ for ℓ_1. There are then two cases to consider: $\mathcal{R}_1$ is satisfied immediately, or it is satisfied at a later time.

In the first case $\mathcal{R}_1$ is satisfied immediately and J_1 issues a nested request $\mathcal{R}_2$ for ℓ_2 at time $t = n+1$. The maximum number of units of execution completed for jobs $J_2, \ldots, J_n$ up to $t = n+1$ is $(n-1) \cdot 2 + 1 = 2n - 1$ for any $m \geq 2$. This is because jobs $J_2, \ldots, J_{n-1}$ can execute for at most 2 units of time before suspending due to a nested request for an already held resource, and because J_n can execute for at most 3 units of time until $\mathcal{R}_2$ is issued. Therefore, at the time $\mathcal{R}_2$ is issued, there

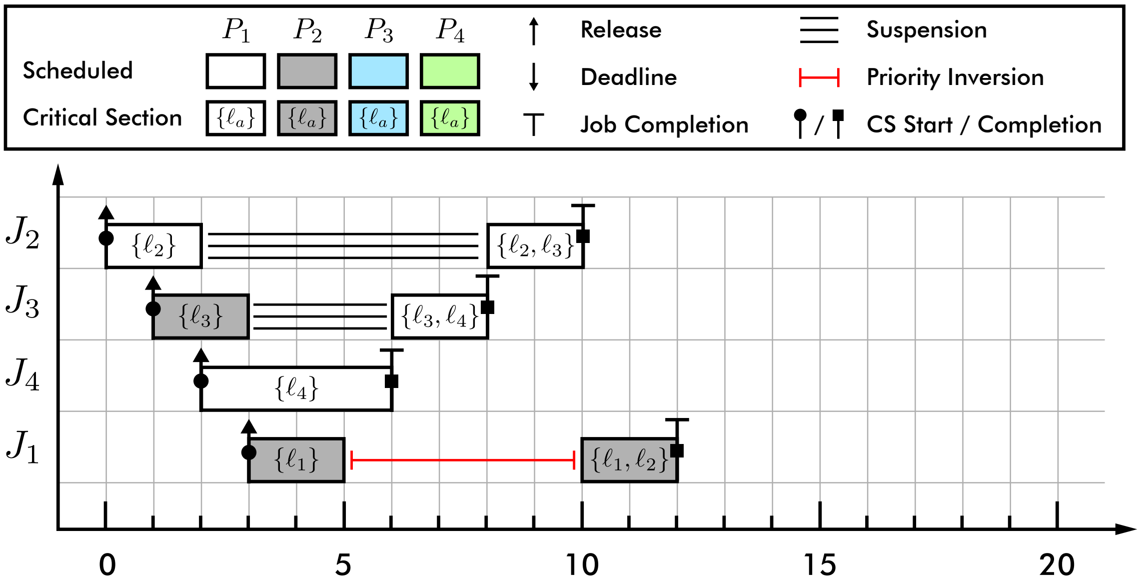


Figure 3.3: One possible G-FP schedule of $\tau^{ol}(4)$ on $m = 2$ processors. The jobs are in ascending priority from top to bottom (*i.e.*, J_2 has the lowest priority, and J_1 has the highest priority).

are $(\sum_{i=2}^{n} e_i) - (2n-1) = (n-1) \cdot 4 - (2n-1) = 2n-3$ units of execution left before jobs $J_2, \ldots, J_n$ complete and ℓ_2 becomes available for acquisition by J_1. Furthermore, as jobs $J_2, \ldots, J_{n-1}$ are all waiting for the job with the next-highest index to release a resource, their executions are serialized. Thus J_1 incurs at the very least $2n-3 = \Omega(n)$ time units of s-oblivious pi-blocking while waiting to acquire ℓ_2.

In the second case, $\mathcal{R}_1$ is assumed to be satisfied satisfied at time $t = n - 1 + \epsilon$ where $\epsilon > 0$ (*i.e.* not immediately). Then, J_1 would begin to incur s-oblivious pi-blocking at $t = n - 1$, as it is the highest-priority job and is not scheduled. This situation cannot result in a reduction of the s-oblivious pi-blocking that J_1 incurs in the first case because **(i)** J_1 is the highest-priority job by construction, and **(ii)** that the serialization of executions described in the first case enforces a minimum of $2n-3$ units of execution before ℓ_2 is released. Therefore, the asymptotic lower-bound in the first case still applies, and J_i still incurs a minimum of $\Omega(n)$ units of s-oblivious pi-blocking while waiting to acquire ℓ_2. $\square$

To conclude, under a standard set of assumptions and commonly used scheduling algorithms, any outer-lock independence-preserving protocol is necessarily non-optimal with respect to s-oblivious pi-blocking. In the rest of this thesis, I focus on an alternative definition for nested independence preservation.

3.2 Group Independence Preservation

With *group independence preservation*, the relationships that exist among shared resources and tasks are defined by relaxing Definitions 3.1.1 and 3.1.2. These relationships are defined as follows.

Definition 3.2.1. Let $\circ$ be a reflexive and symmetric relation on the set of shared resources Γ. For $\ell_a \in \Gamma$ let $\ell_a \circ \ell_a$, and for any $\ell_b, \ell_c \in \Gamma$ let $\ell_b \circ \ell_c$ if $\ell_b \succ \ell_c$ or $\ell_c \succ \ell_b$. The transitive closure of $\circ$ forms an equivalence relation on the resources in Γ, denoted with $\sim$. Then the equivalence class $g(\ell_a) = \{\ell_x \in \Gamma \mid \ell_a \sim \ell_x\}$ is the set of resources that ℓ_a is *associated* with.

I refer to these equivalence classes as *groups*, and let $G = \{g_1, \ldots, g_r\}$ be the set of resource groups in the system. From the definition of a group, it naturally follows that the groups in G are disjoint, and that their union yields Γ. This definition of groups very closely matches the notion of resource groups used in the FMLP [8].

Definition 3.2.2. For a task T_i the set $D_i = \bigcup_{\ell_a \in \gamma_i} g(\ell_a)$ is the set of the shared resources T_i is *associated* with.

One (possibly more intuitive) way to think about the equivalence classes described in Definition 3.2.1, are as components in a simple undirected-graph. For example, consider Fig. 3.4. Each vertex represents a shared resource, and there is an edge between two vertices v_a and v_b if $\ell_a \succ \ell_b$. If task a T_i accesses a shared resource, then T_i is associated with every other resource in the corresponding component, which follows from the necessary properties of an equivalence class (*i.e.*, reflexivity, symmetry, and transitivity).

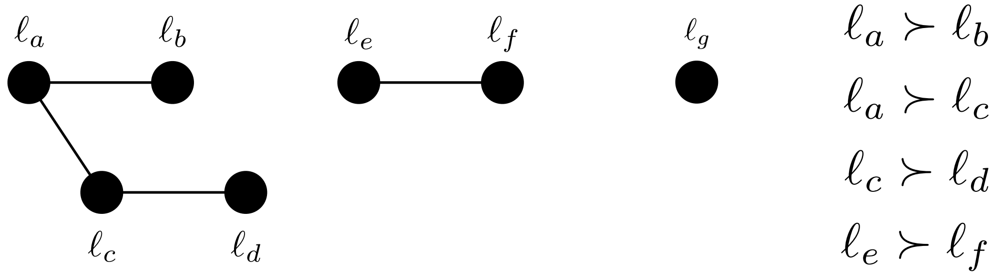


Figure 3.4: Visualization of group relations among shared resources as a simple undirected-graph. The partial ordering used to construct the graph is depicted on the right.

With association defined for shared resources and tasks, I now define group independence preservation as follows.

Definition 3.2.3. Let $b_{i,a}$ denote the pi-blocking incurred by J_i due to requests by any task for a shared resource ℓ_a. Under s-oblivious analysis, a locking protocol is *group independence-preserving* iff $\ell_a \notin D_i$ implies $b_{i,a} = 0$.

Stated differently, group independence is preserved if the overall s-oblivious pi-blocking $b_i = \sum_a b_{i,a}$ of each task does not depend on resources that the task is not associated with.

Chapter 4

The Group Independence-Preserving Protocol

I show that group independence-preserving protocols do not necessarily suffer from the $\Omega(n)$ s-oblivious pi-blocking bound seen with outer-lock independence preservation. I demonstrate this through the construction of group independence-preserving fine-grained nested locking protocol that is asymptotically optimal under s-oblivious analysis; the *Group Independence-Preserving Protocol* (GIPP). In this section I will give a high-level overview of the GIPP before constructing it in the subsequent sections of this thesis. The construction of GIPP relies on both the RNLP and RRPD; the necessary background on them is presented in Section 2.6.1 and Section 2.6.2, respectively. The background material on the RNLP and RRPD includes their structure, rules, properties, necessary lemmas, and necessary theorems.

The GIPP At a very high level, the GIPP works as follows. For each group, a separate instance of the RNLP is instantiated. Crucially, the choice of token lock and RSM used to instantiate each instance of the RNLP must not violate group independence preservation, that is, any progress mechanisms employed must lend themselves to group independence preservation. Progress mechanisms like priority boosting that rely on elevating a job's priority can cause jobs that never request shared resources to incur release-blocking, which precludes the property of group independence preservation; this is highly undesirable in the presence of latency-sensitive tasks. Furthermore, progress mechanisms that rely on the ability to directly compare priorities across clusters can result in unbounded pi-blocking (*i.e.*, the blocking depends on some other task's WCET) [9]. The challenges to realizing the GIPP are then: **(i)** construct an appropriate token lock and RSM, **(ii)** prove the token lock and RSM satisfy the required properties of the RNLP, **(iii)** prove the optimality of the GIPP under s-oblivious analysis, and **(iv)** prove that the GIPP is group independence-preserving.

4.1 The Clustered k-Exclusion Independence-Preserving Protocol

To realize the GIPP I use a single token lock that is common to all the instantiations of the RNLP. If there are r groups (and therefore r instances of the RNLP), then a token lock that arbitrates access to r distinct token types, where each token type has m replicas, will suffice. However, as stated earlier, any such token lock must lend itself to independence preservation. To the best my knowledge, no such k-exclusion locking protocol (*i.e.*, token lock) exists for clustered scheduling. To realize such a token lock, I generalize the R^{2}DGLP [54], which satisfies the requirements just described, with the exception that it was designed for globally-scheduled systems. Ward, Elliott, and Anderson use the term replica instead of token when discussing shared resources in the context of RRPD and R^{2}DGLP; I we will the two terms interchangeably.

As discussed in Section 2.6.2, the R^{2}DGLP uses RRPD as a progress mechanism. Thus, to generalize the R^{2}DGLP to clustered scheduling, the requirement that priorities across all jobs are comparable must be lifted. Additionally, RRPD alone is not enough to ensure progress [54], which means that replica-holders (*i.e.*, token holders), are not guaranteed to be scheduled without the aid of an additional progress mechanism. The R^{2}DGLP solves this with priority inheritance, as the protocol targets globally-scheduled systems. However, the R^{2}DGLP does not strictly mandate the use of priority inheritance, instead, any locking protocol that utilizes RRPD must satisfy property **P1**, which was stated in the background section for the R^{2}DGLP (Section 2.6.2).

I introduce the *Clustered k-Exclusion Independence-Preserving Protocol* (CKIP) as a generalization of R^{2}DGLP that is non-nested independence preserving, asymptotically optimal under s-oblivious analysis, and employable under clustered scheduling. The CKIP is realized by having tasks compete amongst each other in their home clusters under the rules of RRPD, but not across clusters. This is possible as priorities can be directly compared within each cluster. However, this means that priority inheritance can no longer be used to ensure that replica holders make progress. To this end, I employ AI (Section 2.5.1). I will redefine AI in the context of the CKIP and GIPP as replicas (*i.e.*, tokens) must now be taken into account; the definition differs only slightly from the definition in Section 2.5.1.

Definition 4.1.1. Let J_i be a job that holds a replica of a shared resource ℓ_a that has $k \geq 1$ replicas, and W_i be the set of jobs across all clusters waiting to a acquire a replica of ℓ_a. Under *allocation inheritance*, if J_i is not scheduled and there exists a job $J_k \in W_i \cup \{J_i\}$ that has sufficient priority to be scheduled in $C(T_k)$, then J_i migrates to $C(T_k)$ (if necessary) and runs with J_k's priority. While J_i executes in $C(T_k)$ with J_k's priority, J_k is called an *allocation donor*. Once J_i releases the replica of ℓ_a, it migrates back to $C(T_i)$ (if necessary) and resumes execution when it has sufficient priority. Finally, J_i's allocation donor (if any) ceases to be an allocation donor when J_i releases the replica of ℓ_a.

One might ask themselves if it is possible to realize the CKIP using a progress mechanism that does not require inter-cluster migrations; unfortunately, this cannot be done with the model and definitions used in this thesis. In fact, it is not possible for a semaphore-based protocol to avoid inter-cluster migrations, have bounded pi-blocking, and be independence-preserving [9].

Now armed with an independence-preserving progress mechanism [9], I can construct the CKIP by adapting the rules that define the R^{2}DGLP [54, Section 4]. The rules and structure of the CKIP differ enough from the R^{2}DGLP that its rules do not directly apply. Therefore, I present the modified rules and structure in full below.

Structure Tasks compete for a set of q shared resources $\Gamma = \{\ell_1, \ldots, \ell_q\}$ where each resource ℓ_a has $k \geq 1$ replicas. Nested requests are not permitted. Each of the

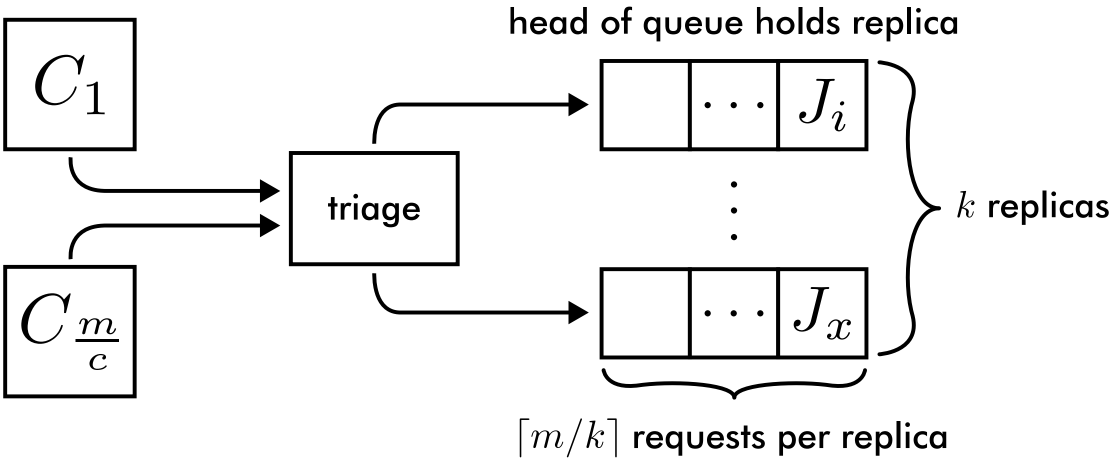


Figure 4.1: Queuing structure of the CKIP. Each of the m/c clusters can have at most c incomplete requests for a replica of a given shared resource. Requests are enqueued into the replica queue with the least number of requests in it.

k replicas has an associated FIFO queue of size $\lceil m/k \rceil$ that jobs are placed in when requesting a replica; I use KQ_a to refer to any one of the queues for ℓ_a. The queuing structure of the CKIP closely resembles the R^{2}DGLP and is depicted in Fig. 4.1. The following rules for CKIP focus on a single replicated resource $\ell_a \in \Gamma$, though they directly apply to all resources in Γ.

K1 Jobs issue requests subject to the rules of RRPD. When J_i issues a request for ℓ_a, it is enqueued into the KQ_a with the fewest number of requests in it, and suspends while it waits.

K2 J_i's request for ℓ_a is satisfied when it becomes the head of KQ_a, and thus becomes ready.

K3 While J_i is the head of KQ_a, it benefits from allocation inheritance, but only with respect to the other jobs in the same KQ_a as J_i (*i.e.*, W_i is comprised of the jobs in KQ_a that J_i is the head of).

K4 When J_i's request for ℓ_a is completed, it is dequeued from KQ_a and the new head (if any) acquires the replica. If J_i had benefited from allocation inheritance, it returns to its home cluster and assumes its former (possibly donated) priority. If J_i has a priority donor due to RRPD in $C(T_i)$, the donor may now issue a request subject to the rules of RRPD.

Now that the rules of the CKIP are defined, I use the remainder of this section to prove the following.

- The CKIP ensures Property **P1**, which is required by any locking protocol that uses RRPD (Section 2.6.2).
- Jobs that interact with the CKIP make progress.
- The CKIP ensures Property **T1** and Property **T2**, which are mandatory properties that any token lock to be used with the RNLP must ensure (Section 2.6.1).

- The CKIP is asymptotically optimal w.r.t. s-oblivious pi-blocking.
- The CKIP is both non-nested independence-preserving and group independence-preserving.

Lemma 4.1.1. Rule **K3** ensures property **P1**.

Proof. If J_i in KQ_a has sufficient priority to be scheduled in $C(T_i)$, then under AI, the head of KQ_a can migrate to $C(T_i)$ and execute with J_i's priority (if necessary). Therefore, the replica-holder is scheduled and J_i makes progress. □

Lemma 4.1.2. A job J_i that incurs pi-blocking while acting as a priority donor under the rules of RRPD makes progress.

Proof. Let J_x be the job that J_i donates its priority to. Then, J_x has an incomplete request for a replica of a shared resource ℓ_a that both jobs require. Because J_i has sufficient priority to be scheduled in $C(T_i)$, then J_x does as well, as $C(T_i) = C(T_x)$. Therefore, J_x makes progress by Lemma 4.1.1, which means J_i does as well. □

Lemma 4.1.3. The CKIP ensures property **T1** with respect to each replicated resource.

Proof. RRPD is orchestrated on a per-cluster basis under the CKIP, and so we can reason about each cluster individually as if it were a lone globally-scheduled system with c processors. Then, by Lemma 2.6.1 there are at most c incomplete requests for a given replicated resource per cluster, and therefore at most m across a clustered system as $\frac{m}{c} \cdot c = m$. □

Lemma 4.1.4. The CKIP ensures property **T2**.

Proof. By Lemma 2.6.2, a job J_i that requires a replica of a shared resource ℓ_a has an incomplete request, or is a priority donor. By Lemma 4.1.2, J_i makes progress while acting as a priority donor, and by Lemma 4.1.1, J_i makes progress while it has an incomplete request. Thus, J_i makes progress if it incurs pi-blocking while waiting for a token (*i.e.*, replica of ℓ_a). □

Lemma 4.1.5. The size of KQ_a need not be larger than $\lceil m/k \rceil$.

Proof. By Lemma 2.6.1 there are never more than c incomplete requests per cluster, and as there are $\frac{m}{c}$ clusters in total, there are never more than $\frac{m}{c} \cdot c = m$ incomplete requests for ℓ_a at any point in time. By Rule **K1**, a request is always enqueued into the KQ_a with the fewest number of requests in it. Thus, if KQ_a needed to larger than $\lceil m/k \rceil$, there would necessarily need to be more than m incomplete requests for the shared resource. Contradiction. □

Lemma 4.1.6. A job J_i incurs at most $(\lceil m/k \rceil - 1) \cdot L^{\max}$ s-oblivious pi-blocking in KQ_a.

Proof. By Lemma 4.1.5, there are at most $\lceil m/k \rceil - 1$ jobs ahead of J_i in KQ_a, and by Lemma 4.1.1 the head of KQ_a is scheduled if any other job in KQ_a has sufficient priority to be scheduled in its own cluster. Therefore, a job incurs at most $(\lceil m/k \rceil - 1) \cdot L^{\max}$ s-oblivious pi-blocking while in KQ_a. □

Theorem 4.1.1. A job J_i incurs at most $(2\lceil m/k\rceil - 1) \cdot L^{\max}$ s-oblivious pi-blocking while waiting to acquire a replica of a shared resource ℓ_a.

Proof. By Lemma 2.6.3 a priority donor can only be pi-blocked for one critical section plus the maximum amount of time a job can be pi-blocked with an incomplete request, and by Lemma 4.1.6 a jobs incurs at most $(\lceil m/k\rceil - 1) \cdot L^{\max}$ s-oblivious pi-blocking while in KQ_a. Thus, a priority donor incurs at most $\lceil m/k\rceil \cdot L^{\max}$ s-oblivious pi-blocking while waiting to acquire ℓ_a. Furthermore, by Lemma 2.6.2 a job that incurs s-oblivious pi-blocking while waiting for a replica of a shared resource ℓ_a either has an incomplete request for ℓ_a, or it is a priority donor. Thus while waiting to acquire ℓ_a a job J_i can incur at most the sum of the pi-blocking it incurs as a priority donor, and the pi-blocking it is subject to while traversing KQ_a, which is $\lceil m/k\rceil \cdot L^{\max} + (\lceil m/k\rceil - 1) \cdot L^{\max} = (2\lceil m/k\rceil - 1) \cdot L^{\max}$. □

Theorem 4.1.2. The CKIP is non-nested independence-preserving under any JLFP scheduler.

Proof. Under the CKIP, requests for replicas of shared resources are arbitrated under the rules of RRPD in each cluster. The rules of RRPD are such that jobs do *not* incur pi-blocking for resources they do not access [54]. Thus, any pi-blocking J_i incurs due to requests for a resource $\ell_a \notin \gamma_i$ would need to be the result of the use of AI as a cross-cluster progress mechanism. However, any job that benefits from AI only executes with the priority of another job currently waiting on a replica of the same resource, which precludes J_i from incurring pi-blocking due to jobs inheriting allocations [9]. Thus, if $N_{i,a} = 0$ then $b_{i,a} = 0$. □

This concludes the section on the CKIP. In the following section a RSM is defined, which is the last component required to realize the GIPP.

4.2 An Independence-Preserving RSM

The GIPP requires that its RSM lends itself to independence preservation, and no such suitable RSM for clustered scheduling has been proposed in prior work. Thus, I introduce the *Allocation Inheritance Resource Satisfaction Mechanism* (AI-RSM). The AI-RSM applies to clustered scheduling, and utilizes AI to ensure progress among jobs competing for shared resources. Let $ts(J_i)$ be the time that J_i acquired its token (and therefore entered the RSM), and let $sr(J_i, t)$ be the set of resources J_i holds at time t. The following equation denotes the set of jobs that can prevent J_i from acquiring ℓ_a at time t, which follows from the rules of the RNLP.

$$A_{i,a,t} = \{J_k \mid ts(J_k) < ts(J_i) \wedge (\ell_a \in sr(J_k, t) \vee \exists \ell_b \in sr(J_k, t) \text{ s.t. } \ell_b \succ \ell_a)\} \quad (4.1)$$

I now define the AI-RSM and subsequently prove that it ensures Property **R1** (Section 2.6.1), which any RSM to be used with the RNLP must ensure. The AI-RSM is defined by the following rule.

A1 When the AI-RSM prevents J_i from acquiring a shared resource ℓ_a at time t, J_i donates its allocation to the job in $A_{i,a,t}$ with the earliest timestamp under the rules of AI.

Lemma 4.2.1. The AI-RSM ensures Property **R1** for clustered scheduling when waiting is realized by suspending.

Proof. Let J_i be a job that is pi-blocked by the RSM at time t while it waits to acquire a shared resource ℓ_a. Then, there must exist some job $J_k \in A_{i,a,t}$ that prevents J_i from acquiring ℓ_a by the rules of the RNLP. By Rule **A1**, the job $J_k \in A_{i,a,t}$ with the earliest timestamp is eligible to inherit J_i's priority in $C(T_i)$. Since J_i incurs s-oblivious pi-blocking, it has one of the c highest priorities in its cluster, and hence the inherited priority enables J_k to be scheduled in $C(T_i)$. Thus, at least one job preventing J_i from acquiring ℓ_a is scheduled and J_i therefore makes progress. □

I have now constructed both a group independence-preserving token lock, and an RSM with an independence-preserving progress mechanism. In the following section I use these two components to realize the GIPP.

4.3 Structure and Analysis of The GIPP

In this section I define the structure of the GIPP. I then subsequently prove its optimality with respect to s-oblivious pi-blocking, and that it is group independence-preserving.

Structure There are m tokens for each group $g_x \subseteq \Gamma$; a token for g_x is denoted with λ_x. A single instance of the CKIP arbitrates access to the set $\Lambda = \{\lambda_1, \ldots, \lambda_r\}$ of replicated tokens, and an instance of the RNLP with the AI-RSM is instantiated for each group. The CKIP instance serves as a common token lock among all the instances of the RNLP. To execute an outermost critical section under the GIPP for resources in g_x a job must **(i)** compete for and acquire a token λ_x under the CKIP, **(ii)** compete in g_x's instance of the AI-RSM under the rules of the RNLP (Section 2.6.1), and **(iii)** release λ_x upon completing its outermost critical section and exiting the AI-RSM. The queuing structure of the GIPP is depicted in Fig. 4.2.

Theorem 4.3.1. The maximum amount of s-oblivious pi-blocking incurred per outermost request under the GIPP is $(2m-1) \cdot L^{\max} = \mathcal{O}(m)$ under any JLFP scheduler.

Proof. The CKIP satisfies property **T1** by Lemma 4.1.3, and the AI-RSM satisfies property **R1** by Lemma 4.2.1. Therefore, the maximum amount of s-oblivious pi-blocking a job incurs while in the AI-RSM is $L^{\mathrm{RSM}} = (m-1) \cdot L^{\max}$ by Theorem 2.6.1, as the corresponding RNLP proof generalizes to any protocol that satisfies these two properties.

Under the rules of the RNLP, a job holds a token for the entire duration it is in the RSM, and releases its token after completing its outermost critical section. The RNLP proof of Theorem 2.6.1 establishes that a job is pi-blocked for at most $m-1$ outermost critical sections while in any RSM. Thus, after a job completes its outermost critical section, the maximum amount of time the job holds a token is $L^{\mathrm{token}} = m \cdot L^{\max}$. By Theorem 4.1.1, a job waiting to acquire a token under the CKIP incurs at most $(2\lceil m/k \rceil - 1) \cdot L^{\mathrm{token}}$ units of s-oblivious pi-blocking. Under the GIPP, there are m tokens for each group (*i.e.*, $k = m$), so the pi-blocking incurred while waiting for a token simplifies to L^{token} as $(2\lceil m/m \rceil -, 1) = 1$. The total s-oblivious pi-blocking a job occurs per outermost request is then the sum of the pi-blocking

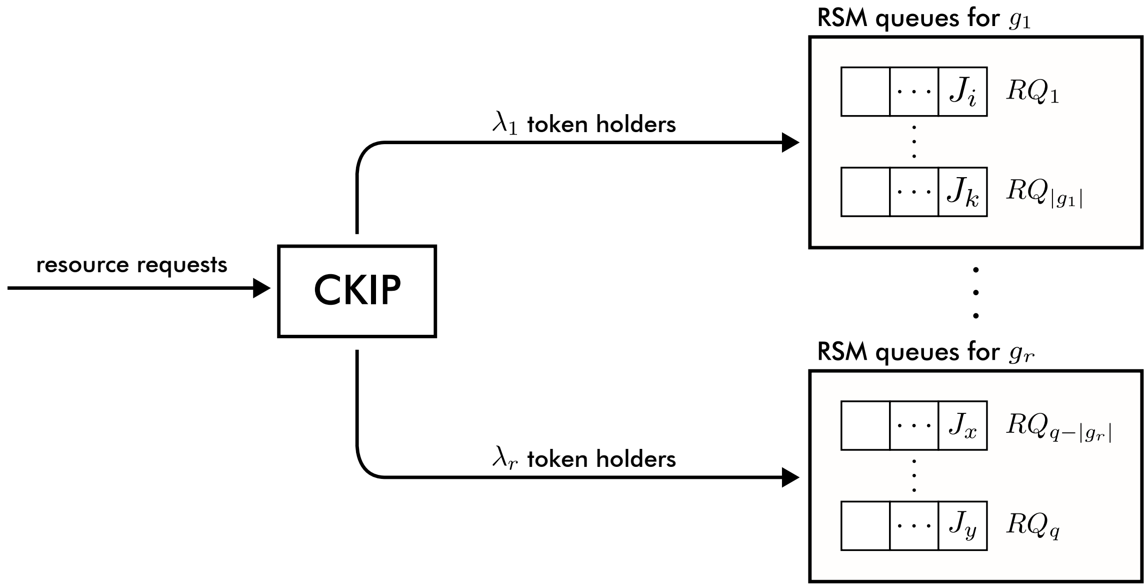


Figure 4.2: Queuing structure of the GIPP. A request for a token of a group is first arbitrated by the CKIP before the request is passed to the group's corresponding instance of the RNLP.

incurred while waiting to acquire a token, and while competing in the AI-RSM, which is $L^{\text{token}} + L^{\text{RSM}} = m \cdot L^{\max} + (m-1) \cdot L^{\max} = (2m-1) \cdot L^{\max}$. □

Theorem 4.3.2. The GIPP is group independence-preserving under any JLFP scheduler.

Proof. Under the CKIP, nested requests are not permitted, so each shared resource (*e.g.*, token type) forms its own group. When each group consists of a single resource, the definition of group independence preservation trivially reduces to non-nested independence preservation. Thus, it follows that the CKIP is group independence-preserving.

By the structure of the GIPP a job interacts with the CKIP for the entire duration it interacts with the GIPP, and as just established the CKIP is group independence-preserving with respect to tokens. Thus, to prove the GIPP is group independence-preserving it suffices to show that the CKIP remains group independence-preserving while token holders compete for shared resources in the AI-RSM. I prove this by contradiction.

Then, a job J_i that does not request a token λ_x for a group g_x incurs pi-blocking due to a request for λ_x from a job J_k. Under the AI-RSM, J_k's priority is only ever elevated to be that of another job competing for resources in g_x. Thus, if J_k's effective priority in $C(T_i)$ is greater than J_i's, there must be another job J_h in $C(T_i)$ that requires resources in g_x and has a higher base-priority than J_i. This precludes J_i from incurring s-oblivious pi-blocking due to J_k's request for λ_x. Contradiction. □

In special cases, the GIPP emulates the behavior of the RNLP and the OMIP. When there is just a single group (*i.e.*, $r = 1$) the GIPP effectively reduces to the RNLP in the sense that there is just a single, global token lock. Conversely, when $r = |\Gamma|$ the GIPP behaves like the OMIP. These cases are examined further in Section 6.3.

Chapter 5

Fine-Grained Pi-Blocking Analysis

I next introduce a fine-grained, non-asymptotic pi-blocking analysis for the GIPP, which is formulated as a *Linear Programming* (LP) problem as in prior work [9, 10, 55]. The asymptotic bound presented in Section 4.3 is coarse-grained as it does not reflect the exact resources each task requests, individual critical section lengths, nor the frequency of critical sections. The following analysis is fine-grained in the sense that it considers these workload-specific aspects to obtain a less pessimistic, but still safe, upper-bound on s-oblivious pi-blocking.

In the following, let T_i denote the task under analysis and let J_i denote an arbitrary job of T_i. For each other task T_x, I let θ_x^i denote a bound on the maximum number of jobs of T_x that overlap with J_i (*i.e.*, that are pending while J_i is pending). Let R_i and R_x be the maximum response times of T_i and T_x, respectively. The bound is then formulated as follows [10, 13].

$$\theta_x^i = \left\lceil \frac{R_i + R_x}{p_x} \right\rceil \tag{5.1}$$

I denote T_x's y^{th} outermost critical section as $O_{x,y}$, its length as $L_{x,y}^O$, the set of resources it accesses as $S_{x,y}$, and define $O_x(g) \triangleq \{O_{x,y} \mid S_{x,y} \subseteq g\}$ to be the outermost critical sections of task T_x that pertain to resources in group g. Note that the index y is used only for enumeration purposes and does *not* imply an order; each job of T_x may execute its outermost critical sections in any order. For each task $T_x \neq T_i$, each outermost request $O_{x,y}$, and $v \in \{1, \ldots, \theta_x^i\}$, I introduce two real-valued variables $X_{x,y,v}^T$ and $X_{x,y,v}^R$, each with domain $[0, 1]$. These variables are called *blocking fractions* [10] and serve to encode the portion of T_x's v^{th} overlapping instance of $O_{x,y}$ that contributes to the pi-blocking that J_i incurs. I use $X_{x,a,v}^T$ and $X_{x,a,v}^R$ to respectively encode the *token* and *RSM blocking* that J_i incurs, where token blocking refers to the time spent waiting to acquire a token, and RSM blocking refers to time spent waiting for a resource within the AI-RSM.

With these definitions in place, the pi-blocking incurred by J_i can be stated as

$$b_i = \sum_{T_x \neq T_i} \sum_{O_{x,y}} \sum_{v=1}^{\theta_x^i} (X_{x,y,v}^T + X_{x,y,v}^R) \cdot L_{x,y}^O \tag{5.2}$$

By interpreting Eq. (5.2) as the **objective function** of an LP maximization problem, I obtain an upper bound b_i on the maximum pi-blocking incurred by any J_i [9, 10, 55]. To avoid excessive pessimism, I introduce in the following LP constraints that reflect both the invariants of the GIPP and properties of the specific task set under analysis.

To start, I prevent any blocking critical section from being counted twice.

Constraint 5.0.1. $\forall T_x \neq T_i : \forall O_{x,y} : \forall v : X^T_{x,y,v} + X^R_{x,y,v} \leq 1$

Proof. A single critical section of T_x cannot cause J_i to experience token blocking and RSM blocking *simultaneously*: to wait for a resource within the AI-RSM, J_i must already hold a token, but while J_i competes for a token it cannot yet interact with the RSM. Thus, the combined token and RSM blocking induced by one of T_x's critical sections cannot exceed the length of the critical section (*i.e.*, the blocking fractions sum to at most one). □

Next, I bound the maximum amount of token blocking that J_i incurs. In preparation, let τ_k be the set of tasks assigned to cluster C_k, and $\tau'_k = \tau_k \setminus \{T_i\}$. Furthermore, Eq. (5.3) defines the number of times J_i issues an outermost request for a resource in g, and Eq. (5.4) defines the number of tasks in C_k that request a resource in g. Based on these definitions, I state a bound on the number of times that J_i must wait for a token. In Eq. (5.5), let k denote the index of $C(T_i)$.

$$\phi_{i,g} \triangleq |\{O_{i,y} \mid S_{i,y} \cap g \neq \emptyset\}| \tag{5.3}$$

$$\beta_{k,g} \triangleq \left| \left\{ T_x \;\middle|\; T_x \in \tau_k \wedge \bigcup_{O_{x,y}} S_{x,y} \cap g \neq \emptyset \right\} \right| \tag{5.4}$$

$$\begin{aligned} W_{i,g} &\triangleq \begin{cases} 0 & \beta_{k,g} \leq c \\ \min(\phi_{i,g}, \phi'_{i,g}) & \text{otherwise} \end{cases} \\ \text{where } \phi'_{i,g} &\triangleq \left(\sum_{T_x \in \tau'_k} (\phi_{x,g} \cdot \theta^i_x) \right) - c + 1 \end{aligned} \tag{5.5}$$

Lemma 5.0.1. $W_{i,g}$ upper-bounds the number of times J_i must wait for a token of group g.

Proof. By case analysis. Let k denote the index of $C(T_i)$. First, if $\beta_{k,g} \leq c$, then there are at most c tasks in $C(T_i)$ that ever require a token for group g (including T_i). There are never more than c token holders per cluster under the CKIP, which effectively reserves c tokens for each cluster. Thus, whenever J_i requires a token for group g, one is always available, and J_i never needs to wait for a token: $W_{i,g} = 0$ if $\beta_{k,g} \leq c$.

Otherwise, if $\beta_{k,g} > c$, then J_i requires a token no more than $\phi_{i,g}$ times, and hence clearly $W_{i,g} \leq \phi_{i,g}$. To obtain $W_{i,g} \leq \phi'_{i,g}$, consider the number of times that other tasks require a token while J_i is pending, which is bounded by $\sum_{T_x \in \tau'_k} (\phi_{x,g} \cdot \theta^i_x)$. Since J_i must wait for a token only if all c tokens are currently held by other tasks, the worst case occurs when $c - 1$ tokens are held "indefinitely" (*i.e.*, if they remain unavailable throughout the interval during which J_i is pending) and the remaining $\phi'_{i,g} = \left(\sum_{T_x \in \tau'_k} (\phi_{x,g} \cdot \theta^i_x) \right) - c + 1$ requests must all share a single token, and thus $W_{i,g} \leq \phi'_{i,g}$. □

I further restrict under which conditions J_i incurs token blocking with the following lemma and constraint.

Lemma 5.0.2. J_i incurs token blocking (*i.e.*, if it incurs pi-blocking while waiting to acquire a token for a group g) only if it is a priority donor under the rules of RRPD.

Proof. Recall that the GIPP allocates group tokens using the CKIP, and that the CKIP employs RRPD. As there are $k = m$ tokens per group (*i.e.*, replicas per token type), the CKIP's per-replica FIFO queues have length $\lceil \frac{m}{k} \rceil = 1$. Since by Rule **K2** the head of each per-replica FIFO queue holds the replica (*i.e.*, a token), and jobs enter a queue immediately when they issue a request (Rule **K1**), it follows that J_i can be waiting for a token only *before* it issues its request for a token, that is, while it serves as a priority donor under the rules of RRPD in the time span between requiring a token and actually issuing a request (recall Fig. 2.14). □

Lemmas 5.0.1 and 5.0.2 then allow a constraint to be established on token blocking due to each other task.

Constraint 5.0.2.

$$\forall g \in G : \forall T_x \neq T_i : \sum_{O_{x,y} \in O_x(g)} \sum_{v=1}^{\theta_x^i} X_{x,y,v}^T \leq W_{i,g}$$

Proof. Suppose not. Then there exists a task T_x that token-blocks J_i with more than $W_{i,g}$ outermost critical sections (w.r.t. some group g). If $W_{i,g} = 0$, then by Lemma 5.0.1 J_i must never wait to acquire a token for group g, which immediately yields a contradiction. Hence assume $W_{i,g} > 0$. As J_i waits for a token for g at most $W_{i,g}$ times (Lemma 5.0.1), this implies that there exists an outermost critical section $O_{i,z}$ executed by J_i such that J_i is delayed, while waiting to acquire a token for g in preparation of $O_{i,z}$, by at least two outermost critical sections of T_x. By Lemma 5.0.2, J_i is a priority donor while it incurs token blocking. According to the rules of RRPD (recall Section 2.6.2), J_i becomes a priority donor at most once per request, and only for a single other request: either immediately when J_i requires a token to commence $O_{i,z}$, or not at all. It follows that T_x must pi-block J_i with *two* distinct outermost critical sections while J_i continuously serves as the priority donor of some job J_l. Since under the rules of RRPD J_i ceases to be a priority donor as soon as J_l finishes its outermost critical section (*i.e.*, when J_l releases its token), J_l cannot be a job of T_x. Hence there remains only one other way for an outermost critical section of T_x to delay J_i, namely by delaying one or more requests of J_l within the RSM, which transitively causes J_i to incur pi-blocking. Consider the later of T_x's two outermost critical sections that cause J_i to incur pi-blocking while donating its priority to J_l. Since it is the second *outermost* critical section of T_x in this interval, T_x necessarily must have acquired a token for group g strictly after the beginning of the interval, when J_l was already holding its token. However, the RSM satisfies resource requests strictly in order of increasing token-acquisition timestamps, and thus T_x's second outermost critical section cannot delay J_l. Contradiction. □

I similarly bound the aggregate token blocking across all tasks in each cluster as follows.

Constraint 5.0.3.

$$\forall g \in G : \forall k \in \{1, \ldots, \frac{m}{c}\} : \sum_{T_x \in \tau'_k} \sum_{O_{x,y} \in O_x(g)} \sum_{v=1}^{\theta_x^i} X_{x,y,v}^T \leq W_{i,g} \cdot \min(c, \beta_{k,g})$$

Proof. Again the case of $W_{i,g} = 0$ is trivial; hence assume $W_{i,g} > 0$ and suppose the invariant does not hold. Then analogously to the proof of Constraint 5.0.2, there exists a contiguous interval $[t_1, t_2)$ and a cluster C_k such that both **(i)** J_i serves as the priority donor of some job J_l throughout $[t_1, t_2)$, and **(ii)** J_i incurs pi-blocking during $[t_1, t_2)$ due to at least $\min(c, \beta_{k,g}) + 1$ outermost critical sections executed by tasks in τ_k. Also analogously to the proof of Constraint 5.0.2, no task delays J_i with more than one outermost critical section during $[t_1, t_2)$. Because the RSM satisfies requests strictly in order of increasing token-acquisition timestamps, any job that delays J_l within the RSM (and hence transitively causes J_i to incur token blocking) must have acquired its token for group g before J_l did so, and hence no later than at time t_1. Furthermore, any such job necessarily releases its token only some time after t_1. At time t_1 there hence exist at least $\min(c, \beta_{k,g}) + 1$ token-holding jobs in cluster C_k. However, the CKIP ensures that no more than c jobs in C_k hold a token at any time, and by definition at most $\beta_{k,g}$ tasks in τ_k require a token for group g. Contradiction. □

This concludes the constraints on token blocking. I next constrain RSM blocking, that is, the pi-blocking incurred by J_i while it holds a token and waits for individual resources. I first introduce two necessary lemmas, and then constraint the RSM blocking that J_i incurs on a per-cluster basis, and subsequently on a per-task basis.

Lemma 5.0.3. While J_i executes an outermost critical section $O_{i,w}$ it can be RSM-blocked by the execution of at most one outermost critical section of T_x.

Proof. Suppose not. Then while J_i executes $O_{i,w}$ it is pi-blocked by two outermost critical sections $O_{x,y}$ and $O_{x,z}$ of T_x. Only upon completion of $O_{x,y}$ can T_x begin the execution of $O_{x,z}$. A new token-acquisition timestamp is recorded when T_x acquires a token during the execution of $O_{x,z}$. However, this timestamp will be strictly larger than the token-acquisition timestamp recorded during the execution of $O_{i,w}$. Thus, all requests J_i issues during the execution of $O_{i,w}$ will be satisfied before any request T_x issues during the execution of $O_{x,z}$ as the RSM satisfies requests in order of increasing token-acquisition timestamps. This precludes $O_{x,z}$ from contributing to the RSM blocking J_i incurs. Contradiction. □

Lemma 5.0.4. Let $O_{i,w}$ and $O_{i,y}$ be two outermost critical sections of J_i, and $O_{x,z}$ be an outermost critical section of T_x. If J_i incurs RSM blocking while executing $O_{i,w}$ due to the execution of $O_{x,z}$, then the execution of $O_{x,z}$ cannot contribute to the RSM blocking J_i incurs while J_i executes $O_{i,y}$.

Proof. If the execution of $O_{x,z}$ contributes to the RSM blocking J_i incurs while J_i executes $O_{i,w}$, then the token-acquisition timestamp recorded during the execution of $O_{i,w}$ is less than the token-acquisition timestamp recorded during the execution of $O_{x,z}$. As the RSM satisfies requests in order of increasing token-acquisition timestamps, T_x necessarily finishes executing $O_{x,z}$ before J_i finishes executing $O_{i,w}$. Thus, the execution $O_{x,z}$ completes before the execution of $O_{i,y}$ begins, which precludes J_i from incurring RSM blocking due to the execution of $O_{x,z}$ while J_i executes $O_{i,y}$. □

Constraint 5.0.4.

$$\forall g \in G : \forall k \in \{1, \ldots, \frac{m}{c}\} :$$

$$\sum_{T_x \in \tau'_k} \sum_{O_{x,y} \in O_x(g)} \sum_{v=1}^{\theta_x^i} X^R_{x,y,v} \leq \begin{cases} \phi_{i,g} \cdot \min(c, \beta_{k,g}) & T_i \notin \tau_k \\ \phi_{i,g} \cdot \min(c-1, \beta_{k,g}-1) & \text{otherwise} \end{cases}$$

Proof. If $\phi_{i,g} = 0$, then J_i does not access resources in group g and the invariant is trivial. Hence assume otherwise and suppose the invariant does not hold. First consider the case $T_i \notin \tau_k$: then there exists an interval $[t_1, t_2)$ during which J_i holds a token for group g such that J_i incurs RSM blocking due to more than $\min(c, \beta_{k,g})$ outermost critical sections executed by jobs in C_k. Analogously to the proof of Constraint 5.0.3, it follows that more than $\min(c, \beta_{k,g})$ jobs must hold a token for group g at time t_1, which is impossible. In the second case, if $T_i \in \tau_k$, then J_i necessarily holds one of the c available tokens (otherwise it could not interact with the RSM), so that there are only $c - 1$ tokens available to other tasks, and only $\beta_{k,g} - 1$ other tasks in τ_k that are also accessing resources in group g. □

Constraint 5.0.5.

$$\forall g \in G : \forall T_x \neq T_i \sum_{O_{x,y} \in O_x(g)} \sum_{v=1}^{\theta_x^i} X_{x,y,v}^R \leq \min(\phi_{i,g}, \phi_{x,g} \cdot \theta_x^i)$$

Proof. Proof by case analysis. Consider the case when $\phi_{i,g} < \phi_{x,g} \cdot \theta_x^i$. By Lemma 5.0.3 J_i is not RSM-blocked by two distinct outermost critical sections of T_x. Thus, at most $\phi_{i,g}$ outermost critical sections of T_x contribute to the RSM blocking that J_i incurs.

Next consider when $\phi_{i,g} \geq \phi_{x,g} \cdot \theta_x^i$. By Lemma 5.0.4 two distinct outermost critical sections of J_i cannot be RSM-blocked by a single outermost critical section of T_x. Thus, at most $\phi_{x,g} \cdot \theta_x^i$ outermost critical sections of T_x contribute to the RSM blocking that J_i incurs. □

I next constrain RSM blocking in a more detailed fashion by considering which critical sections actually conflict within the RSM. The resulting constraint is essential to realizing the benefits of the increased parallelism in nested locking protocols (relative to coarse-grained group-locking) at analysis time, and not just at runtime. To this end, some further terminology and notation are required. First, I say that a set of resources s is *possibly conflicting* with another set of resources s' if either **(i)** $s \cap s' \neq \emptyset$ or **(ii)** $\exists \ell_b \in s, \ell_a \in s'$ such that $\ell_a \succ \ell_b$. Second, Eq. (5.6) counts the number of outermost critical sections of T_i which need resources that the RSM may have to withhold due to other jobs holding resources in s. Finally, the definition in Eq. (5.7) denotes the set of all combinations of resources in group g acquired by other tasks. Based on these definitions, I constrain RSM blocking as follows.

$$F_i(s) \triangleq |\{O_{i,y} \mid S_{i,y} \textit{ possibly conflicts with } s\}| \tag{5.6}$$

$$S^i(g) \triangleq \{S_{x,y} \mid T_x \neq T_i \wedge S_{x,y} \cap g \neq \emptyset\} \tag{5.7}$$

Constraint 5.0.6.

$$\forall g \in G : \forall s \in S^i(g) : \forall k \in \{1, \ldots, \frac{m}{c}\} :$$

$$\sum_{T_x \in \tau_k'} \sum_{\substack{O_{x,y} \text{ s.t.} \\ S_{x,y} \subseteq s}} \sum_{v=1}^{\theta_x^i} X_{x,y,v}^R \leq \begin{cases} F_i(s) \cdot \min(c, \beta_{k,g}) & T_i \notin \tau_k \\ F_i(s) \cdot \min(c-1, \beta_{k,g} - 1) & \text{otherwise} \end{cases}$$

Proof. Consider any group g, set of resources $s \in S^i(g)$, and cluster C_k. For J_i to incur RSM blocking when issuing a request for some resource $\ell_b \in g$, there must exist a job

J_x with an earlier token-acquisition time that either already holds ℓ_b, or that holds a resource $\ell_a \in g$ such that $\ell_a \succ \ell_b$. In other words, J_i incurs RSM blocking only if $\{\ell_b\}$ is possibly conflicting with the set of resources already held by jobs with earlier timestamps. Recall that $F_i(s)$ counts the number of outermost critical sections of T_i accessing resources that are possibly conflicting with s. It follows that J_i executes at most $F_i(s)$ outermost critical sections that may encounter RSM blocking due to outermost critical sections of tasks in τ'_k that access s or a subset of s (*i.e.*, the requests represented on the left-hand side of the constraint). As in the proof of Constraint 5.0.4, it is easy to show that no more than $\min(c, \beta_{k,g})$ (respectively, $\min(c-1, \beta_{k,g}-1)$) outermost critical sections can cause RSM blocking per each outermost critical section of J_i if $T_i \notin \tau_k$ (respectively, $T_i \in \tau_k$). The bound follows. □

Finally, I apply the logic in Constraint 5.0.6 to bounding the RSM blocking J_i incurs on a per-task basis w.r.t. possibly conflicting resource requests.

Constraint 5.0.7.

$$\forall g \in G : \forall T_x \neq T_i : \forall s \in S^i(g) : \sum_{\substack{O_{x,y} \text{ s.t.} \\ S_{x,y} \subseteq s}} \sum_{v=1}^{\theta_x^i} X^R_{x,y,v} \leq F_i(s)$$

Proof. This proof follows analogously from Constraint 5.0.6; J_i executes at most $F_i(s)$ outermost critical sections that may encounter RSM blocking due to outermost critical sections of T_x that accesses s or a subset of s. □

This concludes my fine-grained analysis of the GIPP. The analysis as presented in this thesis contains two constraints not seen in the original work [43]; they are Constraint 5.0.5 and Constraint 5.0.7. In the next chapter I report on an empirical evaluation of the GIPP and two baseline protocols using the just-presented analysis.

Chapter 6

Schedulability Experiments

In this chapter I present the results of large-scale schedulability experiments I conducted to compare the GIPP against the OMIP and the RNLP. The goal of these experiments is to see how these protocols effect the schedulability of task sets when a fine-grained (*i.e.*, non-asymptotic) analysis is applied. As all three of these protocols are asymptotically optimal w.r.t. s-oblivious pi-blocking, a naive schedulability analysis can safely inflate the WCETs of a given task set by the appropriate worst-case upper-bounds on s-oblivious pi-blocking. However, such an analysis would not only be overly pessimistic (in the general case), but it would also fail to provide a meaningful comparison on how the three protocols effect the schedulability of task sets with different properties. For example, task sets with latency-sensitive tasks, or task sets with low contention for all but a few shared resources.

I chose to compare the GIPP against the OMIP and the RNLP as **(i)** they are both asymptotically optimal with respect to s-oblivious pi-blocking, **(ii)** the OMIP [9] is the only prior independence-preserving locking protocol for clustered scheduling, and **(iii)** the RNLP [53] is the only prior fine-grained nested locking protocol that ensures asymptotically optimal pi-blocking bounds under clustered scheduling.

To conduct meaningful schedulability experiments, a fine-grained analysis of the OMIP and the RNLP is also required. The OMIP has such an analysis [9], also formulated as an LP, which is used in these experiments. However, for the RNLP, there are surprisingly no fine-grained bounds available in prior work. I therefore had to adapt the GIPP's fined-grained analysis to the RNLP. To this end, I created an instantiation of the RNLP called the CA-RNLP. The following describes this instantiation and highlights some important details about how the GIPP's analysis can be applied to the CA-RNLP.

- The CA-RNLP uses the CKIP as its token lock, and the AI-RSM as its RSM.
- The RNLP uses a single global token-lock, and thus so does the CA-RNLP.
- In order to apply the GIPP's fine-grained analysis to the CA-RNLP, one must presume that *all* resources belong to a single group, as there is only one token lock. A partial ordering on resources is still constructed as this is required by the RNLP, but the ordering is not used to split the resources into groups.

The basic setup of the experiments is as follows. Large numbers of task sets were generated with Emberson, Stafford, and Davis's method [25] via the SchedCAT [12] library; let τ be one of these task sets. For each of the three protocols the corresponding fined-grained analysis is applied to τ, the WCETs of the tasks in τ are inflated by the produced blocking bounds, and then a schedulability test is applied to τ. In the rest of this chapter I will introduce some necessary definitions, discuss how the experiments presented in this thesis differ from the experiments presented in the original work [43], define the setup of the experiments in detail, and finally discuss the results.

The experiments in this thesis will consider the *resource access patterns* of task sets, *i.e.*, which resources a task issues requests for and with what frequency. The following equation defines the *request symmetry ratio*, which is a measure of how similar the resource access patterns of two given tasks are.

$$symr(T_i, T_k) \triangleq \frac{\sum\limits_{\ell_a \in \gamma_i \cap \gamma_k} N_{i,a} + N_{k,a}}{N_i + N_k} \tag{6.1}$$

Two tasks T_i and T_k with a small request symmetry ratio do not often compete with each other (if at all) for shared resources under the three protocols being examined, which motivates the following definition.

Definition 6.0.1. T_i and T_k have *highly asymmetric access patterns* (HAAPs) if $symr(T_i, T_k) < x$, for a given threshold x. Otherwise, if $symr(T_i, T_k) \geq x$, then they are said to have *uniform access patterns* (UAPs).

To build an intuition for HAAPs, consider the following.

- Two jobs J_i and J_k compete for three shared resources ℓ_1, ℓ_2, and ℓ_3.
- J_i issues $N_{i,1} \geq 1$ outermost requests for ℓ_1, and during one of these requests J_i issues a nested request for ℓ_3.
- J_k issues $N_{k,2} \geq 1$ requests for ℓ_2, and during one of these requests J_k issues a nested request for ℓ_3.
- The partial order on the resources is then $\ell_1 \succ \ell_3$, $\ell_2 \succ \ell_3$.
- As $N_{i,1}$ or $N_{k,2}$ becomes large the request symmetry ratio of T_i and T_k drops. How small the request symmetry ratio needs to be before T_i and T_k have a highly asymmetric access patterns is defined by a system's designer.

When the values of $N_{i,1}$ and $N_{k,2}$ become large in the just presented example, one would expect a fine-grained locking protocol like the GIPP or the RNLP to yield less cumulative pi-blocking than a locking protocol that realizes nested locking with group locks. This is rather intuitive to see. Under the OMIP for example, a single group lock would protect ℓ_1, ℓ_2, and ℓ_3. Thus each request J_i issues can block J_k despite the two jobs rarely issue a request for a common resource (when $N_{i,1}$ or $N_{k,2}$ are large). In contrast, the request J_i issues for ℓ_2 is the only request that can block J_k under the GIPP and the RNLP.

I introduce one more definition before discussing how these experiments differ from those in the original work. The following definition plays a key role in describing how tasks access resources in the experiments.

Definition 6.0.2. A shared resource $\ell_b \in \Gamma$ is a *top-level resource* if there $\nexists \ell_a \in \Gamma$ s.t. $\ell_a \succ \ell_b$.

The large scale experiments performed in the original work did not consider task sets with HAAPs beyond one hand-crafted example, which did indicate that use of coarse-grained locking effected the schedulability of task sets with HAAPs negatively. For this reason there are two separate experiments conducted, one for task sets with UAPs and one for task sets with HAAPs. The original experiments considered considered *wide* groups and *deep* groups [43]. A group is considered to be *wide* if at least half of its resources are top-level, and *deep* otherwise. A visual representation of a wide group and a deep group is depicted in Fig. 6.1. The results from the original experiments did not demonstrate a noticeable effect on schedulability for tasks set that accessed wide groups versus deep groups, and thus I chose to use only wide groups in these experiments. Finally, these experiments use the updated fine-grained analysis for the GIPP presented in Chapter 5, which includes two new constraints not seen in the original work.

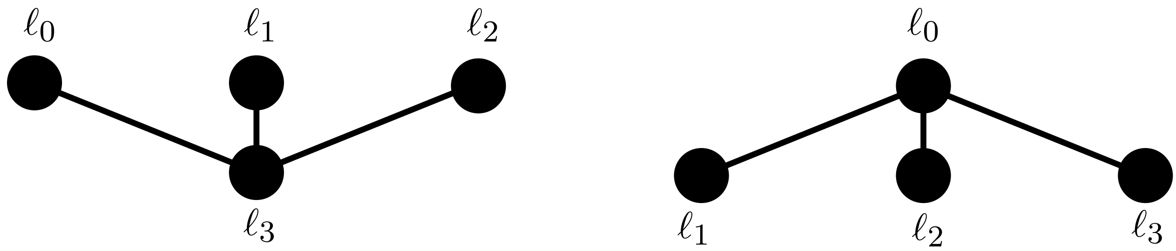


Figure 6.1: Depiction of a wide group on the left, and a deep group on the right. The wide group has the partial order $\ell_0 \succ \ell_3, \ell_1 \succ \ell_3, \ell_2 \succ \ell_3$, and the deep group has the partial order $\ell_0 \succ \ell_1, \ell_0 \succ \ell_2, \ell_0 \succ \ell_3$. Thus the wide group has three top-level resources whereas the deep group has just one top-level resource.

The remainder of this chapter is structured as follows. The next two sections will outline the setup of the experiments for task sets with UAPs and HAAPs, respectively. Afterwards, I will discuss the results and the observations that can be derived from them. Finally, a small section will outline technical details that relate to the implementation and execution of the experiments.

6.1 UAP Experiment Setup

The experiment setup for task sets with UAPs is as follows. The values for the parameters used in the setup are listed in Table 6.1.

- Each task set consisted of n tasks with total utilization U to be scheduled on m processors under P-EDF scheduling.
- There were n^{nl} latency-sensitive tasks in the task set, and $n - n^{\text{nl}}$ non-latency-sensitive (or "regular") tasks.
- Periods were drawn uniformly at random from the set $p^{nls} = \{10ms, 20ms, 25ms, 40ms, 50ms, 100ms, 125ms, 200ms, 250ms, 500ms, 1000ms\}$ for regular tasks. The values of p^{nls} were inspired by Kramer, Ziegenbein, and Hamann's work on producing real-world automotive benchmarks [35].

- Periods were drawn uniformly at random from the set $p^{ls} = \{1ms, 2ms, 4ms, 5ms, 8ms\}$ for latency-sensitive tasks.

- Regular tasks shared twelve resources split into equally sized groups of g^{size}.

- Latency-sensitive tasks shared three resources that belonged to a single group. Each of the tasks issued one or two outermost requests at random for the resources in the group.

- Regular tasks were assigned a minimal set of resource requests at random to ensure the desired groups were formed; each task accessed just one group. Afterwards, tasks were assigned outermost requests for resources in their corresponding groups at random until each task made $N^{\max}$ requests (per job); each of these outermost requests contained a nested request with a probability of tp.

- In each experiment the outermost critical section lengths of regular tasks were drawn from $[1\mu s, mcls]$ uniformly at random where $mcsl$ varied across $[5\mu s, 1000\mu s]$ in increments of $5\mu s$.

- The outermost critical section lengths of latency-sensitive tasks were drawn uniformly at random from $[1\mu s, 15\mu s]$.

Parameter	Values
m	$\{4, 8, 16\}$
n	$\{2.0m, 3.0m\}$
U	$\{0.4m, 0.6m\}$
n^{nl}	$\{0.0m, 0.5m, 1.0m\}$
g^{size}	$\{1, 2, 3, 4\}$
$N^{\max}$	$\{1, 2, 3\}$
tp	$\{0.5\}$

Table 6.1: Parameters values used in the UAP experiments.

There were 432 combinations of the parameters in Table 6.1. For some parameter choices it is not possible to generate task sets with the intended characteristics. For example, it is not possible to generate a task set of $n = 8$ tasks with $q = 12$ resources split into groups of size $g^{\text{size}} = 1$ if each task accesses only one group. In this scenario there would be four groups that are not accessed by any task. After removing such combinations, there are 348 combinations left. For each of these combinations, I generated 500 task sets per $mcsl$ value (*i.e.*, per point on the x-axis of the produced plots seen in Section 6.3), and then tested each task set for schedulability under the GIPP, OMIP, and the CA-RNLP with a P-EDF schedulability test.

6.2 HAAP Experiment Setup

The experiment setup for task sets with HAAPs is the same as the setup for task sets with UAPs (Section 6.1) with a few key differences. The following lists where this setup differs from the UAPs experiments. The values used in this setup are listed in Table 6.2.

- There are no latency-sensitive tasks.
- q is now a parameter.
- The values the parameter g^{size} can take on differs based on the value of q.
- Regular tasks were still assigned the necessary requests at random to ensure the desired groups were formed.
- To bring up the number of requests each regular task makes to $N^{\max}$, tasks were assigned resource requests in the following way. Let g_x be a group with three top-level resources $tl_x = \{\ell_0, \ell_1, \ell_2\}$. When a task T_i is assigned a resource request, the resource it accesses is ℓ_a where $a = i \mod |tl_x|$. For example T_4 would issue outermost requests for ℓ_1.
- The values of $N^{\max}$ are considerably larger, as this value greatly influences the request symmetry ratio of two jobs.

Parameter	Values
m	$\{4, 8, 16\}$
n	$\{2.0m, 3.0m\}$
U	$\{0.4m, 0.6m\}$
q	$\{4, 5, 8, 10, 12, 15, 16, 20\}$
g^{size}	4 when $q \in \{4, 8, 12, 16\}$ 5 when $q \in \{5, 10, 15, 20\}$
$N^{\max}$	$\{5, 10, 15\}$
tp	0.8

Table 6.2: Parameters values used in the HAAP experiments.

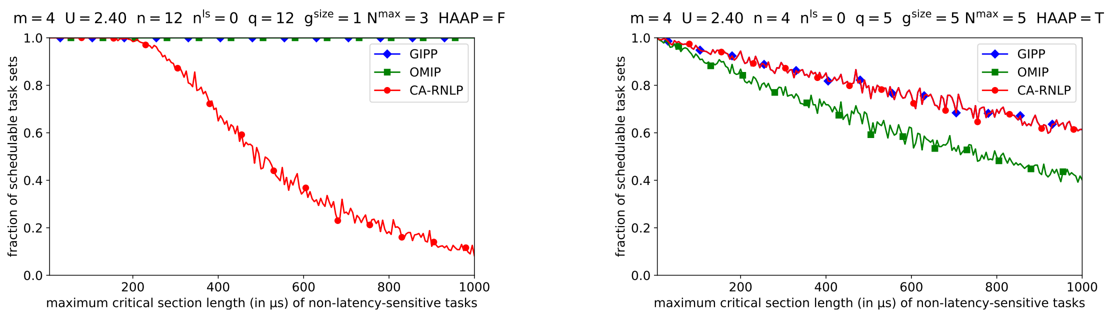
There are 288 total combinations of the parameters in Table 6.2, of which 246 can generate valid task sets. As with the UAPs experiments, I generated 500 task sets per *mcsl* value, and then tested each task set for schedulability under the GIPP, OMIP, and the CA-RNLP assuming P-EDF scheduling.

6.3 Results

In the large-scale experiments, both the GIPP and the OMIP retained a high level of schedulability for most parameter configurations. In most cases, the CA-RNLP provided a substantially lower level of schedulability than the GIPP or the OMIP. The full results of the UAP and HAAP experiments are available in Appendix A.1 and Appendix A.2, respectively. I now outline five key observations in the following sections that are derived from the experiments. Note that in the produced plots I use `HAAP = T` to denote that the task sets generated have HAAPs, and UAPs otherwise.

6.3.1 General Performance

The GIPP performs noticeably better than the OMIP and the CA-RNLP in most of the experiments, and never worse. In corner cases, the performance of the GIPP approaches that of the OMIP when $g^{\text{size}} = 1$, and the CA-RNLP when $g^{\text{size}} = |\Gamma|$ (*i.e.*, the total number of resources); this is apparent in Fig. 6.2(a) and Fig. 6.2(b), respectively. As a result, the GIPP never performs worse than the better-performing of the two baselines.



(a) Each group consists of a single resource. (b) All resources belong to a single group.

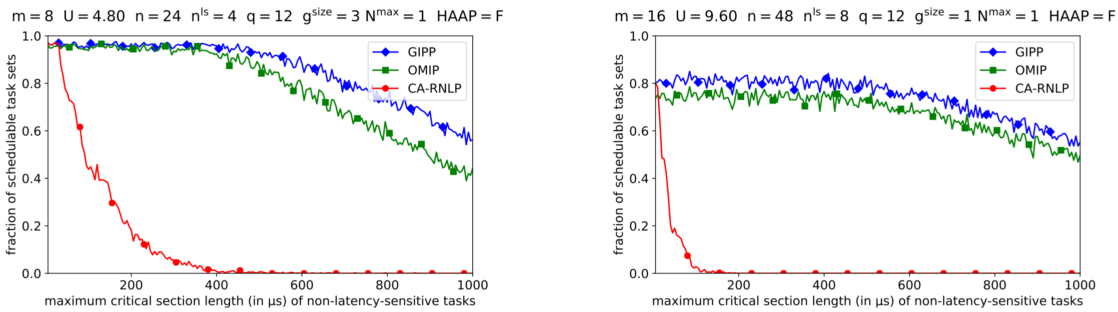
Figure 6.2: The GIPP never performs worse than the CA-RNLP nor the OMIP.

6.3.2 Impact of Latency-Sensitive Tasks

The isolation of latency-sensitive tasks greatly impacts schedulability. Both latency-sensitive and regular tasks compete for the same set of tokens under the CA-RNLP, and as a result they do not benefit from the isolation afforded by the GIPP and the OMIP. The result of this can be seen in Figs. 6.3(a) and 6.3(b) where schedulability quickly drops under the CA-RNLP as *mcsl* increases. The benefit of this isolation is still observable for large task sets that have a relatively large number of latency-sensitive tasks. Consider Fig. 6.3(b). Tasks sets become unschedulable almost immediately under the RNLP, whereas roughly 60% and 50% of task sets remain schedulable under the GIPP and the OMIP, respectively.

6.3.3 Global Token-Lock Bottleneck

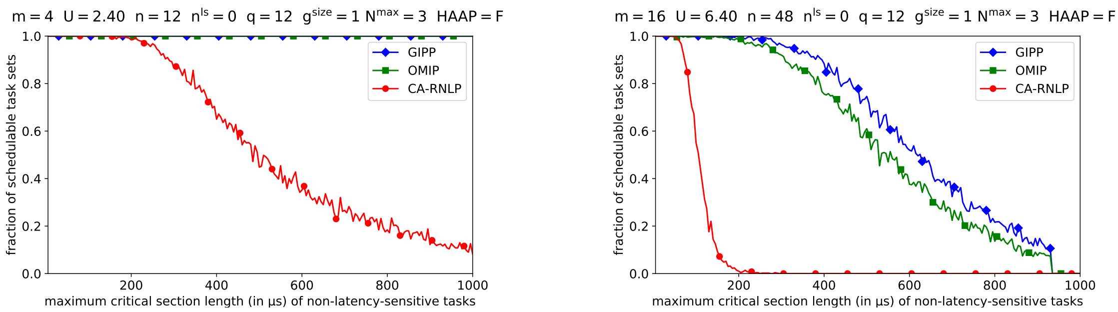
Even in the absence of latency-sensitive tasks, schedulability is greatly affected by the use of a single global token-lock (*i.e.*, tokens are not group-specific). As *mcsl* increases, schedulability under the CA-RNLP drops at a roughly linear rate in Fig. 6.4(a), whereas task sets remain schedulable for the entire range of *mcsl* under the GIPP and the OMIP. As task sets become larger, the effect of token-contention becomes yet more apparent, as shown in Fig. 6.4(b). This demonstrates that a single global token-lock ultimately becomes a bottleneck for otherwise schedulable task sets.



(a) Schedulability quickly decreases under the CA-RNLP.

(b) Almost no schedulability under the CA-RNLP.

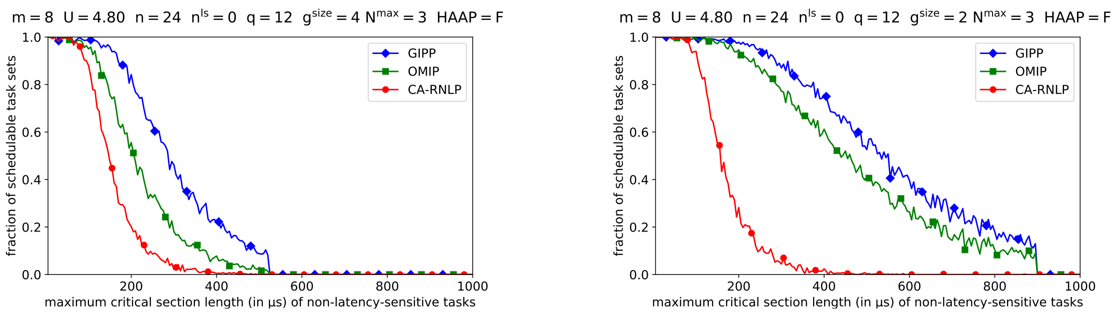
Figure 6.3: The presence of latency-sensitive tasks dominates schedulability.



(a) Task sets are schedulable under the GIPP and the OMIP for entire *mcsl* range.

(b) Effect of bottleneck becomes worse as task sets become larger.

Figure 6.4: The CA-RNLP's use of a single global token-lock becomes a bottleneck for resource acquisition.



(a) Similar behavior among all three protocols under high resource-contention.

(b) Schedulability improves under the GIPP and the OMIP when groups become smaller.

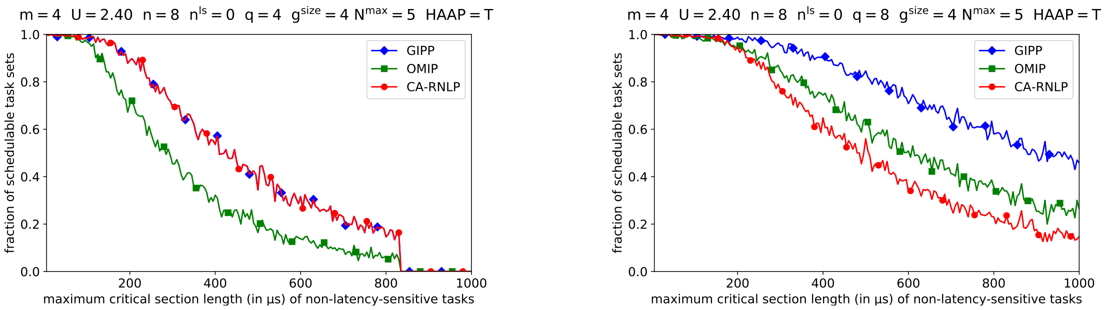
Figure 6.5: Resource contention dominates schedulability.

6.3.4 Performance under High Resource-Contention

The benefits of (group) independence preservation diminish under high contention for all resources. This is shown in Fig. 6.5(a), where roughly the same pattern of schedulability is seen under all three protocols. In contrast, the benefits of independence preservation are more clearly seen when there is a greater degree of isolation as in Fig. 6.5(b).

6.3.5 Performance under Varying Access Patterns

The rules and structure of the GIPP and the CA-RNLP allow for top-level resources to be acquired independently, which is not possible with the OMIP's group locks. Fig. 6.6(a) demonstrates that fine-grained locking offers a noticeable increase in schedulability over group locks in the presence of tasks with HAAPs; the GIPP and the CA-RNLP perform identically, whereas schedulability under the OMIP suffers due to group-lock contention. However, once resources are split into two groups as in Fig. 6.6(b), the CA-RNLP performs worse than the OMIP, which further suggests that the use of a single global token-lock serves as a bottleneck for schedulability.



(a) Schedulability under the OMIP suffers due to the use of group locks.

(b) The bottleneck effect of a global token-lock outweighs the advantages of fine-grained locking.

Figure 6.6: Group-lock and token contention dominate schedulability.

6.4 Technical Details

In the final section of this chapter, I provide a brief outline on some of the technical details regarding the actual execution of the large-scale experiments. The fined-grained analysis I implemented for the GIPP, as well as the OMIP's analysis are available in the SchedCAT library; both are implemented in C++ with the use of the GNU Linear Programming Kit (GLPK) [28]. The SchedCAT code used to generate the task sets is written in python. The python code generates task sets for a given set of parameters and then passes the task sets onto the C++ code via bindings generated with the Simplified Wrapper and Interface Generator (SWIG) [46]. The general idea was to use python for task set generation, plot production, and other data manipulation tasks due to its rich package database and quick development time, and then outsourcing the "heavy lifting" to the C++ code.

Running the experiments took considerable processing power. In particular, the code that solves the LPs ran for roughly one week on 288 Intel Xeon E7-8857 v2 cores. The computations were almost entirely CPU bound and thus did not require any significant amount of memory or disk space.

All of the code used to conduct the experiments, process the results, and produce the plots is available at `http://www.jamesrobb.ca/downloads/technical_writing/thesis/gipp_code.tar.bz2`.

Chapter 7

Conclusion

I have examined and defined what it means to be independence-preserving in the context of fine-grained nested locking. On the one hand, I have established that *outer-lock independence preservation* yields non-optimal bounds on s-oblivious pi-blocking. On the other hand, I demonstrate that *group independence preservation* and support for fine-grained nested locking can be realized jointly with asymptotically optimal pi-blocking bounds (under s-oblivious analysis) via the GIPP, the first multiprocessor real-time protocol to accomplish this trifecta.

To realize the GIPP, I constructed the CKIP as a building block, which is noteworthy in itself as it is the first asymptotically optimal, non-nested independence-preserving, k-exclusion lock for clustered scheduling.

Finally, I demonstrated via fined-grained pi-blocking analysis and empirical experiments that group independence preservation alleviates the bottleneck imposed by a single token-lock in the RNLP (as well as group locks under the OMIP), while also being able to support workloads with latency-sensitive tasks.

In future work, it would be interesting to extend the GIPP to semi-partitioned scheduling [3, 6, 18]. It will also be necessary to study the real-world overheads (*e.g.*, cache misses, TLB flushes, inter-processor interrupts, *etc.*), which the GIPP is particularly exposed to due to its use of allocation inheritance, in a practical system such as LITMUS$^{\mathrm{RT}}$ [13, 21].

Bibliography

[1] Jaume Abella et al. "Towards Improved Survivability in Safety-Critical Systems". In: *IEEE International On-Line Testing Symposium (IOLTS'11)*. 2011. DOI: 10.1109/IOLTS.2011.5994536.

[2] AbsInt. *aiT Worst-Case Execution Time Analyzers*. en. 2020. URL: https://www.absint.com/ait/index.htm (visited on 03/24/2020).

[3] James H. Anderson, Vasile Bud, and UmaMaheswari C. Devi. "An EDF-Based Scheduling Algorithm for Multiprocessor Soft Real-Time Systems". In: *17th Euromicro Conference on Real-Time Systems (ECRTS'05)*. 2005. DOI: 10.1109/ECRTS.2005.6.

[4] AUTOSAR. "AUTOSAR Release 4.1, Specification of Operating System." In: (2013). URL: www.autosar.org.

[5] Sanjoy K. Baruah, Aloysius K. Mok, and Louis E. Rosier. "Preemptively Scheduling Hard-Real-Time Sporadic Tasks on One Processor". In: *11th Real-Time Systems Symposium (RTSS'90)*. 1990. DOI: 10.1109/REAL.1990.128746.

[6] Andrea Bastoni, Björn B. Brandenburg, and James H. Anderson. "Is Semi-Partitioned Scheduling Practical?" In: *23rd Euromicro Conference on Real-Time Systems (ECRTS'11)*. 2011. DOI: 10.1109/ECRTS.2011.20.

[7] Alessandro Biondi, Björn B. Brandenburg, and Alexander Wieder. "A Blocking Bound for Nested FIFO Spin Locks". In: *38th Real-Time Systems Symposium (RTSS'17)*. 2017. DOI: 10.1109/RTSS.2016.036.

[8] Aaron Block et al. "A Flexible Real-Time Locking Protocol for Multiprocessors". In: *13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'07)*. IEEE, 2007. ISBN: 0-7695-2975-5. DOI: 10.1109/RTCSA.2007.8.

[9] Björn B. Brandenburg. "A Fully Preemptive Multiprocessor Semaphore Protocol for Latency-Sensitive Real-Time Applications". In: *25th Euromicro Conference on Real-Time Systems (ECRTS'13)*. 2013. DOI: 10.1109/ECRTS.2013.38.

[10] Björn B. Brandenburg. "Improved Analysis and Evaluation of Real-Time Semaphore Protocols for P-FP Scheduling". In: *19th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'13)*. 2013. DOI: 10.1109/RTAS.2013.6531087.

[11] Björn B. Brandenburg. "Multiprocessor Real-Time Locking Protocols: A Systematic Review". In: *arXiv:1909.09600 [cs]* (2019). arXiv: 1909.09600 [cs]. URL: http://arxiv.org/abs/1909.09600 (visited on 04/20/2020).

[12] Björn B. Brandenburg. *SchedCAT: The Schedulability Test Collection and Toolkit.* Jan. 2020. URL: https://github.com/brandenburg/schedcat (visited on 03/17/2020).

[13] Björn B. Brandenburg. "Scheduling and Locking in Multiprocessor Real-Time Operating Systems". PhD thesis. University of North Carolina at Chapel Hill, 2011.

[14] Björn B. Brandenburg and James H. Anderson. "Optimality Results for Multiprocessor Real-Time Locking". In: *31st IEEE Real-Time Systems Symposium (RTSS'10)*. IEEE, 2010. ISBN: 978-0-7695-4298-0. DOI: `10.1109/RTSS.2010.17`.

[15] Björn B. Brandenburg and James H. Anderson. "Real-Time Resource-Sharing under Clustered Scheduling: Mutex, Reader-Writer, and k-Exclusion Locks". In: *9th ACM International Conference on Embedded Software (EMSOFT'11)*. 2011. ISBN: 978-1-4503-0714-7. DOI: `10.1145/2038642.2038655`.

[16] Björn B. Brandenburg and Andrea Bastoni. "The Case for Migratory Priority Inheritance in Linux: Bounded Priority Inversions on Multiprocessors". In: *14th Real Time Linux Workshop (RTLWS'12)*. 2012.

[17] Björn B. Brandenburg, John M. Calandrino, and James H. Anderson. "On the Scalability of Real-Time Scheduling Algorithms on Multicore Platforms: A Case Study". In: *29th Real-Time Systems Symposium (RTSS'08)*. 2008. DOI: `10.1109/RTSS.2008.23`.

[18] Björn B. Brandenburg and Mahircan Gul. "Global Scheduling Not Required: Simple, Near-Optimal Multiprocessor Real-Time Scheduling with Semi-Partitioned Reservations". In: *37th IEEE Real-Time Systems Symposium (RTSS'16)*. IEEE, 2016. ISBN: 978-1-5090-5303-2. DOI: `10.1109/RTSS.2016.019`.

[19] Alan Burns and Andy J. Wellings. "A Schedulability Compatible Multiprocessor Resource Sharing Protocol - MrsP". In: *25th Euromicro Conference on Real-Time Systems (ECRTS'13)*. 2013. DOI: `10.1109/ECRTS.2013.37`.

[20] Giorgio C. Buttazzo. *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*. Vol. 24. Springer Science & Business Media, 2011.

[21] John M. Calandrino et al. "LITMUS$^{\mathrm{RT}}$: A Testbed for Empirically Comparing Real-Time Multiprocessor Schedulers". In: *27th IEEE Real-Time Systems Symposium (RTSS'06)*. 2006. DOI: `10.1109/RTSS.2006.27`.

[22] Robert I. Davis and Alan Burns. *A Survey of Hard Real-Time Scheduling for Multiprocessor Systems*. 2011. URL: `https://doi.org/10.1145/1978802.1978814` (visited on 04/14/2020).

[23] Sudarshan K. Dhall and Chung L. Liu. "On a Real-Time Scheduling Problem". In: *Operations Research* 26.1 (1978), pp. 127–140. ISSN: 0030-364X. DOI: `10.1287/opre.26.1.127`.

[24] Arvind Easwaran and Björn Andersson. "Resource Sharing in Global Fixed-Priority Preemptive Multiprocessor Scheduling". In: *30th IEEE Real-Time Systems Symposium (RTSS'09)*. 2009. DOI: `10.1109/RTSS.2009.37`.

[25] Paul Emberson, Roger Stafford, and Robert I. Davis. "Techniques for the Synthesis of Multiprocessor Tasksets". In: *Proceedings of the 1st International Workshop on Analysis Tools and Methodologies for Embedded and Real-Time Systems (WATERS'10)*. 2010.

[26] Dario Faggioli, Giuseppe Lipari, and Tommaso Cucinotta. "Analysis and Implementation of the Multiprocessor Bandwidth Inheritance Protocol". en. In: *Real-Time Systems* 48.6 (2012), pp. 789–825. ISSN: 1573-1383. DOI: 10.1007/s11241-012-9162-0.

[27] Dario Faggioli, Giuseppe Lipari, and Tommaso Cucinotta. "The Multiprocessor Bandwidth Inheritance Protocol". In: *22nd Euromicro Conference on Real-Time Systems (ECRTS'10)*. 2010. DOI: 10.1109/ECRTS.2010.19.

[28] Free Software Foundation (FSF) GNU Project. *GLPK (GNU Linear Programming Kit)*. Apr. 2020. URL: https://www.gnu.org/software/glpk/ (visited on 04/05/2020).

[29] Philip Holman. "On the Implementation of Pfair-Scheduled Multiprocessor Systems". PhD thesis. University of North Carolina at Chapel Hill, 2004.

[30] Philip Holman and James H. Anderson. "Locking Under Pfair Scheduling". In: *ACM Transactions on Computer Systems* 24.2 (2006), pp. 140–174. ISSN: 0734-2071. DOI: 10.1145/1132026.1132028.

[31] Philip Holman and James H. Anderson. "Object Sharing in Pfair-Scheduled Multiprocessor Systems". In: *14th Euromicro Conference on Real-Time Systems (ECRTS'02)*. 2002. DOI: 10.1109/EMRTS.2002.1019191.

[32] Catherine E. Jarrett, Bryan C. Ward, and James H. Anderson. "A Contention-Sensitive Fine-Grained Locking Protocol for Multiprocessor Real-Time Systems". In: *23rd International Conference on Real Time and Networks Systems (RTNS'15)*. 2015. ISBN: 978-1-4503-3591-1. DOI: 10.1145/2834848.2834874.

[33] Mathai Joseph and Paritosh K. Pandya. "Finding Response Times in a Real-Time System". en. In: *The Computer Journal* 29.5 (1986), pp. 390–395. ISSN: 0010-4620. DOI: 10.1093/comjnl/29.5.390.

[34] Bernhard Korte and Jens Vygen. "Bin-Packing". In: *Kombinatorische Optimierung*. Springer, 2012, pp. 499–516. DOI: 10.1007/3-540-29297-7_18.

[35] Simon Kramer, Dirk Ziegenbein, and Arne Hamann. "Real World Automotive Benchmarks for Free". In: *6th International Workshop on Analysis Tools and Methodologies for Embedded and Real-Time Systems (WATERS'15)*. 2015.

[36] Joseph Y. -T. Leung and Jennifer Whitehead. "On the Complexity of Fixed-Priority Scheduling of Periodic, Real-Time Tasks". en. In: *Performance Evaluation* 2.4 (1982), pp. 237–250. ISSN: 0166-5316. DOI: 10.1016/0166-5316(82)90024-4.

[37] Chung L. Liu and James W. Layland. "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment". In: *Journal of the ACM* 20.1 (1973), pp. 46–61. ISSN: 0004-5411. DOI: 10.1145/321738.321743.

[38] Aloysius K. Mok. "Fundamental Design Problems of Distributed Systems for the Hard-Real-Time Environment." PhD thesis. Massachusetts Institute of Technology, 1983. URL: http://hdl.%20handle.%20net/1721.1/15670.

[39] Catherine E. Nemitz, Tanya Amert, and James H. Anderson. "Real-Time Multiprocessor Locks with Nesting: Optimizing the Common Case". en. In: *Real-Time Systems* 55.2 (2019), pp. 296–348. ISSN: 1573-1383. DOI: 10.1007/s11241-019-09328-w.

[40] Catherine E. Nemitz, Tanya Amert, and James H. Anderson. “Using Lock Servers to Scale Real-Time Locking Protocols: Chasing Ever-Increasing Core Counts”. In: *30th Euromicro Conference on Real-Time Systems (ECRTS’18)*. Ed. by Sebastian Altmeyer. 2018. ISBN: 978-3-95977-075-0. DOI: `10.4230/LIPIcs.ECRTS.2018.25`.

[41] R. Rajkumar. “Real-Time Synchronization Protocols for Shared Memory Multiprocessors”. In: *10th International Conference on Distributed Computing Systems (ICDCS’90)*. 1990. ISBN: 0-8186-2048-X. DOI: `10.1109/ICDCS.1990.89257`.

[42] Ragunathan Rajkumar. *Synchronization in Real-Time Systems: A Priority Inheritance Approach*. USA: Kluwer Academic Publishers, 1991. ISBN: 0-7923-9211-6.

[43] James Robb and Björn B. Brandenburg. “Nested, but Separate: Isolating Unrelated Critical Sections in Real-Time Nested Locking”. In: *32nd Euromicro Conference on Real-Time Systems (ECRTS’20)*. 2020.

[44] Liu Sha, Ragunathan Rajkumar, and John P. Lehoczky. “Priority Inheritance Protocols: An Approach to Real-Time Synchronization”. In: *IEEE Transactions on Computers* 39.9 (1990), pp. 1175–1185. ISSN: 00189340. DOI: `10.1109/12.57058`.

[45] Lui Sha et al. “Real-Time Scheduling Theory: A Historical Perspective”. In: *Real-Time Systems* 28.2-3 SPEC. ISS. (2004), pp. 101–155. ISSN: 09226443. DOI: `10.1023/B:TIME.0000045315.61234.1e`.

[46] *Simplified Wrapper and Interface Generator*. Apr. 2020. URL: `http://www.swig.org/` (visited on 04/05/2020).

[47] Anand Srinivasan and James H. Anderson. “Optimal Rate-Based Scheduling on Multiprocessors”. en. In: *Journal of Computer and System Sciences* 72.6 (2006), pp. 1094–1117. ISSN: 0022-0000. DOI: `10.1016/j.jcss.2006.03.001`.

[48] Hiroaki Takada and Ken Sakamura. “Real-Time Scalability of Nested Spin Locks”. In: *2nd IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA’95)*. 1995. DOI: `10.1109/rtcsa.1995.528766`.

[49] Ken Tindell and Alan Burns. “Guaranteeing Message Latencies on Control Area Network (CAN)”. In: *1st International CAN Conference*. Citeseer, 1994.

[50] Vasanth Venkatachalam and Michael Franz. *Power Reduction Techniques for Microprocessor Systems*. 2005. URL: `https://doi.org/10.1145/1108956.1108957` (visited on 04/01/2020).

[51] Bryan C. Ward and James H. Anderson. “Fine-Grained Multiprocessor Real-Time Locking with Improved Blocking”. In: *21st International Conference on Real Time and Networks Systems (RTNS’13)*. 2013. ISBN: 978-1-4503-2058-0. DOI: `10.1145/2516821.2516843`.

[52] Bryan C. Ward and James H. Anderson. “Multi-Resource Real-Time Reader/Writer Locks for Multiprocessors”. In: *28th IEEE International Parallel and Distributed Processing Symposium (IPDPS’14)*. 2014. DOI: `10.1109/IPDPS.2014.29`.

[53] Bryan C. Ward and James H. Anderson. "Supporting Nested Locking in Multiprocessor Real-Time Systems". In: *24th Euromicro Conference on Real-Time Systems (ECRTS'12)*. 2012. DOI: 10.1109/ECRTS.2012.17.

[54] Bryan C. Ward, Glenn A. Elliott, and James H. Anderson. "Replica-Request Priority Donation: A Real-Time Progress Mechanism for Global Locking Protocols". In: *18th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*. 2012. DOI: 10.1109/RTCSA.2012.26.

[55] Alexander Wieder and Björn B. Brandenburg. "On Spin Locks in AUTOSAR: Blocking Analysis of FIFO, Unordered, and Priority-Ordered Spin Locks". In: *34th IEEE Real-Time Systems Symposium (RTSS'13)*. 2013. DOI: 10.1109/RTSS.2013.13.

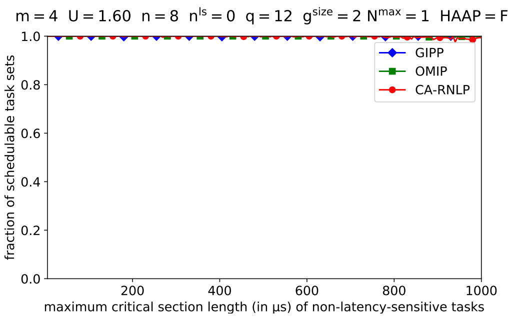
[56] Maolin Yang, Alexander Wieder, and Björn B. Brandenburg. "Global Real-Time Semaphore Protocols: A Survey, Unified Analysis, and Comparison". In: *36th IEEE Real-Time Systems Symposium (RTSS'15)*. 2015. DOI: 10.1109/RTSS.2015.8.

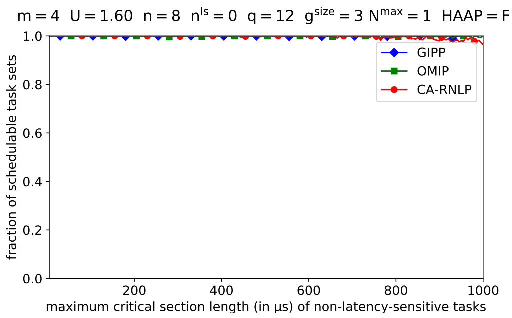
Appendix A

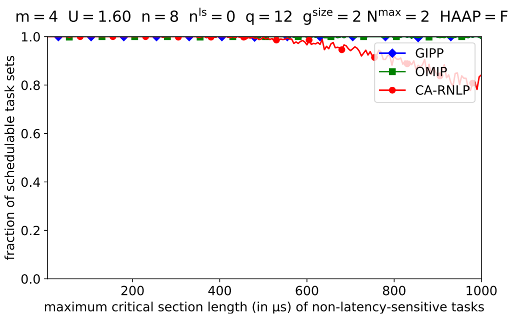
Full Results for Schedulability Experiments

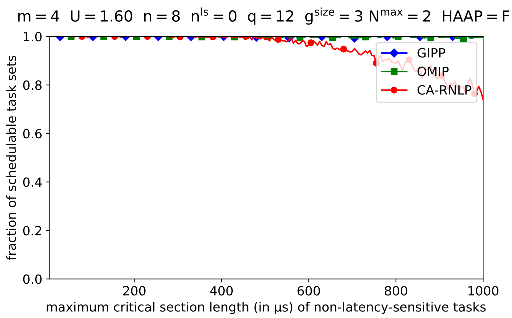
For the sake of completeness and transparency, the following two sections present the full results for the large-scale schedulability experiments presented in Chapter 6. The full results for the UAP experiments and the HAAP experiments are shown in Appendix A.1 and Appendix A.2, respectively.

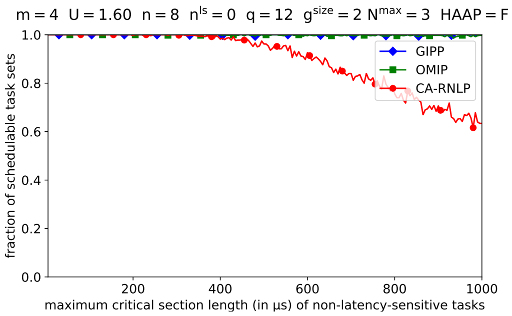
A.1 UAP Experiment Results

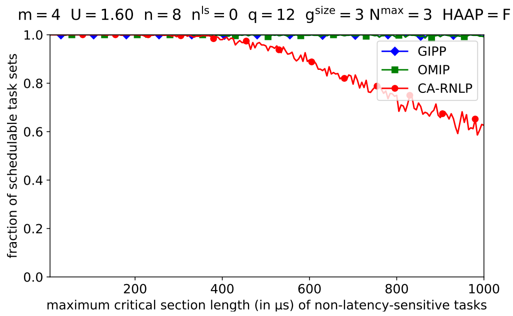


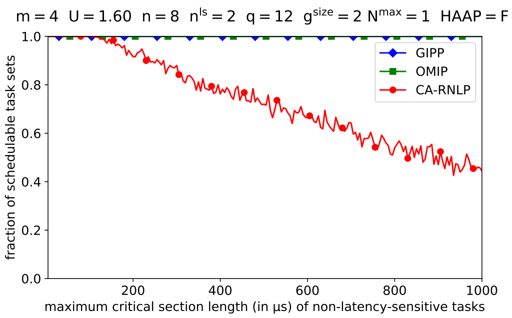


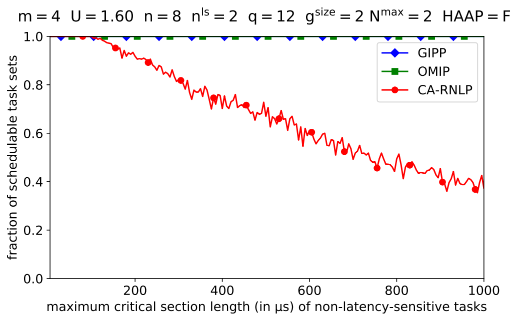


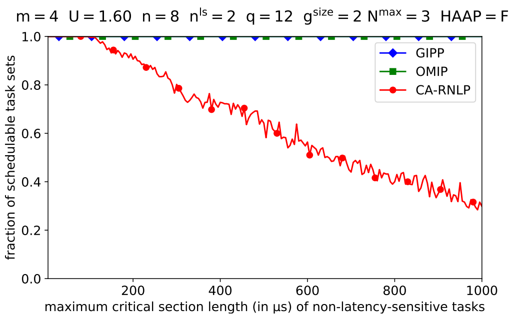


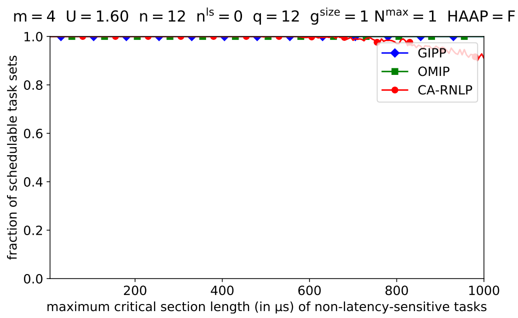

$m = 4 \quad U = 1.60 \quad n = 8 \quad n^{ls} = 0 \quad q = 12 \quad g^{size} = 2 \; N^{max} = 3 \quad HAAP = F$
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

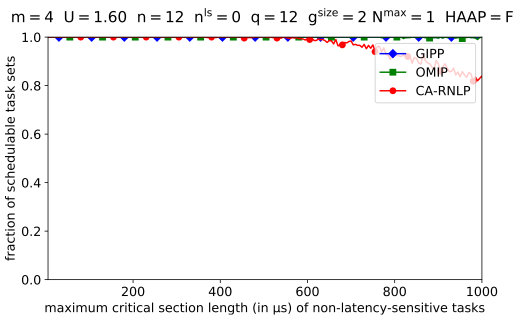

$m = 4 \quad U = 1.60 \quad n = 8 \quad n^{ls} = 0 \quad q = 12 \quad g^{size} = 3 \; N^{max} = 3 \quad HAAP = F$
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

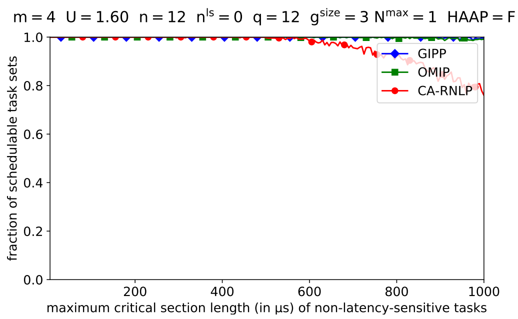

$m = 4 \quad U = 1.60 \quad n = 8 \quad n^{ls} = 2 \quad q = 12 \quad g^{size} = 2 \; N^{max} = 1 \quad HAAP = F$
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

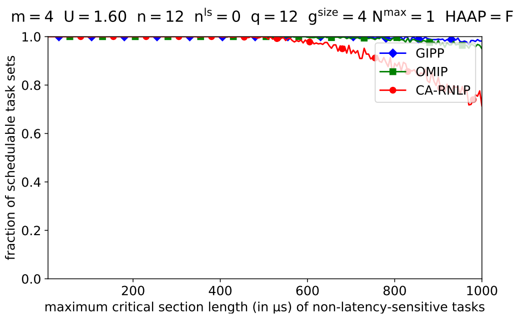

$m = 4 \quad U = 1.60 \quad n = 8 \quad n^{ls} = 2 \quad q = 12 \quad g^{size} = 2 \; N^{max} = 2 \quad HAAP = F$
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

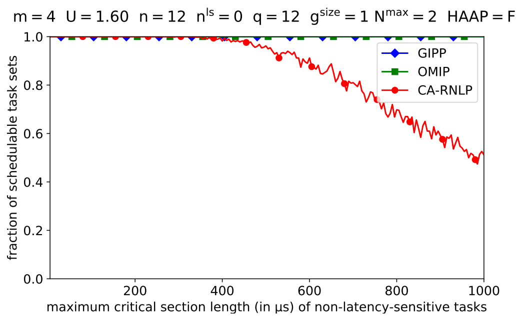

$m = 4 \quad U = 1.60 \quad n = 8 \quad n^{ls} = 2 \quad q = 12 \quad g^{size} = 2 \; N^{max} = 3 \quad HAAP = F$
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

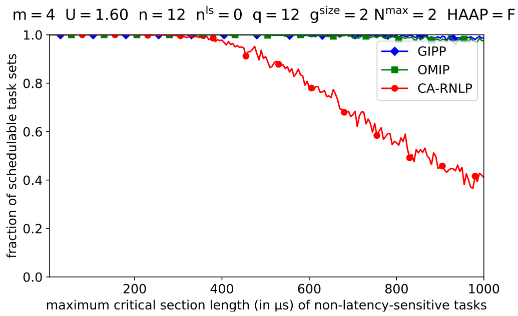

$m = 4 \quad U = 1.60 \quad n = 12 \quad n^{ls} = 0 \quad q = 12 \quad g^{size} = 1 \; N^{max} = 1 \quad HAAP = F$
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

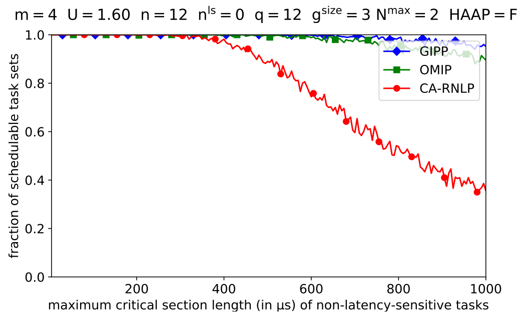

$m = 4 \quad U = 1.60 \quad n = 12 \quad n^{ls} = 0 \quad q = 12 \quad g^{size} = 2 \; N^{max} = 1 \quad HAAP = F$
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

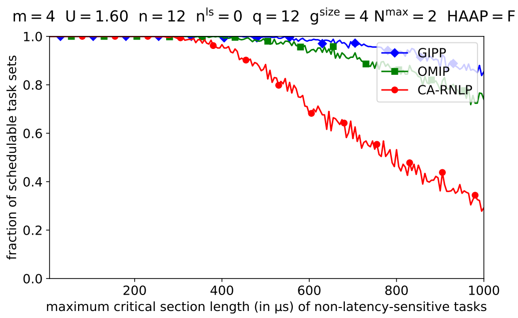

$m = 4 \quad U = 1.60 \quad n = 12 \quad n^{ls} = 0 \quad q = 12 \quad g^{size} = 3 \; N^{max} = 1 \quad HAAP = F$
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

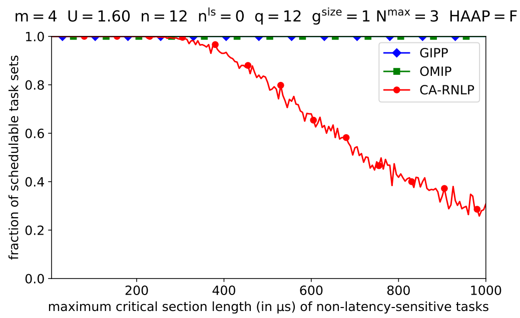

$m = 4 \quad U = 1.60 \quad n = 12 \quad n^{ls} = 0 \quad q = 12 \quad g^{size} = 4 \; N^{max} = 1 \quad HAAP = F$
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

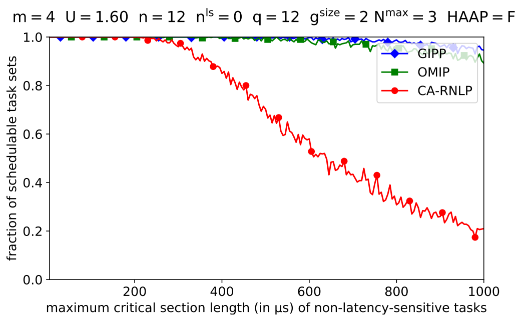

$m = 4 \quad U = 1.60 \quad n = 12 \quad n^{ls} = 0 \quad q = 12 \quad g^{size} = 1 \; N^{max} = 2 \quad HAAP = F$
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

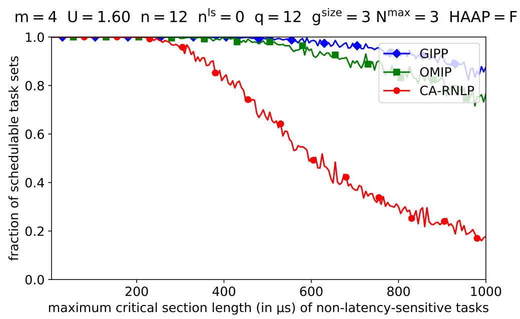

m = 4 U = 1.60 n = 12 n^{ls} = 0 q = 12 g^{size} = 2 N^{max} = 2 HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks

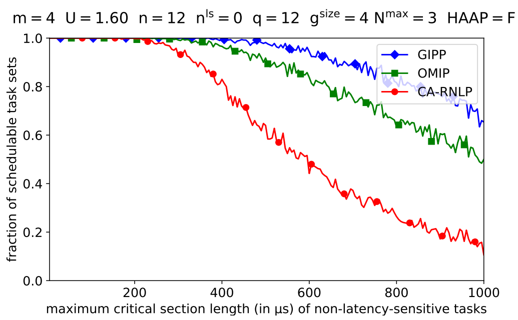

m = 4 U = 1.60 n = 12 n^{ls} = 0 q = 12 g^{size} = 3 N^{max} = 2 HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks

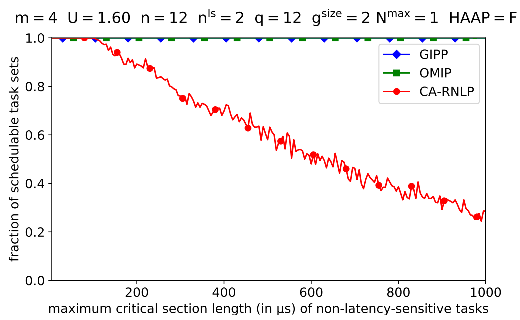

m = 4 U = 1.60 n = 12 n^{ls} = 0 q = 12 g^{size} = 4 N^{max} = 2 HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks

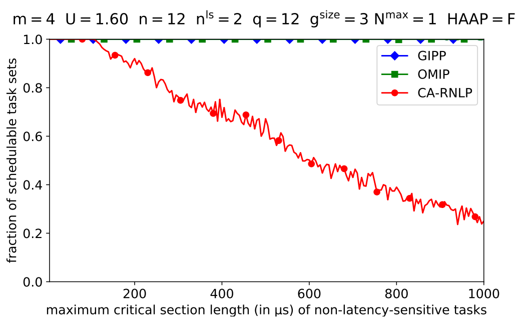

m = 4 U = 1.60 n = 12 n^{ls} = 0 q = 12 g^{size} = 1 N^{max} = 3 HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks

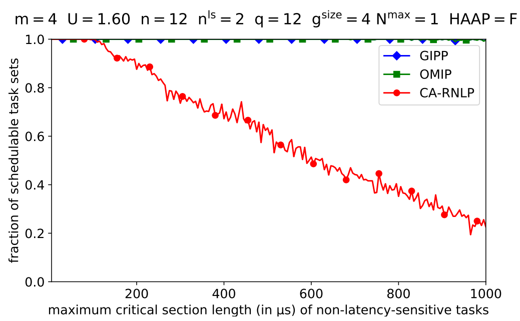

m = 4 U = 1.60 n = 12 n^{ls} = 0 q = 12 g^{size} = 2 N^{max} = 3 HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks

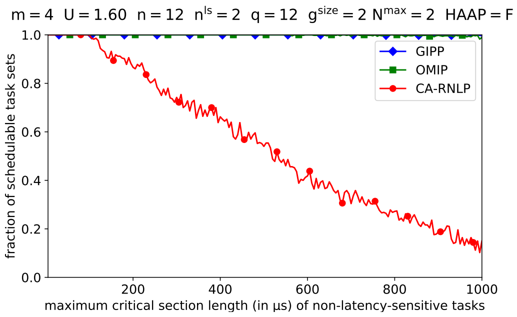

m = 4 U = 1.60 n = 12 n^{ls} = 0 q = 12 g^{size} = 3 N^{max} = 3 HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks

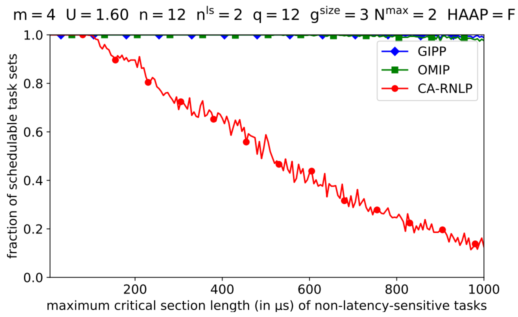

m = 4 U = 1.60 n = 12 n^{ls} = 0 q = 12 g^{size} = 4 N^{max} = 3 HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks

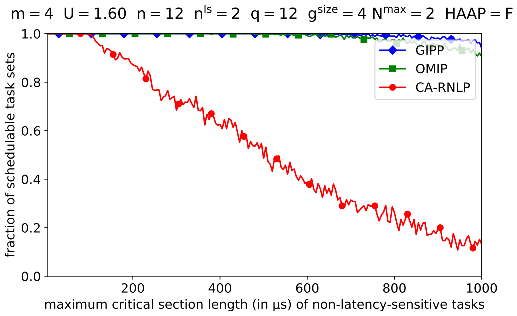

m = 4 U = 1.60 n = 12 n^{ls} = 2 q = 12 g^{size} = 2 N^{max} = 1 HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks

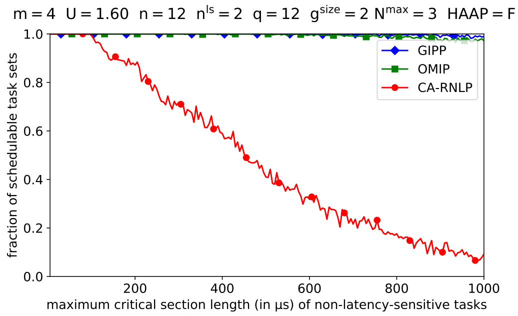

m = 4 U = 1.60 n = 12 n^{ls} = 2 q = 12 g^{size} = 3 N^{max} = 1 HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks

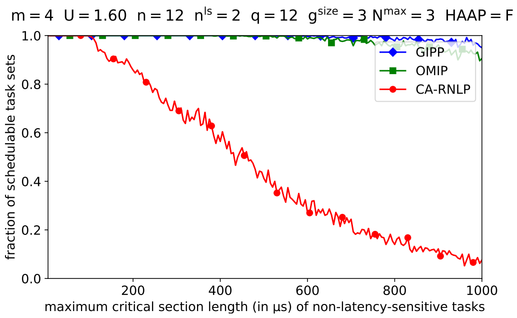

m = 4 U = 1.60 n = 12 n^{ls} = 2 q = 12 g^{size} = 4 N^{max} = 1 HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks

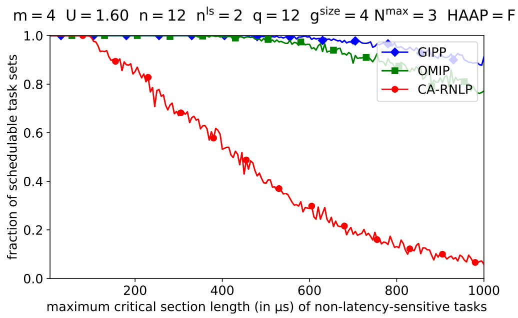

$m = 4$ $U = 1.60$ $n = 12$ $n^{ls} = 2$ $q = 12$ $g^{size} = 2$ $N^{max} = 2$ $HAAP = F$
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

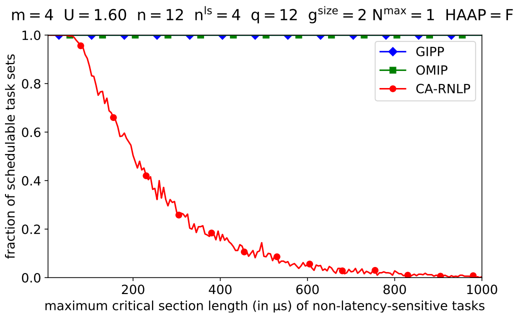

$m = 4$ $U = 1.60$ $n = 12$ $n^{ls} = 2$ $q = 12$ $g^{size} = 3$ $N^{max} = 2$ $HAAP = F$
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

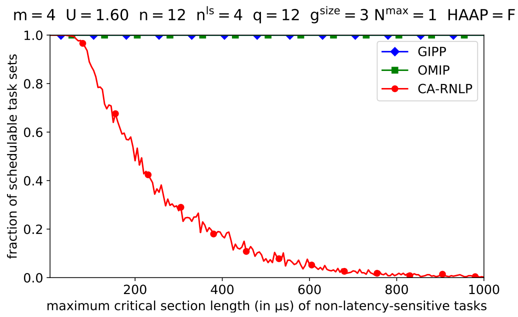

$m = 4$ $U = 1.60$ $n = 12$ $n^{ls} = 2$ $q = 12$ $g^{size} = 4$ $N^{max} = 2$ $HAAP = F$
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

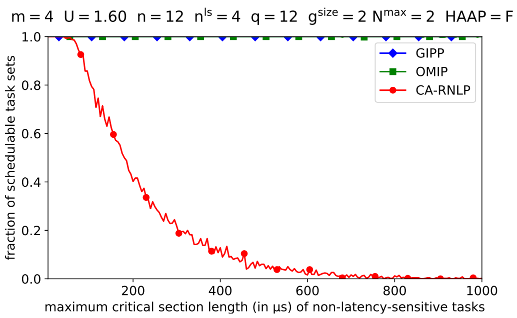

$m = 4$ $U = 1.60$ $n = 12$ $n^{ls} = 2$ $q = 12$ $g^{size} = 2$ $N^{max} = 3$ $HAAP = F$
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

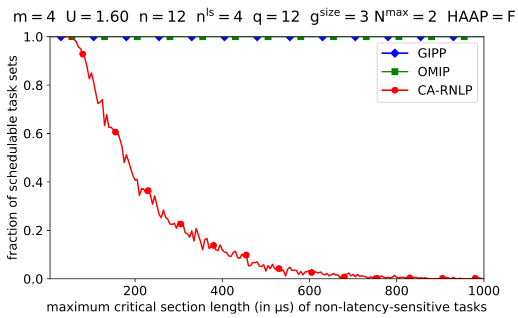

$m = 4$ $U = 1.60$ $n = 12$ $n^{ls} = 2$ $q = 12$ $g^{size} = 3$ $N^{max} = 3$ $HAAP = F$
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

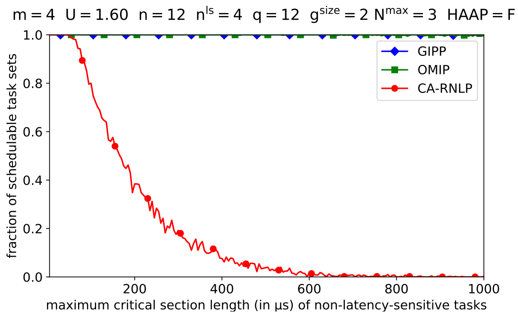

$m = 4$ $U = 1.60$ $n = 12$ $n^{ls} = 2$ $q = 12$ $g^{size} = 4$ $N^{max} = 3$ $HAAP = F$
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

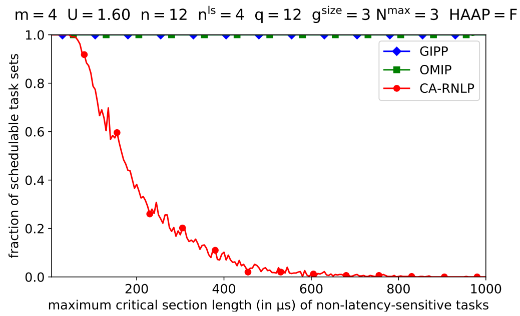

$m = 4$ $U = 1.60$ $n = 12$ $n^{ls} = 4$ $q = 12$ $g^{size} = 2$ $N^{max} = 1$ $HAAP = F$
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

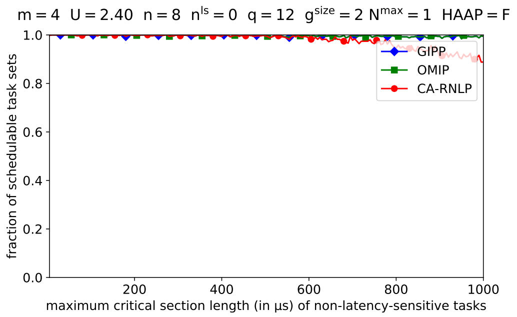

$m = 4$ $U = 1.60$ $n = 12$ $n^{ls} = 4$ $q = 12$ $g^{size} = 3$ $N^{max} = 1$ $HAAP = F$
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

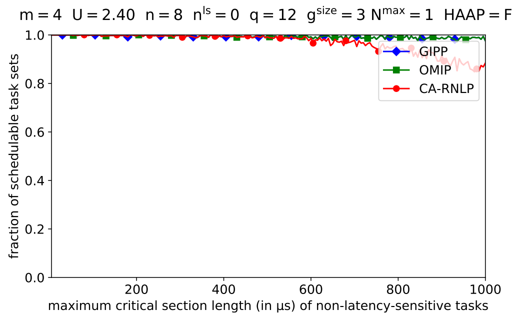

$m = 4$ $U = 1.60$ $n = 12$ $n^{ls} = 4$ $q = 12$ $g^{size} = 2$ $N^{max} = 2$ $HAAP = F$
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

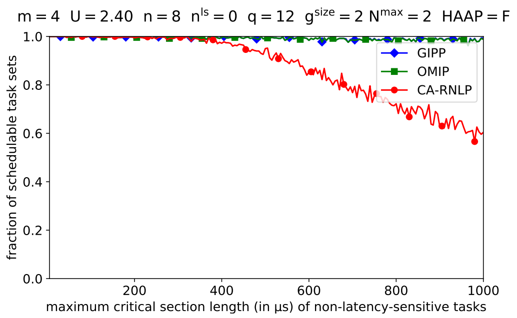

$m = 4$ $U = 1.60$ $n = 12$ $n^{ls} = 4$ $q = 12$ $g^{size} = 3$ $N^{max} = 2$ $HAAP = F$
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

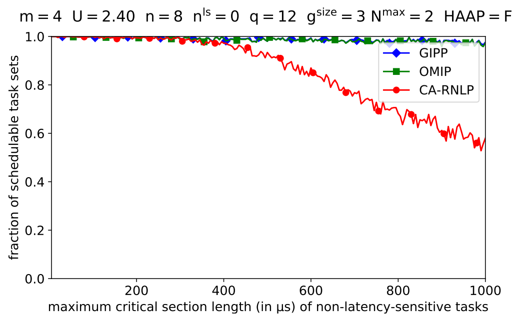

m = 4 U = 1.60 n = 12 n^ls = 4 q = 12 g^size = 2 N^max = 3 HAAP = F
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

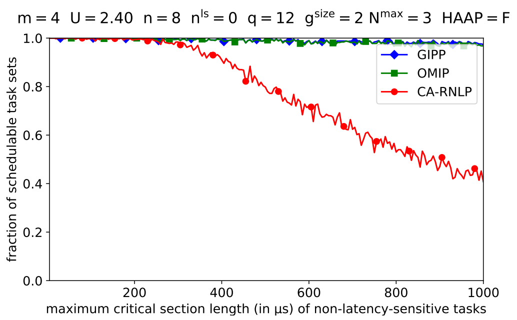

m = 4 U = 1.60 n = 12 n^ls = 4 q = 12 g^size = 3 N^max = 3 HAAP = F
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

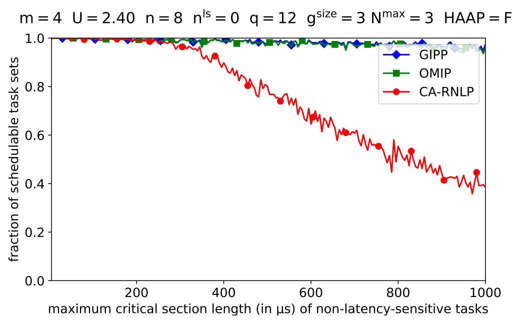

m = 4 U = 2.40 n = 8 n^ls = 0 q = 12 g^size = 2 N^max = 1 HAAP = F
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

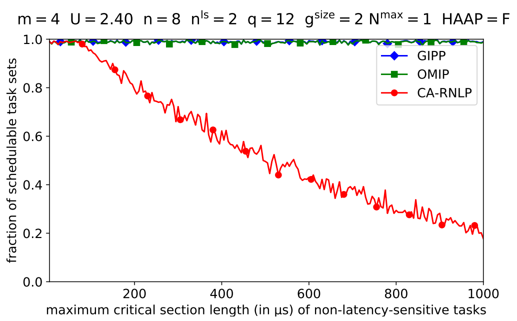

m = 4 U = 2.40 n = 8 n^ls = 0 q = 12 g^size = 3 N^max = 1 HAAP = F
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

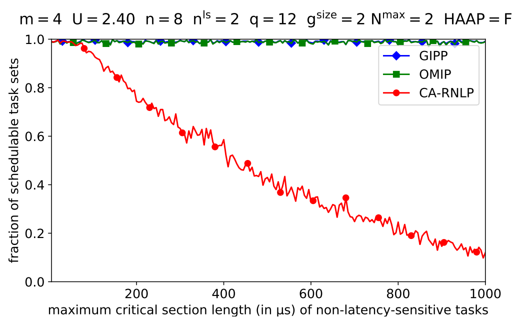

m = 4 U = 2.40 n = 8 n^ls = 0 q = 12 g^size = 2 N^max = 2 HAAP = F
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

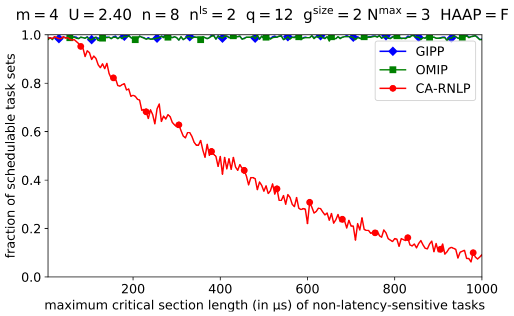

m = 4 U = 2.40 n = 8 n^ls = 0 q = 12 g^size = 3 N^max = 2 HAAP = F
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

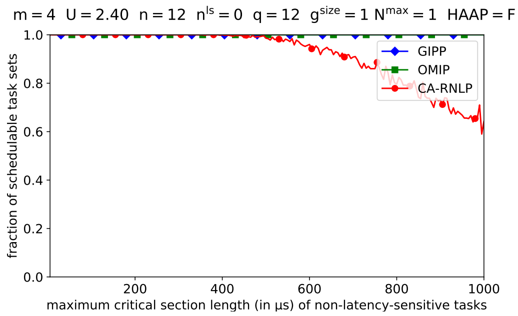

m = 4 U = 2.40 n = 8 n^ls = 0 q = 12 g^size = 2 N^max = 3 HAAP = F
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

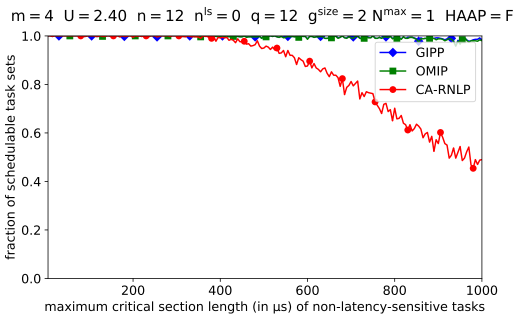

m = 4 U = 2.40 n = 8 n^ls = 0 q = 12 g^size = 3 N^max = 3 HAAP = F
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

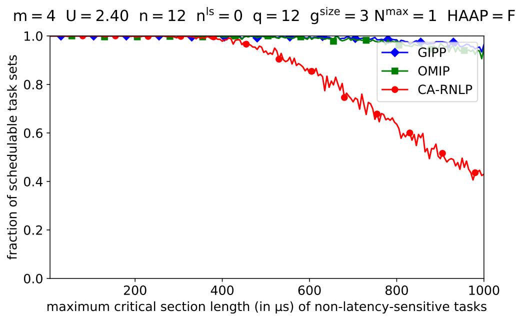

m = 4 U = 2.40 n = 8 n^ls = 2 q = 12 g^size = 2 N^max = 1 HAAP = F
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

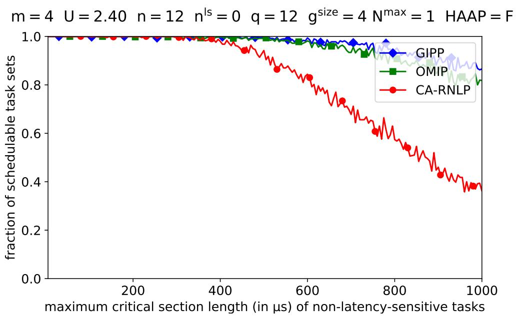

m = 4 U = 2.40 n = 8 n^ls = 2 q = 12 g^size = 2 N^max = 2 HAAP = F
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

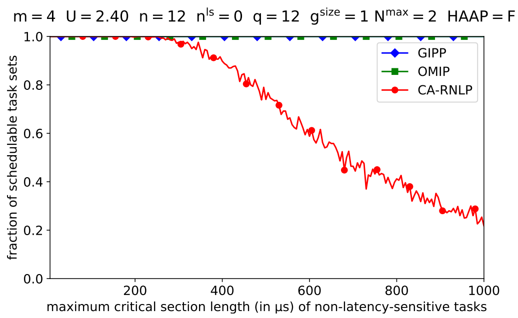

m = 4 U = 2.40 n = 8 n^{ls} = 2 q = 12 g^{size} = 2 N^{max} = 3 HAAP = F
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

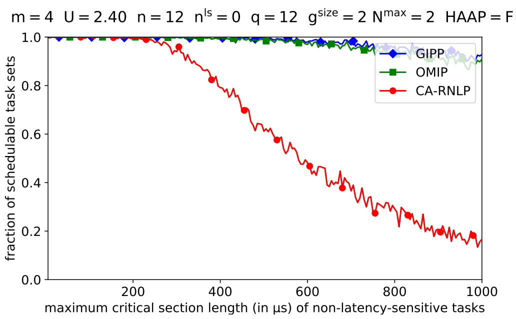

m = 4 U = 2.40 n = 12 n^{ls} = 0 q = 12 g^{size} = 1 N^{max} = 1 HAAP = F
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

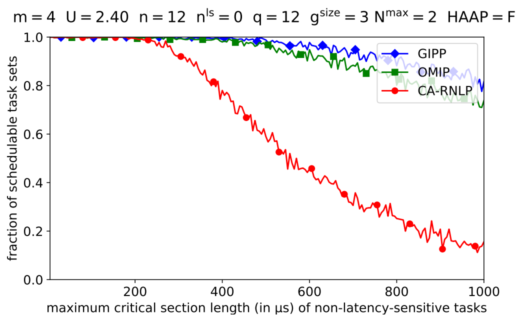

m = 4 U = 2.40 n = 12 n^{ls} = 0 q = 12 g^{size} = 2 N^{max} = 1 HAAP = F
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

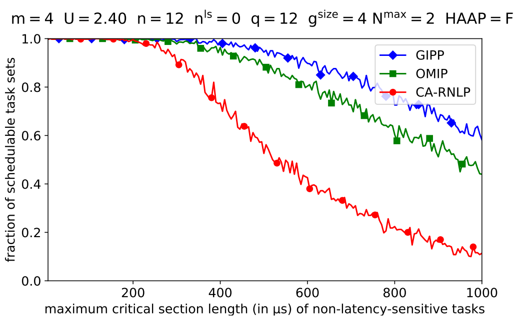

m = 4 U = 2.40 n = 12 n^{ls} = 0 q = 12 g^{size} = 3 N^{max} = 1 HAAP = F
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

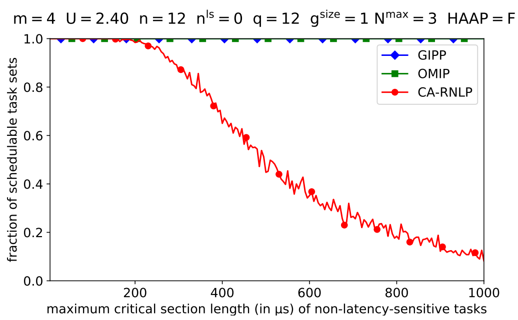

m = 4 U = 2.40 n = 12 n^{ls} = 0 q = 12 g^{size} = 4 N^{max} = 1 HAAP = F
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

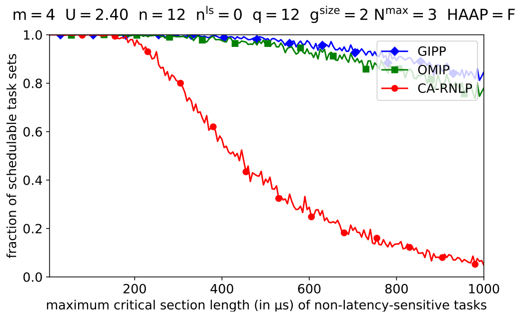

m = 4 U = 2.40 n = 12 n^{ls} = 0 q = 12 g^{size} = 1 N^{max} = 2 HAAP = F
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

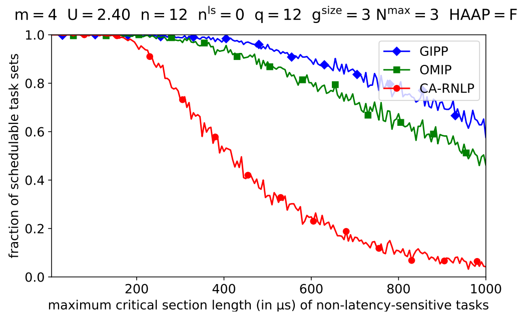

m = 4 U = 2.40 n = 12 n^{ls} = 0 q = 12 g^{size} = 2 N^{max} = 2 HAAP = F
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

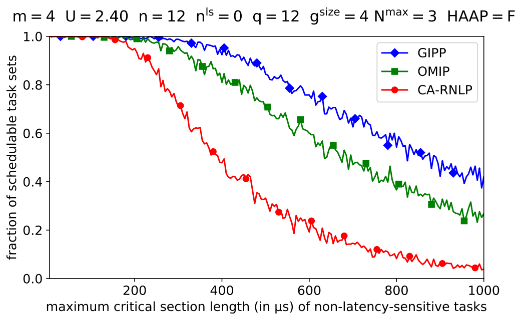

m = 4 U = 2.40 n = 12 n^{ls} = 0 q = 12 g^{size} = 3 N^{max} = 2 HAAP = F
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

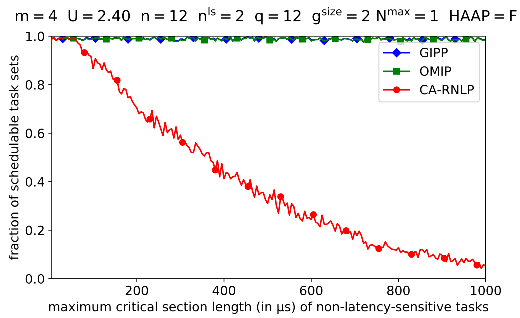

m = 4 U = 2.40 n = 12 n^{ls} = 0 q = 12 g^{size} = 4 N^{max} = 2 HAAP = F
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

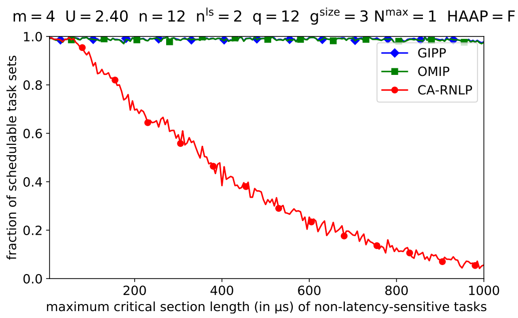

m = 4 U = 2.40 n = 12 n^{ls} = 0 q = 12 g^{size} = 1 N^{max} = 3 HAAP = F
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

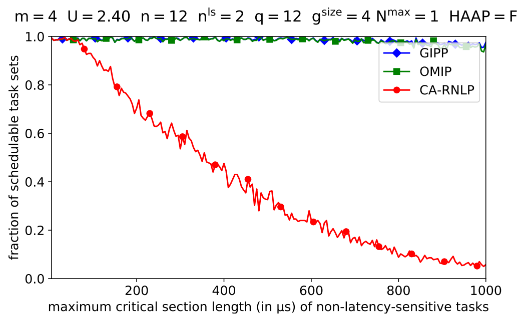

$m = 4 \quad U = 2.40 \quad n = 12 \quad n^{ls} = 0 \quad q = 12 \quad g^{size} = 2 \; N^{max} = 3 \quad HAAP = F$
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

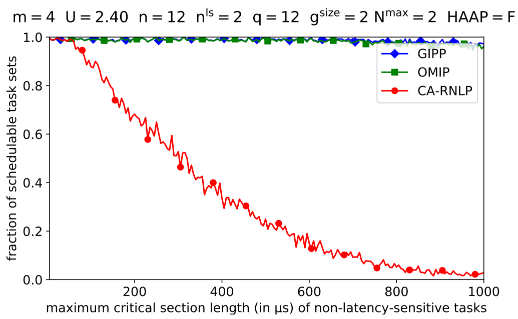

$m = 4 \quad U = 2.40 \quad n = 12 \quad n^{ls} = 0 \quad q = 12 \quad g^{size} = 3 \; N^{max} = 3 \quad HAAP = F$
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

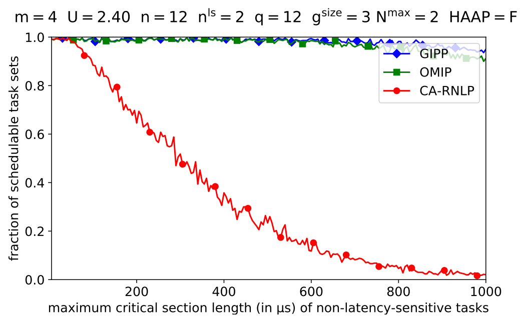

$m = 4 \quad U = 2.40 \quad n = 12 \quad n^{ls} = 0 \quad q = 12 \quad g^{size} = 4 \; N^{max} = 3 \quad HAAP = F$
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

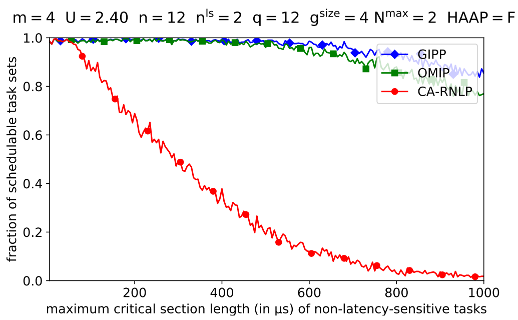

$m = 4 \quad U = 2.40 \quad n = 12 \quad n^{ls} = 2 \quad q = 12 \quad g^{size} = 2 \; N^{max} = 1 \quad HAAP = F$
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

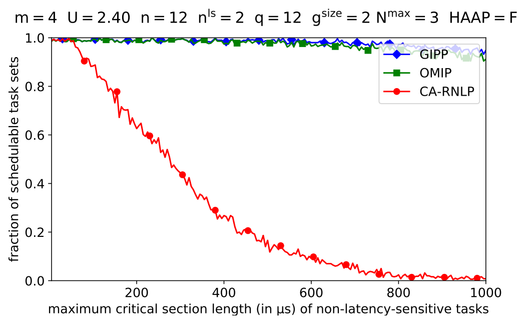

$m = 4 \quad U = 2.40 \quad n = 12 \quad n^{ls} = 2 \quad q = 12 \quad g^{size} = 3 \; N^{max} = 1 \quad HAAP = F$
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

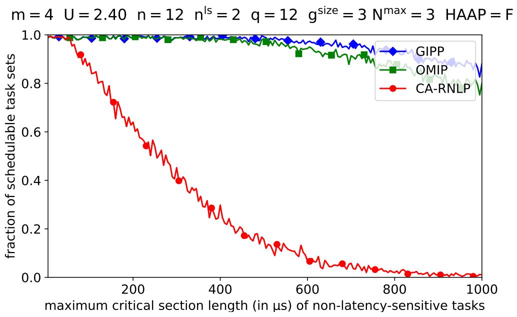

$m = 4 \quad U = 2.40 \quad n = 12 \quad n^{ls} = 2 \quad q = 12 \quad g^{size} = 4 \; N^{max} = 1 \quad HAAP = F$
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

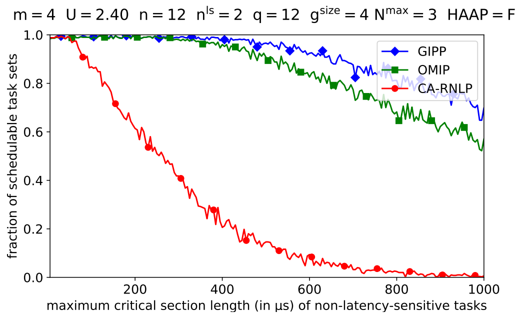

$m = 4 \quad U = 2.40 \quad n = 12 \quad n^{ls} = 2 \quad q = 12 \quad g^{size} = 2 \; N^{max} = 2 \quad HAAP = F$
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

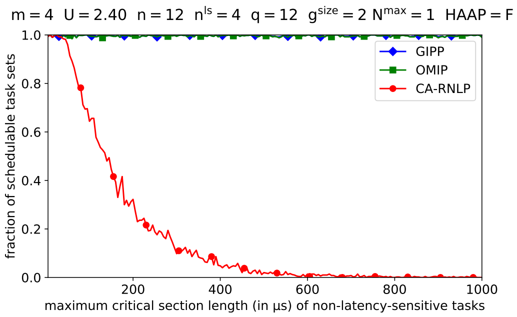

$m = 4 \quad U = 2.40 \quad n = 12 \quad n^{ls} = 2 \quad q = 12 \quad g^{size} = 3 \; N^{max} = 2 \quad HAAP = F$
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

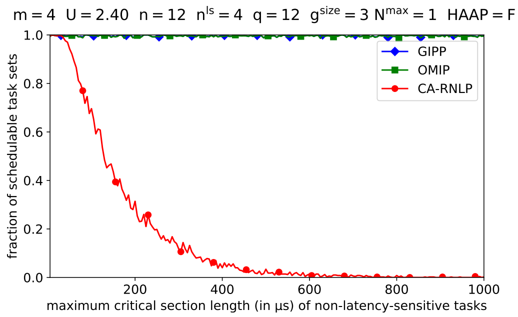

$m = 4 \quad U = 2.40 \quad n = 12 \quad n^{ls} = 2 \quad q = 12 \quad g^{size} = 4 \; N^{max} = 2 \quad HAAP = F$
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

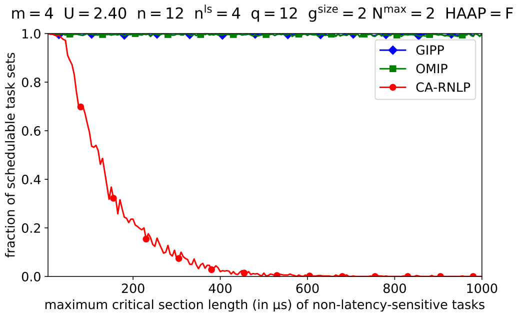

$m = 4 \quad U = 2.40 \quad n = 12 \quad n^{ls} = 2 \quad q = 12 \quad g^{size} = 2 \; N^{max} = 3 \quad HAAP = F$
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

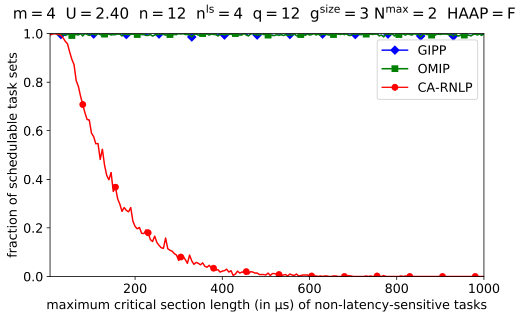

m = 4 U = 2.40 n = 12 $n^{ls} = 2$ q = 12 $g^{size} = 3$ $N^{max} = 3$ HAAP = F
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

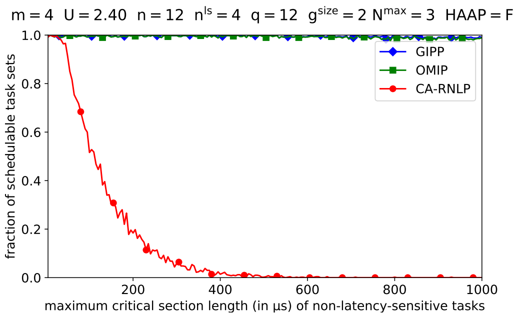

m = 4 U = 2.40 n = 12 $n^{ls} = 2$ q = 12 $g^{size} = 4$ $N^{max} = 3$ HAAP = F
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

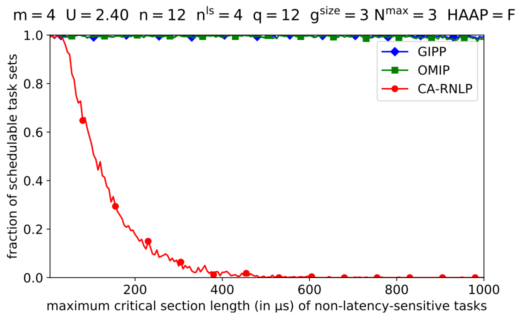

m = 4 U = 2.40 n = 12 $n^{ls} = 4$ q = 12 $g^{size} = 2$ $N^{max} = 1$ HAAP = F
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

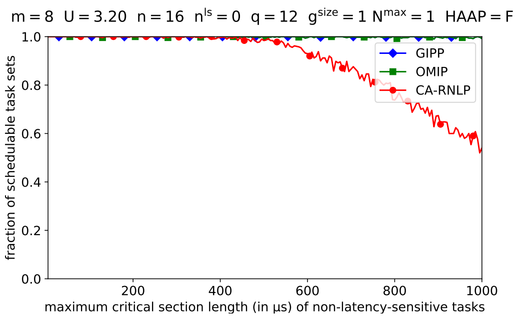

m = 4 U = 2.40 n = 12 $n^{ls} = 4$ q = 12 $g^{size} = 3$ $N^{max} = 1$ HAAP = F
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

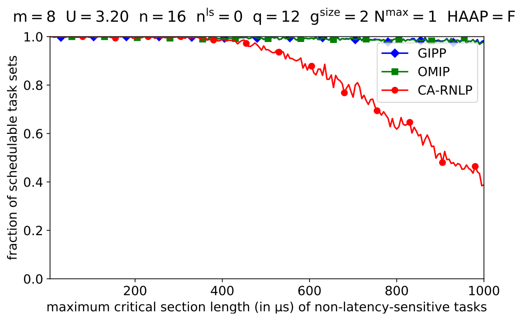

m = 4 U = 2.40 n = 12 $n^{ls} = 4$ q = 12 $g^{size} = 2$ $N^{max} = 2$ HAAP = F
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

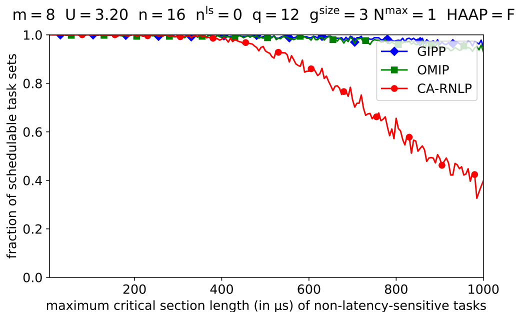

m = 4 U = 2.40 n = 12 $n^{ls} = 4$ q = 12 $g^{size} = 3$ $N^{max} = 2$ HAAP = F
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

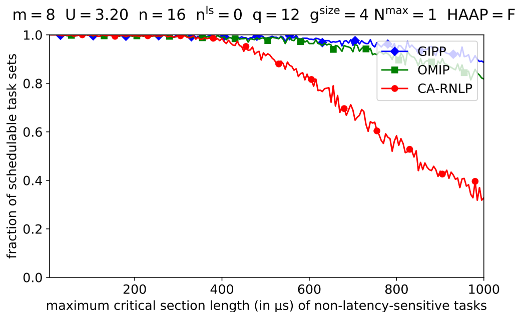

m = 4 U = 2.40 n = 12 $n^{ls} = 4$ q = 12 $g^{size} = 2$ $N^{max} = 3$ HAAP = F
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

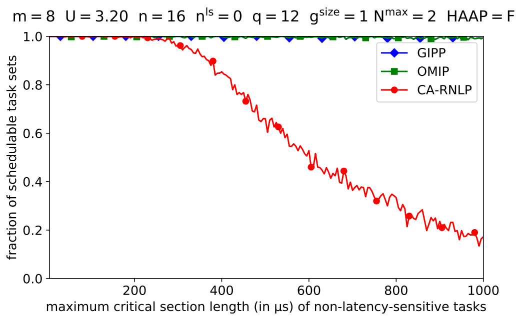

m = 4 U = 2.40 n = 12 $n^{ls} = 4$ q = 12 $g^{size} = 3$ $N^{max} = 3$ HAAP = F
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

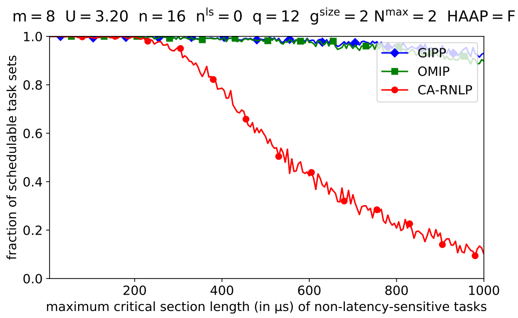

m = 8 U = 3.20 n = 16 $n^{ls} = 0$ q = 12 $g^{size} = 1$ $N^{max} = 1$ HAAP = F
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

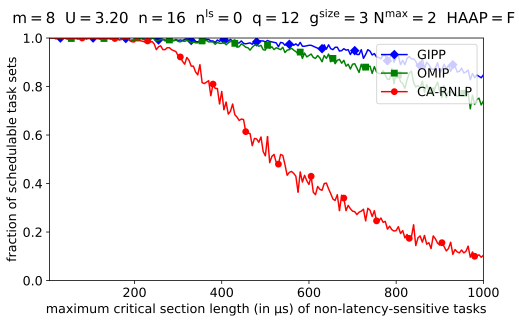

m = 8 U = 3.20 n = 16 $n^{ls} = 0$ q = 12 $g^{size} = 2$ $N^{max} = 1$ HAAP = F
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

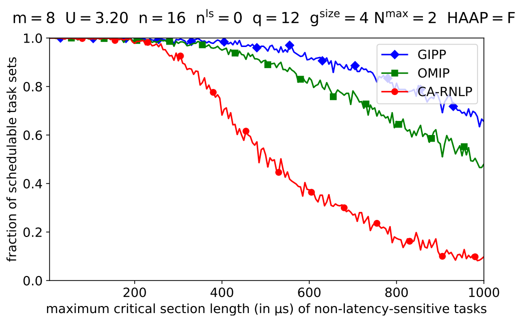

m = 8 U = 3.20 n = 16 $n^{ls} = 0$ q = 12 $g^{size} = 3$ $N^{max} = 1$ HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

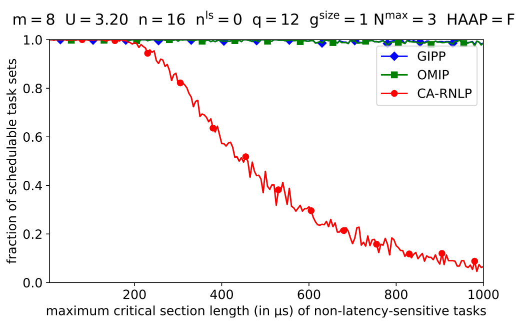

m = 8 U = 3.20 n = 16 $n^{ls} = 0$ q = 12 $g^{size} = 4$ $N^{max} = 1$ HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

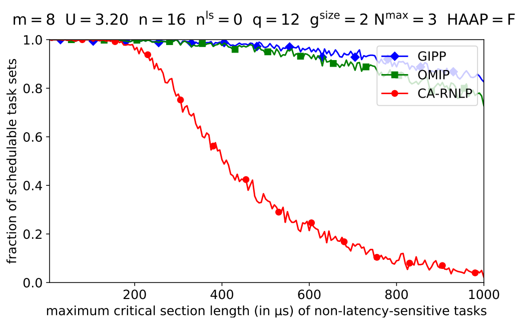

m = 8 U = 3.20 n = 16 $n^{ls} = 0$ q = 12 $g^{size} = 1$ $N^{max} = 2$ HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

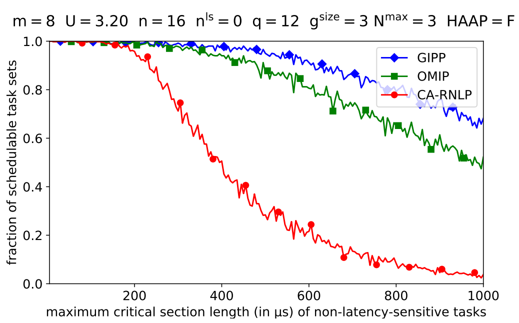

m = 8 U = 3.20 n = 16 $n^{ls} = 0$ q = 12 $g^{size} = 2$ $N^{max} = 2$ HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

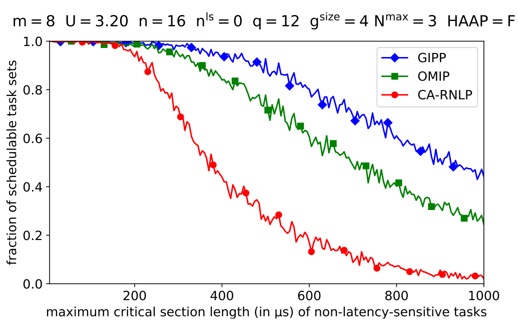

m = 8 U = 3.20 n = 16 $n^{ls} = 0$ q = 12 $g^{size} = 3$ $N^{max} = 2$ HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

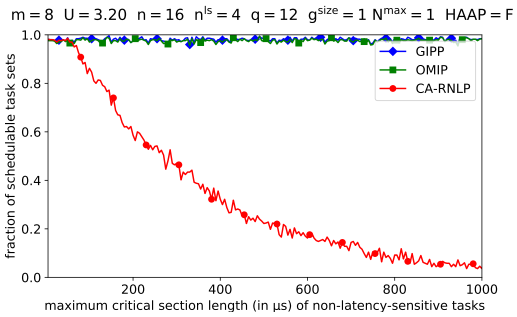

m = 8 U = 3.20 n = 16 $n^{ls} = 0$ q = 12 $g^{size} = 4$ $N^{max} = 2$ HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

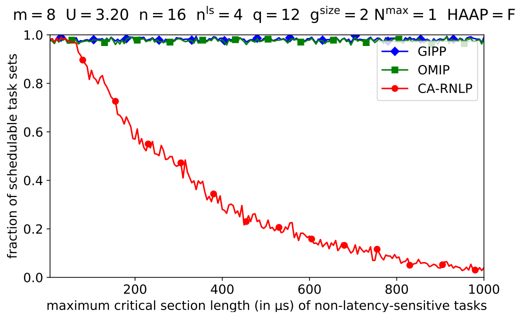

m = 8 U = 3.20 n = 16 $n^{ls} = 0$ q = 12 $g^{size} = 1$ $N^{max} = 3$ HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

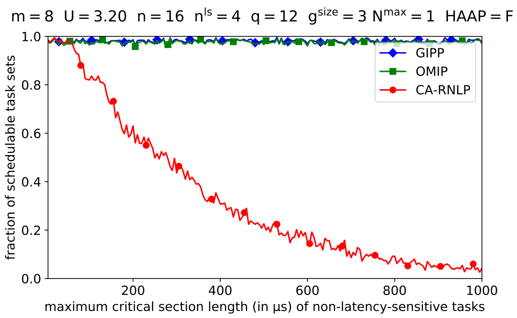

m = 8 U = 3.20 n = 16 $n^{ls} = 0$ q = 12 $g^{size} = 2$ $N^{max} = 3$ HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

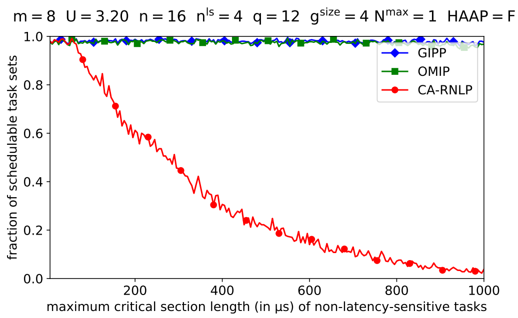

m = 8 U = 3.20 n = 16 $n^{ls} = 0$ q = 12 $g^{size} = 3$ $N^{max} = 3$ HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

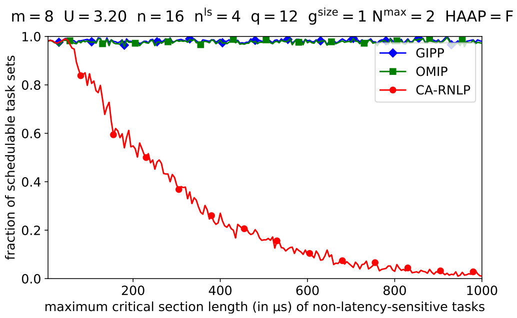

m = 8 U = 3.20 n = 16 $n^{ls} = 0$ q = 12 $g^{size} = 4$ $N^{max} = 3$ HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

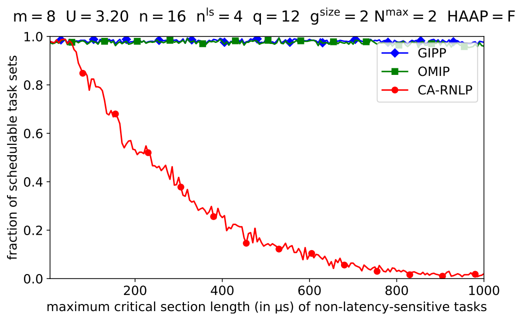

m = 8 U = 3.20 n = 16 n^{ls} = 4 q = 12 g^{size} = 1 N^{max} = 1 HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

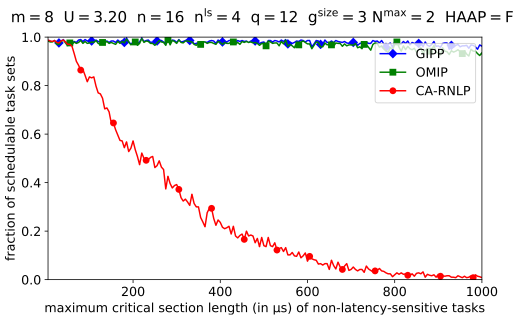

m = 8 U = 3.20 n = 16 n^{ls} = 4 q = 12 g^{size} = 2 N^{max} = 1 HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

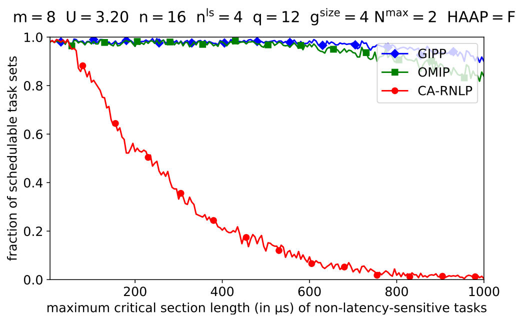

m = 8 U = 3.20 n = 16 n^{ls} = 4 q = 12 g^{size} = 3 N^{max} = 1 HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

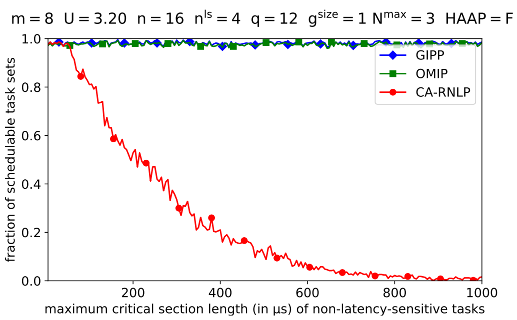

m = 8 U = 3.20 n = 16 n^{ls} = 4 q = 12 g^{size} = 4 N^{max} = 1 HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

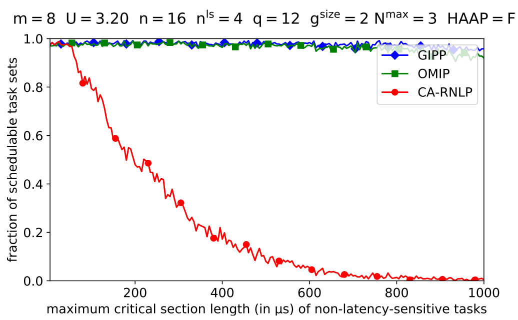

m = 8 U = 3.20 n = 16 n^{ls} = 4 q = 12 g^{size} = 1 N^{max} = 2 HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

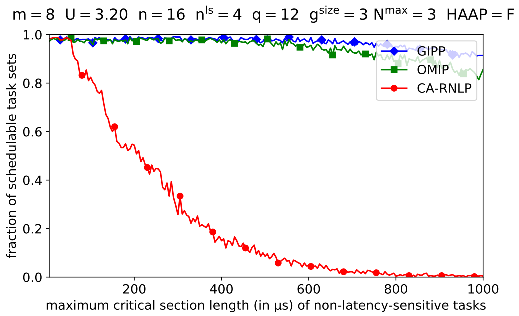

m = 8 U = 3.20 n = 16 n^{ls} = 4 q = 12 g^{size} = 2 N^{max} = 2 HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

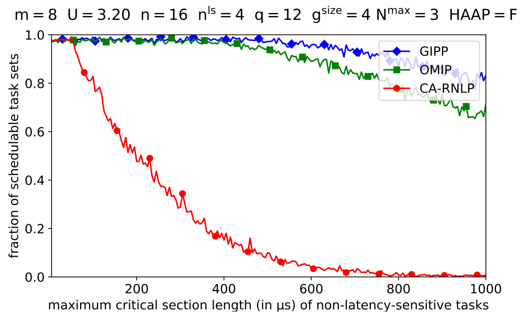

m = 8 U = 3.20 n = 16 n^{ls} = 4 q = 12 g^{size} = 3 N^{max} = 2 HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

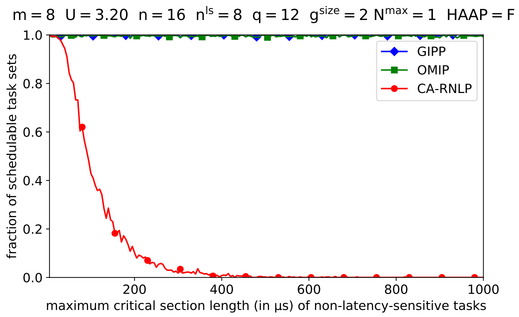

m = 8 U = 3.20 n = 16 n^{ls} = 4 q = 12 g^{size} = 4 N^{max} = 2 HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

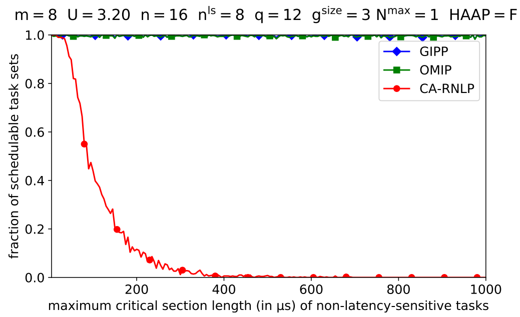

m = 8 U = 3.20 n = 16 n^{ls} = 4 q = 12 g^{size} = 1 N^{max} = 3 HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

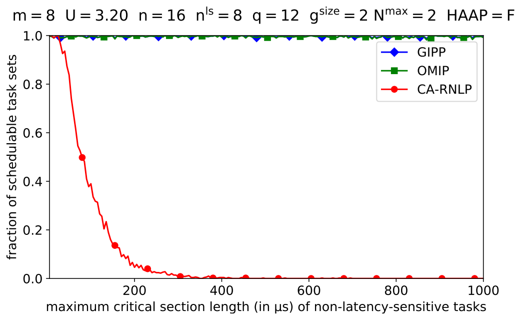

m = 8 U = 3.20 n = 16 n^{ls} = 4 q = 12 g^{size} = 2 N^{max} = 3 HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

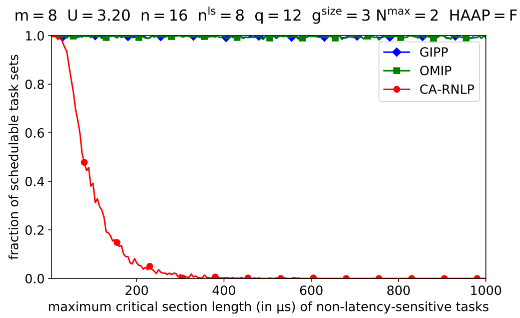

m = 8 U = 3.20 n = 16 n^{ls} = 4 q = 12 g^{size} = 3 N^{max} = 3 HAAP = F
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

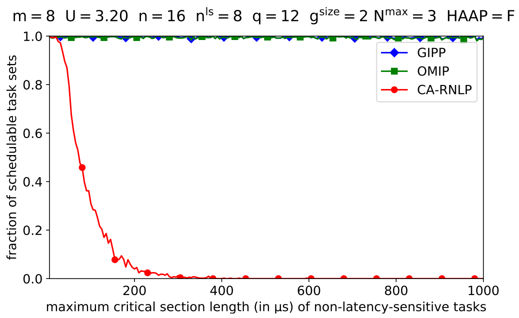

m = 8 U = 3.20 n = 16 n^{ls} = 4 q = 12 g^{size} = 4 N^{max} = 3 HAAP = F
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

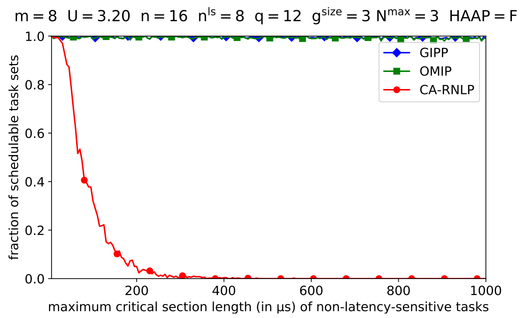

m = 8 U = 3.20 n = 16 n^{ls} = 8 q = 12 g^{size} = 2 N^{max} = 1 HAAP = F
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

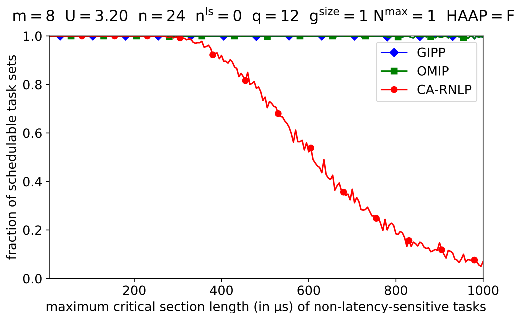

m = 8 U = 3.20 n = 16 n^{ls} = 8 q = 12 g^{size} = 3 N^{max} = 1 HAAP = F
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

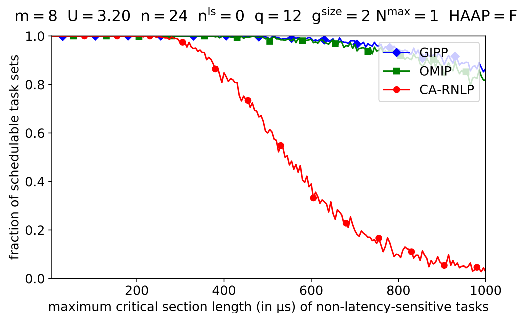

m = 8 U = 3.20 n = 16 n^{ls} = 8 q = 12 g^{size} = 2 N^{max} = 2 HAAP = F
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

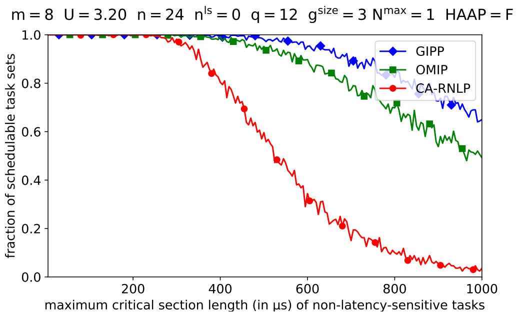

m = 8 U = 3.20 n = 16 n^{ls} = 8 q = 12 g^{size} = 3 N^{max} = 2 HAAP = F
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

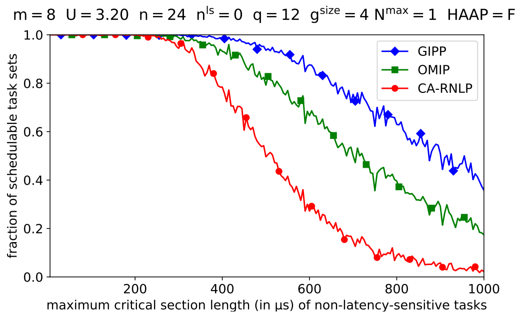

m = 8 U = 3.20 n = 16 n^{ls} = 8 q = 12 g^{size} = 2 N^{max} = 3 HAAP = F
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

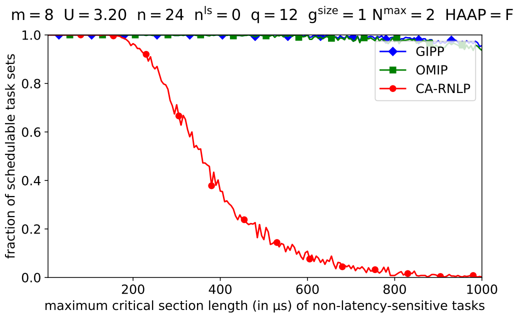

m = 8 U = 3.20 n = 16 n^{ls} = 8 q = 12 g^{size} = 3 N^{max} = 3 HAAP = F
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

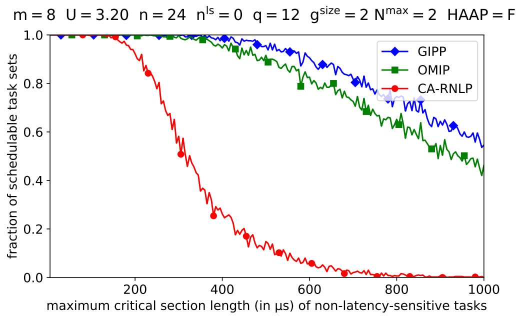

m = 8 U = 3.20 n = 24 n^{ls} = 0 q = 12 g^{size} = 1 N^{max} = 1 HAAP = F
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

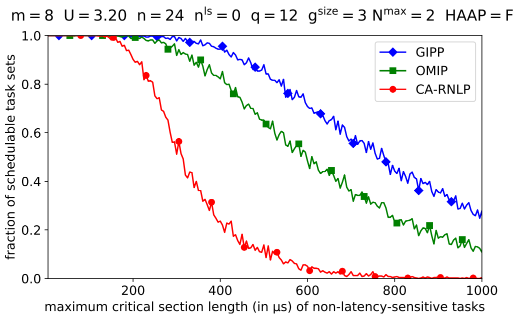

m = 8 U = 3.20 n = 24 n^{ls} = 0 q = 12 g^{size} = 2 N^{max} = 1 HAAP = F
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

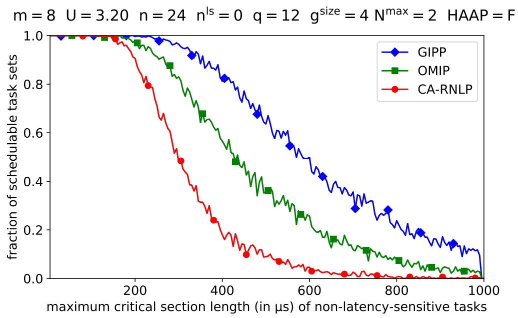

m = 8 U = 3.20 n = 24 n^ls = 0 q = 12 g^size = 3 N^max = 1 HAAP = F
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

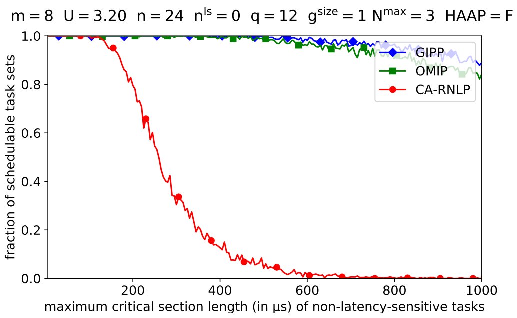

m = 8 U = 3.20 n = 24 n^ls = 0 q = 12 g^size = 4 N^max = 1 HAAP = F
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

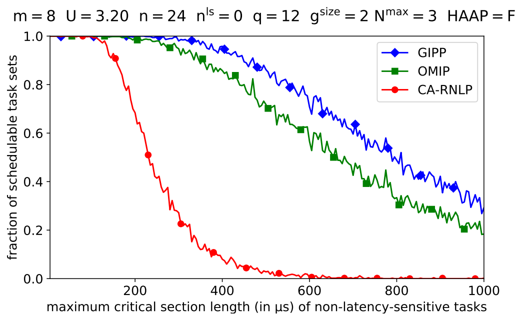

m = 8 U = 3.20 n = 24 n^ls = 0 q = 12 g^size = 1 N^max = 2 HAAP = F
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

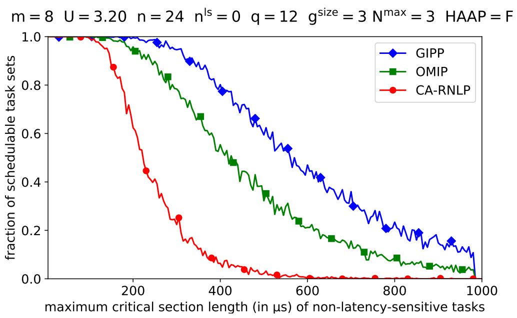

m = 8 U = 3.20 n = 24 n^ls = 0 q = 12 g^size = 2 N^max = 2 HAAP = F
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

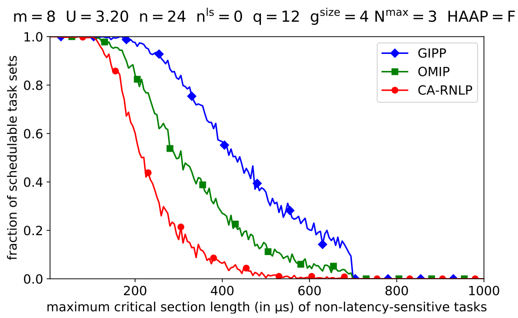

m = 8 U = 3.20 n = 24 n^ls = 0 q = 12 g^size = 3 N^max = 2 HAAP = F
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

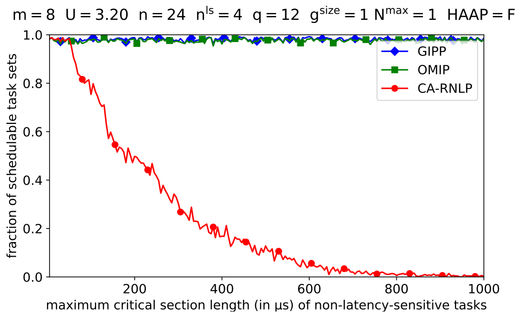

m = 8 U = 3.20 n = 24 n^ls = 0 q = 12 g^size = 4 N^max = 2 HAAP = F
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

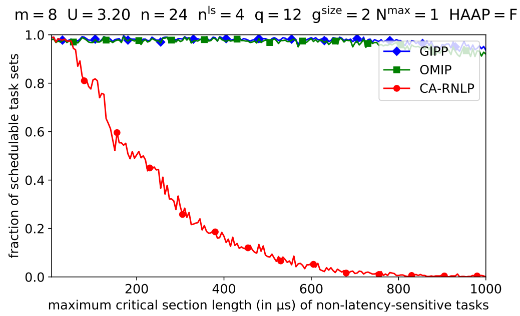

m = 8 U = 3.20 n = 24 n^ls = 0 q = 12 g^size = 1 N^max = 3 HAAP = F
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

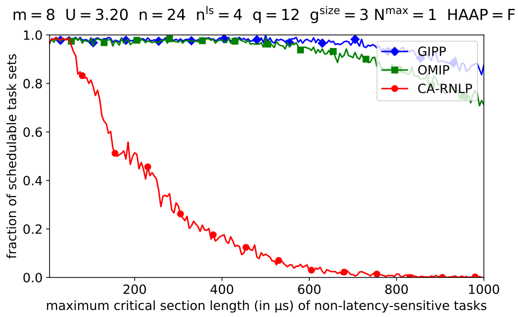

m = 8 U = 3.20 n = 24 n^ls = 0 q = 12 g^size = 2 N^max = 3 HAAP = F
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

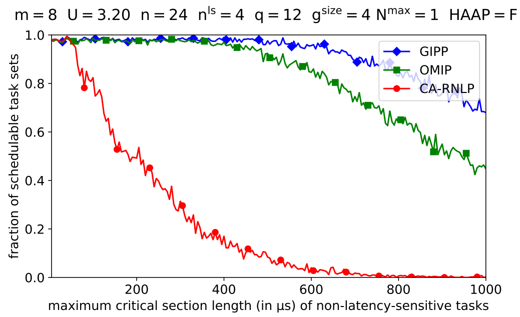

m = 8 U = 3.20 n = 24 n^ls = 0 q = 12 g^size = 3 N^max = 3 HAAP = F
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

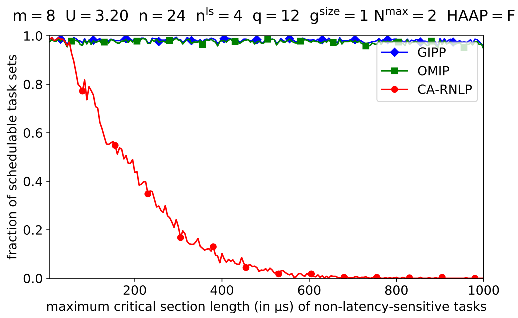

m = 8 U = 3.20 n = 24 n^ls = 0 q = 12 g^size = 4 N^max = 3 HAAP = F
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

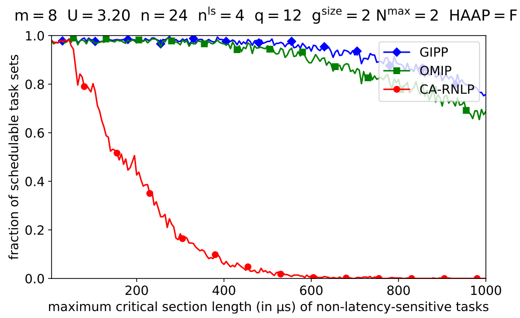

m = 8 U = 3.20 n = 24 $n^{ls} = 4$ q = 12 $g^{size} = 1$ $N^{max} = 1$ HAAP = F
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

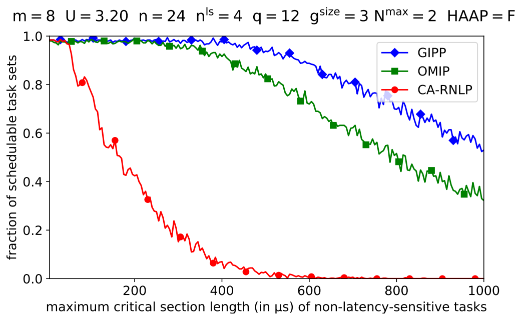

m = 8 U = 3.20 n = 24 $n^{ls} = 4$ q = 12 $g^{size} = 2$ $N^{max} = 1$ HAAP = F
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

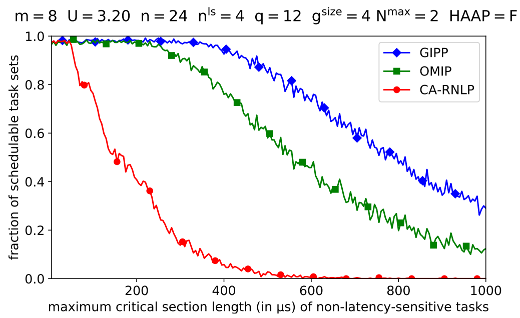

m = 8 U = 3.20 n = 24 $n^{ls} = 4$ q = 12 $g^{size} = 3$ $N^{max} = 1$ HAAP = F
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

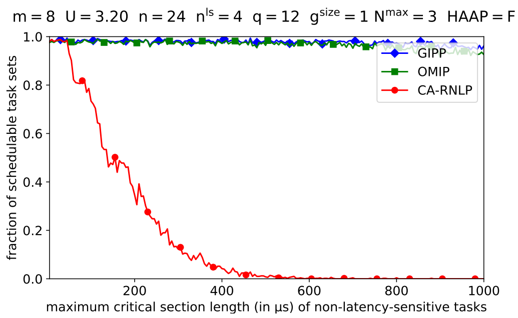

m = 8 U = 3.20 n = 24 $n^{ls} = 4$ q = 12 $g^{size} = 4$ $N^{max} = 1$ HAAP = F
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

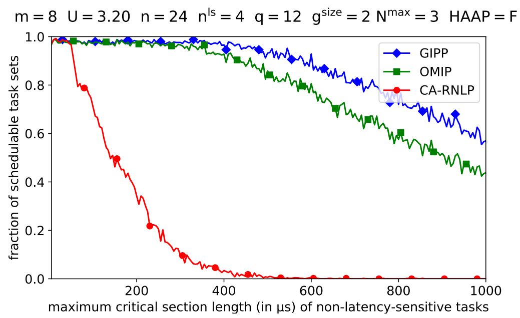

m = 8 U = 3.20 n = 24 $n^{ls} = 4$ q = 12 $g^{size} = 1$ $N^{max} = 2$ HAAP = F
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

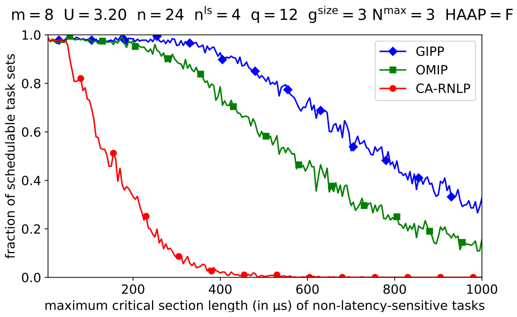

m = 8 U = 3.20 n = 24 $n^{ls} = 4$ q = 12 $g^{size} = 2$ $N^{max} = 2$ HAAP = F
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

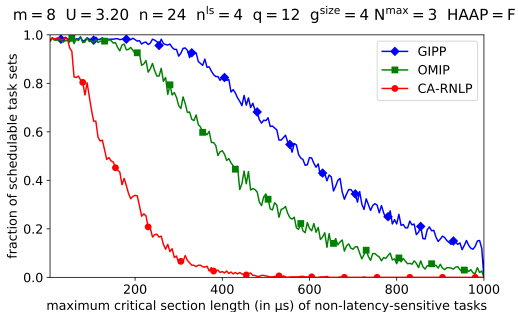

m = 8 U = 3.20 n = 24 $n^{ls} = 4$ q = 12 $g^{size} = 3$ $N^{max} = 2$ HAAP = F
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

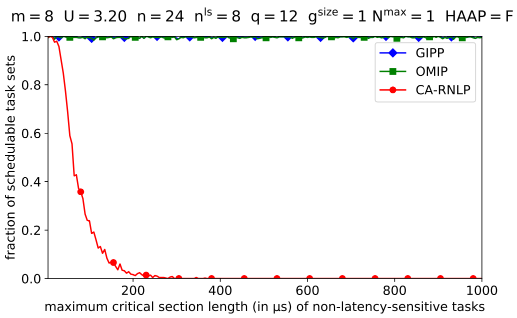

m = 8 U = 3.20 n = 24 $n^{ls} = 4$ q = 12 $g^{size} = 4$ $N^{max} = 2$ HAAP = F
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

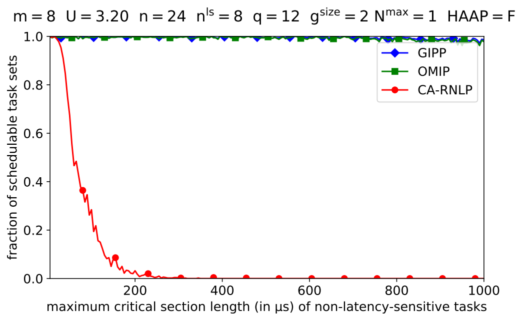

m = 8 U = 3.20 n = 24 $n^{ls} = 4$ q = 12 $g^{size} = 1$ $N^{max} = 3$ HAAP = F
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

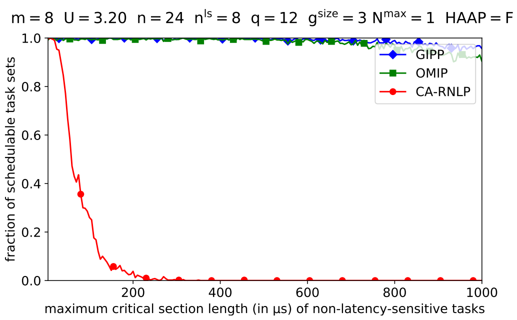

m = 8 U = 3.20 n = 24 $n^{ls} = 4$ q = 12 $g^{size} = 2$ $N^{max} = 3$ HAAP = F
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

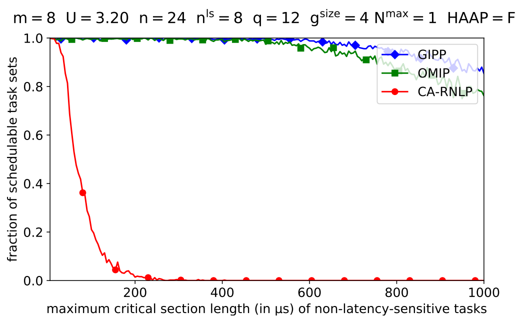

m = 8 U = 3.20 n = 24 n^{ls} = 4 q = 12 g^{size} = 3 N^{max} = 3 HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks
0.0 0.2 0.4 0.6 0.8 1.0
200 400 600 800 1000

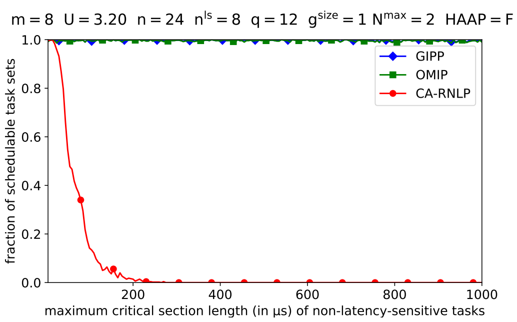

m = 8 U = 3.20 n = 24 n^{ls} = 4 q = 12 g^{size} = 4 N^{max} = 3 HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks
0.0 0.2 0.4 0.6 0.8 1.0
200 400 600 800 1000

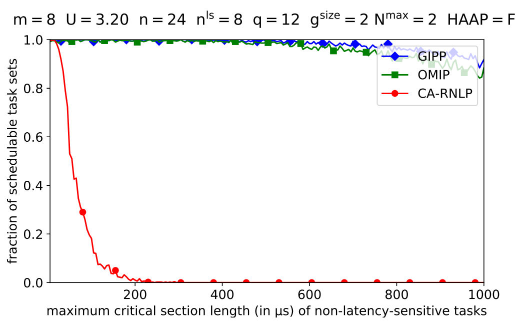

m = 8 U = 3.20 n = 24 n^{ls} = 8 q = 12 g^{size} = 1 N^{max} = 1 HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks
0.0 0.2 0.4 0.6 0.8 1.0
200 400 600 800 1000

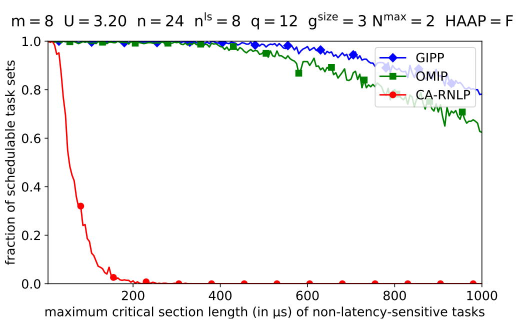

m = 8 U = 3.20 n = 24 n^{ls} = 8 q = 12 g^{size} = 2 N^{max} = 1 HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks
0.0 0.2 0.4 0.6 0.8 1.0
200 400 600 800 1000

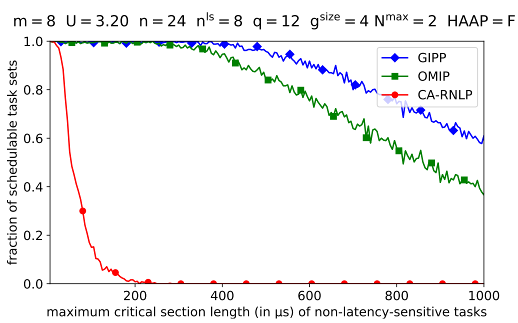

m = 8 U = 3.20 n = 24 n^{ls} = 8 q = 12 g^{size} = 3 N^{max} = 1 HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks
0.0 0.2 0.4 0.6 0.8 1.0
200 400 600 800 1000

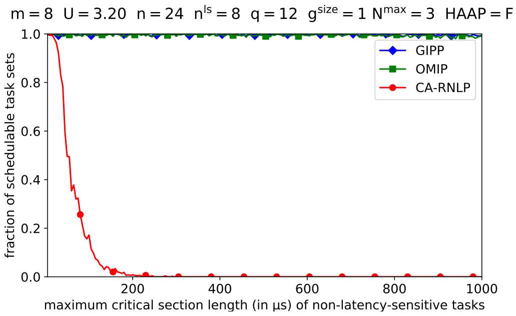

m = 8 U = 3.20 n = 24 n^{ls} = 8 q = 12 g^{size} = 4 N^{max} = 1 HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks
0.0 0.2 0.4 0.6 0.8 1.0
200 400 600 800 1000

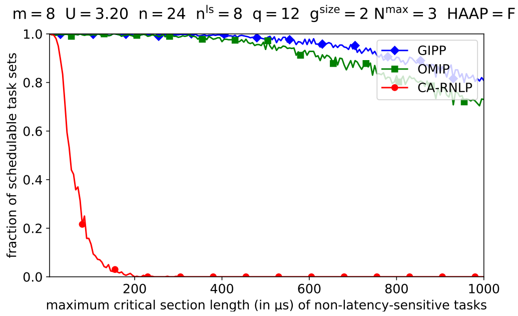

m = 8 U = 3.20 n = 24 n^{ls} = 8 q = 12 g^{size} = 1 N^{max} = 2 HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks
0.0 0.2 0.4 0.6 0.8 1.0
200 400 600 800 1000

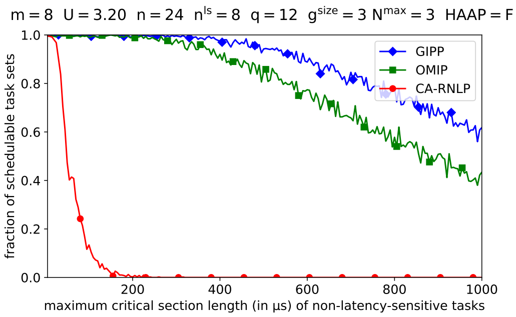

m = 8 U = 3.20 n = 24 n^{ls} = 8 q = 12 g^{size} = 2 N^{max} = 2 HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks
0.0 0.2 0.4 0.6 0.8 1.0
200 400 600 800 1000

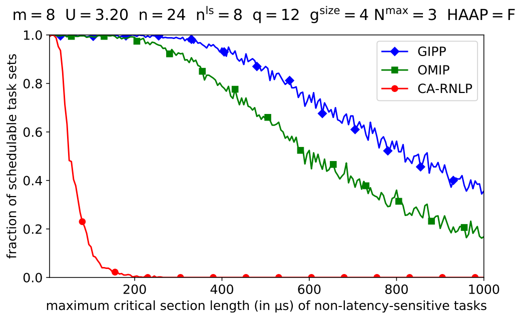

m = 8 U = 3.20 n = 24 n^{ls} = 8 q = 12 g^{size} = 3 N^{max} = 2 HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks
0.0 0.2 0.4 0.6 0.8 1.0
200 400 600 800 1000

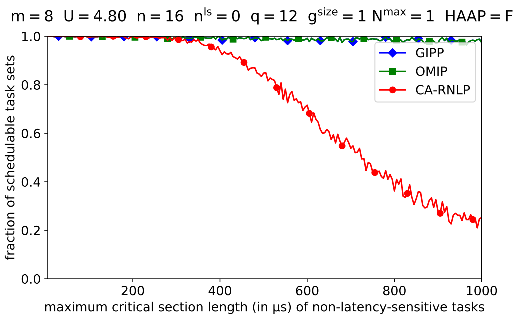

m = 8 U = 3.20 n = 24 n^{ls} = 8 q = 12 g^{size} = 4 N^{max} = 2 HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks
0.0 0.2 0.4 0.6 0.8 1.0
200 400 600 800 1000

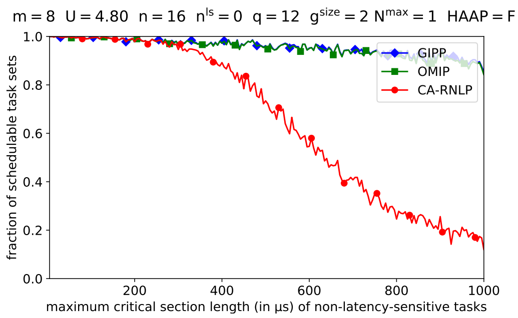

m = 8 U = 3.20 n = 24 $n^{ls} = 8$ q = 12 $g^{size} = 1$ $N^{max} = 3$ HAAP = F
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

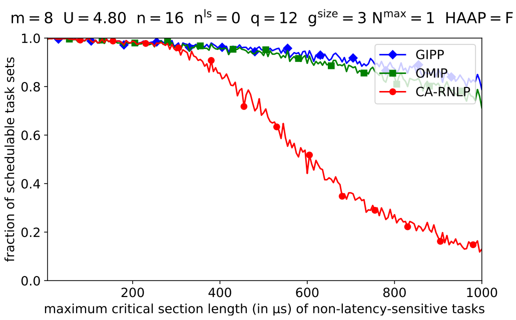

m = 8 U = 3.20 n = 24 $n^{ls} = 8$ q = 12 $g^{size} = 2$ $N^{max} = 3$ HAAP = F
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

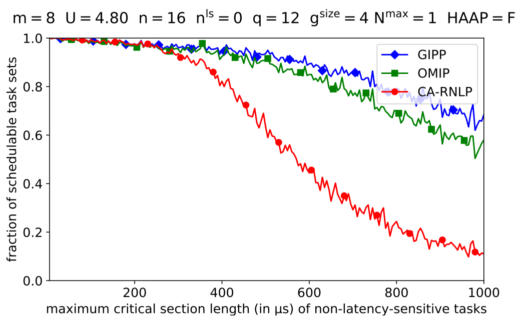

m = 8 U = 3.20 n = 24 $n^{ls} = 8$ q = 12 $g^{size} = 3$ $N^{max} = 3$ HAAP = F
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

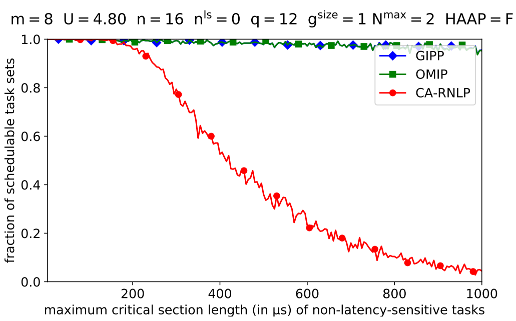

m = 8 U = 3.20 n = 24 $n^{ls} = 8$ q = 12 $g^{size} = 4$ $N^{max} = 3$ HAAP = F
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

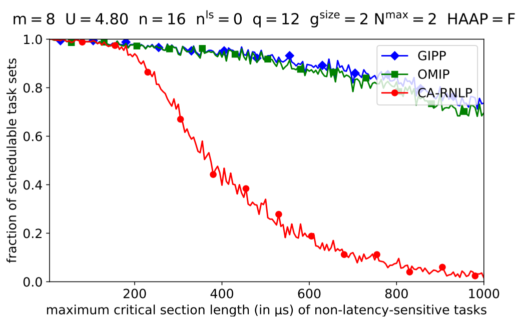

m = 8 U = 4.80 n = 16 $n^{ls} = 0$ q = 12 $g^{size} = 1$ $N^{max} = 1$ HAAP = F
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

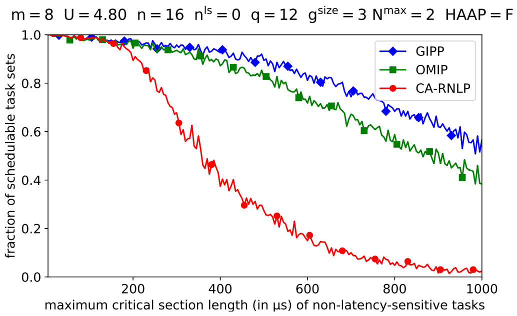

m = 8 U = 4.80 n = 16 $n^{ls} = 0$ q = 12 $g^{size} = 2$ $N^{max} = 1$ HAAP = F
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

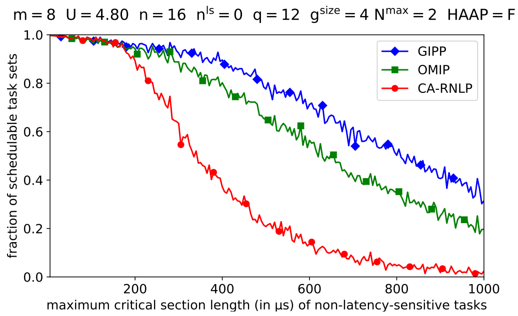

m = 8 U = 4.80 n = 16 $n^{ls} = 0$ q = 12 $g^{size} = 3$ $N^{max} = 1$ HAAP = F
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

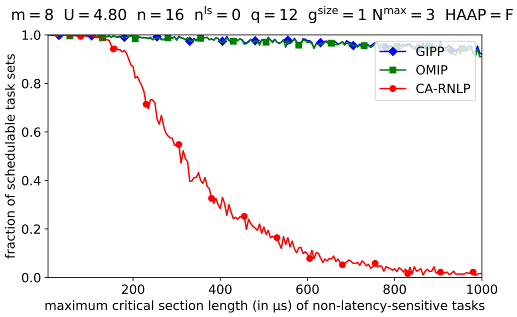

m = 8 U = 4.80 n = 16 $n^{ls} = 0$ q = 12 $g^{size} = 4$ $N^{max} = 1$ HAAP = F
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

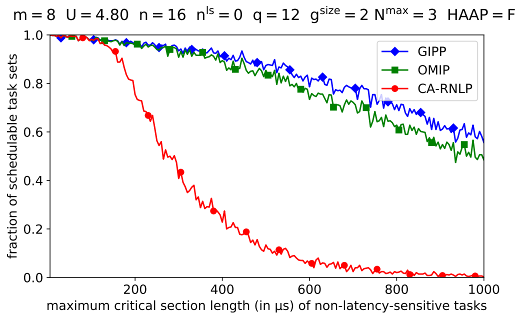

m = 8 U = 4.80 n = 16 $n^{ls} = 0$ q = 12 $g^{size} = 1$ $N^{max} = 2$ HAAP = F
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

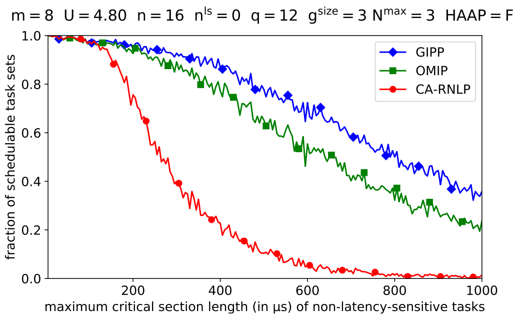

m = 8 U = 4.80 n = 16 $n^{ls} = 0$ q = 12 $g^{size} = 2$ $N^{max} = 2$ HAAP = F
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

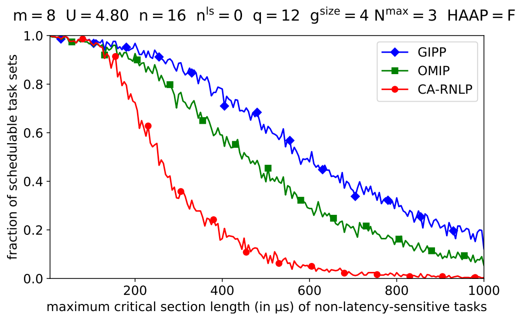

m = 8 U = 4.80 n = 16 n^ls = 0 q = 12 g^size = 3 N^max = 2 HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

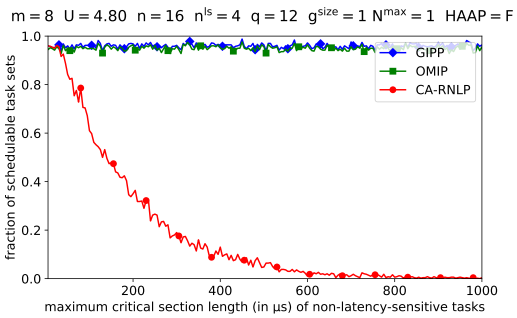

m = 8 U = 4.80 n = 16 n^ls = 0 q = 12 g^size = 4 N^max = 2 HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

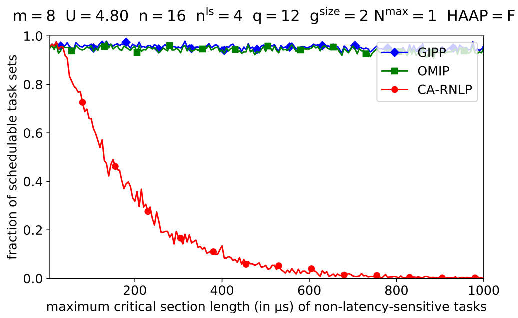

m = 8 U = 4.80 n = 16 n^ls = 0 q = 12 g^size = 1 N^max = 3 HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

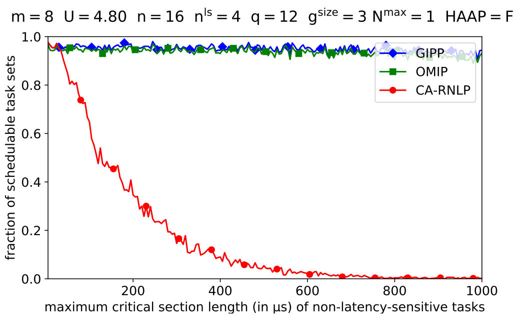

m = 8 U = 4.80 n = 16 n^ls = 0 q = 12 g^size = 2 N^max = 3 HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

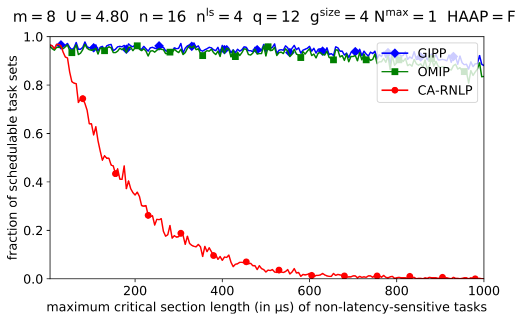

m = 8 U = 4.80 n = 16 n^ls = 0 q = 12 g^size = 3 N^max = 3 HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

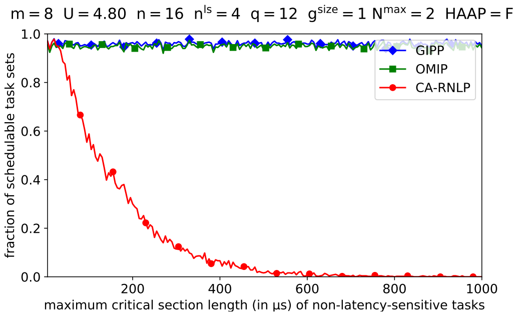

m = 8 U = 4.80 n = 16 n^ls = 0 q = 12 g^size = 4 N^max = 3 HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

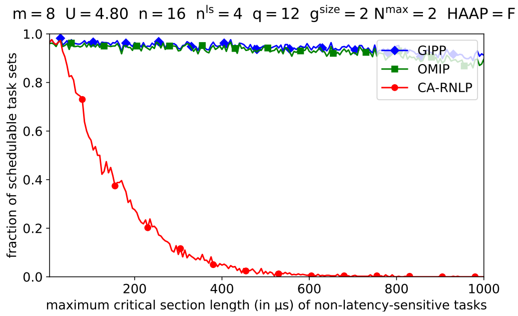

m = 8 U = 4.80 n = 16 n^ls = 4 q = 12 g^size = 1 N^max = 1 HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

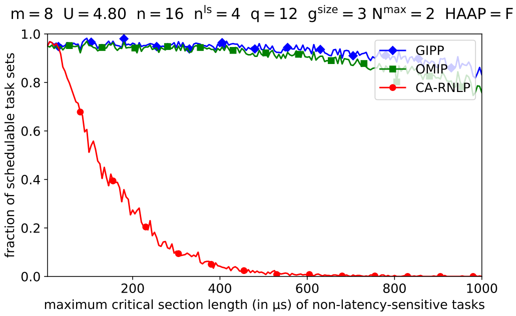

m = 8 U = 4.80 n = 16 n^ls = 4 q = 12 g^size = 2 N^max = 1 HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

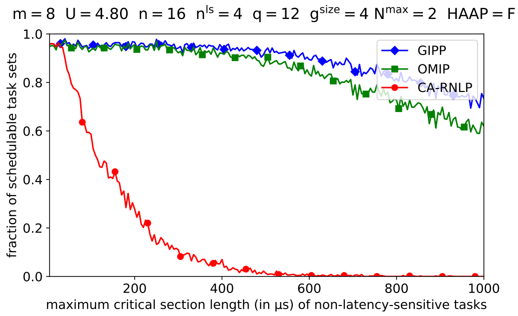

m = 8 U = 4.80 n = 16 n^ls = 4 q = 12 g^size = 3 N^max = 1 HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

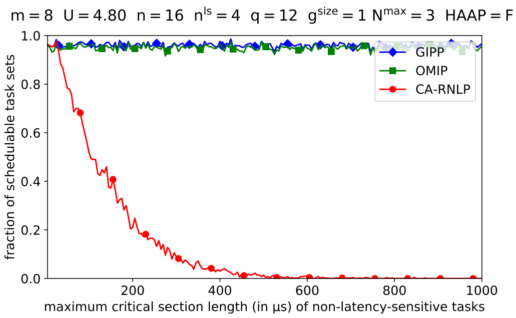

m = 8 U = 4.80 n = 16 n^ls = 4 q = 12 g^size = 4 N^max = 1 HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

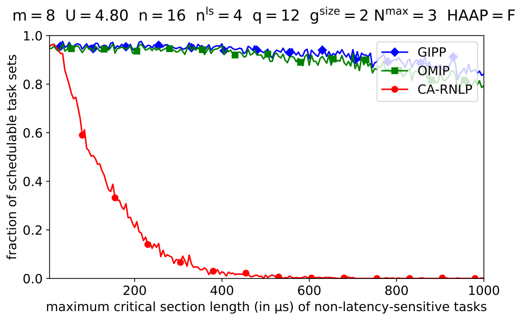

m = 8 U = 4.80 n = 16 nls = 4 q = 12 gsize = 1 Nmax = 2 HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks

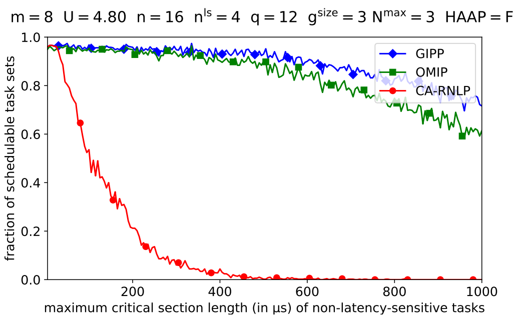

m = 8 U = 4.80 n = 16 nls = 4 q = 12 gsize = 2 Nmax = 2 HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks

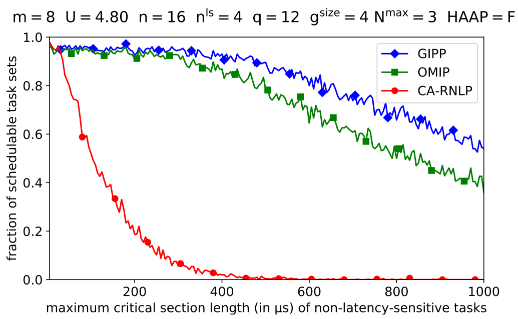

m = 8 U = 4.80 n = 16 nls = 4 q = 12 gsize = 3 Nmax = 2 HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks

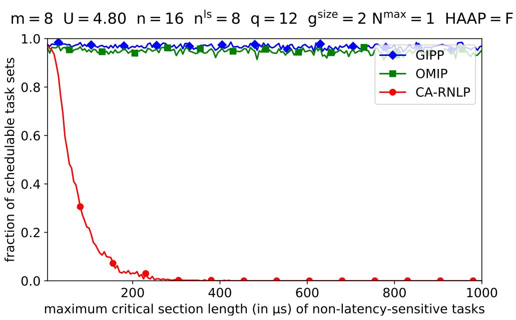

m = 8 U = 4.80 n = 16 nls = 4 q = 12 gsize = 4 Nmax = 2 HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks

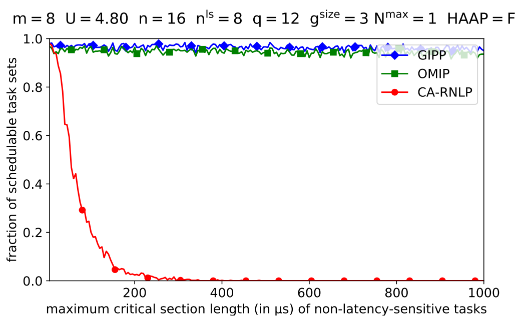

m = 8 U = 4.80 n = 16 nls = 4 q = 12 gsize = 1 Nmax = 3 HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks

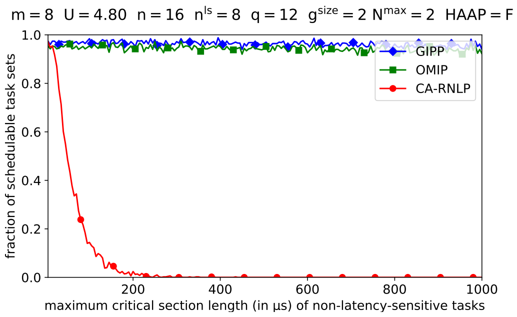

m = 8 U = 4.80 n = 16 nls = 4 q = 12 gsize = 2 Nmax = 3 HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks

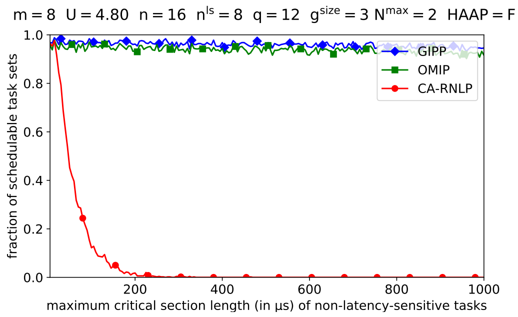

m = 8 U = 4.80 n = 16 nls = 4 q = 12 gsize = 3 Nmax = 3 HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks

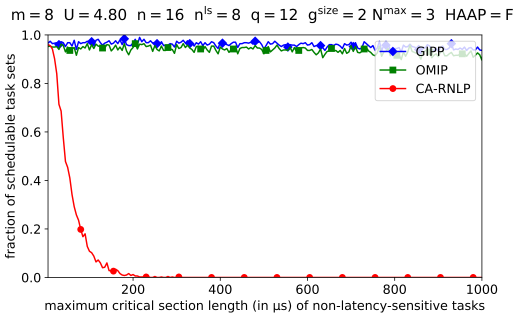

m = 8 U = 4.80 n = 16 nls = 4 q = 12 gsize = 4 Nmax = 3 HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks

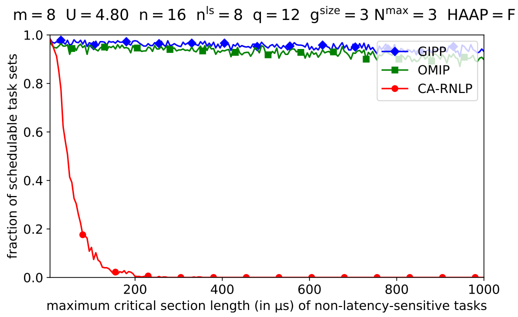

m = 8 U = 4.80 n = 16 nls = 8 q = 12 gsize = 2 Nmax = 1 HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks

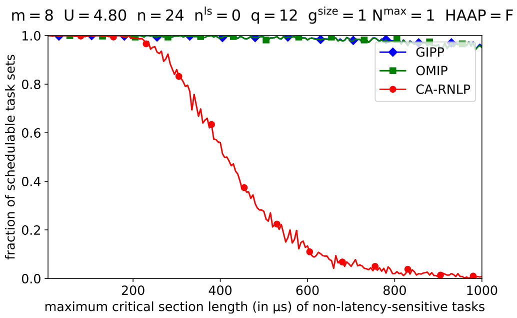

m = 8 U = 4.80 n = 16 nls = 8 q = 12 gsize = 3 Nmax = 1 HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks

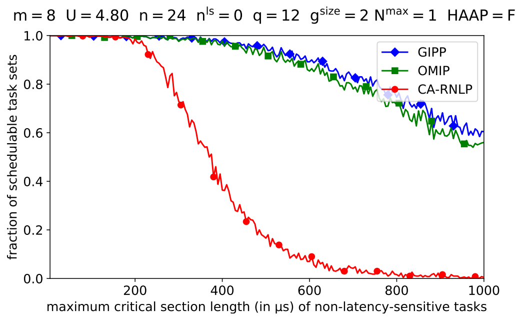

m = 8 U = 4.80 n = 16 $n^{ls} = 8$ q = 12 $g^{size} = 2$ $N^{max} = 2$ HAAP = F
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

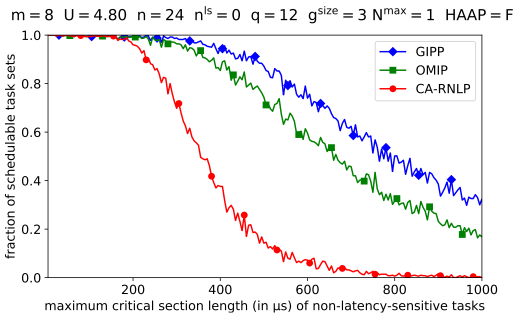

m = 8 U = 4.80 n = 16 $n^{ls} = 8$ q = 12 $g^{size} = 3$ $N^{max} = 2$ HAAP = F
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

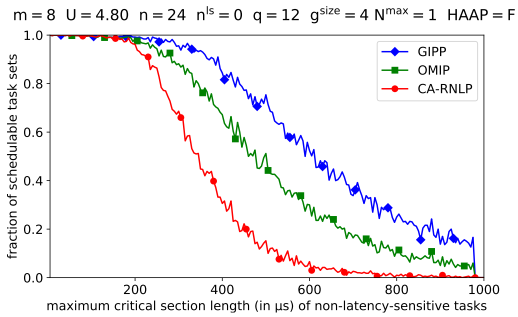

m = 8 U = 4.80 n = 16 $n^{ls} = 8$ q = 12 $g^{size} = 2$ $N^{max} = 3$ HAAP = F
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

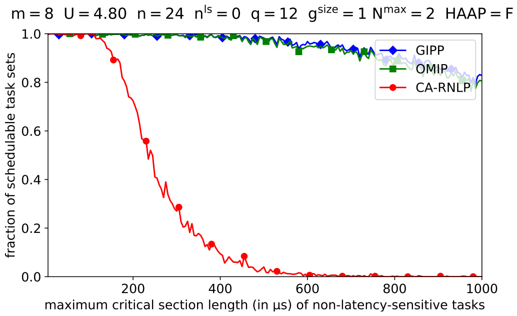

m = 8 U = 4.80 n = 16 $n^{ls} = 8$ q = 12 $g^{size} = 3$ $N^{max} = 3$ HAAP = F
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

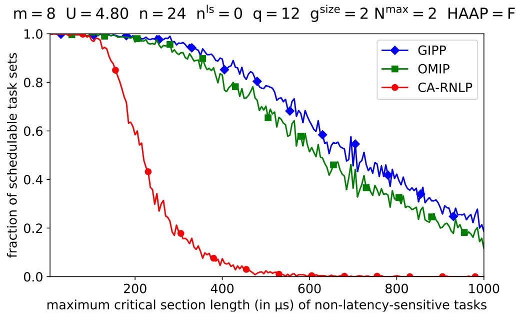

m = 8 U = 4.80 n = 24 $n^{ls} = 0$ q = 12 $g^{size} = 1$ $N^{max} = 1$ HAAP = F
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

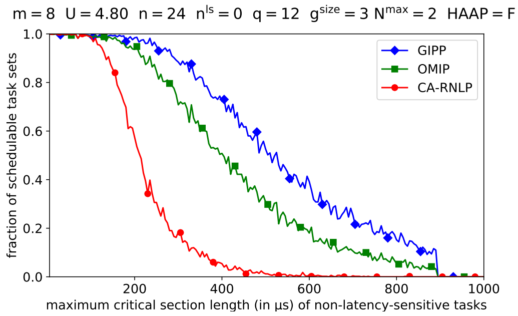

m = 8 U = 4.80 n = 24 $n^{ls} = 0$ q = 12 $g^{size} = 2$ $N^{max} = 1$ HAAP = F
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

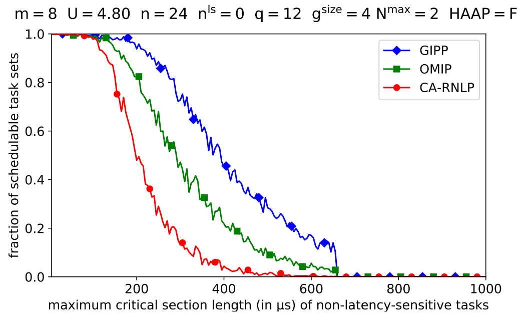

m = 8 U = 4.80 n = 24 $n^{ls} = 0$ q = 12 $g^{size} = 3$ $N^{max} = 1$ HAAP = F
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

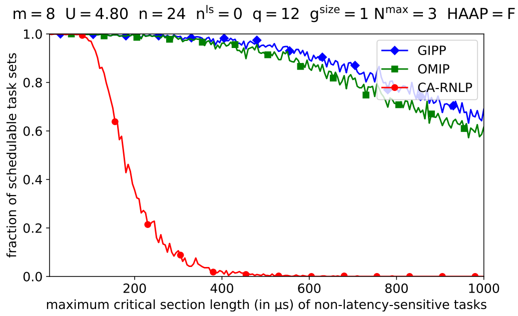

m = 8 U = 4.80 n = 24 $n^{ls} = 0$ q = 12 $g^{size} = 4$ $N^{max} = 1$ HAAP = F
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

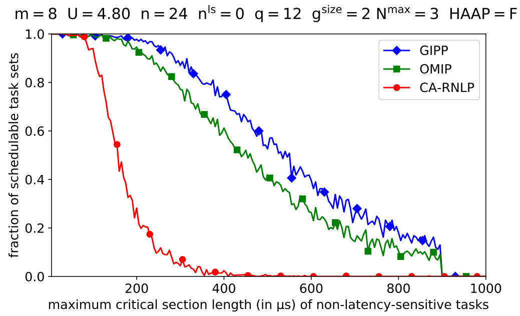

m = 8 U = 4.80 n = 24 $n^{ls} = 0$ q = 12 $g^{size} = 1$ $N^{max} = 2$ HAAP = F
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

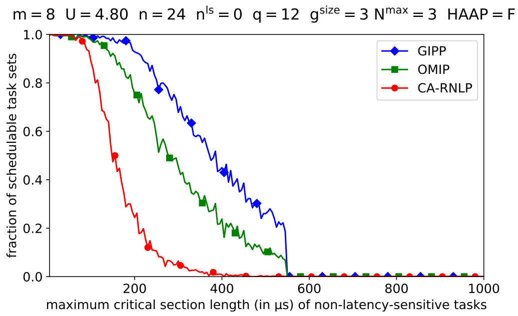

m = 8 U = 4.80 n = 24 $n^{ls} = 0$ q = 12 $g^{size} = 2$ $N^{max} = 2$ HAAP = F
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

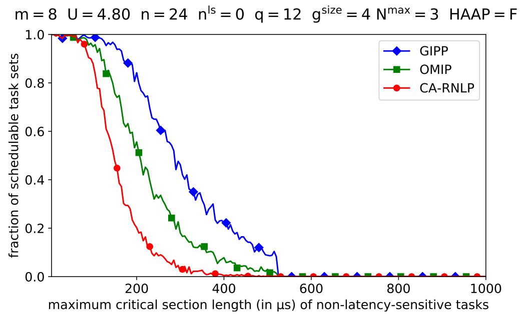

$m = 8$ $U = 4.80$ $n = 24$ $n^{ls} = 0$ $q = 12$ $g^{size} = 3$ $N^{max} = 2$ HAAP = F
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

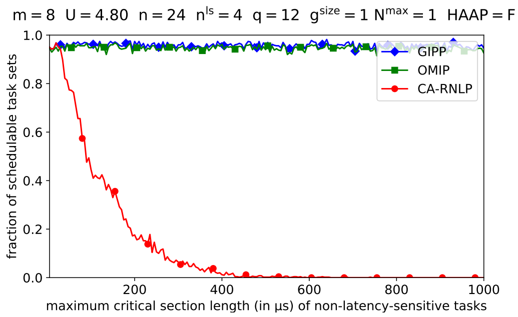

$m = 8$ $U = 4.80$ $n = 24$ $n^{ls} = 0$ $q = 12$ $g^{size} = 4$ $N^{max} = 2$ HAAP = F
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

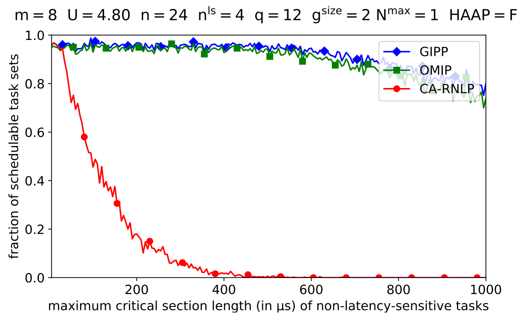

$m = 8$ $U = 4.80$ $n = 24$ $n^{ls} = 0$ $q = 12$ $g^{size} = 1$ $N^{max} = 3$ HAAP = F
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

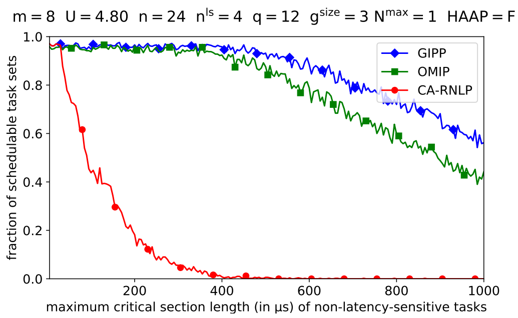

$m = 8$ $U = 4.80$ $n = 24$ $n^{ls} = 0$ $q = 12$ $g^{size} = 2$ $N^{max} = 3$ HAAP = F
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

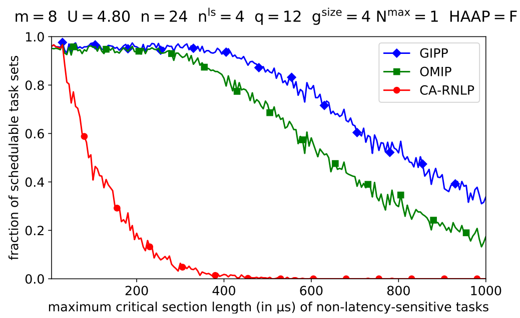

$m = 8$ $U = 4.80$ $n = 24$ $n^{ls} = 0$ $q = 12$ $g^{size} = 3$ $N^{max} = 3$ HAAP = F
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

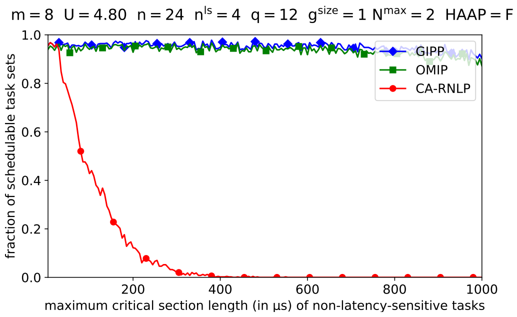

$m = 8$ $U = 4.80$ $n = 24$ $n^{ls} = 0$ $q = 12$ $g^{size} = 4$ $N^{max} = 3$ HAAP = F
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

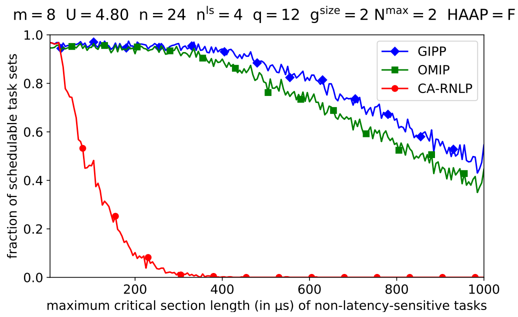

$m = 8$ $U = 4.80$ $n = 24$ $n^{ls} = 4$ $q = 12$ $g^{size} = 1$ $N^{max} = 1$ HAAP = F
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

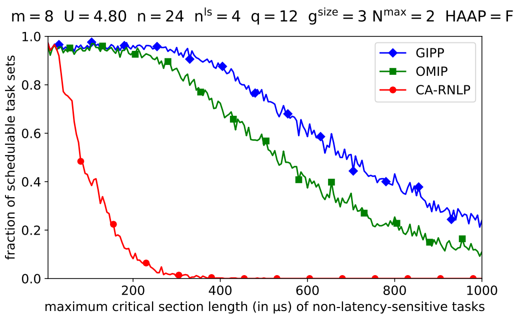

$m = 8$ $U = 4.80$ $n = 24$ $n^{ls} = 4$ $q = 12$ $g^{size} = 2$ $N^{max} = 1$ HAAP = F
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

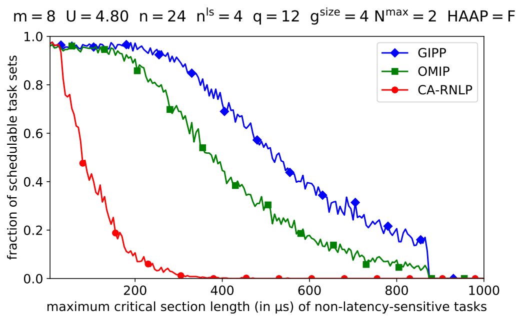

$m = 8$ $U = 4.80$ $n = 24$ $n^{ls} = 4$ $q = 12$ $g^{size} = 3$ $N^{max} = 1$ HAAP = F
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

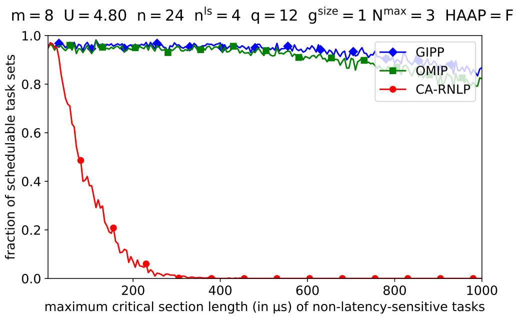

$m = 8$ $U = 4.80$ $n = 24$ $n^{ls} = 4$ $q = 12$ $g^{size} = 4$ $N^{max} = 1$ HAAP = F
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

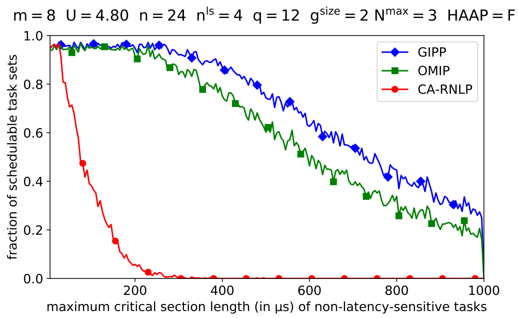

m = 8 U = 4.80 n = 24 $n^{ls} = 4$ q = 12 $g^{size} = 1$ $N^{max} = 2$ HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

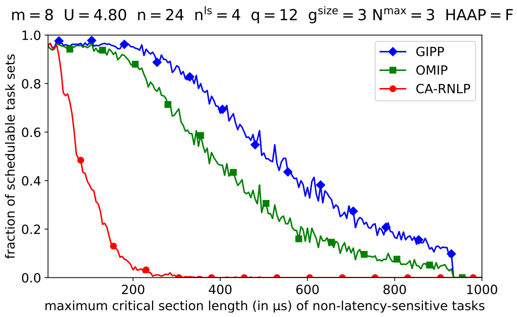

m = 8 U = 4.80 n = 24 $n^{ls} = 4$ q = 12 $g^{size} = 2$ $N^{max} = 2$ HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

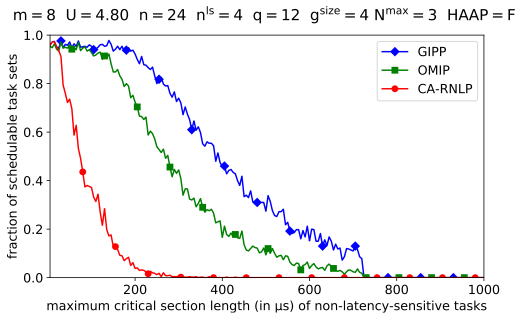

m = 8 U = 4.80 n = 24 $n^{ls} = 4$ q = 12 $g^{size} = 3$ $N^{max} = 2$ HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

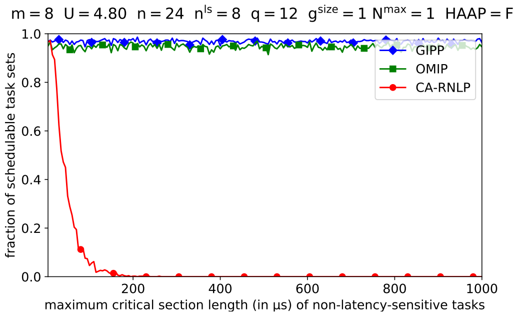

m = 8 U = 4.80 n = 24 $n^{ls} = 4$ q = 12 $g^{size} = 4$ $N^{max} = 2$ HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

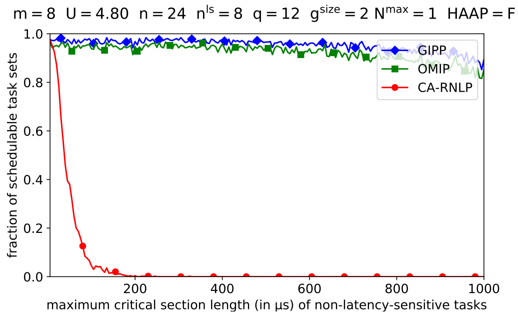

m = 8 U = 4.80 n = 24 $n^{ls} = 4$ q = 12 $g^{size} = 1$ $N^{max} = 3$ HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

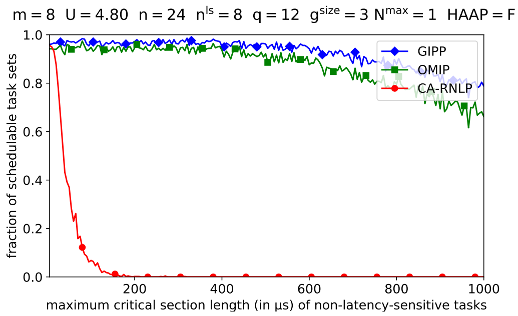

m = 8 U = 4.80 n = 24 $n^{ls} = 4$ q = 12 $g^{size} = 2$ $N^{max} = 3$ HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

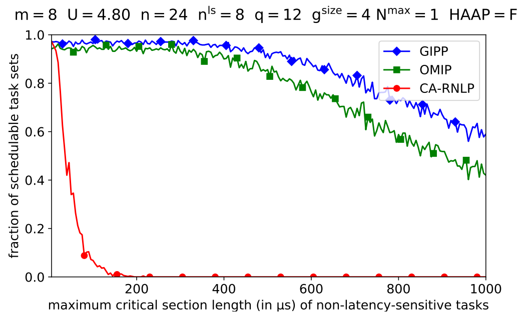

m = 8 U = 4.80 n = 24 $n^{ls} = 4$ q = 12 $g^{size} = 3$ $N^{max} = 3$ HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

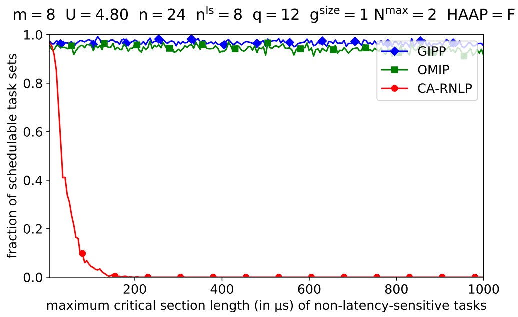

m = 8 U = 4.80 n = 24 $n^{ls} = 4$ q = 12 $g^{size} = 4$ $N^{max} = 3$ HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

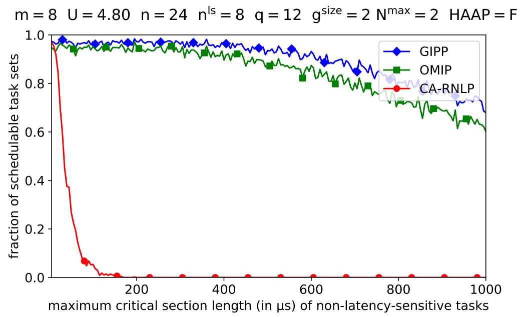

m = 8 U = 4.80 n = 24 $n^{ls} = 8$ q = 12 $g^{size} = 1$ $N^{max} = 1$ HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

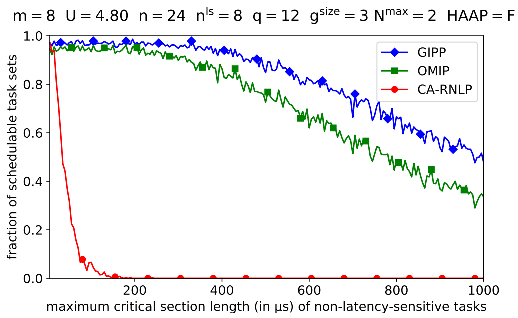

m = 8 U = 4.80 n = 24 $n^{ls} = 8$ q = 12 $g^{size} = 2$ $N^{max} = 1$ HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

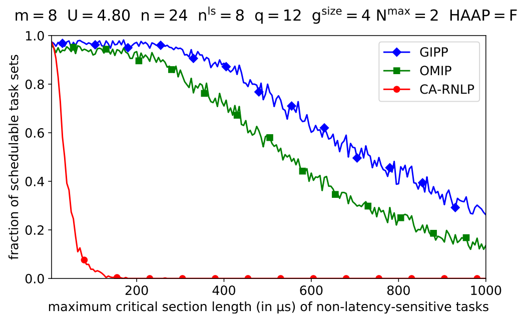

m = 8 U = 4.80 n = 24 n^{ls} = 8 q = 12 g^{size} = 3 N^{max} = 1 HAAP = F
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

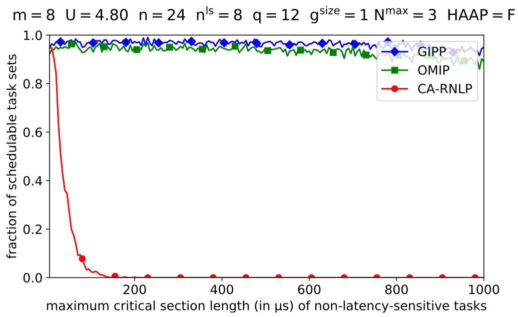

m = 8 U = 4.80 n = 24 n^{ls} = 8 q = 12 g^{size} = 4 N^{max} = 1 HAAP = F
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

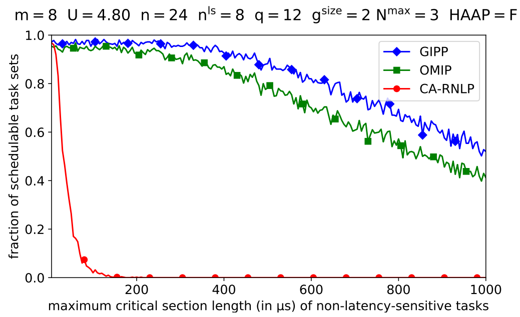

m = 8 U = 4.80 n = 24 n^{ls} = 8 q = 12 g^{size} = 1 N^{max} = 2 HAAP = F
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

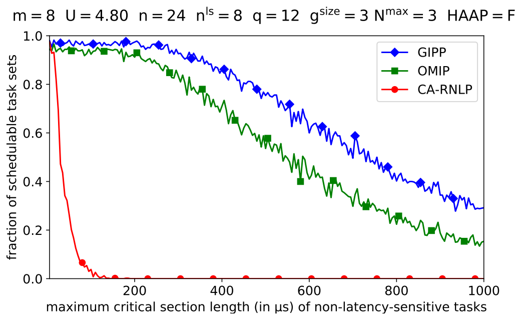

m = 8 U = 4.80 n = 24 n^{ls} = 8 q = 12 g^{size} = 2 N^{max} = 2 HAAP = F
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

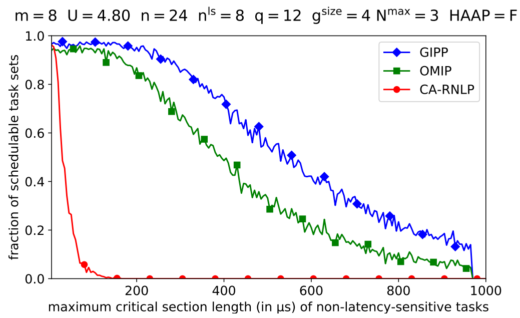

m = 8 U = 4.80 n = 24 n^{ls} = 8 q = 12 g^{size} = 3 N^{max} = 2 HAAP = F
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

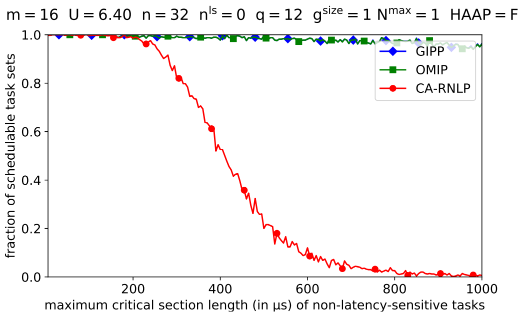

m = 8 U = 4.80 n = 24 n^{ls} = 8 q = 12 g^{size} = 4 N^{max} = 2 HAAP = F
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

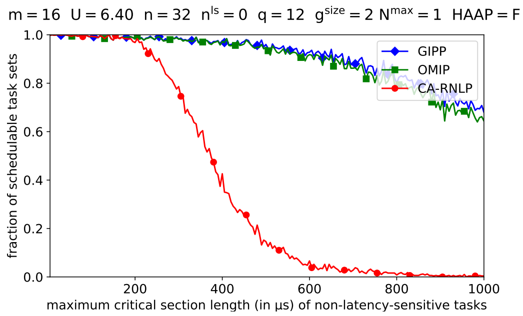

m = 8 U = 4.80 n = 24 n^{ls} = 8 q = 12 g^{size} = 1 N^{max} = 3 HAAP = F
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

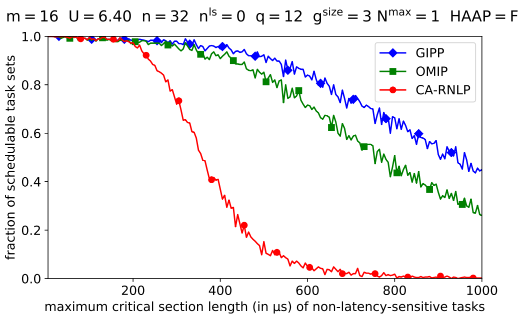

m = 8 U = 4.80 n = 24 n^{ls} = 8 q = 12 g^{size} = 2 N^{max} = 3 HAAP = F
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

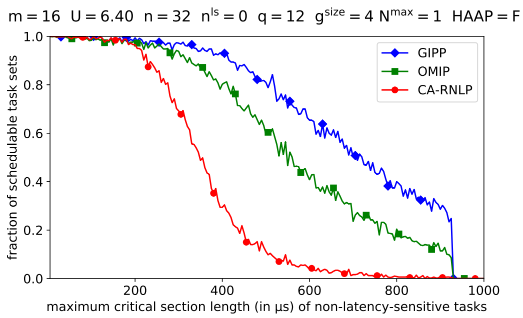

m = 8 U = 4.80 n = 24 n^{ls} = 8 q = 12 g^{size} = 3 N^{max} = 3 HAAP = F
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

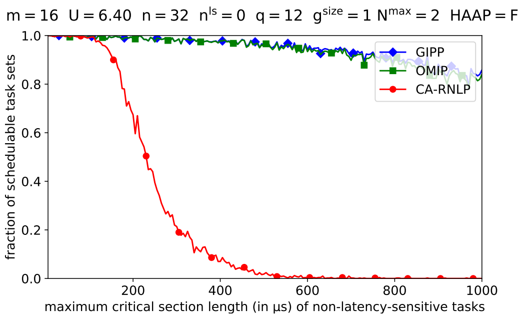

m = 8 U = 4.80 n = 24 n^{ls} = 8 q = 12 g^{size} = 4 N^{max} = 3 HAAP = F
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

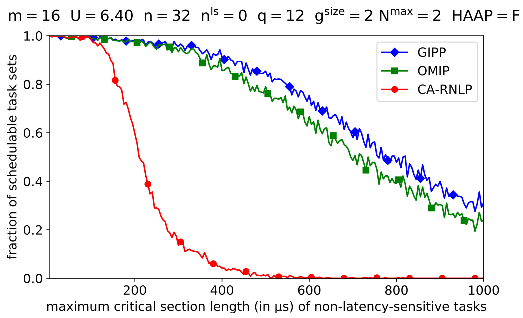

m = 16 U = 6.40 n = 32 $n^{ls} = 0$ q = 12 $g^{size} = 1$ $N^{max} = 1$ HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

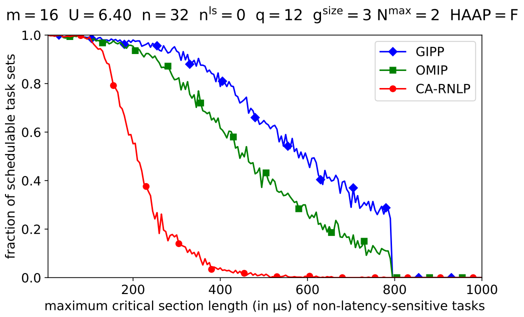

m = 16 U = 6.40 n = 32 $n^{ls} = 0$ q = 12 $g^{size} = 2$ $N^{max} = 1$ HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

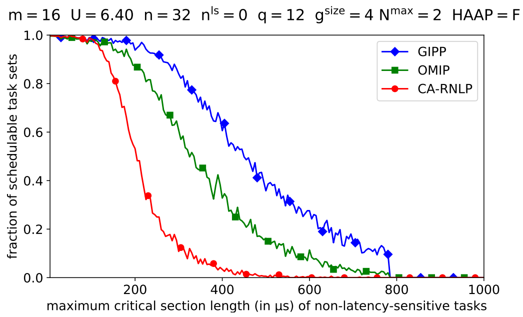

m = 16 U = 6.40 n = 32 $n^{ls} = 0$ q = 12 $g^{size} = 3$ $N^{max} = 1$ HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

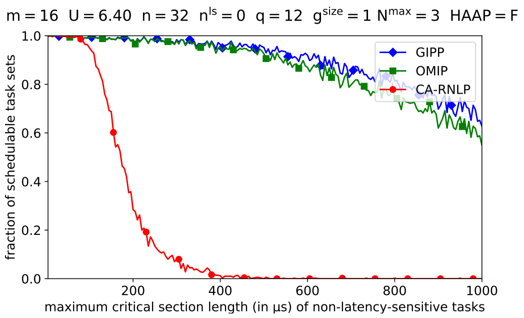

m = 16 U = 6.40 n = 32 $n^{ls} = 0$ q = 12 $g^{size} = 4$ $N^{max} = 1$ HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

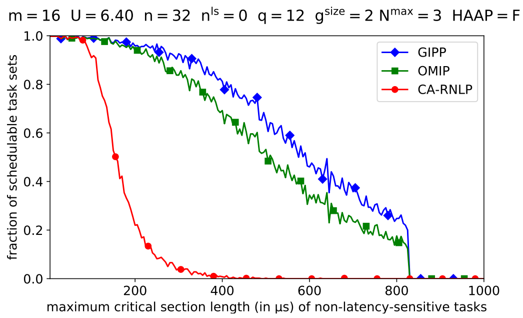

m = 16 U = 6.40 n = 32 $n^{ls} = 0$ q = 12 $g^{size} = 1$ $N^{max} = 2$ HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

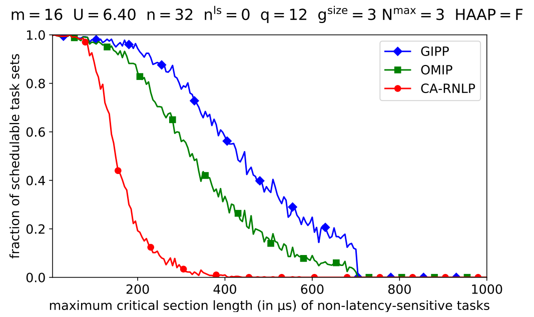

m = 16 U = 6.40 n = 32 $n^{ls} = 0$ q = 12 $g^{size} = 2$ $N^{max} = 2$ HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

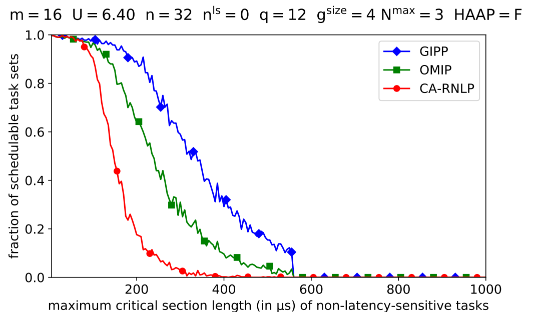

m = 16 U = 6.40 n = 32 $n^{ls} = 0$ q = 12 $g^{size} = 3$ $N^{max} = 2$ HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

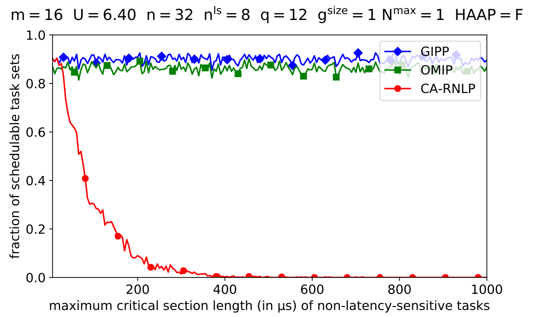

m = 16 U = 6.40 n = 32 $n^{ls} = 0$ q = 12 $g^{size} = 4$ $N^{max} = 2$ HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

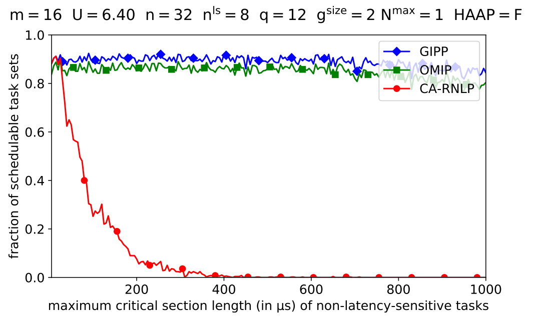

m = 16 U = 6.40 n = 32 $n^{ls} = 0$ q = 12 $g^{size} = 1$ $N^{max} = 3$ HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

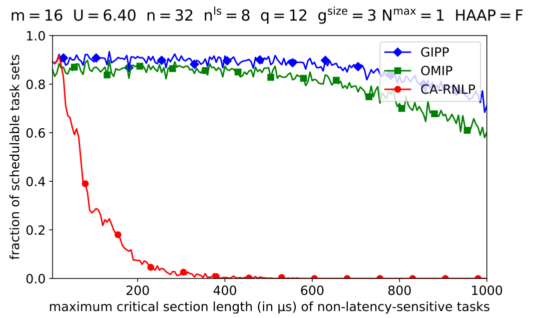

m = 16 U = 6.40 n = 32 $n^{ls} = 0$ q = 12 $g^{size} = 2$ $N^{max} = 3$ HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

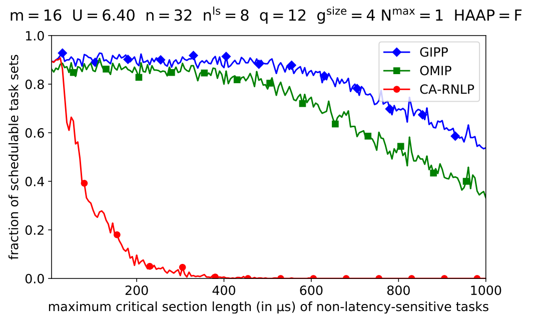

m = 16 U = 6.40 n = 32 $n^{ls} = 0$ q = 12 $g^{size} = 3$ $N^{max} = 3$ HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks

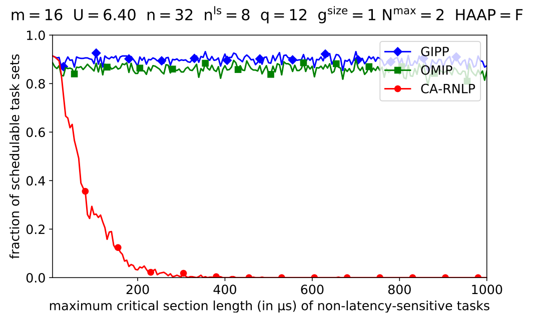

m = 16 U = 6.40 n = 32 $n^{ls} = 0$ q = 12 $g^{size} = 4$ $N^{max} = 3$ HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks

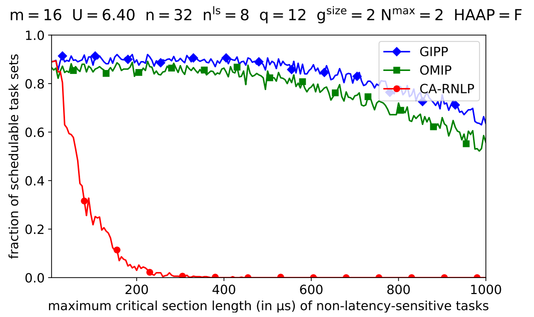

m = 16 U = 6.40 n = 32 $n^{ls} = 8$ q = 12 $g^{size} = 1$ $N^{max} = 1$ HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks

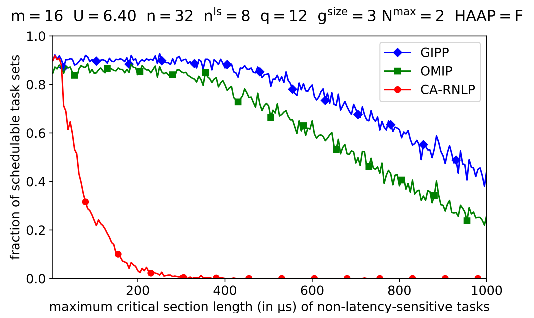

m = 16 U = 6.40 n = 32 $n^{ls} = 8$ q = 12 $g^{size} = 2$ $N^{max} = 1$ HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks

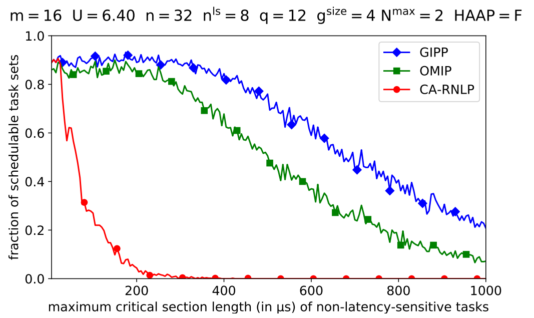

m = 16 U = 6.40 n = 32 $n^{ls} = 8$ q = 12 $g^{size} = 3$ $N^{max} = 1$ HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks

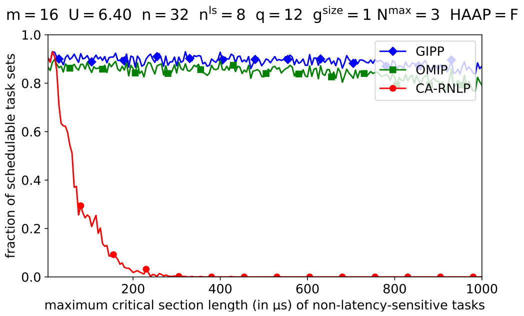

m = 16 U = 6.40 n = 32 $n^{ls} = 8$ q = 12 $g^{size} = 4$ $N^{max} = 1$ HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks

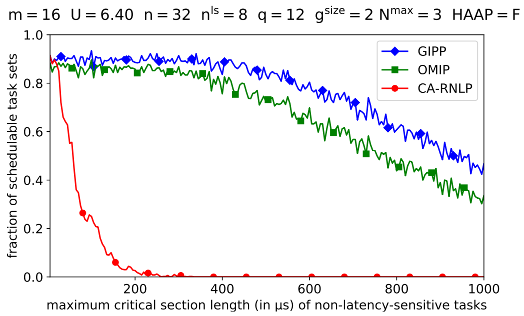

m = 16 U = 6.40 n = 32 $n^{ls} = 8$ q = 12 $g^{size} = 1$ $N^{max} = 2$ HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks

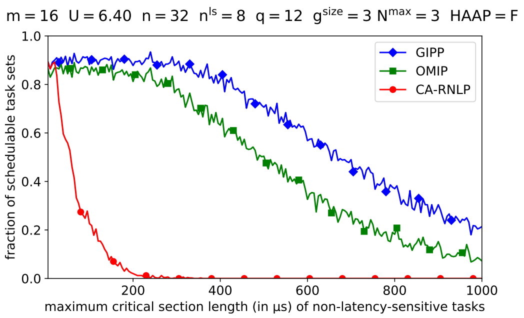

m = 16 U = 6.40 n = 32 $n^{ls} = 8$ q = 12 $g^{size} = 2$ $N^{max} = 2$ HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks

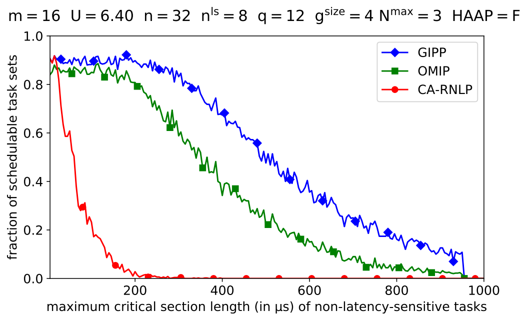

m = 16 U = 6.40 n = 32 $n^{ls} = 8$ q = 12 $g^{size} = 3$ $N^{max} = 2$ HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks

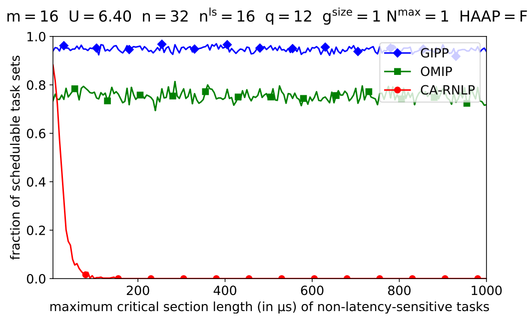

m = 16 U = 6.40 n = 32 $n^{ls} = 8$ q = 12 $g^{size} = 4$ $N^{max} = 2$ HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks

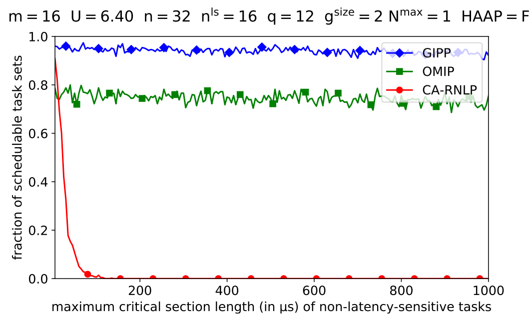

$m = 16 \quad U = 6.40 \quad n = 32 \quad n^{ls} = 8 \quad q = 12 \quad g^{size} = 1 \; N^{max} = 3 \quad HAAP = F$
fraction of schedulable task sets
GIPP
OMIP
CA-RNLP
maximum critical section length (in μs) of non-latency-sensitive tasks

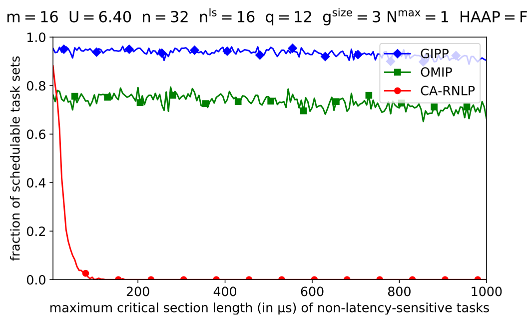

$m = 16 \quad U = 6.40 \quad n = 32 \quad n^{ls} = 8 \quad q = 12 \quad g^{size} = 2 \; N^{max} = 3 \quad HAAP = F$
fraction of schedulable task sets
GIPP
OMIP
CA-RNLP
maximum critical section length (in μs) of non-latency-sensitive tasks

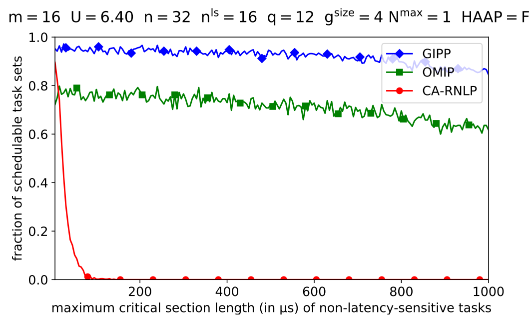

$m = 16 \quad U = 6.40 \quad n = 32 \quad n^{ls} = 8 \quad q = 12 \quad g^{size} = 3 \; N^{max} = 3 \quad HAAP = F$
fraction of schedulable task sets
GIPP
OMIP
CA-RNLP
maximum critical section length (in μs) of non-latency-sensitive tasks

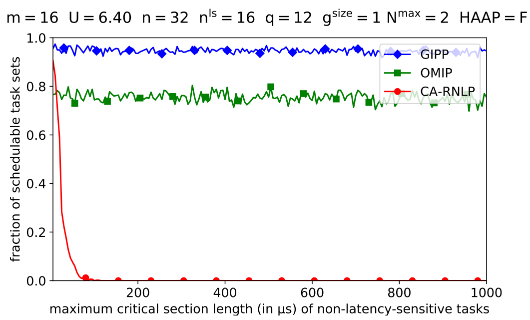

$m = 16 \quad U = 6.40 \quad n = 32 \quad n^{ls} = 8 \quad q = 12 \quad g^{size} = 4 \; N^{max} = 3 \quad HAAP = F$
fraction of schedulable task sets
GIPP
OMIP
CA-RNLP
maximum critical section length (in μs) of non-latency-sensitive tasks

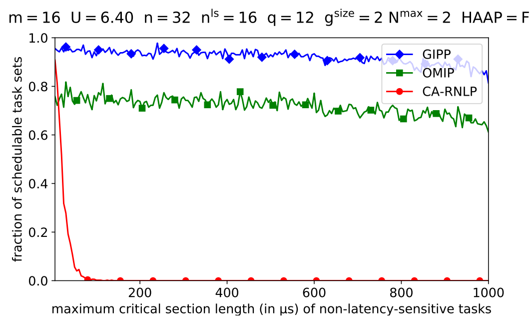

$m = 16 \quad U = 6.40 \quad n = 32 \quad n^{ls} = 16 \quad q = 12 \quad g^{size} = 1 \; N^{max} = 1 \quad HAAP = F$
fraction of schedulable task sets
GIPP
OMIP
CA-RNLP
maximum critical section length (in μs) of non-latency-sensitive tasks

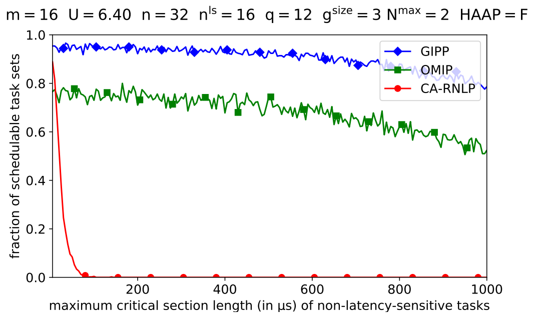

$m = 16 \quad U = 6.40 \quad n = 32 \quad n^{ls} = 16 \quad q = 12 \quad g^{size} = 2 \; N^{max} = 1 \quad HAAP = F$
fraction of schedulable task sets
GIPP
OMIP
CA-RNLP
maximum critical section length (in μs) of non-latency-sensitive tasks

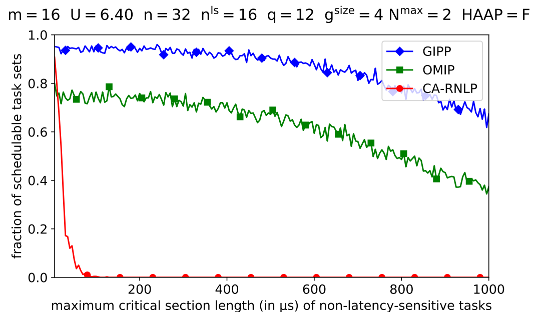

$m = 16 \quad U = 6.40 \quad n = 32 \quad n^{ls} = 16 \quad q = 12 \quad g^{size} = 3 \; N^{max} = 1 \quad HAAP = F$
fraction of schedulable task sets
GIPP
OMIP
CA-RNLP
maximum critical section length (in μs) of non-latency-sensitive tasks

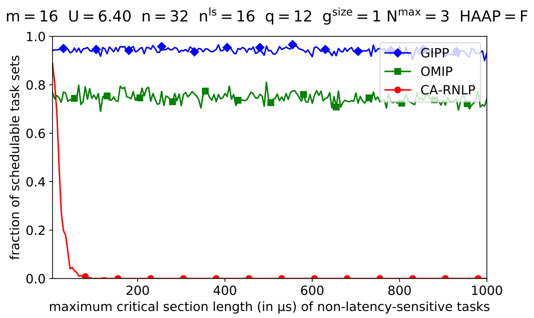

$m = 16 \quad U = 6.40 \quad n = 32 \quad n^{ls} = 16 \quad q = 12 \quad g^{size} = 4 \; N^{max} = 1 \quad HAAP = F$
fraction of schedulable task sets
GIPP
OMIP
CA-RNLP
maximum critical section length (in μs) of non-latency-sensitive tasks

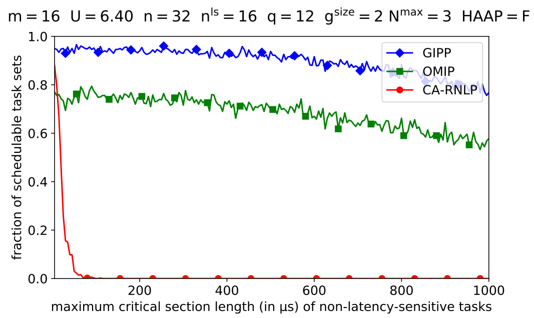

$m = 16 \quad U = 6.40 \quad n = 32 \quad n^{ls} = 16 \quad q = 12 \quad g^{size} = 1 \; N^{max} = 2 \quad HAAP = F$
fraction of schedulable task sets
GIPP
OMIP
CA-RNLP
maximum critical section length (in μs) of non-latency-sensitive tasks

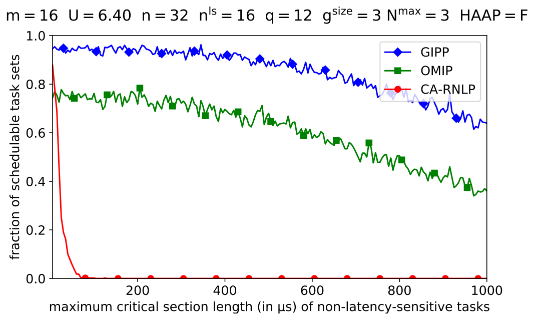

$m = 16 \quad U = 6.40 \quad n = 32 \quad n^{ls} = 16 \quad q = 12 \quad g^{size} = 2 \; N^{max} = 2 \quad HAAP = F$
fraction of schedulable task sets
GIPP
OMIP
CA-RNLP
maximum critical section length (in μs) of non-latency-sensitive tasks

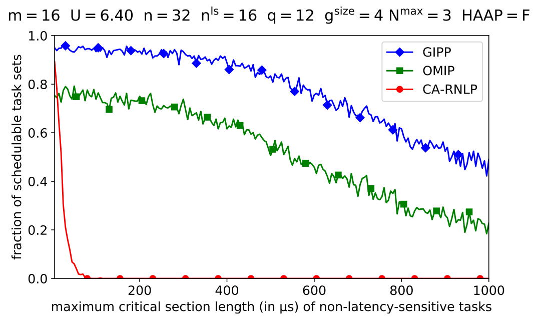

m = 16 U = 6.40 n = 32 n^{ls} = 16 q = 12 g^{size} = 3 N^{max} = 2 HAAP = F
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

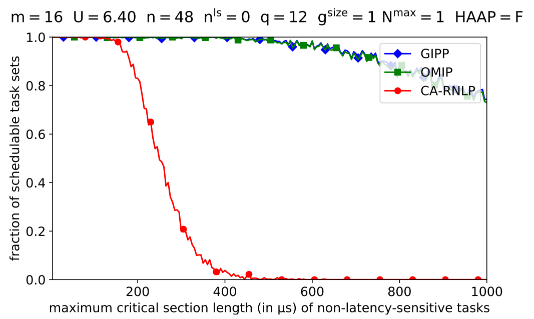

m = 16 U = 6.40 n = 32 n^{ls} = 16 q = 12 g^{size} = 4 N^{max} = 2 HAAP = F
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

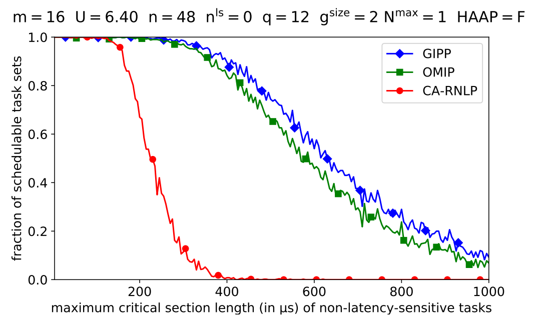

m = 16 U = 6.40 n = 32 n^{ls} = 16 q = 12 g^{size} = 1 N^{max} = 3 HAAP = F
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

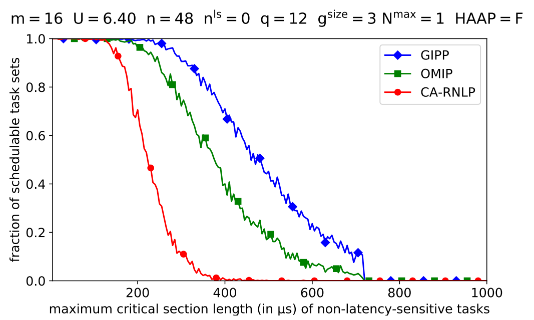

m = 16 U = 6.40 n = 32 n^{ls} = 16 q = 12 g^{size} = 2 N^{max} = 3 HAAP = F
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

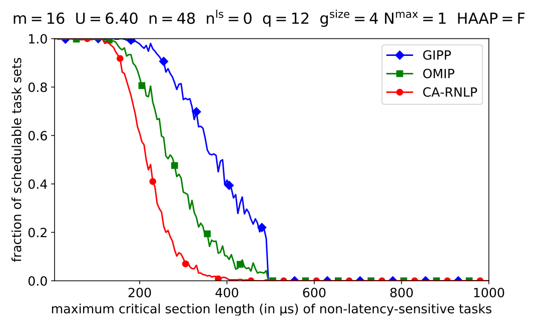

m = 16 U = 6.40 n = 32 n^{ls} = 16 q = 12 g^{size} = 3 N^{max} = 3 HAAP = F
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

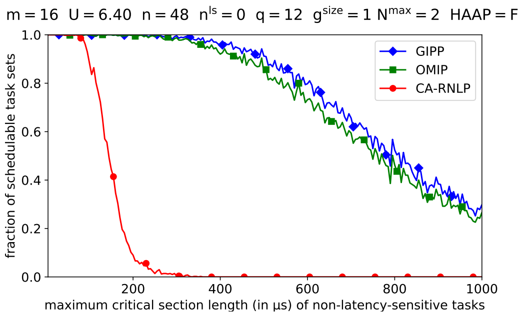

m = 16 U = 6.40 n = 32 n^{ls} = 16 q = 12 g^{size} = 4 N^{max} = 3 HAAP = F
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

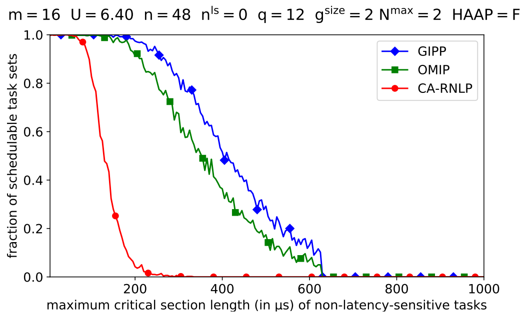

m = 16 U = 6.40 n = 48 n^{ls} = 0 q = 12 g^{size} = 1 N^{max} = 1 HAAP = F
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

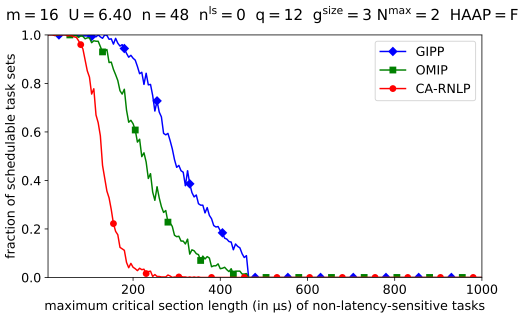

m = 16 U = 6.40 n = 48 n^{ls} = 0 q = 12 g^{size} = 2 N^{max} = 1 HAAP = F
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

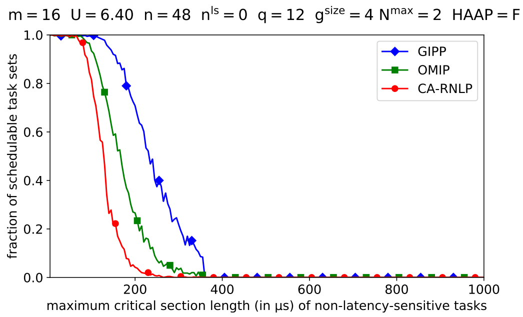

m = 16 U = 6.40 n = 48 n^{ls} = 0 q = 12 g^{size} = 3 N^{max} = 1 HAAP = F
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

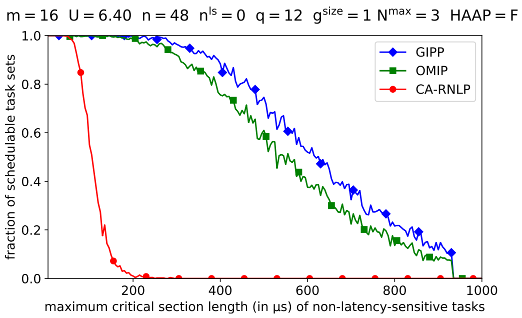

m = 16 U = 6.40 n = 48 n^{ls} = 0 q = 12 g^{size} = 4 N^{max} = 1 HAAP = F
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

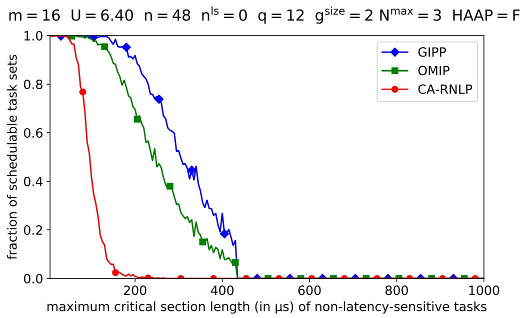

m = 16 U = 6.40 n = 48 n^{ls} = 0 q = 12 g^{size} = 1 N^{max} = 2 HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
0.0
0.2
0.4
0.6
0.8
1.0
200
400
600
800
1000
maximum critical section length (in µs) of non-latency-sensitive tasks

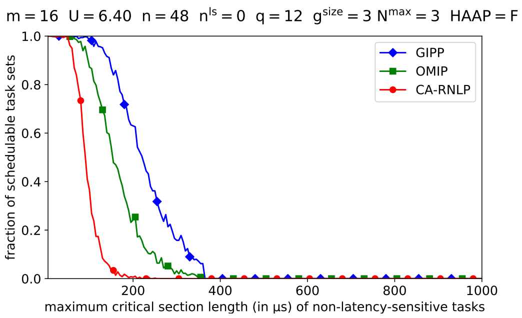

m = 16 U = 6.40 n = 48 n^{ls} = 0 q = 12 g^{size} = 2 N^{max} = 2 HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
0.0
0.2
0.4
0.6
0.8
1.0
200
400
600
800
1000
maximum critical section length (in µs) of non-latency-sensitive tasks

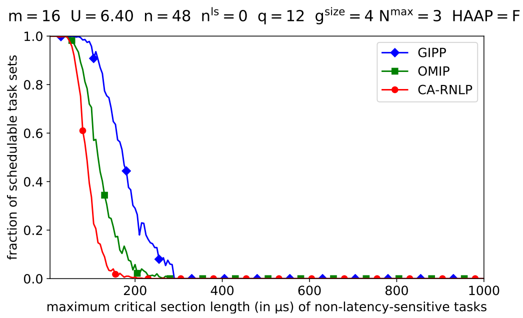

m = 16 U = 6.40 n = 48 n^{ls} = 0 q = 12 g^{size} = 3 N^{max} = 2 HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
0.0
0.2
0.4
0.6
0.8
1.0
200
400
600
800
1000
maximum critical section length (in µs) of non-latency-sensitive tasks

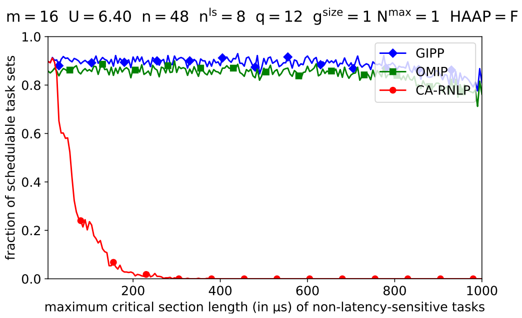

m = 16 U = 6.40 n = 48 n^{ls} = 0 q = 12 g^{size} = 4 N^{max} = 2 HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
0.0
0.2
0.4
0.6
0.8
1.0
200
400
600
800
1000
maximum critical section length (in µs) of non-latency-sensitive tasks

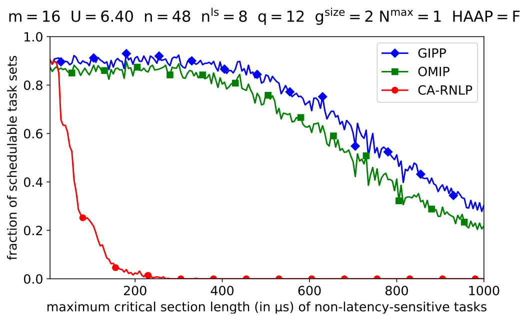

m = 16 U = 6.40 n = 48 n^{ls} = 0 q = 12 g^{size} = 1 N^{max} = 3 HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
0.0
0.2
0.4
0.6
0.8
1.0
200
400
600
800
1000
maximum critical section length (in µs) of non-latency-sensitive tasks

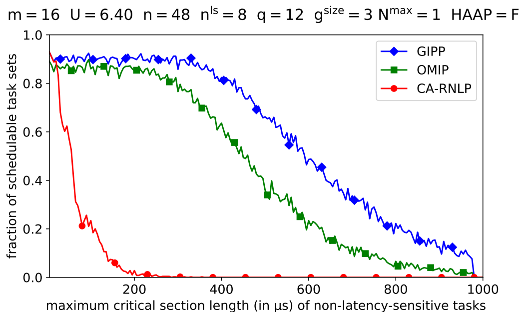

m = 16 U = 6.40 n = 48 n^{ls} = 0 q = 12 g^{size} = 2 N^{max} = 3 HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
0.0
0.2
0.4
0.6
0.8
1.0
200
400
600
800
1000
maximum critical section length (in µs) of non-latency-sensitive tasks

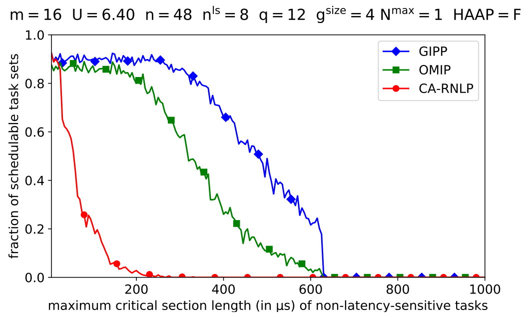

m = 16 U = 6.40 n = 48 n^{ls} = 0 q = 12 g^{size} = 3 N^{max} = 3 HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
0.0
0.2
0.4
0.6
0.8
1.0
200
400
600
800
1000
maximum critical section length (in µs) of non-latency-sensitive tasks

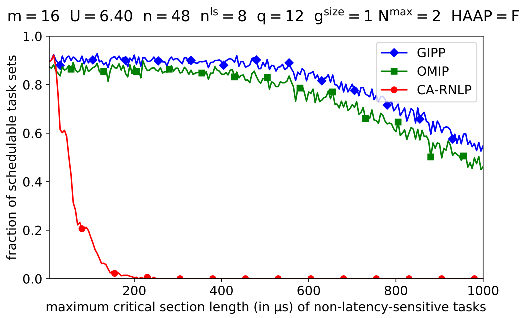

m = 16 U = 6.40 n = 48 n^{ls} = 0 q = 12 g^{size} = 4 N^{max} = 3 HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
0.0
0.2
0.4
0.6
0.8
1.0
200
400
600
800
1000
maximum critical section length (in µs) of non-latency-sensitive tasks

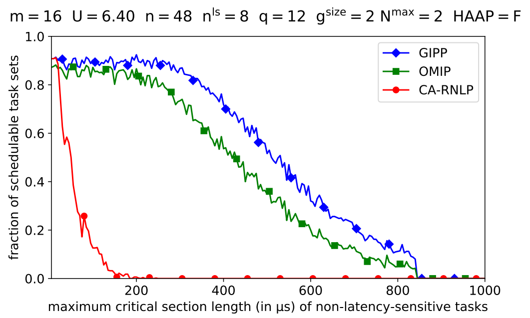

m = 16 U = 6.40 n = 48 n^{ls} = 8 q = 12 g^{size} = 1 N^{max} = 1 HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
0.0
0.2
0.4
0.6
0.8
1.0
200
400
600
800
1000
maximum critical section length (in µs) of non-latency-sensitive tasks

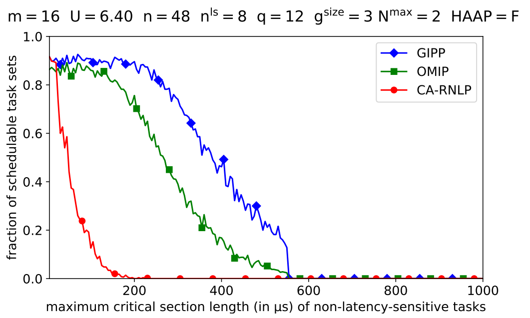

m = 16 U = 6.40 n = 48 n^{ls} = 8 q = 12 g^{size} = 2 N^{max} = 1 HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
0.0
0.2
0.4
0.6
0.8
1.0
200
400
600
800
1000
maximum critical section length (in µs) of non-latency-sensitive tasks

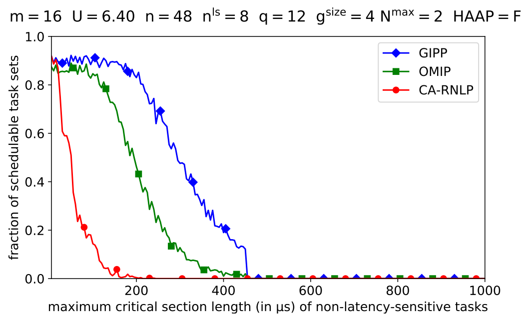

m = 16 U = 6.40 n = 48 n^ls = 8 q = 12 g^size = 3 N^max = 1 HAAP = F
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

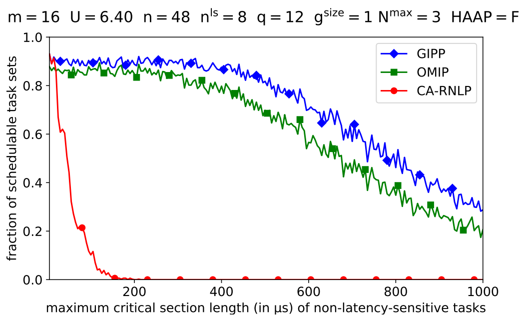

m = 16 U = 6.40 n = 48 n^ls = 8 q = 12 g^size = 4 N^max = 1 HAAP = F
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

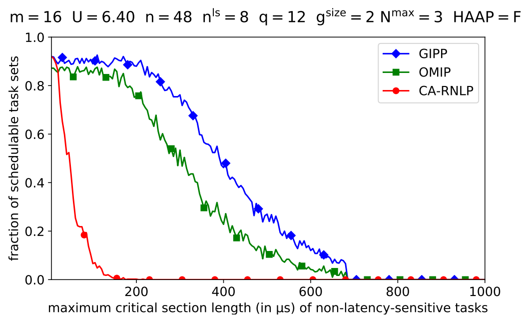

m = 16 U = 6.40 n = 48 n^ls = 8 q = 12 g^size = 1 N^max = 2 HAAP = F
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

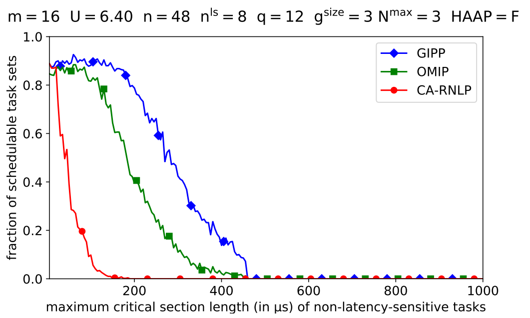

m = 16 U = 6.40 n = 48 n^ls = 8 q = 12 g^size = 2 N^max = 2 HAAP = F
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

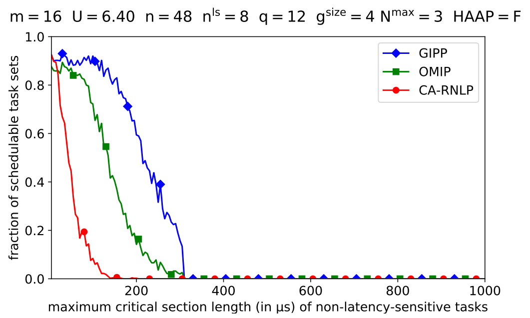

m = 16 U = 6.40 n = 48 n^ls = 8 q = 12 g^size = 3 N^max = 2 HAAP = F
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

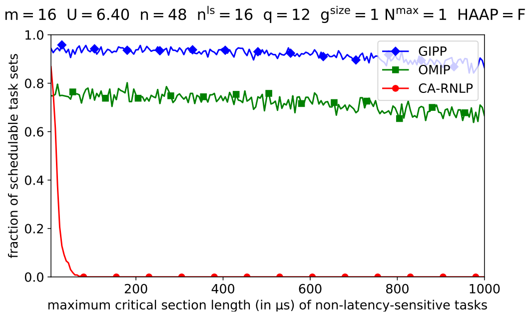

m = 16 U = 6.40 n = 48 n^ls = 8 q = 12 g^size = 4 N^max = 2 HAAP = F
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

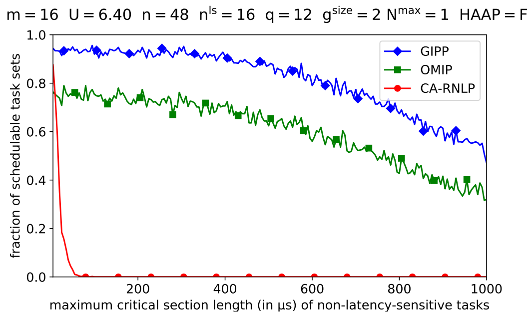

m = 16 U = 6.40 n = 48 n^ls = 8 q = 12 g^size = 1 N^max = 3 HAAP = F
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

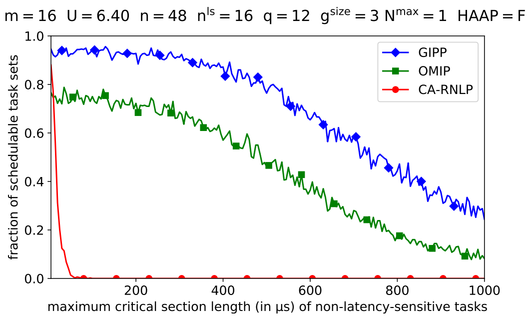

m = 16 U = 6.40 n = 48 n^ls = 8 q = 12 g^size = 2 N^max = 3 HAAP = F
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

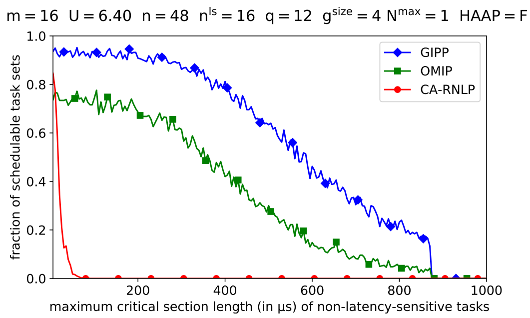

m = 16 U = 6.40 n = 48 n^ls = 8 q = 12 g^size = 3 N^max = 3 HAAP = F
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

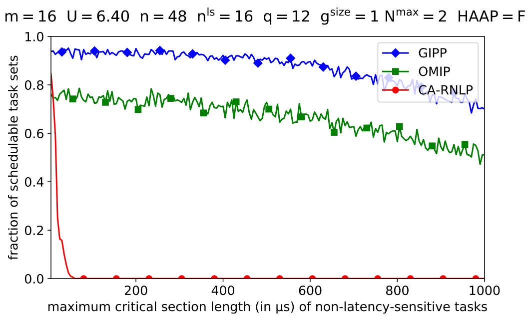

m = 16 U = 6.40 n = 48 n^ls = 8 q = 12 g^size = 4 N^max = 3 HAAP = F
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

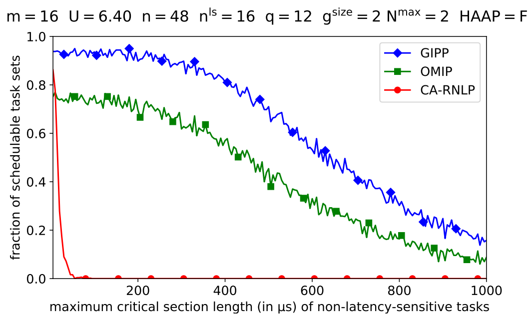

m = 16 U = 6.40 n = 48 n^{ls} = 16 q = 12 g^{size} = 1 N^{max} = 1 HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

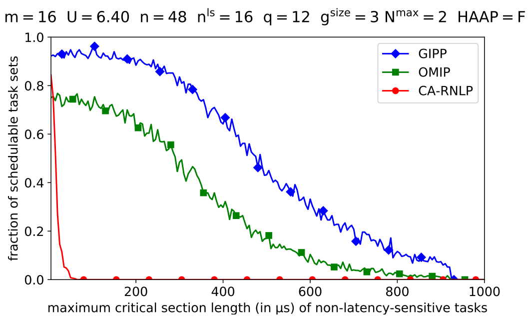

m = 16 U = 6.40 n = 48 n^{ls} = 16 q = 12 g^{size} = 2 N^{max} = 1 HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

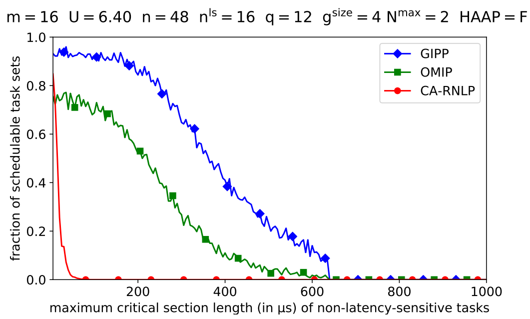

m = 16 U = 6.40 n = 48 n^{ls} = 16 q = 12 g^{size} = 3 N^{max} = 1 HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

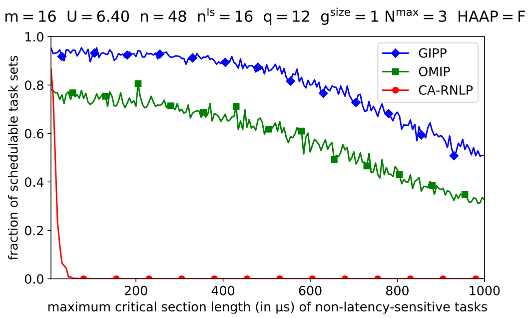

m = 16 U = 6.40 n = 48 n^{ls} = 16 q = 12 g^{size} = 4 N^{max} = 1 HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

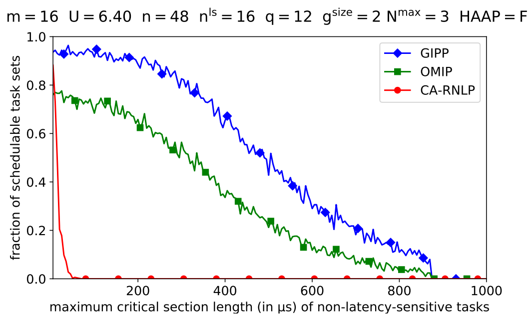

m = 16 U = 6.40 n = 48 n^{ls} = 16 q = 12 g^{size} = 1 N^{max} = 2 HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

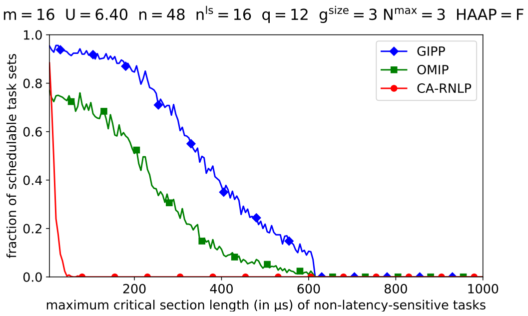

m = 16 U = 6.40 n = 48 n^{ls} = 16 q = 12 g^{size} = 2 N^{max} = 2 HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

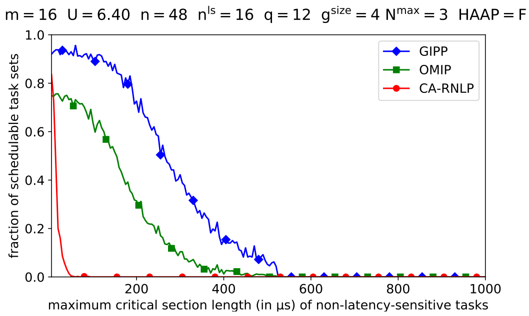

m = 16 U = 6.40 n = 48 n^{ls} = 16 q = 12 g^{size} = 3 N^{max} = 2 HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

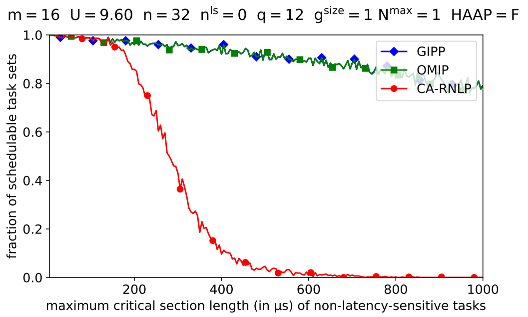

m = 16 U = 6.40 n = 48 n^{ls} = 16 q = 12 g^{size} = 4 N^{max} = 2 HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

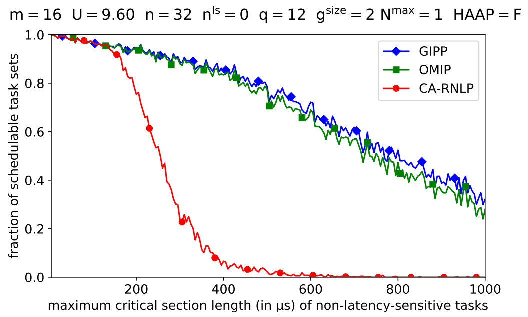

m = 16 U = 6.40 n = 48 n^{ls} = 16 q = 12 g^{size} = 1 N^{max} = 3 HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

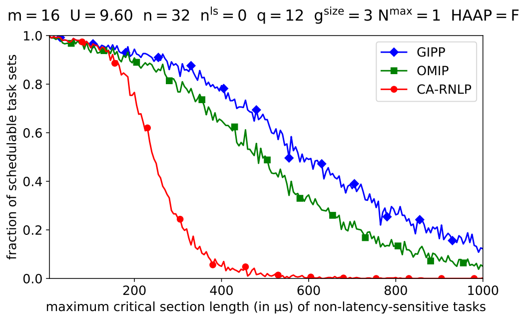

m = 16 U = 6.40 n = 48 n^{ls} = 16 q = 12 g^{size} = 2 N^{max} = 3 HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

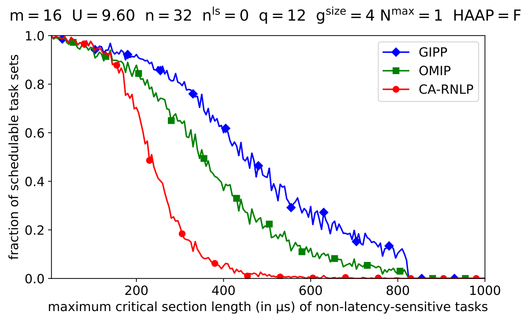

m = 16 U = 6.40 n = 48 n^{ls} = 16 q = 12 g^{size} = 3 N^{max} = 3 HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

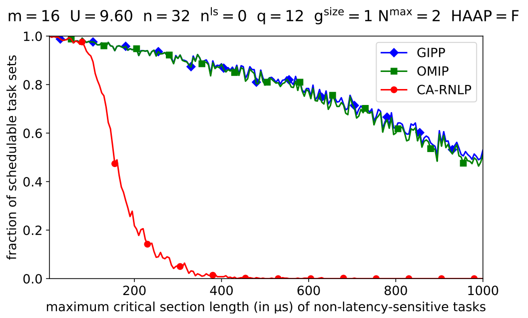

m = 16 U = 6.40 n = 48 n^{ls} = 16 q = 12 g^{size} = 4 N^{max} = 3 HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

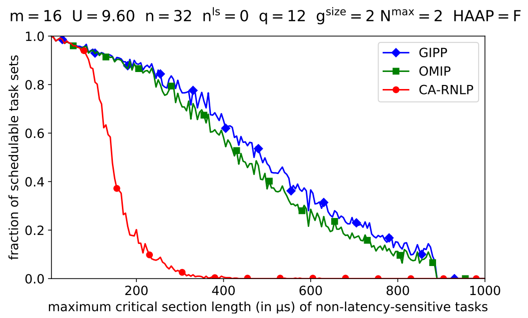

m = 16 U = 9.60 n = 32 n^{ls} = 0 q = 12 g^{size} = 1 N^{max} = 1 HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

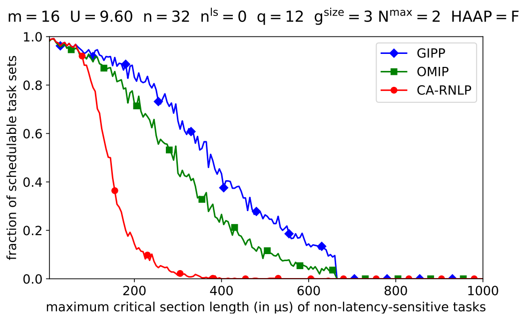

m = 16 U = 9.60 n = 32 n^{ls} = 0 q = 12 g^{size} = 2 N^{max} = 1 HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

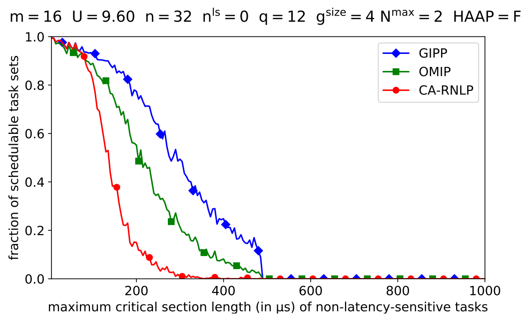

m = 16 U = 9.60 n = 32 n^{ls} = 0 q = 12 g^{size} = 3 N^{max} = 1 HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

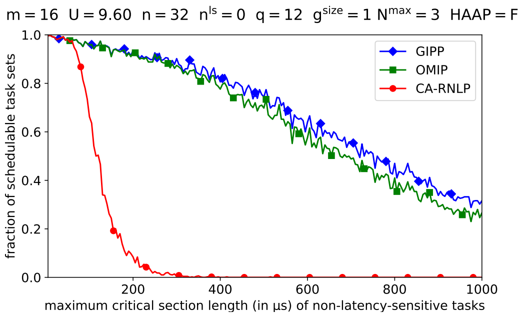

m = 16 U = 9.60 n = 32 n^{ls} = 0 q = 12 g^{size} = 4 N^{max} = 1 HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

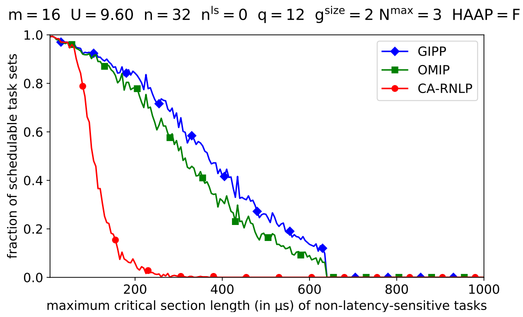

m = 16 U = 9.60 n = 32 n^{ls} = 0 q = 12 g^{size} = 1 N^{max} = 2 HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

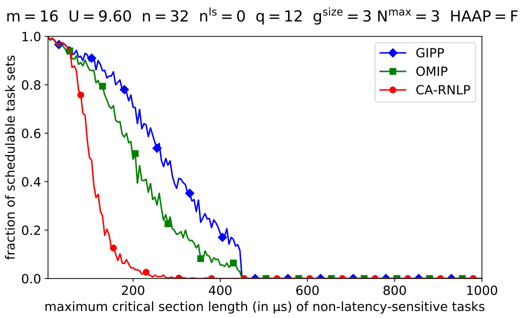

m = 16 U = 9.60 n = 32 n^{ls} = 0 q = 12 g^{size} = 2 N^{max} = 2 HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

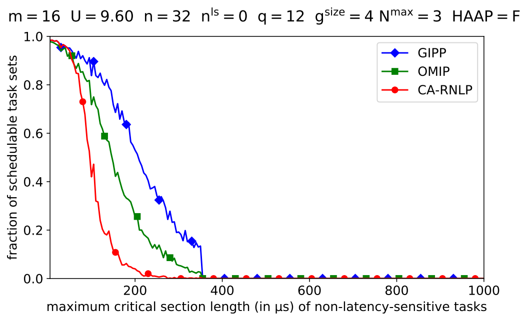

m = 16 U = 9.60 n = 32 n^{ls} = 0 q = 12 g^{size} = 3 N^{max} = 2 HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

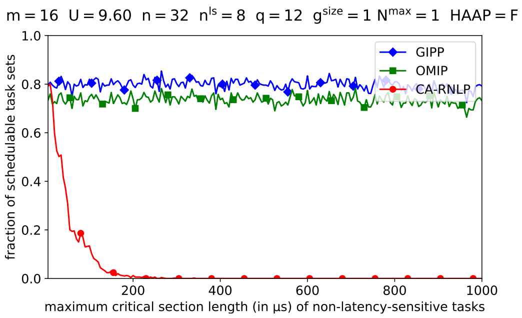

m = 16 U = 9.60 n = 32 n^{ls} = 0 q = 12 g^{size} = 4 N^{max} = 2 HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

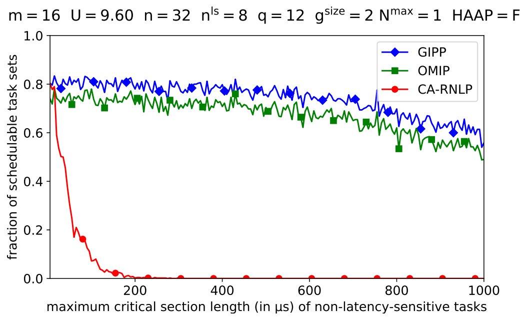

m = 16 U = 9.60 n = 32 n^{ls} = 0 q = 12 g^{size} = 1 N^{max} = 3 HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks

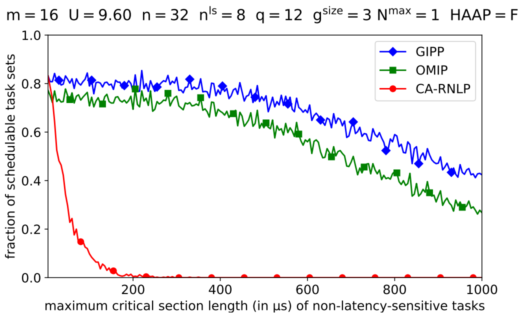

m = 16 U = 9.60 n = 32 n^{ls} = 0 q = 12 g^{size} = 2 N^{max} = 3 HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks

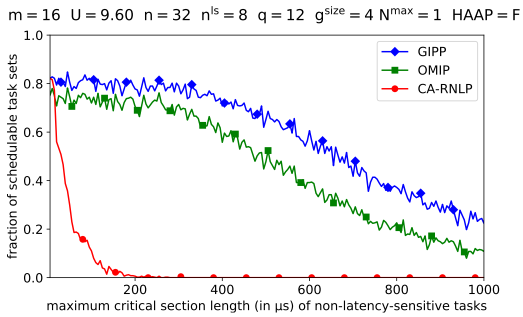

m = 16 U = 9.60 n = 32 n^{ls} = 0 q = 12 g^{size} = 3 N^{max} = 3 HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks

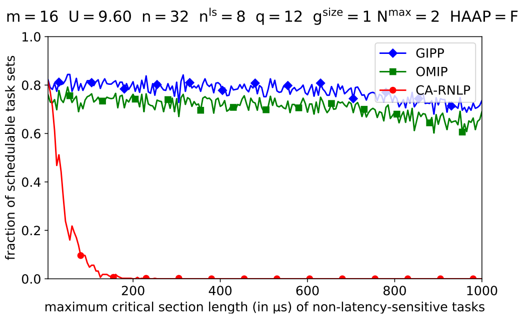

m = 16 U = 9.60 n = 32 n^{ls} = 0 q = 12 g^{size} = 4 N^{max} = 3 HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks

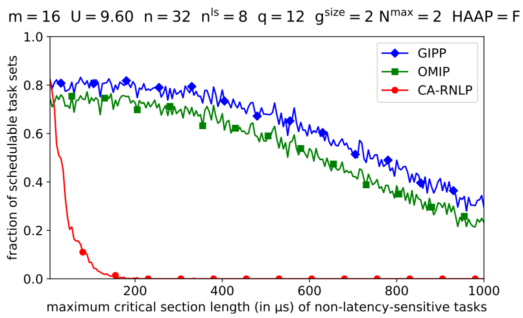

m = 16 U = 9.60 n = 32 n^{ls} = 8 q = 12 g^{size} = 1 N^{max} = 1 HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks

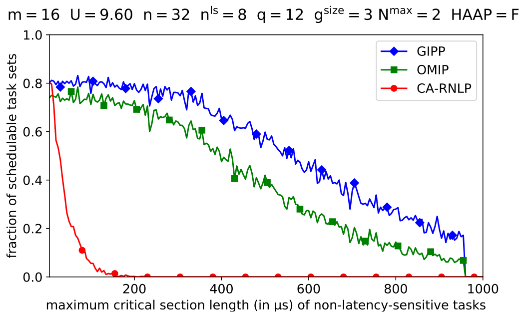

m = 16 U = 9.60 n = 32 n^{ls} = 8 q = 12 g^{size} = 2 N^{max} = 1 HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks

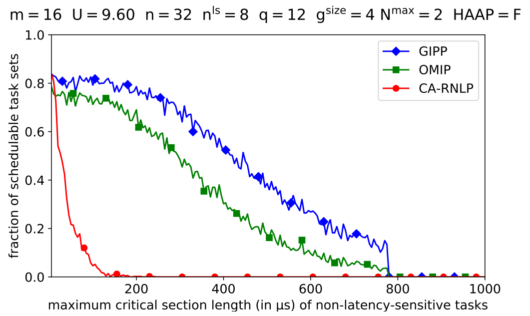

m = 16 U = 9.60 n = 32 n^{ls} = 8 q = 12 g^{size} = 3 N^{max} = 1 HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks

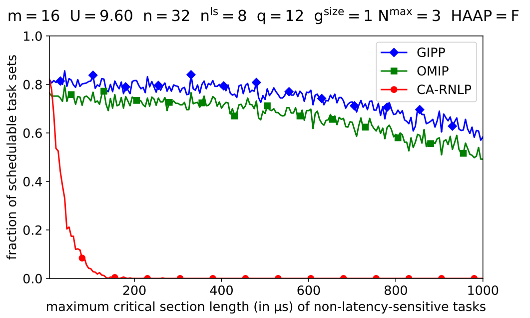

m = 16 U = 9.60 n = 32 n^{ls} = 8 q = 12 g^{size} = 4 N^{max} = 1 HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks

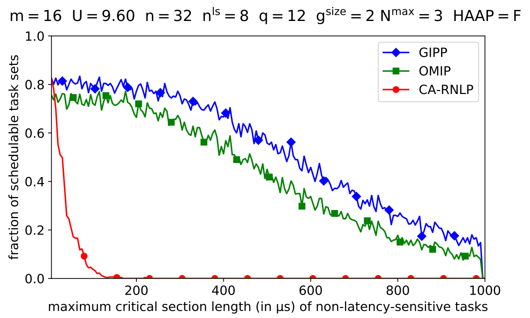

m = 16 U = 9.60 n = 32 n^{ls} = 8 q = 12 g^{size} = 1 N^{max} = 2 HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks

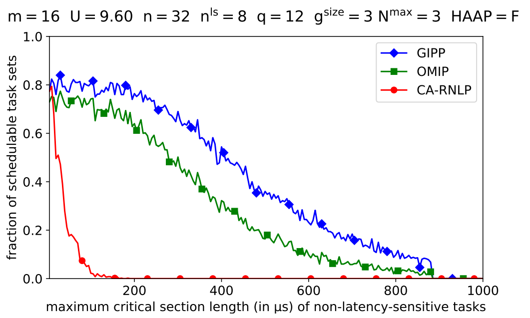

m = 16 U = 9.60 n = 32 n^{ls} = 8 q = 12 g^{size} = 2 N^{max} = 2 HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks

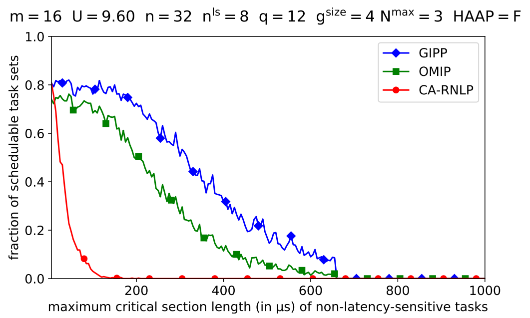

$m = 16$ $U = 9.60$ $n = 32$ $n^{ls} = 8$ $q = 12$ $g^{size} = 3$ $N^{max} = 2$ HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

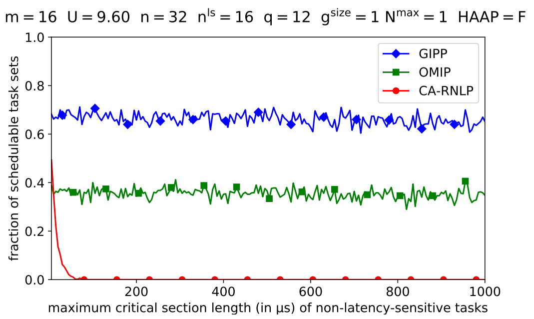

$m = 16$ $U = 9.60$ $n = 32$ $n^{ls} = 8$ $q = 12$ $g^{size} = 4$ $N^{max} = 2$ HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

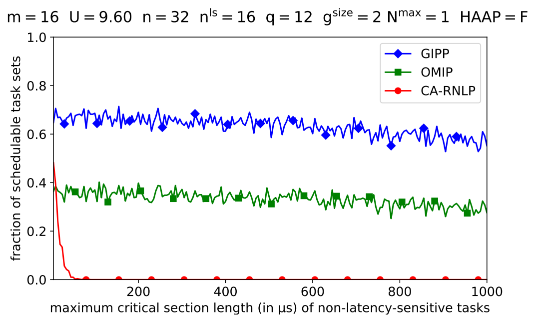

$m = 16$ $U = 9.60$ $n = 32$ $n^{ls} = 8$ $q = 12$ $g^{size} = 1$ $N^{max} = 3$ HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

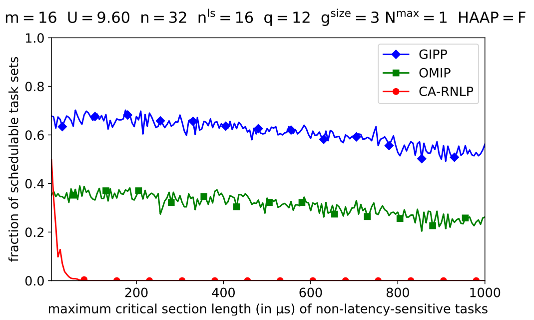

$m = 16$ $U = 9.60$ $n = 32$ $n^{ls} = 8$ $q = 12$ $g^{size} = 2$ $N^{max} = 3$ HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

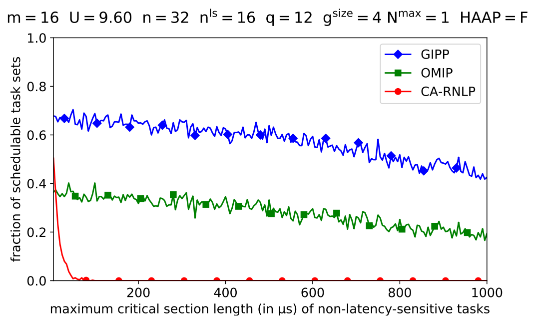

$m = 16$ $U = 9.60$ $n = 32$ $n^{ls} = 8$ $q = 12$ $g^{size} = 3$ $N^{max} = 3$ HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

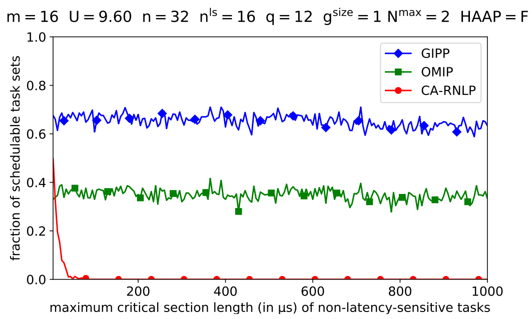

$m = 16$ $U = 9.60$ $n = 32$ $n^{ls} = 8$ $q = 12$ $g^{size} = 4$ $N^{max} = 3$ HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

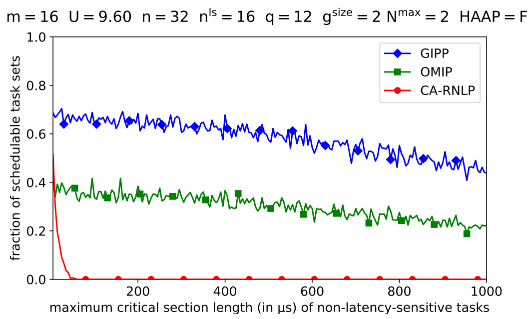

$m = 16$ $U = 9.60$ $n = 32$ $n^{ls} = 16$ $q = 12$ $g^{size} = 1$ $N^{max} = 1$ HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

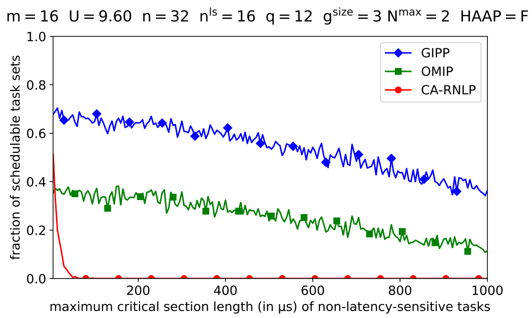

$m = 16$ $U = 9.60$ $n = 32$ $n^{ls} = 16$ $q = 12$ $g^{size} = 2$ $N^{max} = 1$ HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

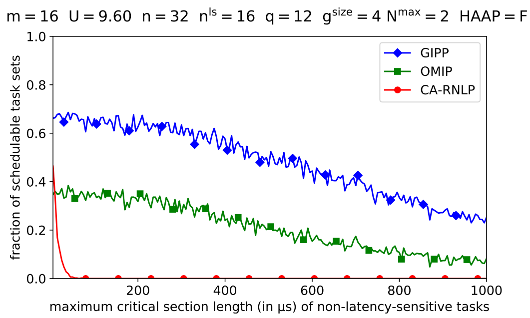

$m = 16$ $U = 9.60$ $n = 32$ $n^{ls} = 16$ $q = 12$ $g^{size} = 3$ $N^{max} = 1$ HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

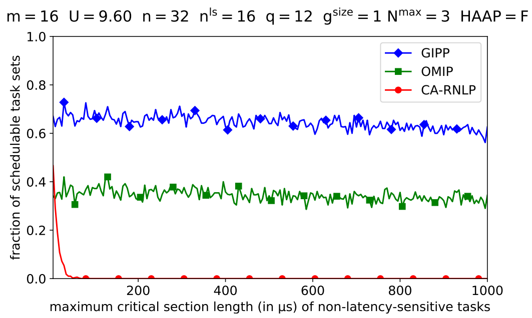

$m = 16$ $U = 9.60$ $n = 32$ $n^{ls} = 16$ $q = 12$ $g^{size} = 4$ $N^{max} = 1$ HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

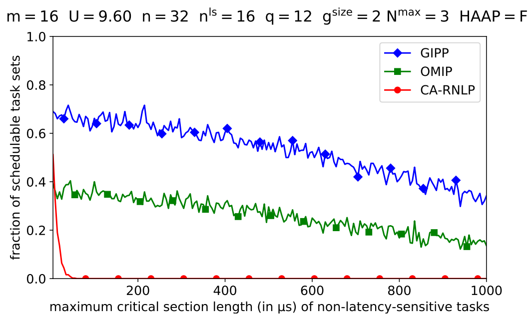

$m = 16 \;\; U = 9.60 \;\; n = 32 \;\; n^{ls} = 16 \;\; q = 12 \;\; g^{size} = 1 \; N^{max} = 2 \;\; HAAP = F$
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

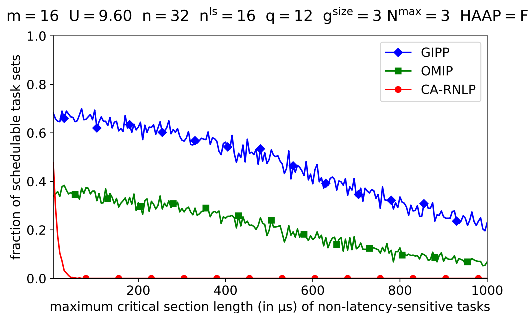

$m = 16 \;\; U = 9.60 \;\; n = 32 \;\; n^{ls} = 16 \;\; q = 12 \;\; g^{size} = 2 \; N^{max} = 2 \;\; HAAP = F$
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

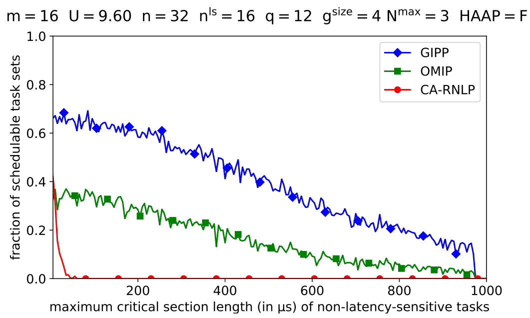

$m = 16 \;\; U = 9.60 \;\; n = 32 \;\; n^{ls} = 16 \;\; q = 12 \;\; g^{size} = 3 \; N^{max} = 2 \;\; HAAP = F$
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

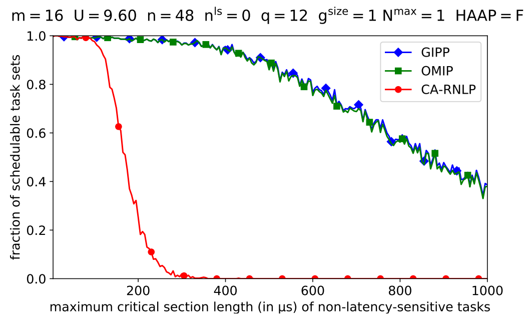

$m = 16 \;\; U = 9.60 \;\; n = 32 \;\; n^{ls} = 16 \;\; q = 12 \;\; g^{size} = 4 \; N^{max} = 2 \;\; HAAP = F$
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

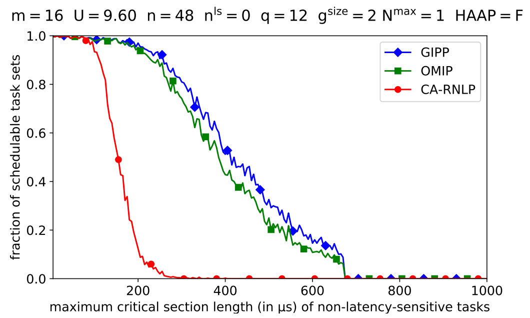

$m = 16 \;\; U = 9.60 \;\; n = 32 \;\; n^{ls} = 16 \;\; q = 12 \;\; g^{size} = 1 \; N^{max} = 3 \;\; HAAP = F$
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

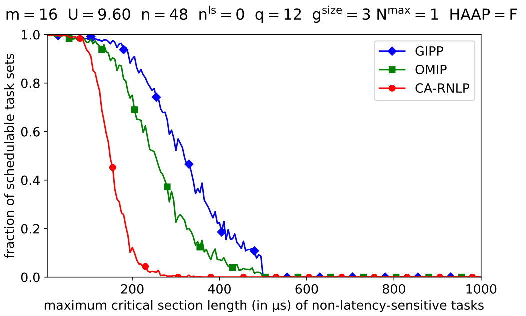

$m = 16 \;\; U = 9.60 \;\; n = 32 \;\; n^{ls} = 16 \;\; q = 12 \;\; g^{size} = 2 \; N^{max} = 3 \;\; HAAP = F$
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

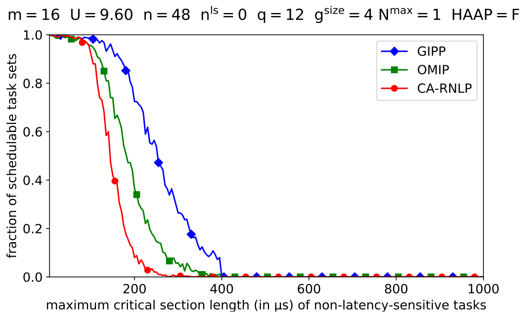

$m = 16 \;\; U = 9.60 \;\; n = 32 \;\; n^{ls} = 16 \;\; q = 12 \;\; g^{size} = 3 \; N^{max} = 3 \;\; HAAP = F$
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

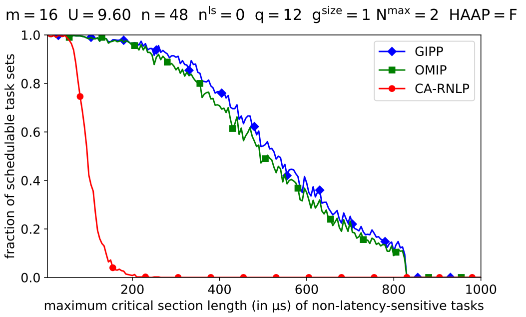

$m = 16 \;\; U = 9.60 \;\; n = 32 \;\; n^{ls} = 16 \;\; q = 12 \;\; g^{size} = 4 \; N^{max} = 3 \;\; HAAP = F$
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

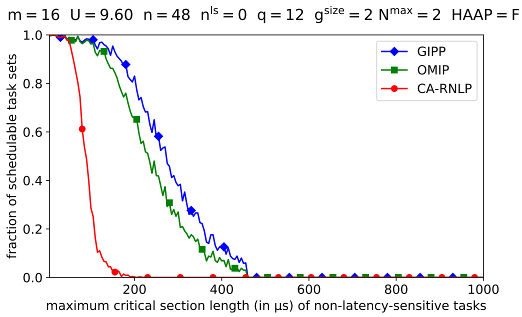

$m = 16 \;\; U = 9.60 \;\; n = 48 \;\; n^{ls} = 0 \;\; q = 12 \;\; g^{size} = 1 \; N^{max} = 1 \;\; HAAP = F$
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

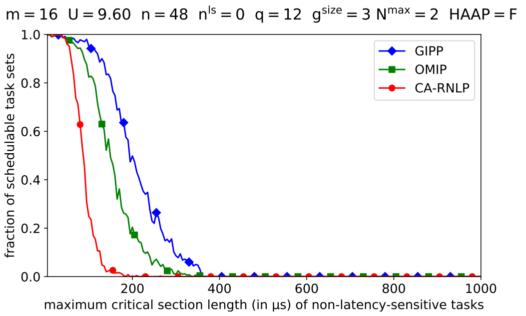

$m = 16 \;\; U = 9.60 \;\; n = 48 \;\; n^{ls} = 0 \;\; q = 12 \;\; g^{size} = 2 \; N^{max} = 1 \;\; HAAP = F$
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

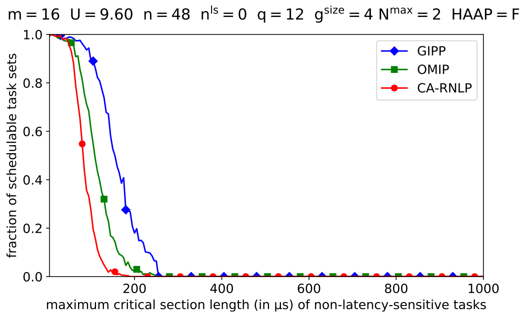

$m = 16 \;\; U = 9.60 \;\; n = 48 \;\; n^{ls} = 0 \;\; q = 12 \;\; g^{size} = 3 \; N^{max} = 1 \;\; HAAP = F$
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

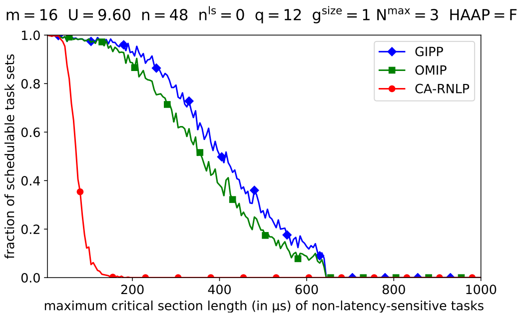

$m = 16 \;\; U = 9.60 \;\; n = 48 \;\; n^{ls} = 0 \;\; q = 12 \;\; g^{size} = 4 \; N^{max} = 1 \;\; HAAP = F$
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

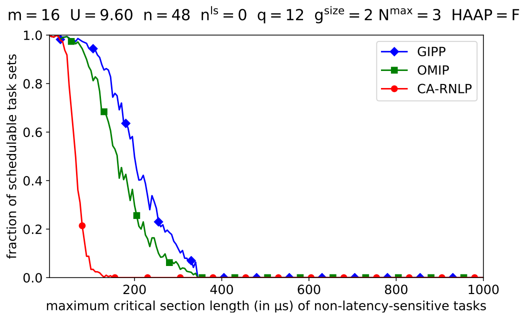

$m = 16 \;\; U = 9.60 \;\; n = 48 \;\; n^{ls} = 0 \;\; q = 12 \;\; g^{size} = 1 \; N^{max} = 2 \;\; HAAP = F$
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

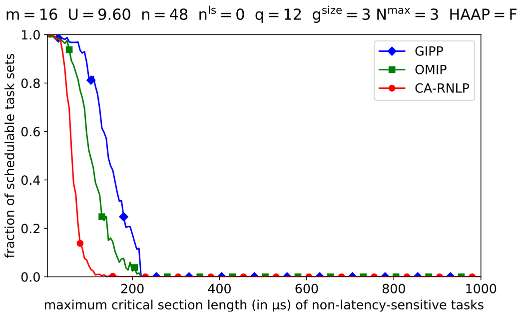

$m = 16 \;\; U = 9.60 \;\; n = 48 \;\; n^{ls} = 0 \;\; q = 12 \;\; g^{size} = 2 \; N^{max} = 2 \;\; HAAP = F$
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

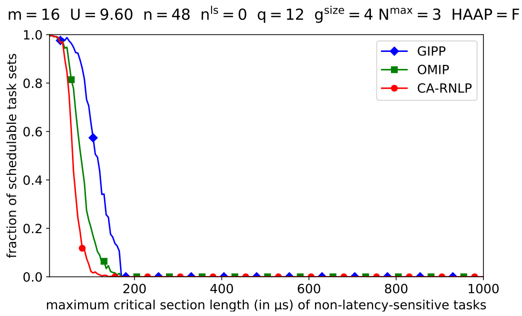

$m = 16 \;\; U = 9.60 \;\; n = 48 \;\; n^{ls} = 0 \;\; q = 12 \;\; g^{size} = 3 \; N^{max} = 2 \;\; HAAP = F$
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

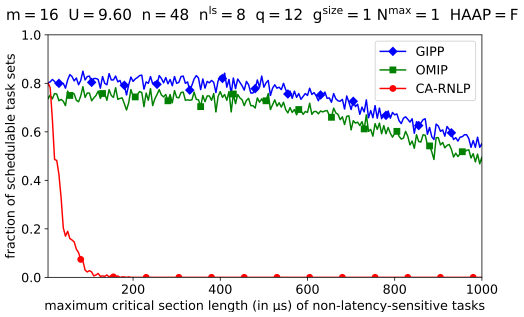

$m = 16 \;\; U = 9.60 \;\; n = 48 \;\; n^{ls} = 0 \;\; q = 12 \;\; g^{size} = 4 \; N^{max} = 2 \;\; HAAP = F$
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

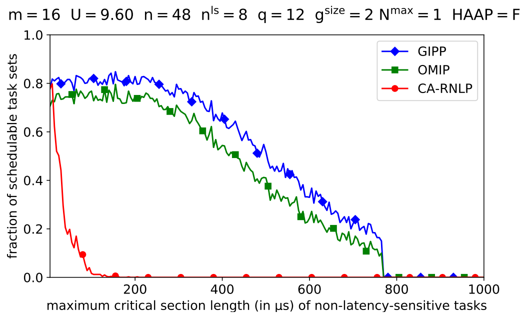

$m = 16 \;\; U = 9.60 \;\; n = 48 \;\; n^{ls} = 0 \;\; q = 12 \;\; g^{size} = 1 \; N^{max} = 3 \;\; HAAP = F$
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

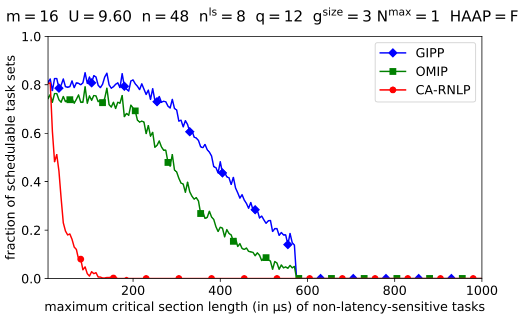

$m = 16 \;\; U = 9.60 \;\; n = 48 \;\; n^{ls} = 0 \;\; q = 12 \;\; g^{size} = 2 \; N^{max} = 3 \;\; HAAP = F$
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

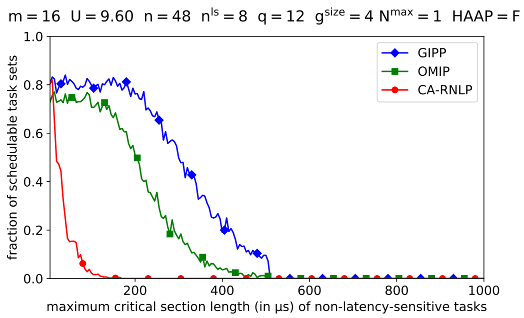

$m = 16 \;\; U = 9.60 \;\; n = 48 \;\; n^{ls} = 0 \;\; q = 12 \;\; g^{size} = 3 \; N^{max} = 3 \;\; HAAP = F$
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

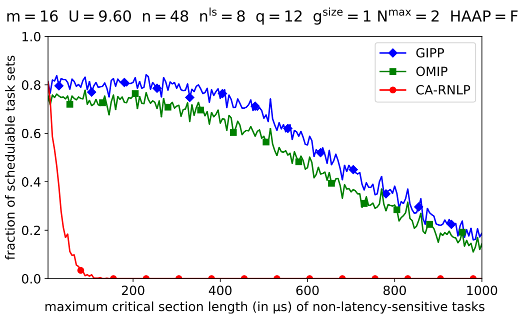

$m = 16 \;\; U = 9.60 \;\; n = 48 \;\; n^{ls} = 0 \;\; q = 12 \;\; g^{size} = 4 \; N^{max} = 3 \;\; HAAP = F$
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

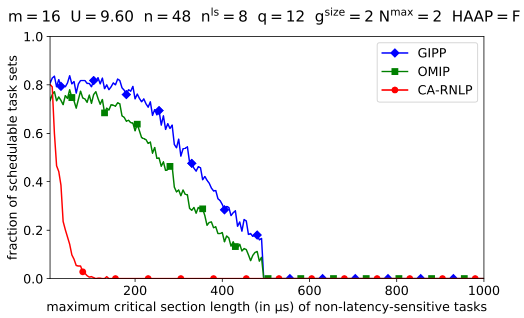

m = 16 U = 9.60 n = 48 n^{ls} = 8 q = 12 g^{size} = 1 N^{max} = 1 HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks

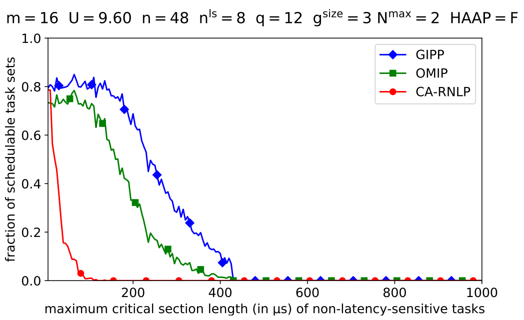

m = 16 U = 9.60 n = 48 n^{ls} = 8 q = 12 g^{size} = 2 N^{max} = 1 HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks

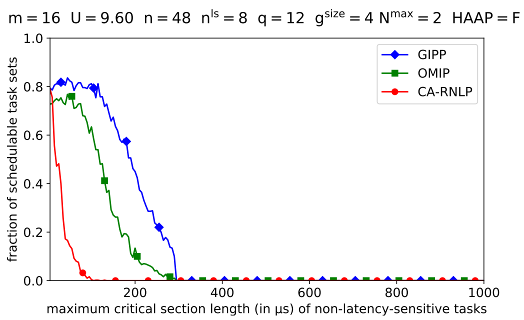

m = 16 U = 9.60 n = 48 n^{ls} = 8 q = 12 g^{size} = 3 N^{max} = 1 HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks

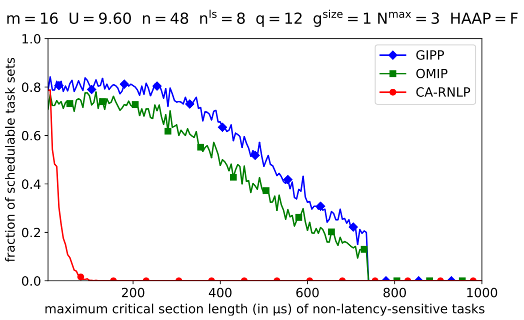

m = 16 U = 9.60 n = 48 n^{ls} = 8 q = 12 g^{size} = 4 N^{max} = 1 HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks

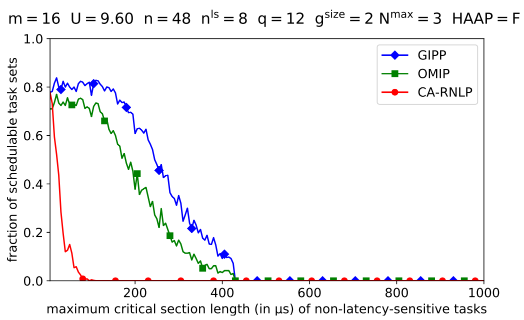

m = 16 U = 9.60 n = 48 n^{ls} = 8 q = 12 g^{size} = 1 N^{max} = 2 HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks

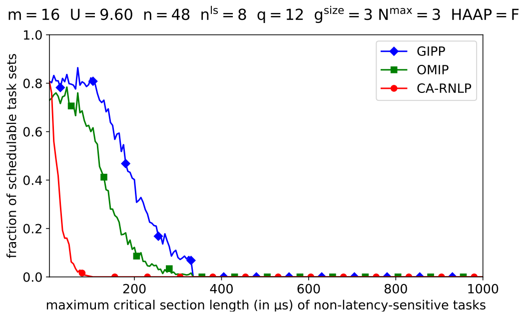

m = 16 U = 9.60 n = 48 n^{ls} = 8 q = 12 g^{size} = 2 N^{max} = 2 HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks

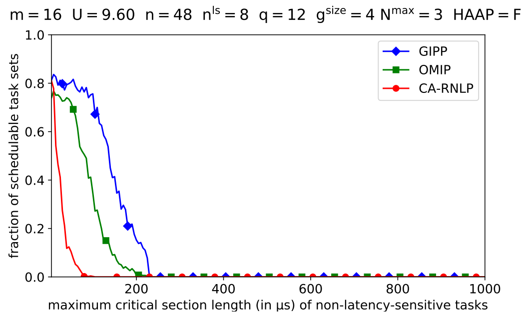

m = 16 U = 9.60 n = 48 n^{ls} = 8 q = 12 g^{size} = 3 N^{max} = 2 HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks

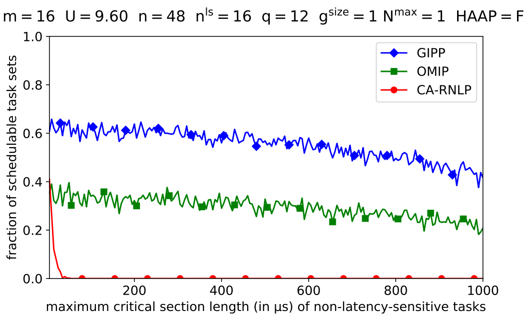

m = 16 U = 9.60 n = 48 n^{ls} = 8 q = 12 g^{size} = 4 N^{max} = 2 HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks

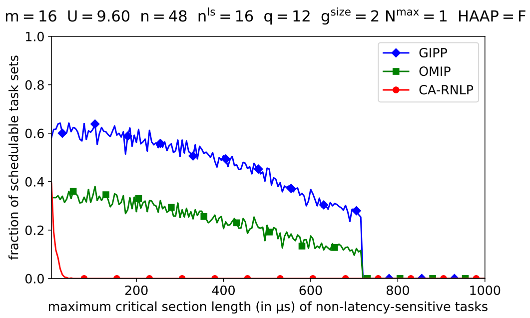

m = 16 U = 9.60 n = 48 n^{ls} = 8 q = 12 g^{size} = 1 N^{max} = 3 HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks

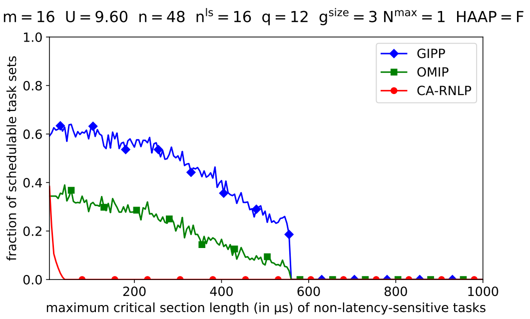

m = 16 U = 9.60 n = 48 n^{ls} = 8 q = 12 g^{size} = 2 N^{max} = 3 HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks

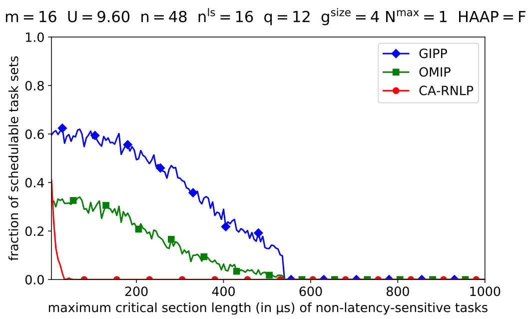

m = 16 U = 9.60 n = 48 n^{ls} = 8 q = 12 g^{size} = 3 N^{max} = 3 HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

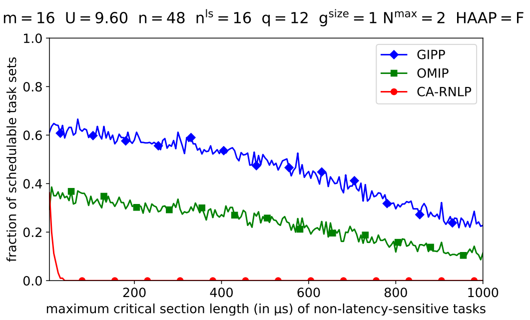

m = 16 U = 9.60 n = 48 n^{ls} = 8 q = 12 g^{size} = 4 N^{max} = 3 HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

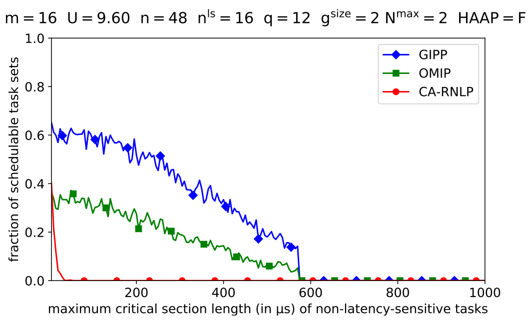

m = 16 U = 9.60 n = 48 n^{ls} = 16 q = 12 g^{size} = 1 N^{max} = 1 HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

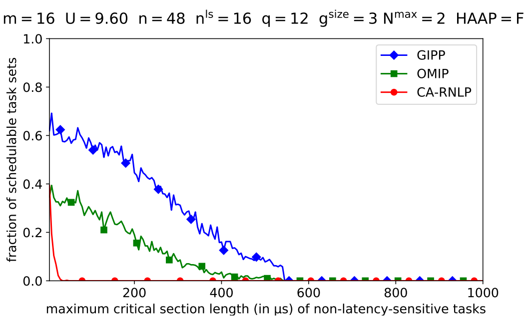

m = 16 U = 9.60 n = 48 n^{ls} = 16 q = 12 g^{size} = 2 N^{max} = 1 HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

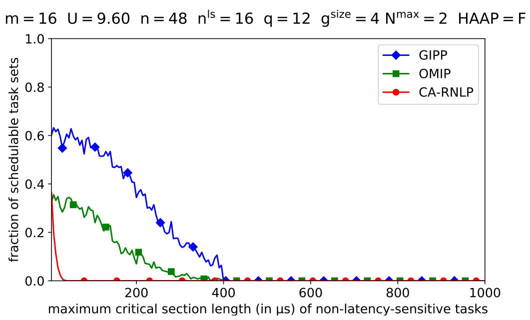

m = 16 U = 9.60 n = 48 n^{ls} = 16 q = 12 g^{size} = 3 N^{max} = 1 HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

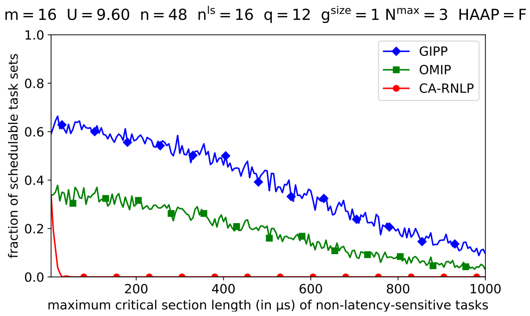

m = 16 U = 9.60 n = 48 n^{ls} = 16 q = 12 g^{size} = 4 N^{max} = 1 HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

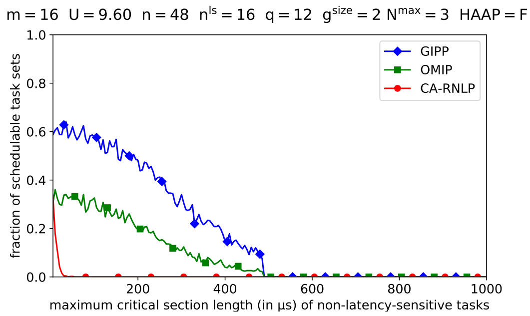

m = 16 U = 9.60 n = 48 n^{ls} = 16 q = 12 g^{size} = 1 N^{max} = 2 HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

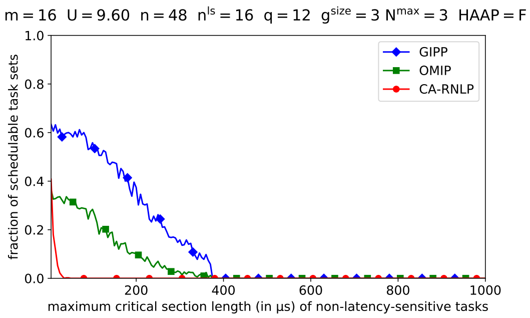

m = 16 U = 9.60 n = 48 n^{ls} = 16 q = 12 g^{size} = 2 N^{max} = 2 HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

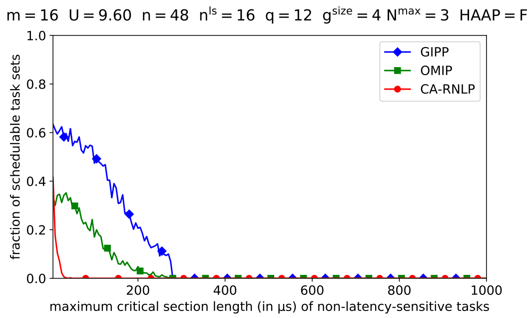

m = 16 U = 9.60 n = 48 n^{ls} = 16 q = 12 g^{size} = 3 N^{max} = 2 HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks


m = 16 U = 9.60 n = 48 n^{ls} = 16 q = 12 g^{size} = 4 N^{max} = 2 HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

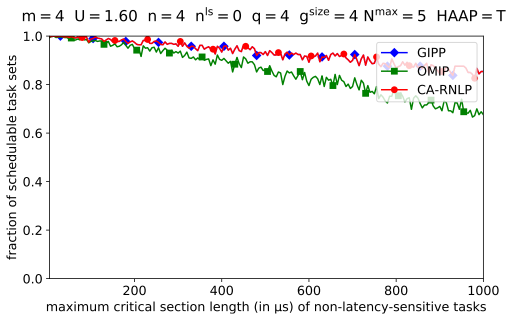

m = 16 U = 9.60 n = 48 n^ls = 16 q = 12 g^size = 1 N^max = 3 HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

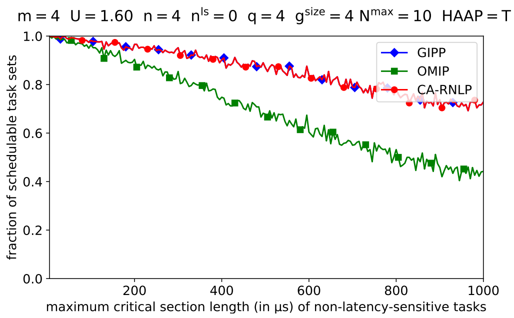

m = 16 U = 9.60 n = 48 n^ls = 16 q = 12 g^size = 2 N^max = 3 HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

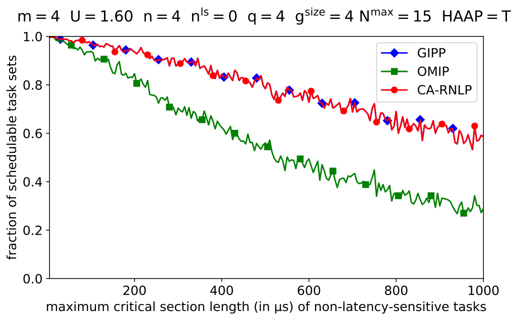

m = 16 U = 9.60 n = 48 n^ls = 16 q = 12 g^size = 3 N^max = 3 HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

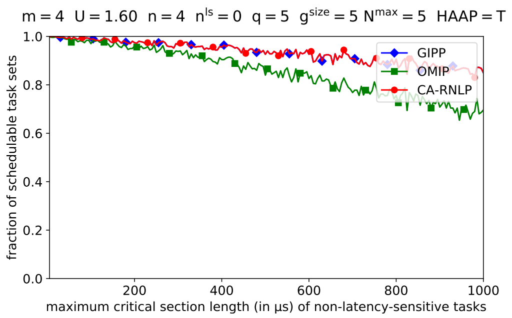

m = 16 U = 9.60 n = 48 n^ls = 16 q = 12 g^size = 4 N^max = 3 HAAP = F
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

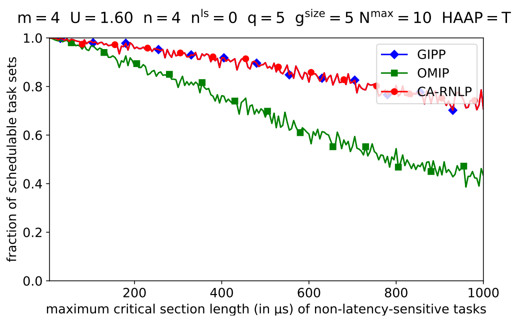
A.2 HAAP Experiment Results

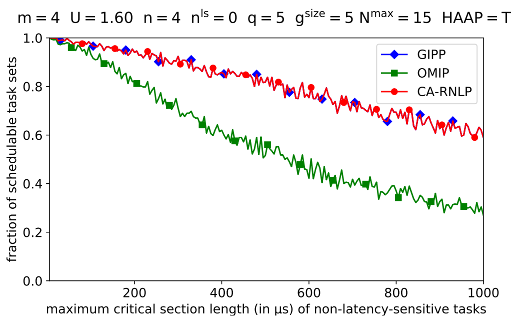


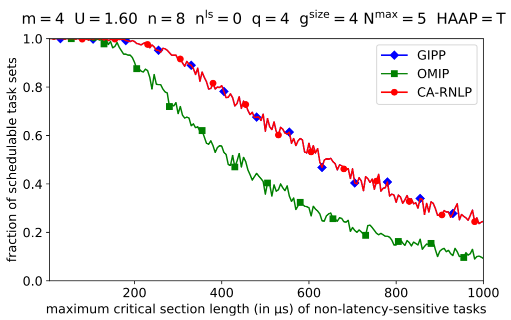


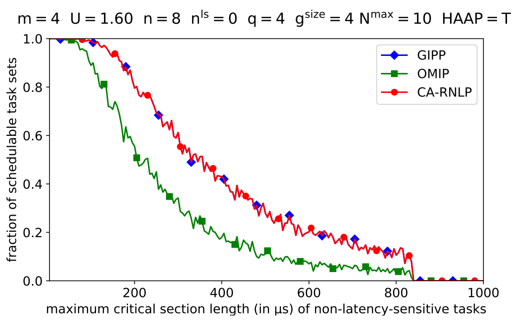


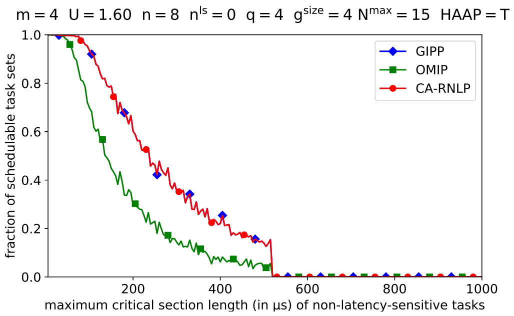


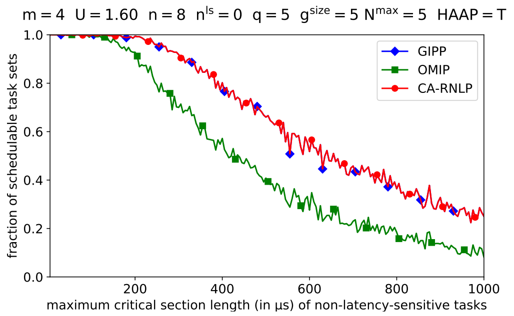


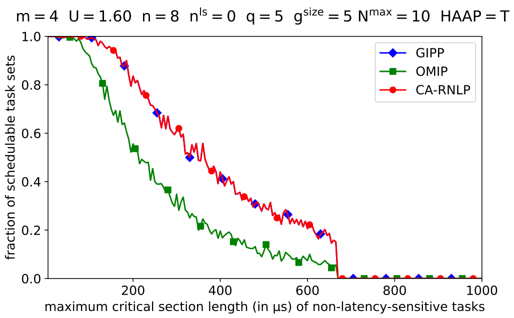


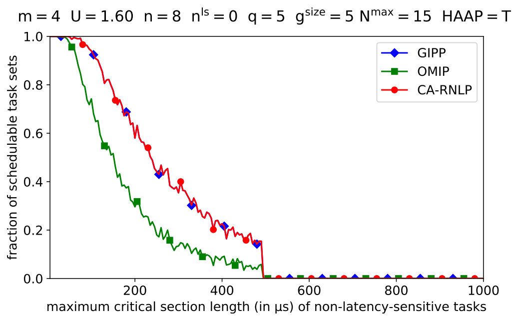


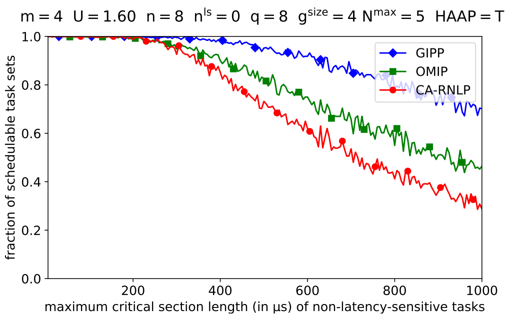


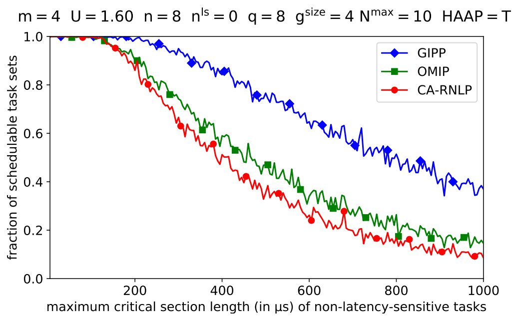

$m = 4$ $U = 1.60$ $n = 8$ $n^{ls} = 0$ $q = 4$ $g^{size} = 4$ $N^{max} = 15$ HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

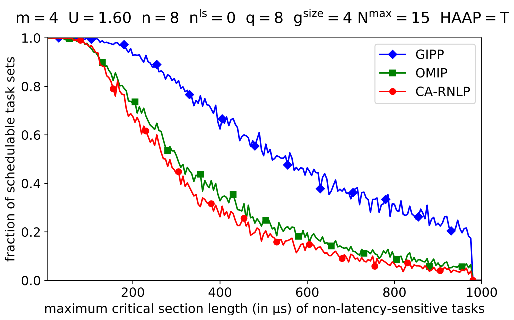

$m = 4$ $U = 1.60$ $n = 8$ $n^{ls} = 0$ $q = 5$ $g^{size} = 5$ $N^{max} = 5$ HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

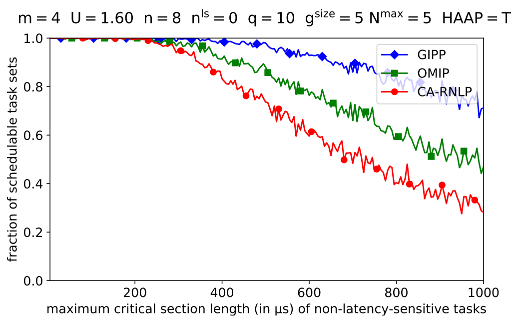

$m = 4$ $U = 1.60$ $n = 8$ $n^{ls} = 0$ $q = 5$ $g^{size} = 5$ $N^{max} = 10$ HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

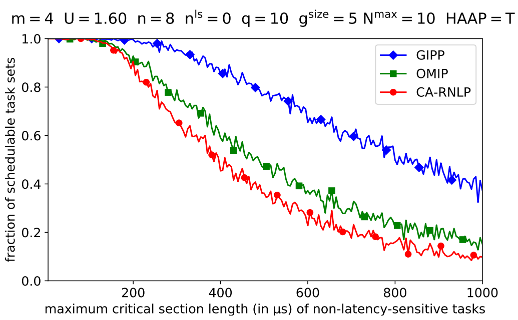

$m = 4$ $U = 1.60$ $n = 8$ $n^{ls} = 0$ $q = 5$ $g^{size} = 5$ $N^{max} = 15$ HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

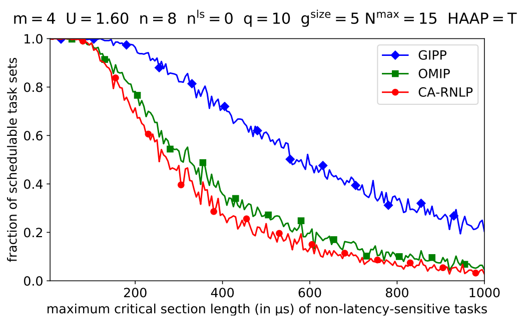

$m = 4$ $U = 1.60$ $n = 8$ $n^{ls} = 0$ $q = 8$ $g^{size} = 4$ $N^{max} = 5$ HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

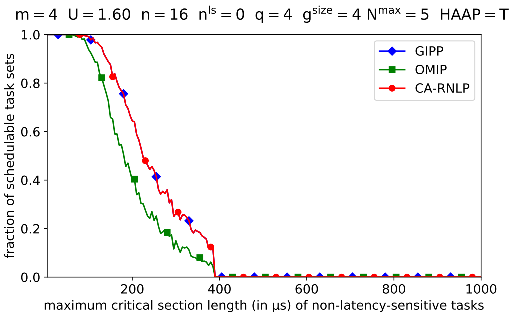

$m = 4$ $U = 1.60$ $n = 8$ $n^{ls} = 0$ $q = 8$ $g^{size} = 4$ $N^{max} = 10$ HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

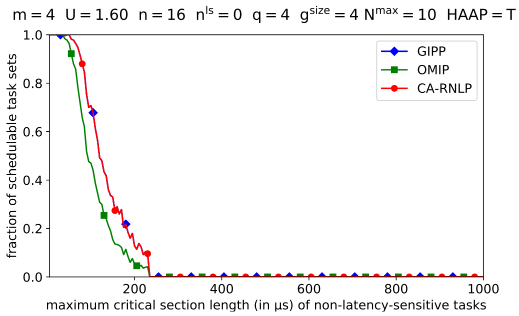

$m = 4$ $U = 1.60$ $n = 8$ $n^{ls} = 0$ $q = 8$ $g^{size} = 4$ $N^{max} = 15$ HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

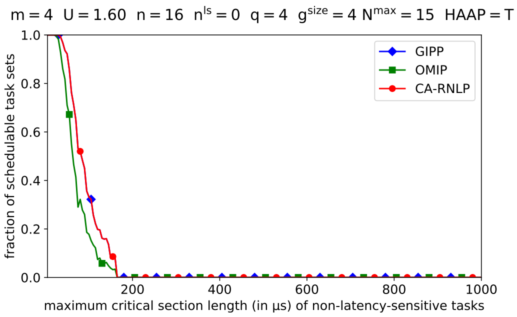

$m = 4$ $U = 1.60$ $n = 8$ $n^{ls} = 0$ $q = 10$ $g^{size} = 5$ $N^{max} = 5$ HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

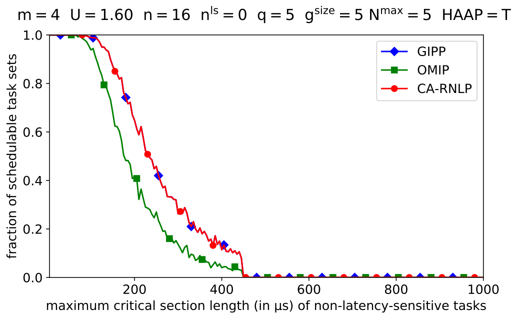

$m = 4$ $U = 1.60$ $n = 8$ $n^{ls} = 0$ $q = 10$ $g^{size} = 5$ $N^{max} = 10$ HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

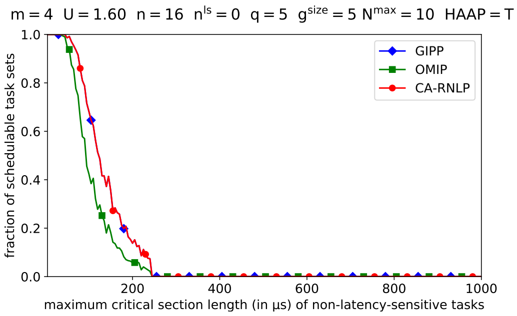

$m = 4$ $U = 1.60$ $n = 8$ $n^{ls} = 0$ $q = 10$ $g^{size} = 5$ $N^{max} = 15$ HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

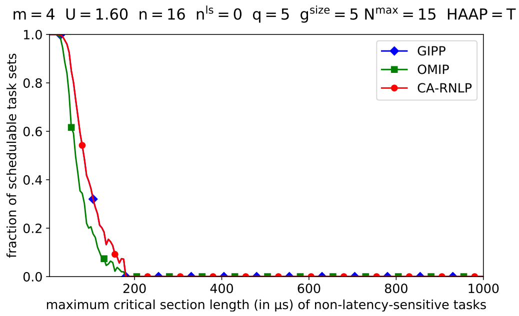

m = 4 U = 1.60 n = 16 n^ls = 0 q = 4 g^size = 4 N^max = 5 HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

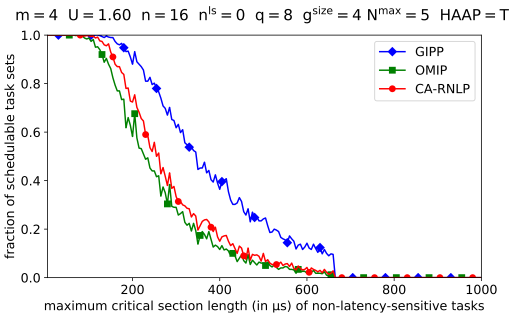

m = 4 U = 1.60 n = 16 n^ls = 0 q = 4 g^size = 4 N^max = 10 HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

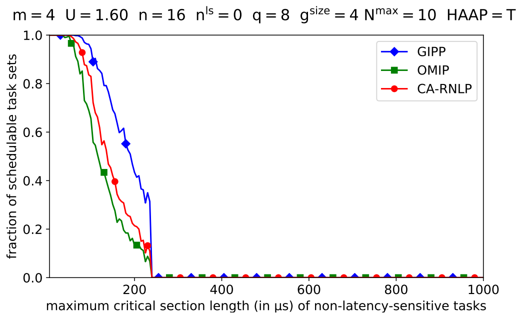

m = 4 U = 1.60 n = 16 n^ls = 0 q = 4 g^size = 4 N^max = 15 HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

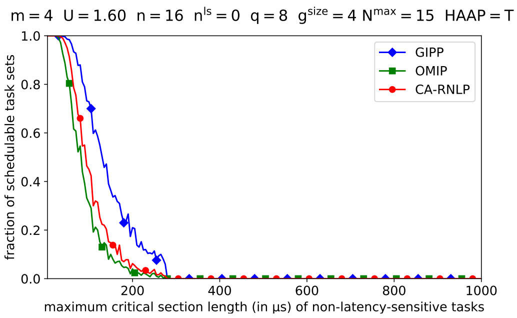

m = 4 U = 1.60 n = 16 n^ls = 0 q = 5 g^size = 5 N^max = 5 HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

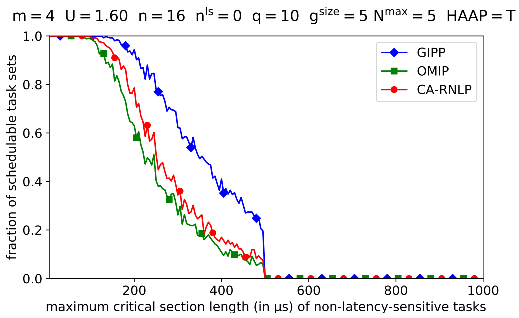

m = 4 U = 1.60 n = 16 n^ls = 0 q = 5 g^size = 5 N^max = 10 HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

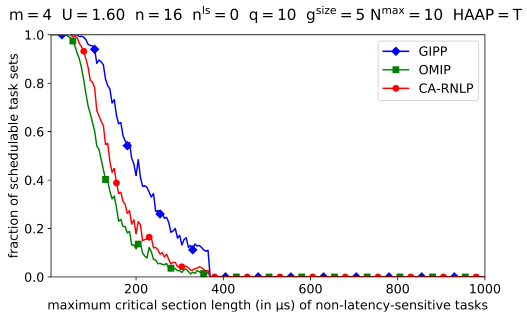

m = 4 U = 1.60 n = 16 n^ls = 0 q = 5 g^size = 5 N^max = 15 HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

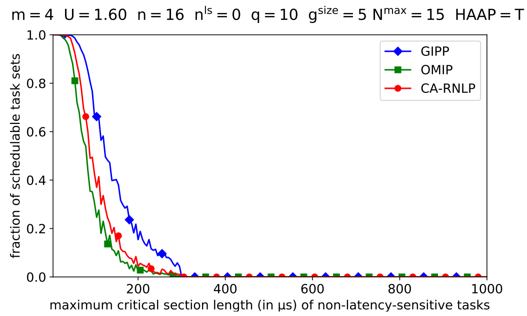

m = 4 U = 1.60 n = 16 n^ls = 0 q = 8 g^size = 4 N^max = 5 HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

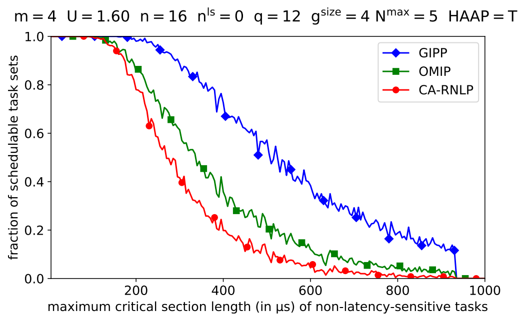

m = 4 U = 1.60 n = 16 n^ls = 0 q = 8 g^size = 4 N^max = 10 HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

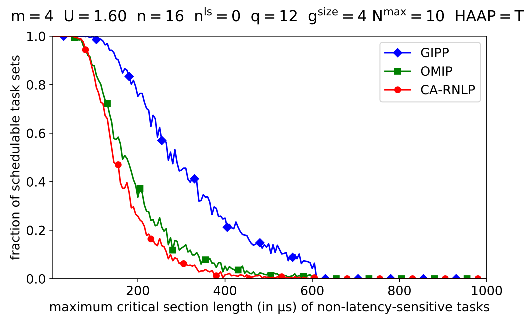

m = 4 U = 1.60 n = 16 n^ls = 0 q = 8 g^size = 4 N^max = 15 HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

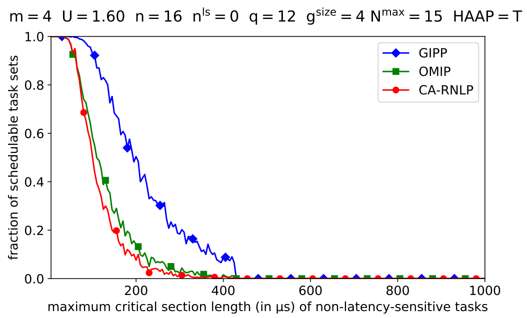

m = 4 U = 1.60 n = 16 n^ls = 0 q = 10 g^size = 5 N^max = 5 HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

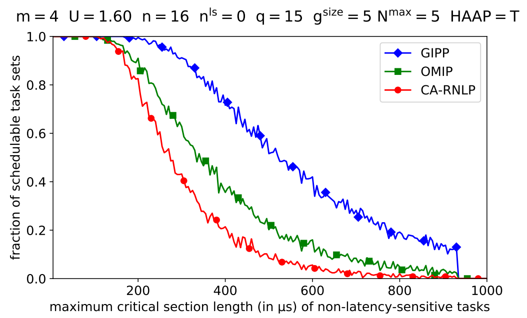

m = 4 U = 1.60 n = 16 $n^{ls} = 0$ q = 10 $g^{size} = 5$ $N^{max} = 10$ HAAP = T
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

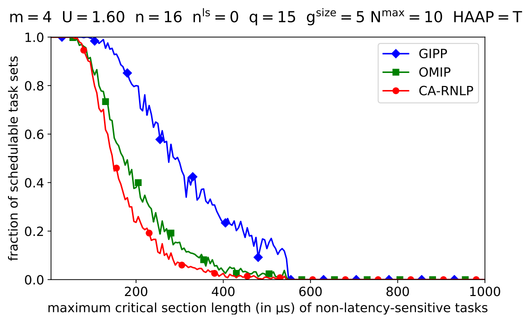

m = 4 U = 1.60 n = 16 $n^{ls} = 0$ q = 10 $g^{size} = 5$ $N^{max} = 15$ HAAP = T
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

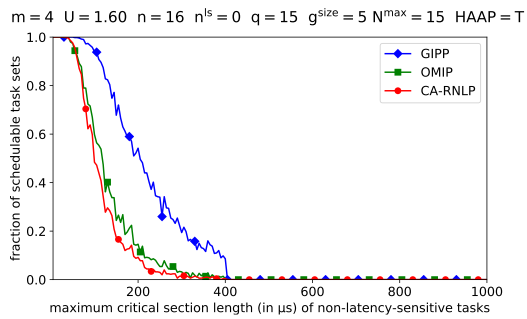

m = 4 U = 1.60 n = 16 $n^{ls} = 0$ q = 12 $g^{size} = 4$ $N^{max} = 5$ HAAP = T
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

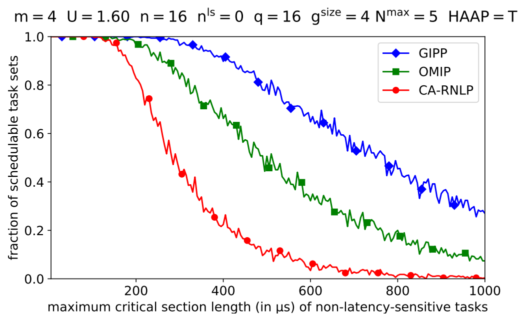

m = 4 U = 1.60 n = 16 $n^{ls} = 0$ q = 12 $g^{size} = 4$ $N^{max} = 10$ HAAP = T
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

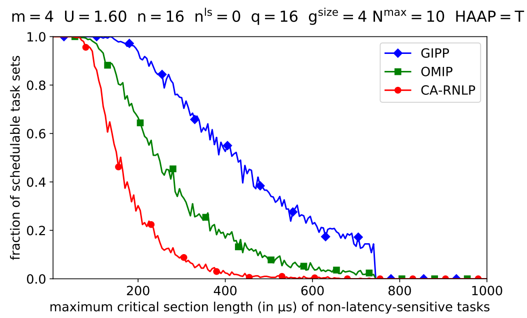

m = 4 U = 1.60 n = 16 $n^{ls} = 0$ q = 12 $g^{size} = 4$ $N^{max} = 15$ HAAP = T
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

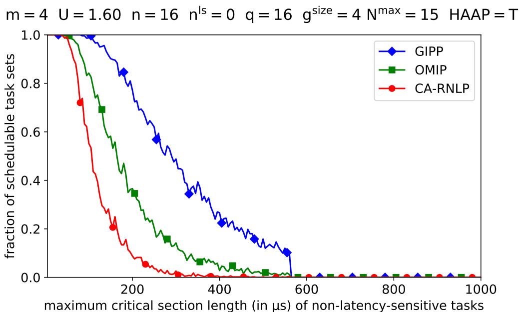

m = 4 U = 1.60 n = 16 $n^{ls} = 0$ q = 15 $g^{size} = 5$ $N^{max} = 5$ HAAP = T
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

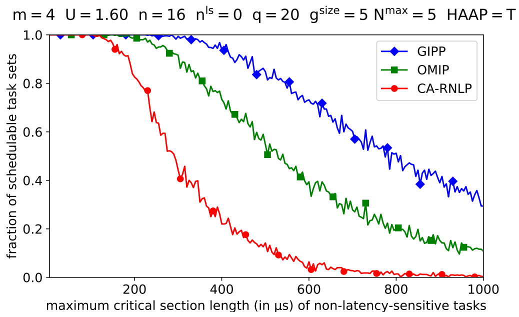

m = 4 U = 1.60 n = 16 $n^{ls} = 0$ q = 15 $g^{size} = 5$ $N^{max} = 10$ HAAP = T
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

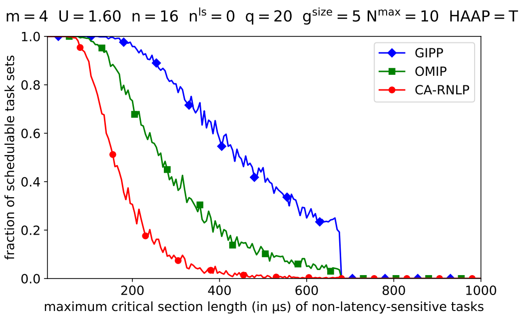

m = 4 U = 1.60 n = 16 $n^{ls} = 0$ q = 15 $g^{size} = 5$ $N^{max} = 15$ HAAP = T
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

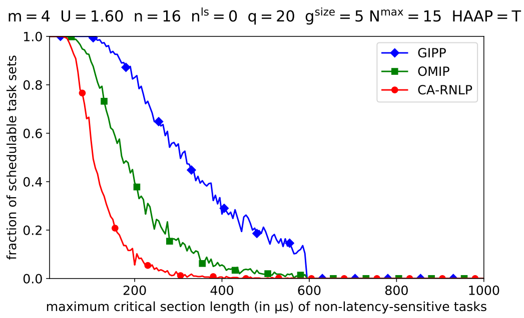

m = 4 U = 1.60 n = 16 $n^{ls} = 0$ q = 16 $g^{size} = 4$ $N^{max} = 5$ HAAP = T
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

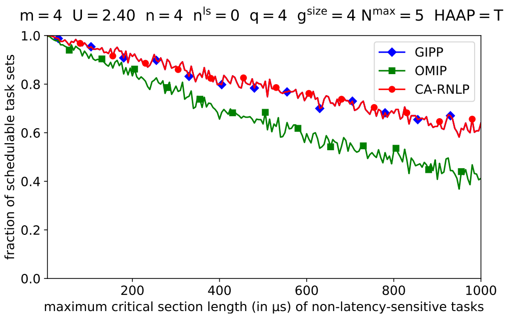

m = 4 U = 1.60 n = 16 $n^{ls} = 0$ q = 16 $g^{size} = 4$ $N^{max} = 10$ HAAP = T
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

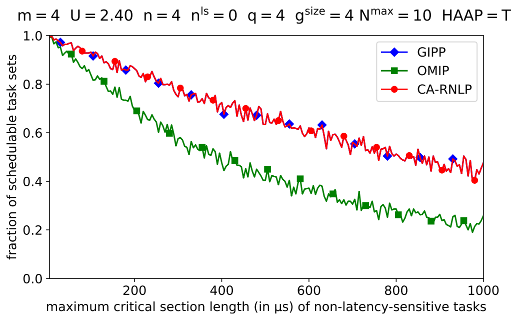

m = 4 U = 1.60 n = 16 n^{ls} = 0 q = 16 g^{size} = 4 N^{max} = 15 HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

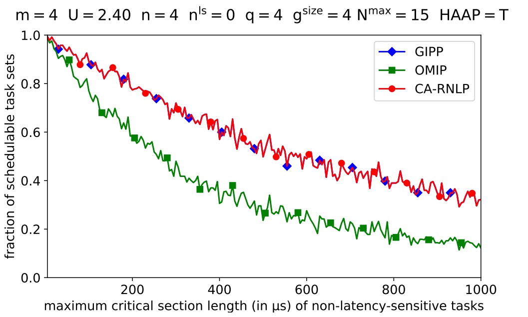

m = 4 U = 1.60 n = 16 n^{ls} = 0 q = 20 g^{size} = 5 N^{max} = 5 HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

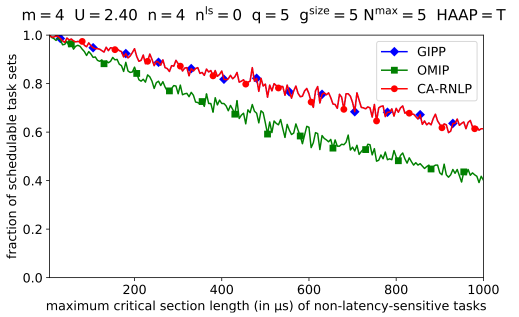

m = 4 U = 1.60 n = 16 n^{ls} = 0 q = 20 g^{size} = 5 N^{max} = 10 HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

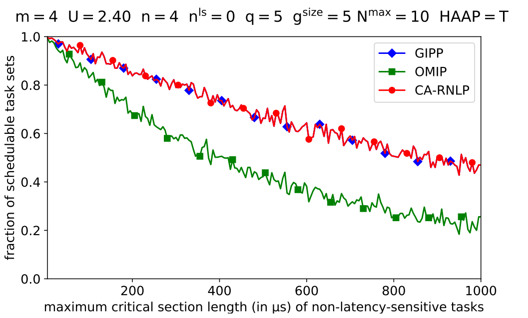

m = 4 U = 1.60 n = 16 n^{ls} = 0 q = 20 g^{size} = 5 N^{max} = 15 HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

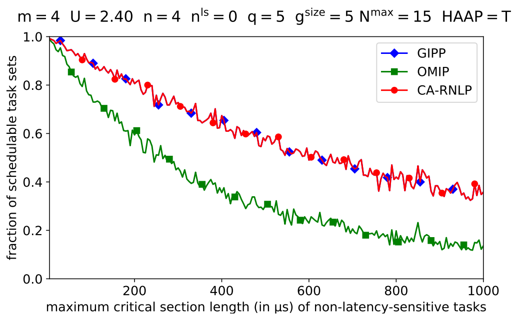

m = 4 U = 2.40 n = 4 n^{ls} = 0 q = 4 g^{size} = 4 N^{max} = 5 HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

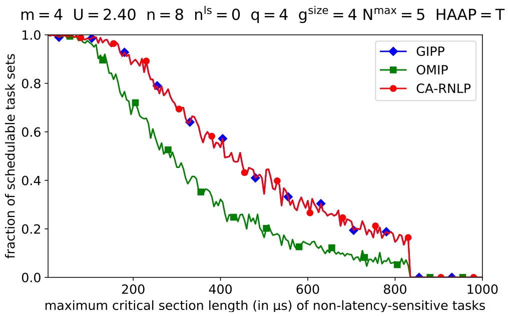

m = 4 U = 2.40 n = 4 n^{ls} = 0 q = 4 g^{size} = 4 N^{max} = 10 HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

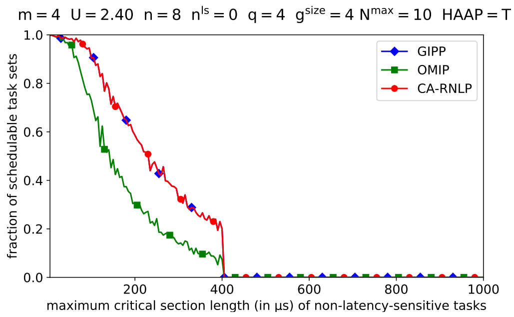

m = 4 U = 2.40 n = 4 n^{ls} = 0 q = 4 g^{size} = 4 N^{max} = 15 HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

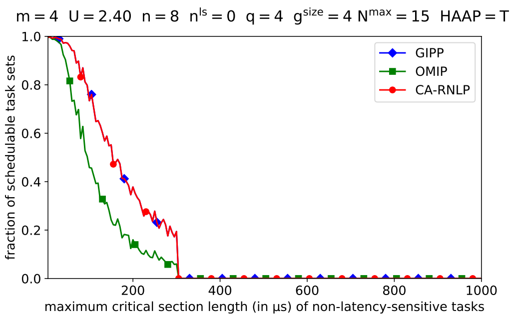

m = 4 U = 2.40 n = 4 n^{ls} = 0 q = 5 g^{size} = 5 N^{max} = 5 HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

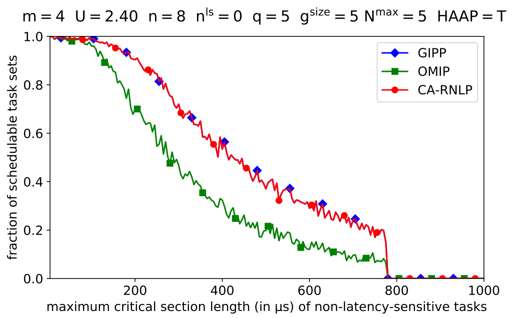

m = 4 U = 2.40 n = 4 n^{ls} = 0 q = 5 g^{size} = 5 N^{max} = 10 HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

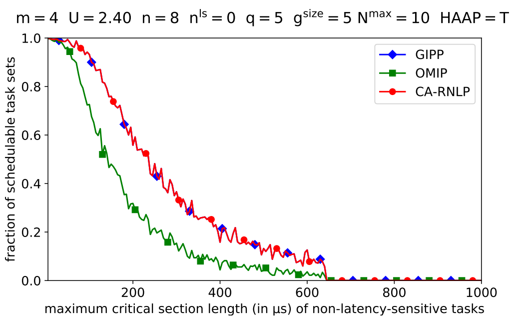

m = 4 U = 2.40 n = 4 n^{ls} = 0 q = 5 g^{size} = 5 N^{max} = 15 HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

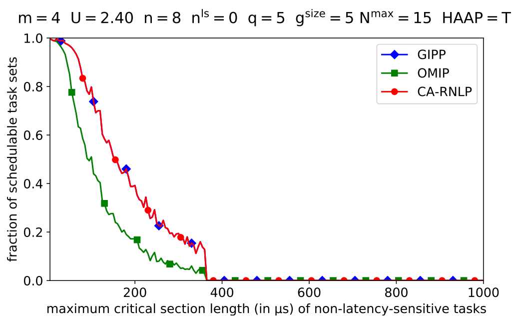

m = 4 U = 2.40 n = 8 $n^{ls} = 0$ q = 4 $g^{size} = 4$ $N^{max} = 5$ HAAP = T
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

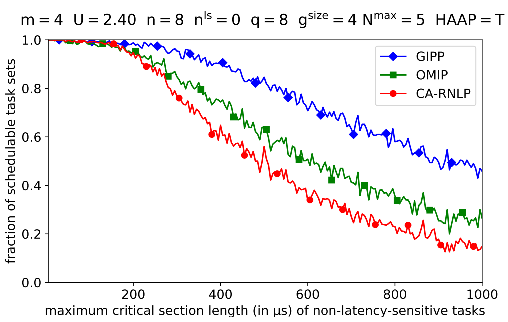

m = 4 U = 2.40 n = 8 $n^{ls} = 0$ q = 4 $g^{size} = 4$ $N^{max} = 10$ HAAP = T
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

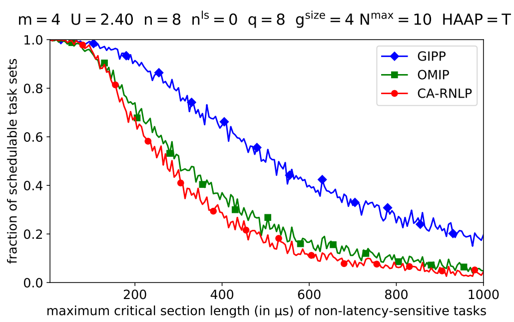

m = 4 U = 2.40 n = 8 $n^{ls} = 0$ q = 4 $g^{size} = 4$ $N^{max} = 15$ HAAP = T
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

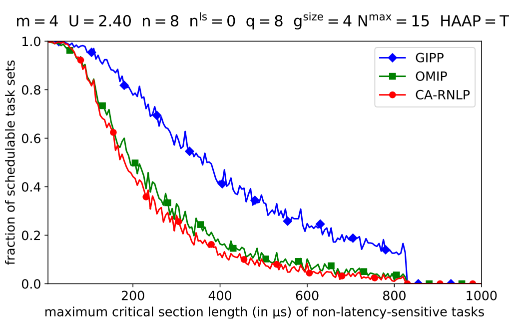

m = 4 U = 2.40 n = 8 $n^{ls} = 0$ q = 5 $g^{size} = 5$ $N^{max} = 5$ HAAP = T
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

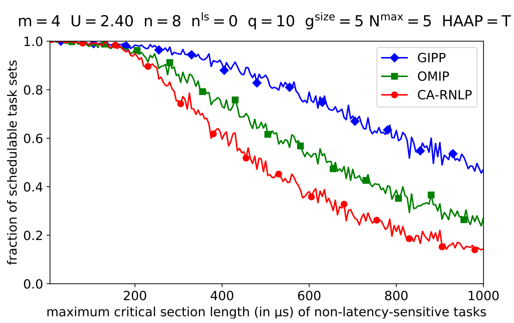

m = 4 U = 2.40 n = 8 $n^{ls} = 0$ q = 5 $g^{size} = 5$ $N^{max} = 10$ HAAP = T
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

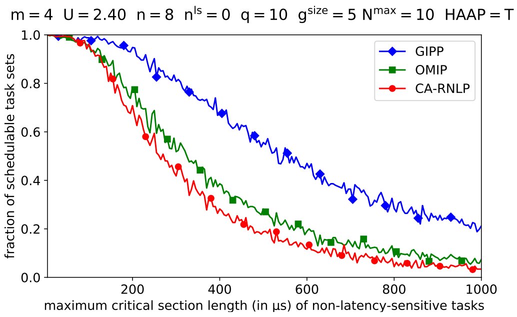

m = 4 U = 2.40 n = 8 $n^{ls} = 0$ q = 5 $g^{size} = 5$ $N^{max} = 15$ HAAP = T
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

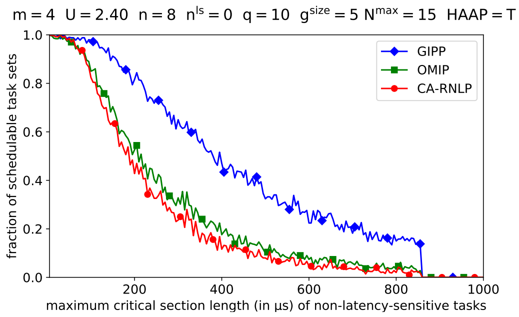

m = 4 U = 2.40 n = 8 $n^{ls} = 0$ q = 8 $g^{size} = 4$ $N^{max} = 5$ HAAP = T
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

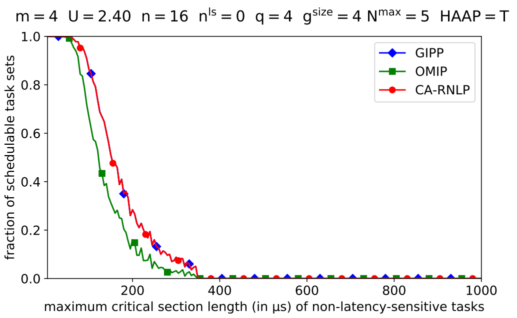

m = 4 U = 2.40 n = 8 $n^{ls} = 0$ q = 8 $g^{size} = 4$ $N^{max} = 10$ HAAP = T
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

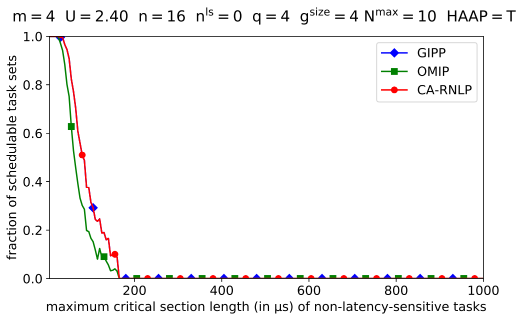

m = 4 U = 2.40 n = 8 $n^{ls} = 0$ q = 8 $g^{size} = 4$ $N^{max} = 15$ HAAP = T
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

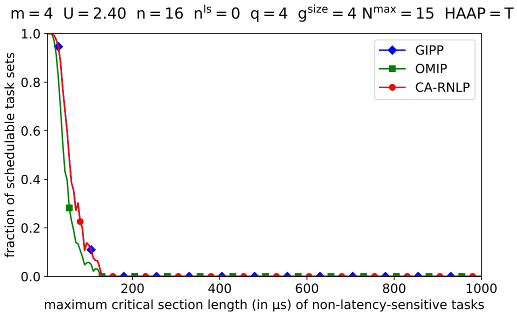

m = 4 U = 2.40 n = 8 $n^{ls} = 0$ q = 10 $g^{size} = 5$ $N^{max} = 5$ HAAP = T
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

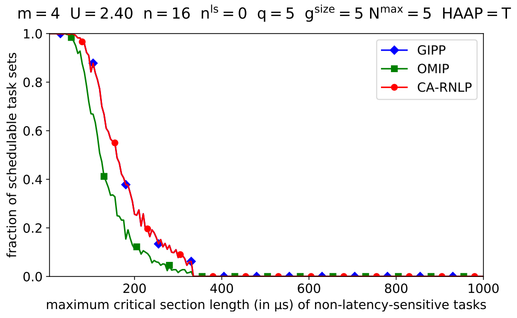

m = 4 U = 2.40 n = 8 $n^{ls} = 0$ q = 10 $g^{size} = 5$ $N^{max} = 10$ HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

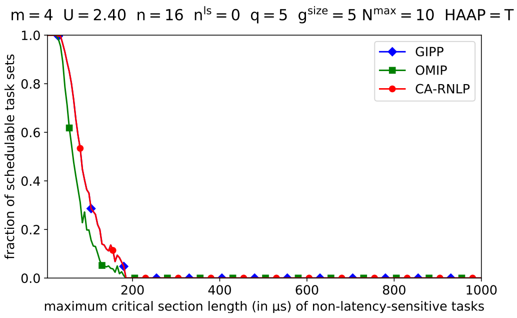

m = 4 U = 2.40 n = 8 $n^{ls} = 0$ q = 10 $g^{size} = 5$ $N^{max} = 15$ HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

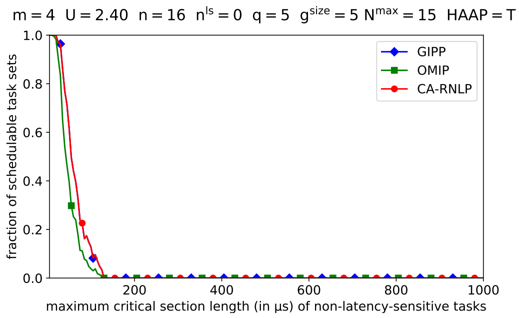

m = 4 U = 2.40 n = 16 $n^{ls} = 0$ q = 4 $g^{size} = 4$ $N^{max} = 5$ HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

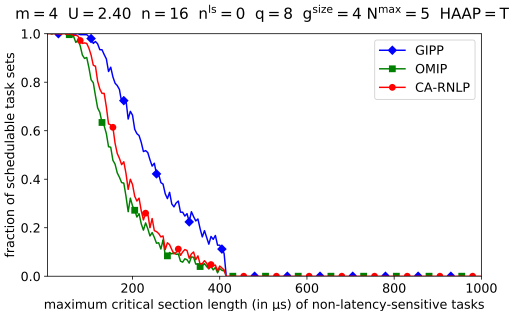

m = 4 U = 2.40 n = 16 $n^{ls} = 0$ q = 4 $g^{size} = 4$ $N^{max} = 10$ HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

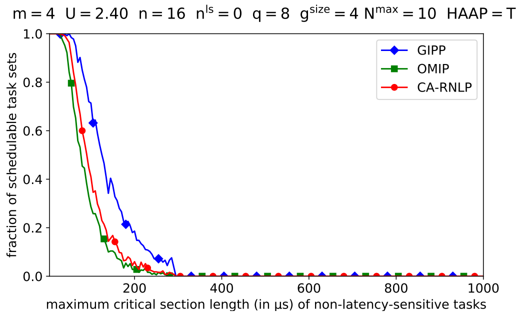

m = 4 U = 2.40 n = 16 $n^{ls} = 0$ q = 4 $g^{size} = 4$ $N^{max} = 15$ HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

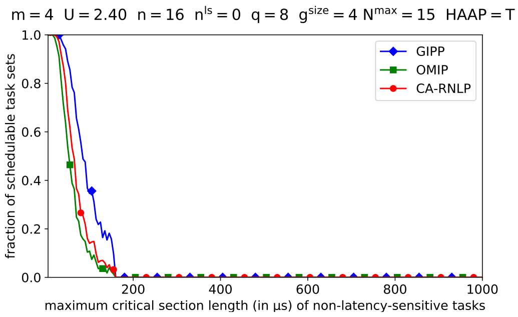

m = 4 U = 2.40 n = 16 $n^{ls} = 0$ q = 5 $g^{size} = 5$ $N^{max} = 5$ HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

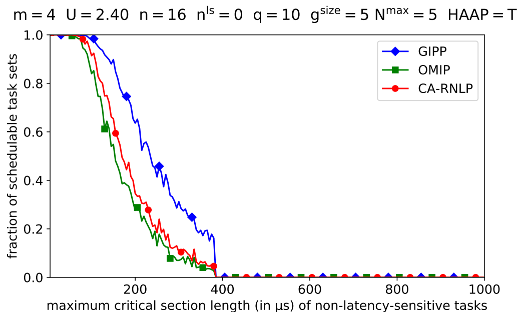

m = 4 U = 2.40 n = 16 $n^{ls} = 0$ q = 5 $g^{size} = 5$ $N^{max} = 10$ HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

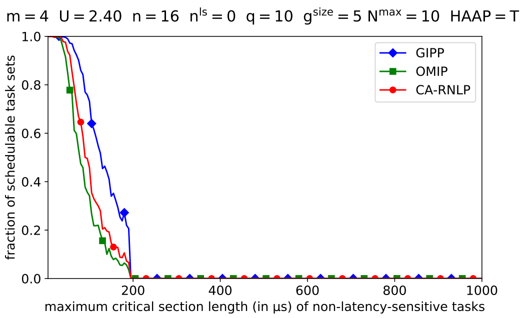

m = 4 U = 2.40 n = 16 $n^{ls} = 0$ q = 5 $g^{size} = 5$ $N^{max} = 15$ HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

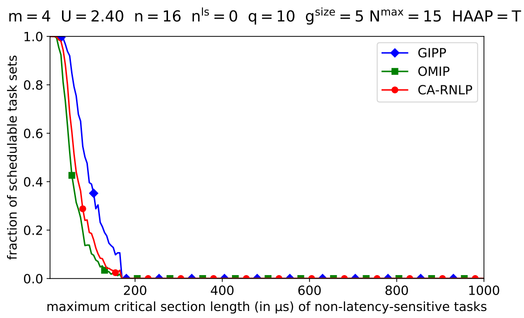

m = 4 U = 2.40 n = 16 $n^{ls} = 0$ q = 8 $g^{size} = 4$ $N^{max} = 5$ HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

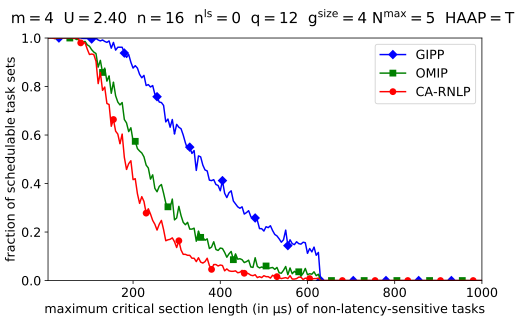

m = 4 U = 2.40 n = 16 $n^{ls} = 0$ q = 8 $g^{size} = 4$ $N^{max} = 10$ HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

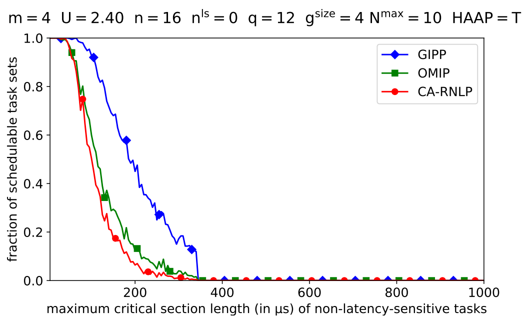

$m = 4$ $U = 2.40$ $n = 16$ $n^{ls} = 0$ $q = 8$ $g^{size} = 4$ $N^{max} = 15$ HAAP = T
fraction of schedulable task sets
GIPP
OMIP
CA-RNLP
maximum critical section length (in µs) of non-latency-sensitive tasks

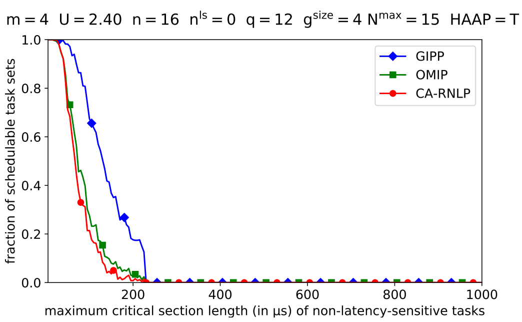

$m = 4$ $U = 2.40$ $n = 16$ $n^{ls} = 0$ $q = 10$ $g^{size} = 5$ $N^{max} = 5$ HAAP = T
fraction of schedulable task sets
GIPP
OMIP
CA-RNLP
maximum critical section length (in µs) of non-latency-sensitive tasks

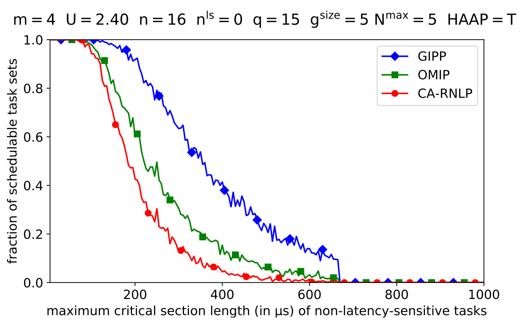

$m = 4$ $U = 2.40$ $n = 16$ $n^{ls} = 0$ $q = 10$ $g^{size} = 5$ $N^{max} = 10$ HAAP = T
fraction of schedulable task sets
GIPP
OMIP
CA-RNLP
maximum critical section length (in µs) of non-latency-sensitive tasks

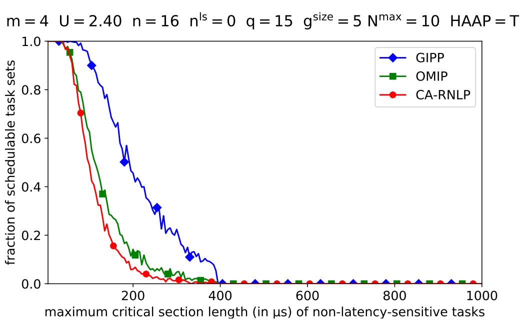

$m = 4$ $U = 2.40$ $n = 16$ $n^{ls} = 0$ $q = 10$ $g^{size} = 5$ $N^{max} = 15$ HAAP = T
fraction of schedulable task sets
GIPP
OMIP
CA-RNLP
maximum critical section length (in µs) of non-latency-sensitive tasks

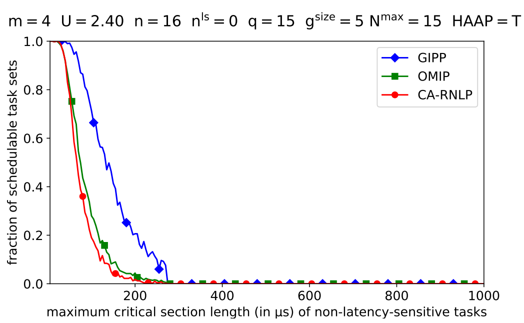

$m = 4$ $U = 2.40$ $n = 16$ $n^{ls} = 0$ $q = 12$ $g^{size} = 4$ $N^{max} = 5$ HAAP = T
fraction of schedulable task sets
GIPP
OMIP
CA-RNLP
maximum critical section length (in µs) of non-latency-sensitive tasks

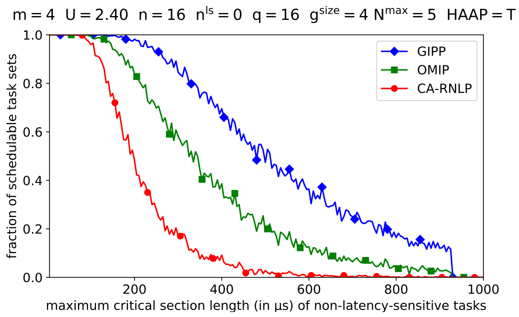

$m = 4$ $U = 2.40$ $n = 16$ $n^{ls} = 0$ $q = 12$ $g^{size} = 4$ $N^{max} = 10$ HAAP = T
fraction of schedulable task sets
GIPP
OMIP
CA-RNLP
maximum critical section length (in µs) of non-latency-sensitive tasks

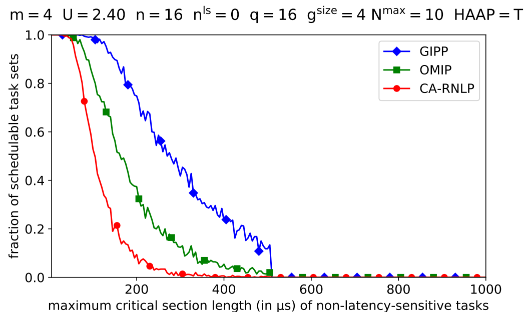

$m = 4$ $U = 2.40$ $n = 16$ $n^{ls} = 0$ $q = 12$ $g^{size} = 4$ $N^{max} = 15$ HAAP = T
fraction of schedulable task sets
GIPP
OMIP
CA-RNLP
maximum critical section length (in µs) of non-latency-sensitive tasks

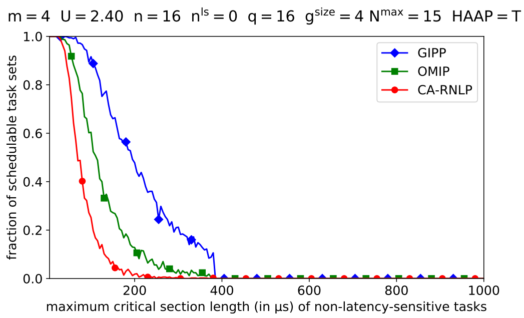

$m = 4$ $U = 2.40$ $n = 16$ $n^{ls} = 0$ $q = 15$ $g^{size} = 5$ $N^{max} = 5$ HAAP = T
fraction of schedulable task sets
GIPP
OMIP
CA-RNLP
maximum critical section length (in µs) of non-latency-sensitive tasks

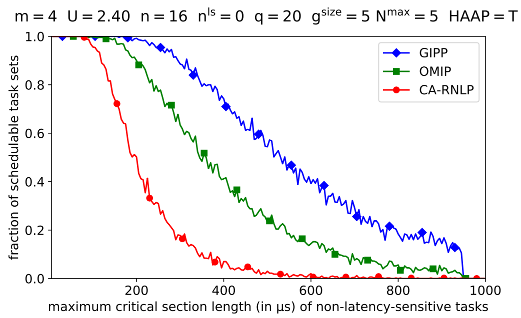

$m = 4$ $U = 2.40$ $n = 16$ $n^{ls} = 0$ $q = 15$ $g^{size} = 5$ $N^{max} = 10$ HAAP = T
fraction of schedulable task sets
GIPP
OMIP
CA-RNLP
maximum critical section length (in µs) of non-latency-sensitive tasks

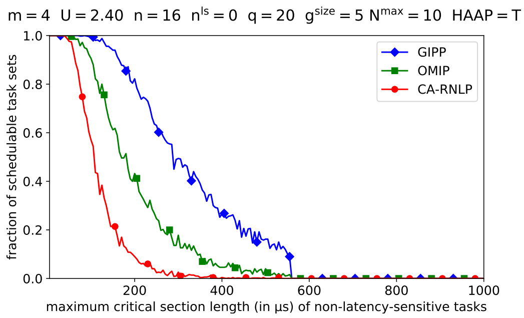

$m = 4$ $U = 2.40$ $n = 16$ $n^{ls} = 0$ $q = 15$ $g^{size} = 5$ $N^{max} = 15$ HAAP = T
fraction of schedulable task sets
GIPP
OMIP
CA-RNLP
maximum critical section length (in µs) of non-latency-sensitive tasks

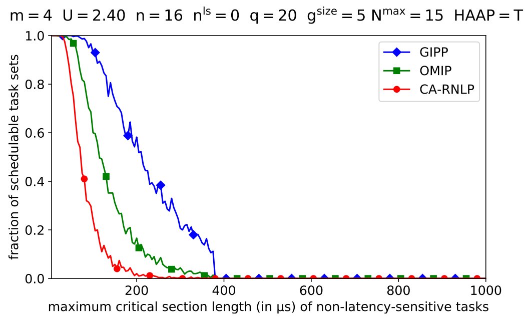

m = 4 U = 2.40 n = 16 n^{ls} = 0 q = 16 g^{size} = 4 N^{max} = 5 HAAP = T
fraction of schedulable task sets
0.0
0.2
0.4
0.6
0.8
1.0
200
400
600
800
1000
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

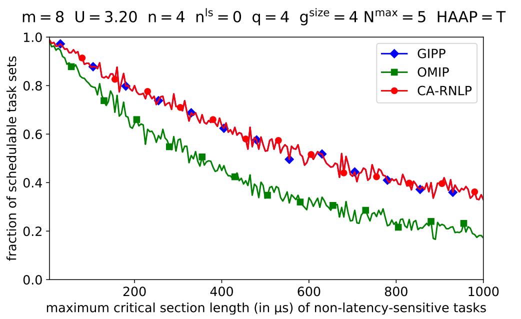

m = 4 U = 2.40 n = 16 n^{ls} = 0 q = 16 g^{size} = 4 N^{max} = 10 HAAP = T
fraction of schedulable task sets
0.0
0.2
0.4
0.6
0.8
1.0
200
400
600
800
1000
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

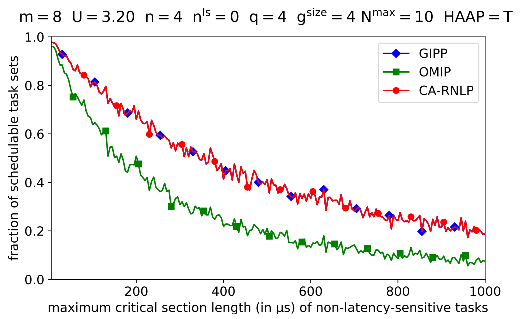

m = 4 U = 2.40 n = 16 n^{ls} = 0 q = 16 g^{size} = 4 N^{max} = 15 HAAP = T
fraction of schedulable task sets
0.0
0.2
0.4
0.6
0.8
1.0
200
400
600
800
1000
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

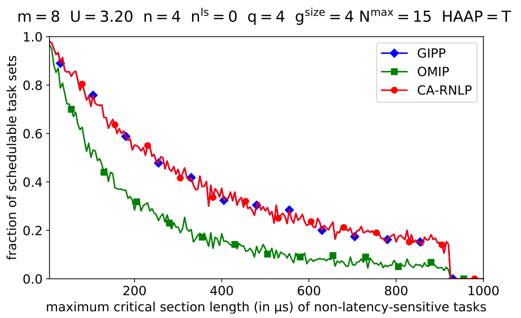

m = 4 U = 2.40 n = 16 n^{ls} = 0 q = 20 g^{size} = 5 N^{max} = 5 HAAP = T
fraction of schedulable task sets
0.0
0.2
0.4
0.6
0.8
1.0
200
400
600
800
1000
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

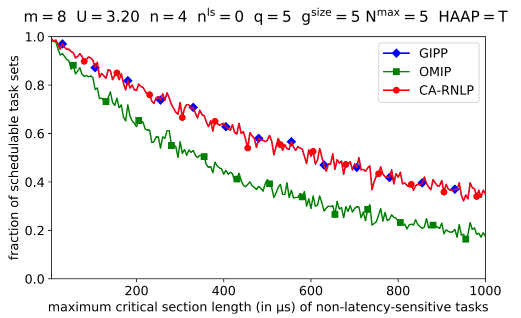

m = 4 U = 2.40 n = 16 n^{ls} = 0 q = 20 g^{size} = 5 N^{max} = 10 HAAP = T
fraction of schedulable task sets
0.0
0.2
0.4
0.6
0.8
1.0
200
400
600
800
1000
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

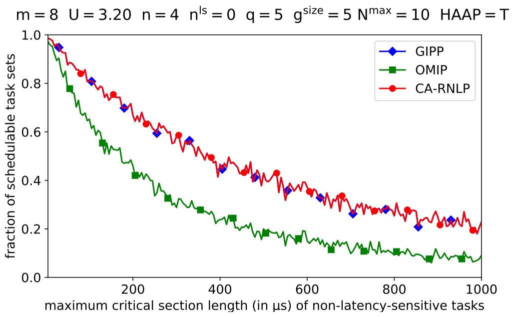

m = 4 U = 2.40 n = 16 n^{ls} = 0 q = 20 g^{size} = 5 N^{max} = 15 HAAP = T
fraction of schedulable task sets
0.0
0.2
0.4
0.6
0.8
1.0
200
400
600
800
1000
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

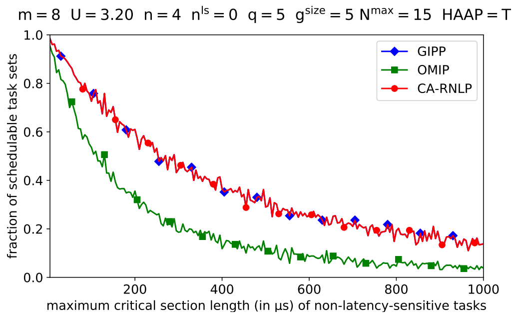

m = 8 U = 3.20 n = 4 n^{ls} = 0 q = 4 g^{size} = 4 N^{max} = 5 HAAP = T
fraction of schedulable task sets
0.0
0.2
0.4
0.6
0.8
1.0
200
400
600
800
1000
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

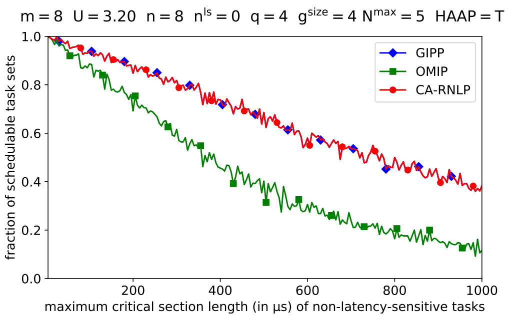

m = 8 U = 3.20 n = 4 n^{ls} = 0 q = 4 g^{size} = 4 N^{max} = 10 HAAP = T
fraction of schedulable task sets
0.0
0.2
0.4
0.6
0.8
1.0
200
400
600
800
1000
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

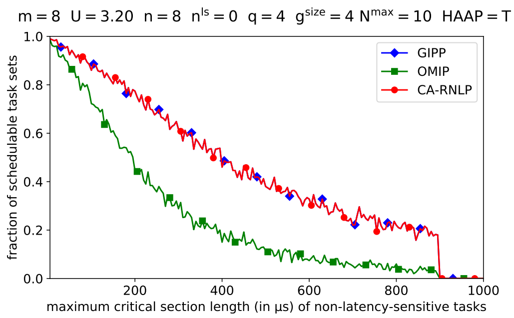

m = 8 U = 3.20 n = 4 n^{ls} = 0 q = 4 g^{size} = 4 N^{max} = 15 HAAP = T
fraction of schedulable task sets
0.0
0.2
0.4
0.6
0.8
1.0
200
400
600
800
1000
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

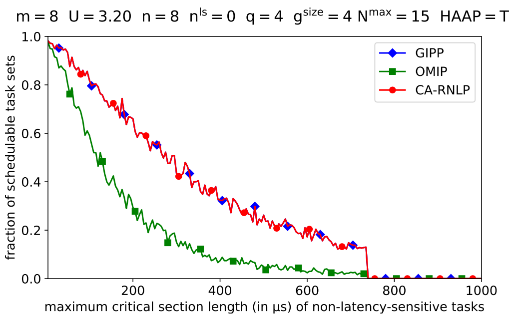

m = 8 U = 3.20 n = 4 n^{ls} = 0 q = 5 g^{size} = 5 N^{max} = 5 HAAP = T
fraction of schedulable task sets
0.0
0.2
0.4
0.6
0.8
1.0
200
400
600
800
1000
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

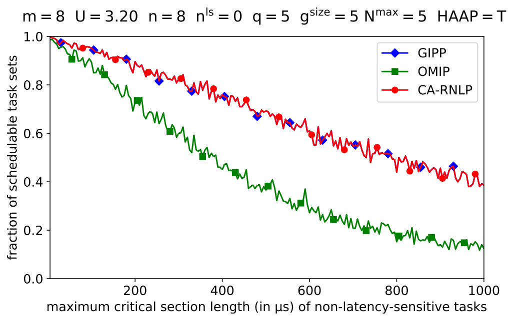

m = 8 U = 3.20 n = 4 n^{ls} = 0 q = 5 g^{size} = 5 N^{max} = 10 HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks

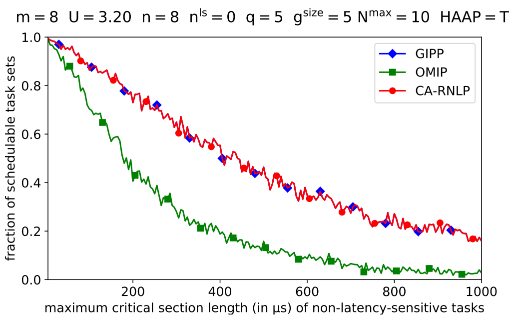

m = 8 U = 3.20 n = 4 n^{ls} = 0 q = 5 g^{size} = 5 N^{max} = 15 HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks

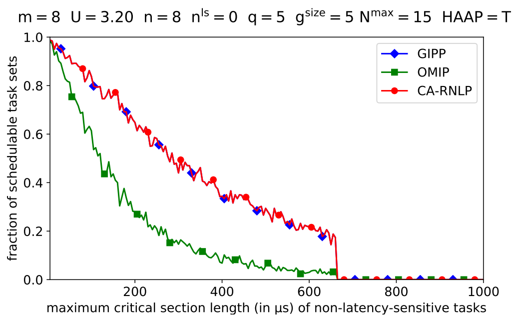

m = 8 U = 3.20 n = 8 n^{ls} = 0 q = 4 g^{size} = 4 N^{max} = 5 HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks

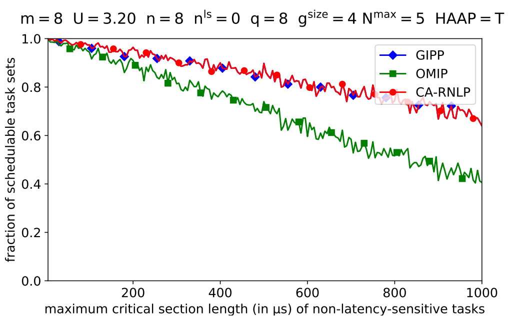

m = 8 U = 3.20 n = 8 n^{ls} = 0 q = 4 g^{size} = 4 N^{max} = 10 HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks

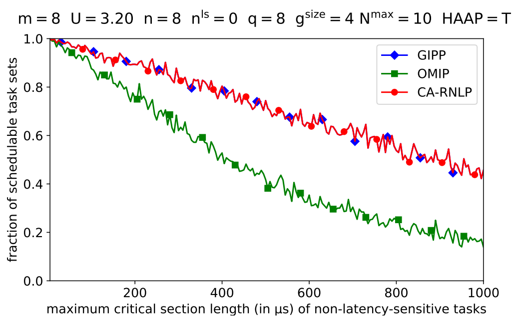

m = 8 U = 3.20 n = 8 n^{ls} = 0 q = 4 g^{size} = 4 N^{max} = 15 HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks

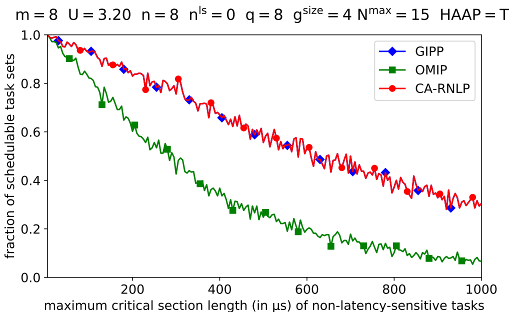

m = 8 U = 3.20 n = 8 n^{ls} = 0 q = 5 g^{size} = 5 N^{max} = 5 HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks

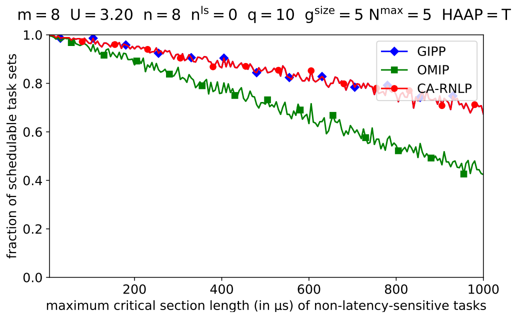

m = 8 U = 3.20 n = 8 n^{ls} = 0 q = 5 g^{size} = 5 N^{max} = 10 HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks

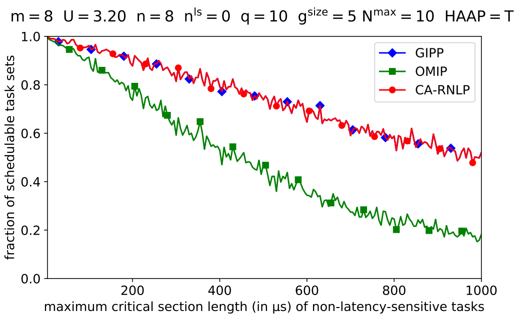

m = 8 U = 3.20 n = 8 n^{ls} = 0 q = 5 g^{size} = 5 N^{max} = 15 HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks

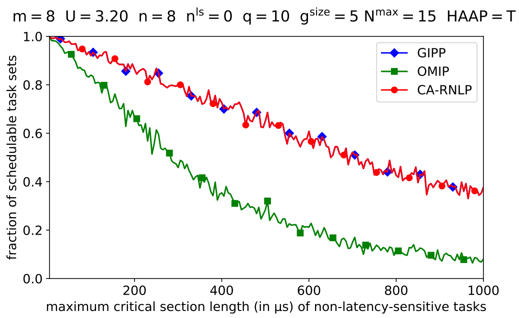

m = 8 U = 3.20 n = 8 n^{ls} = 0 q = 8 g^{size} = 4 N^{max} = 5 HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks

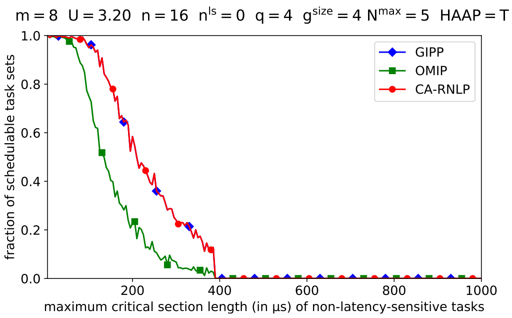

m = 8 U = 3.20 n = 8 n^{ls} = 0 q = 8 g^{size} = 4 N^{max} = 10 HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks

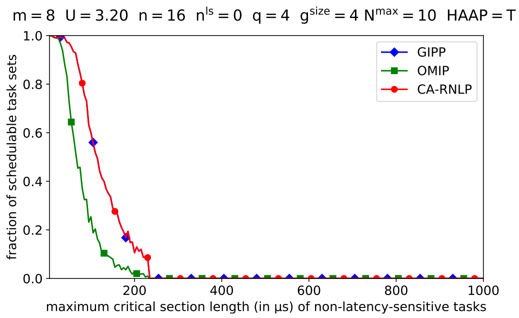

m = 8 U = 3.20 n = 8 n^{ls} = 0 q = 8 g^{size} = 4 N^{max} = 15 HAAP = T
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

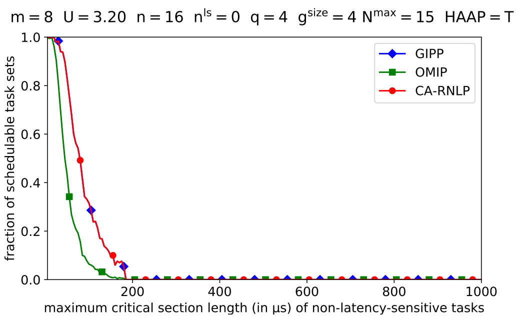

m = 8 U = 3.20 n = 8 n^{ls} = 0 q = 10 g^{size} = 5 N^{max} = 5 HAAP = T
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

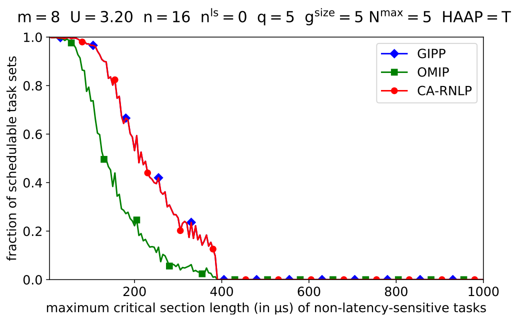

m = 8 U = 3.20 n = 8 n^{ls} = 0 q = 10 g^{size} = 5 N^{max} = 10 HAAP = T
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

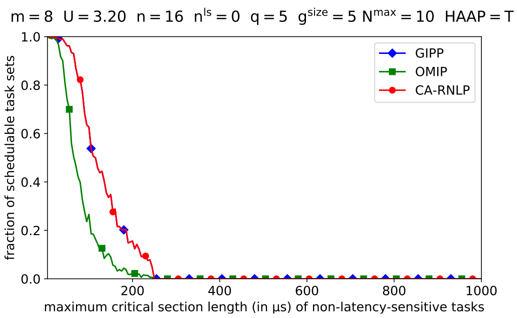

m = 8 U = 3.20 n = 8 n^{ls} = 0 q = 10 g^{size} = 5 N^{max} = 15 HAAP = T
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

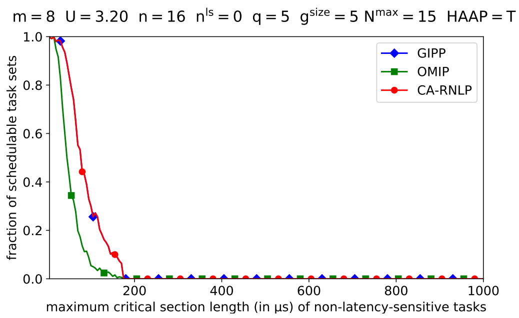

m = 8 U = 3.20 n = 16 n^{ls} = 0 q = 4 g^{size} = 4 N^{max} = 5 HAAP = T
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

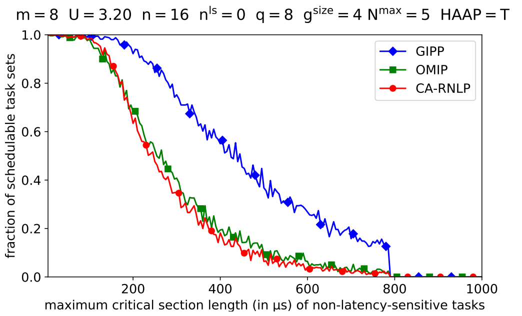

m = 8 U = 3.20 n = 16 n^{ls} = 0 q = 4 g^{size} = 4 N^{max} = 10 HAAP = T
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

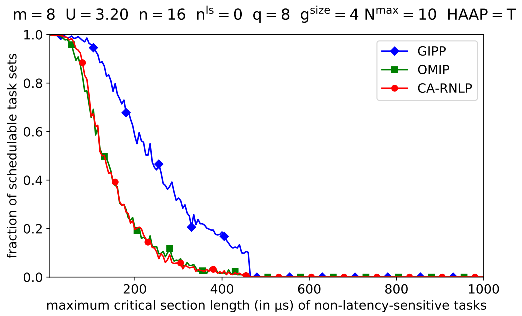

m = 8 U = 3.20 n = 16 n^{ls} = 0 q = 4 g^{size} = 4 N^{max} = 15 HAAP = T
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

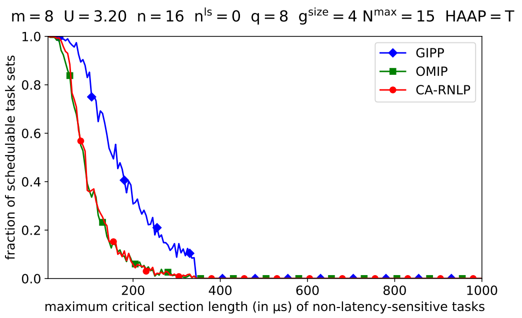

m = 8 U = 3.20 n = 16 n^{ls} = 0 q = 5 g^{size} = 5 N^{max} = 5 HAAP = T
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

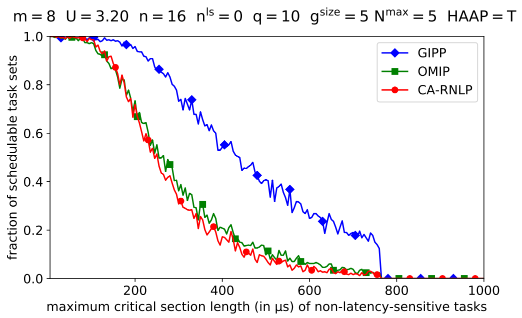

m = 8 U = 3.20 n = 16 n^{ls} = 0 q = 5 g^{size} = 5 N^{max} = 10 HAAP = T
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

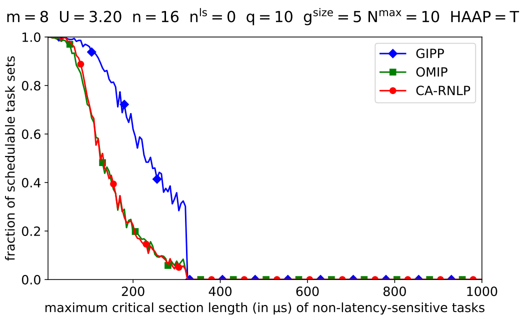

m = 8 U = 3.20 n = 16 n^{ls} = 0 q = 5 g^{size} = 5 N^{max} = 15 HAAP = T
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

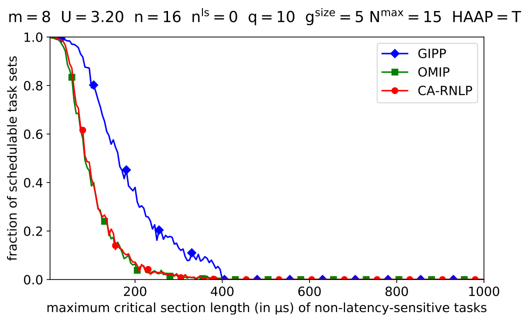

$m = 8$ $U = 3.20$ $n = 16$ $n^{ls} = 0$ $q = 8$ $g^{size} = 4$ $N^{max} = 5$ HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks

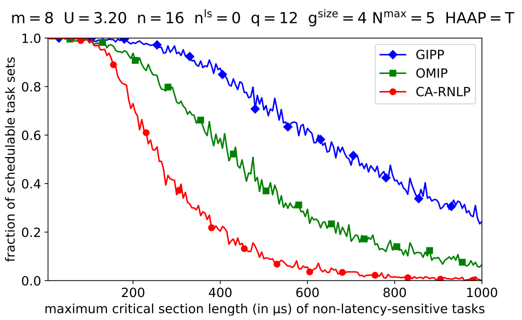

$m = 8$ $U = 3.20$ $n = 16$ $n^{ls} = 0$ $q = 8$ $g^{size} = 4$ $N^{max} = 10$ HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks

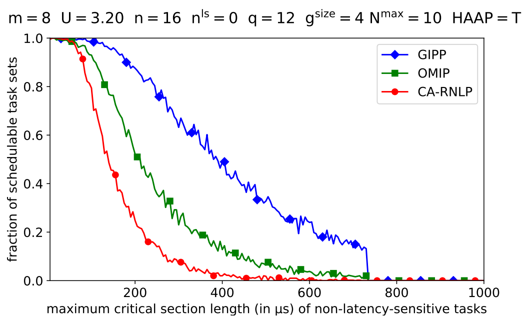

$m = 8$ $U = 3.20$ $n = 16$ $n^{ls} = 0$ $q = 8$ $g^{size} = 4$ $N^{max} = 15$ HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks

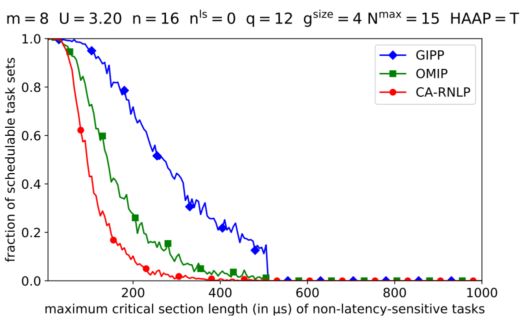

$m = 8$ $U = 3.20$ $n = 16$ $n^{ls} = 0$ $q = 10$ $g^{size} = 5$ $N^{max} = 5$ HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks

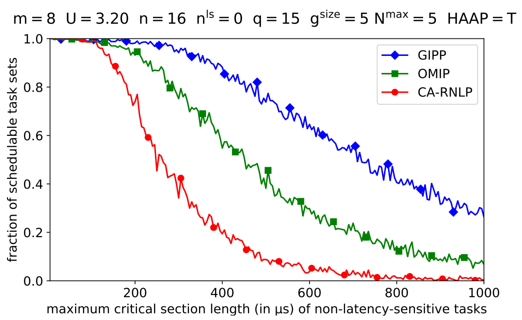

$m = 8$ $U = 3.20$ $n = 16$ $n^{ls} = 0$ $q = 10$ $g^{size} = 5$ $N^{max} = 10$ HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks

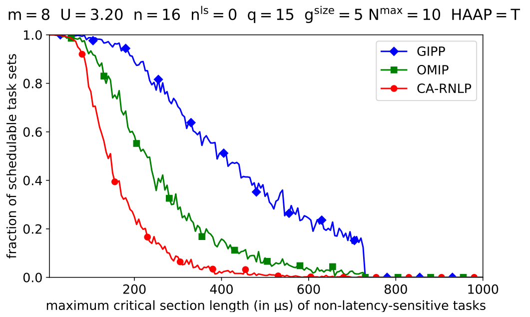

$m = 8$ $U = 3.20$ $n = 16$ $n^{ls} = 0$ $q = 10$ $g^{size} = 5$ $N^{max} = 15$ HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks

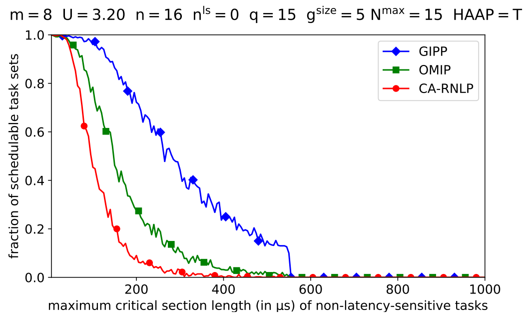

$m = 8$ $U = 3.20$ $n = 16$ $n^{ls} = 0$ $q = 12$ $g^{size} = 4$ $N^{max} = 5$ HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks

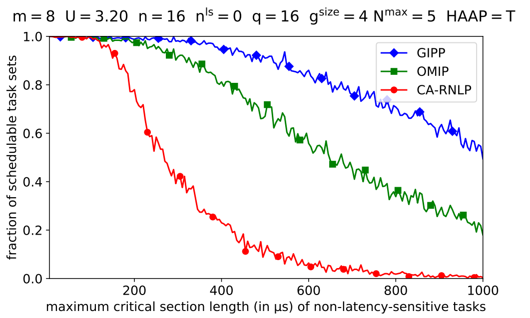

$m = 8$ $U = 3.20$ $n = 16$ $n^{ls} = 0$ $q = 12$ $g^{size} = 4$ $N^{max} = 10$ HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks

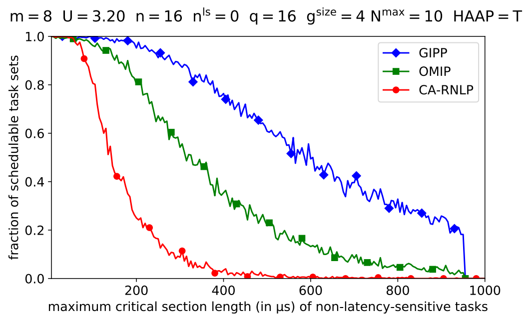

$m = 8$ $U = 3.20$ $n = 16$ $n^{ls} = 0$ $q = 12$ $g^{size} = 4$ $N^{max} = 15$ HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks

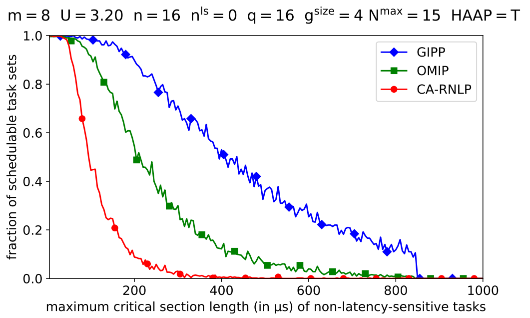

$m = 8$ $U = 3.20$ $n = 16$ $n^{ls} = 0$ $q = 15$ $g^{size} = 5$ $N^{max} = 5$ HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks

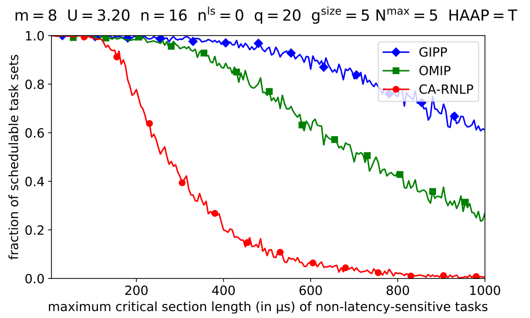

$m = 8$ $U = 3.20$ $n = 16$ $n^{ls} = 0$ $q = 15$ $g^{size} = 5$ $N^{max} = 10$ HAAP = T
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

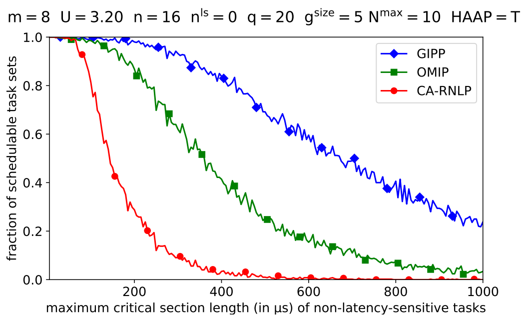

$m = 8$ $U = 3.20$ $n = 16$ $n^{ls} = 0$ $q = 15$ $g^{size} = 5$ $N^{max} = 15$ HAAP = T
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

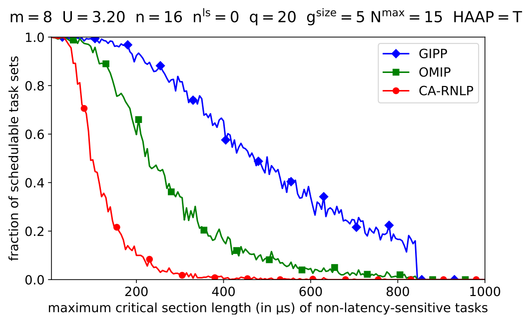

$m = 8$ $U = 3.20$ $n = 16$ $n^{ls} = 0$ $q = 16$ $g^{size} = 4$ $N^{max} = 5$ HAAP = T
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

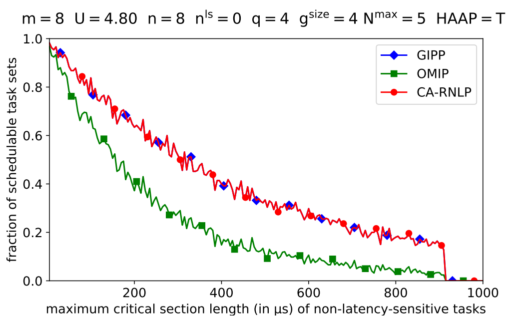

$m = 8$ $U = 3.20$ $n = 16$ $n^{ls} = 0$ $q = 16$ $g^{size} = 4$ $N^{max} = 10$ HAAP = T
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

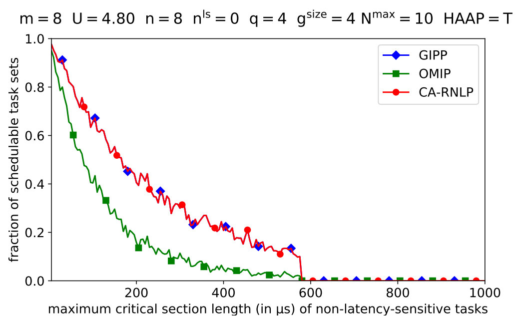

$m = 8$ $U = 3.20$ $n = 16$ $n^{ls} = 0$ $q = 16$ $g^{size} = 4$ $N^{max} = 15$ HAAP = T
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

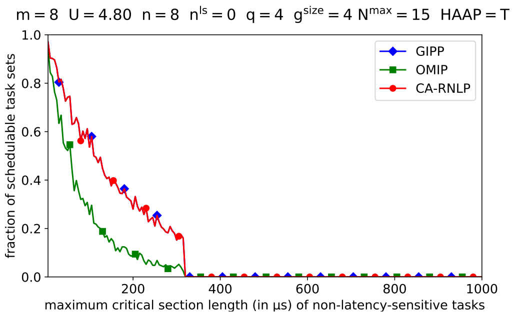

$m = 8$ $U = 3.20$ $n = 16$ $n^{ls} = 0$ $q = 20$ $g^{size} = 5$ $N^{max} = 5$ HAAP = T
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

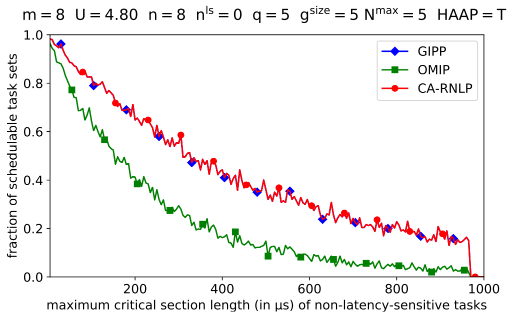

$m = 8$ $U = 3.20$ $n = 16$ $n^{ls} = 0$ $q = 20$ $g^{size} = 5$ $N^{max} = 10$ HAAP = T
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

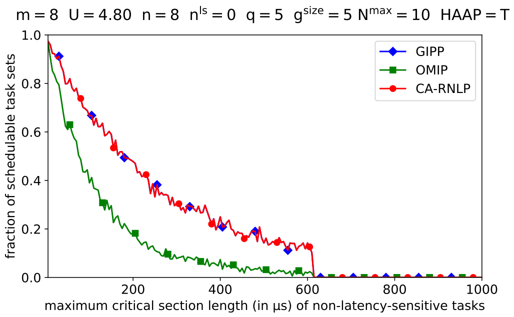

$m = 8$ $U = 3.20$ $n = 16$ $n^{ls} = 0$ $q = 20$ $g^{size} = 5$ $N^{max} = 15$ HAAP = T
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

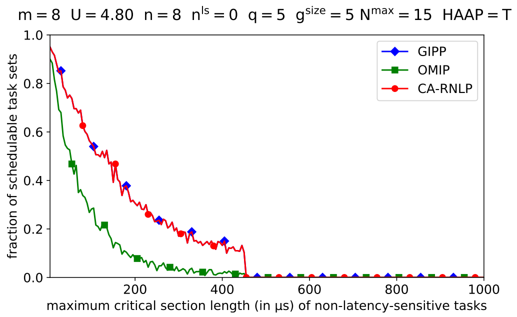

$m = 8$ $U = 4.80$ $n = 8$ $n^{ls} = 0$ $q = 4$ $g^{size} = 4$ $N^{max} = 5$ HAAP = T
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

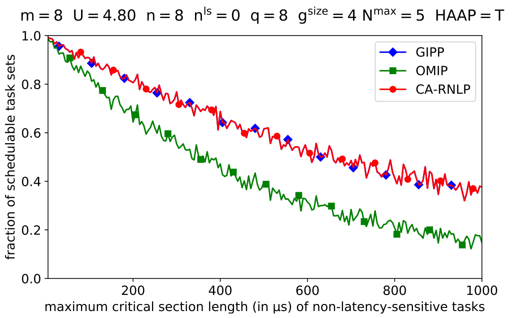

$m = 8$ $U = 4.80$ $n = 8$ $n^{ls} = 0$ $q = 4$ $g^{size} = 4$ $N^{max} = 10$ HAAP = T
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

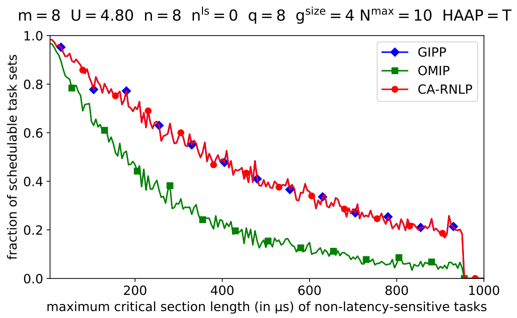

$m = 8$ $U = 4.80$ $n = 8$ $n^{ls} = 0$ $q = 4$ $g^{size} = 4$ $N^{max} = 15$ HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks

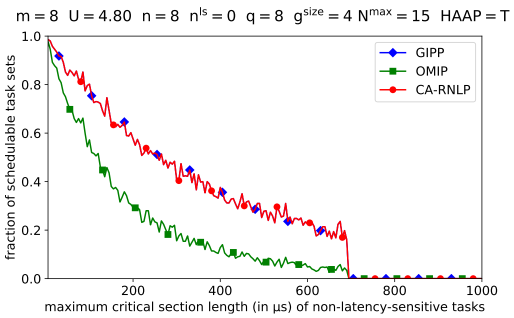

$m = 8$ $U = 4.80$ $n = 8$ $n^{ls} = 0$ $q = 5$ $g^{size} = 5$ $N^{max} = 5$ HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks

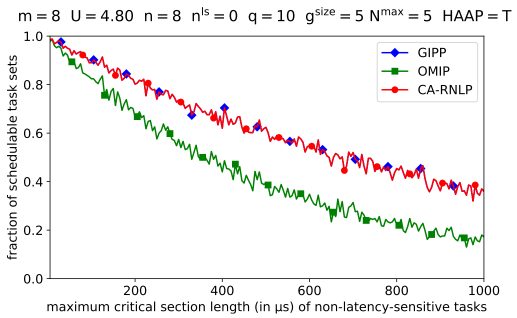

$m = 8$ $U = 4.80$ $n = 8$ $n^{ls} = 0$ $q = 5$ $g^{size} = 5$ $N^{max} = 10$ HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks

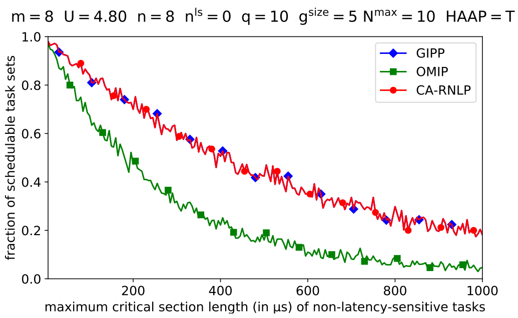

$m = 8$ $U = 4.80$ $n = 8$ $n^{ls} = 0$ $q = 5$ $g^{size} = 5$ $N^{max} = 15$ HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks

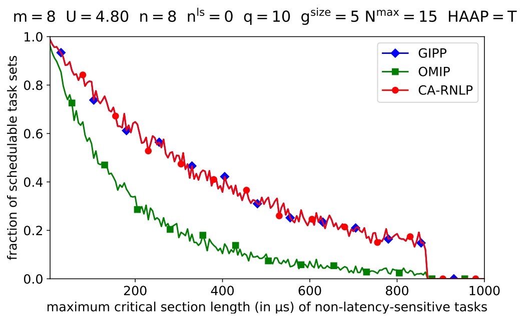

$m = 8$ $U = 4.80$ $n = 8$ $n^{ls} = 0$ $q = 8$ $g^{size} = 4$ $N^{max} = 5$ HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks


$m = 8$ $U = 4.80$ $n = 8$ $n^{ls} = 0$ $q = 8$ $g^{size} = 4$ $N^{max} = 10$ HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks

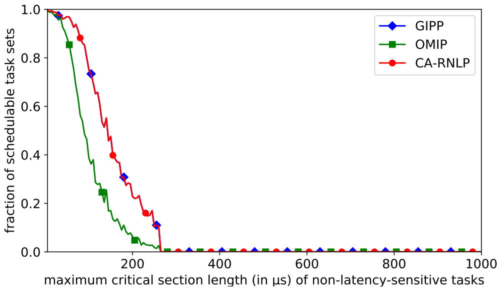

$m = 8$ $U = 4.80$ $n = 8$ $n^{ls} = 0$ $q = 8$ $g^{size} = 4$ $N^{max} = 15$ HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks


$m = 8$ $U = 4.80$ $n = 8$ $n^{ls} = 0$ $q = 10$ $g^{size} = 5$ $N^{max} = 5$ HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks

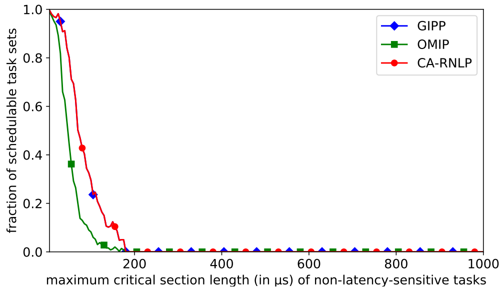

$m = 8$ $U = 4.80$ $n = 8$ $n^{ls} = 0$ $q = 10$ $g^{size} = 5$ $N^{max} = 10$ HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks


$m = 8$ $U = 4.80$ $n = 8$ $n^{ls} = 0$ $q = 10$ $g^{size} = 5$ $N^{max} = 15$ HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks

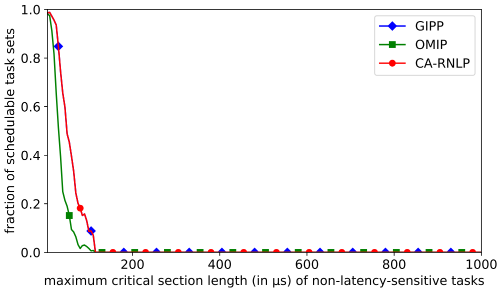
$m = 8$ $U = 4.80$ $n = 16$ $n^{ls} = 0$ $q = 4$ $g^{size} = 4$ $N^{max} = 5$ $HAAP = T$



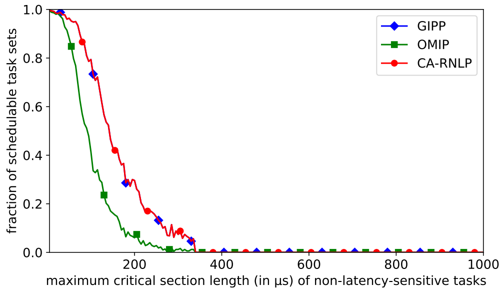
$m = 8$ $U = 4.80$ $n = 16$ $n^{ls} = 0$ $q = 4$ $g^{size} = 4$ $N^{max} = 10$ $HAAP = T$



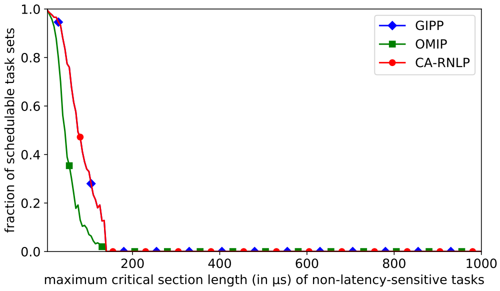
$m = 8$ $U = 4.80$ $n = 16$ $n^{ls} = 0$ $q = 4$ $g^{size} = 4$ $N^{max} = 15$ $HAAP = T$



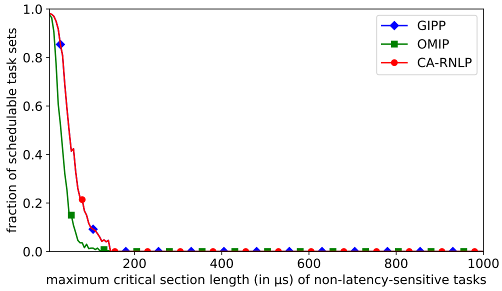
$m = 8$ $U = 4.80$ $n = 16$ $n^{ls} = 0$ $q = 5$ $g^{size} = 5$ $N^{max} = 5$ $HAAP = T$



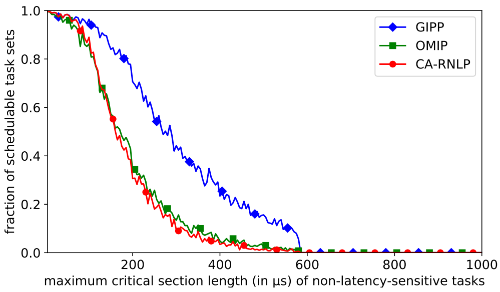
$m = 8$ $U = 4.80$ $n = 16$ $n^{ls} = 0$ $q = 5$ $g^{size} = 5$ $N^{max} = 10$ $HAAP = T$



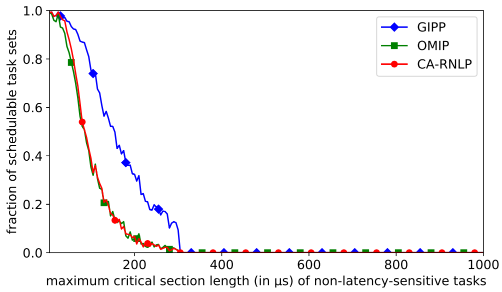
$m = 8$ $U = 4.80$ $n = 16$ $n^{ls} = 0$ $q = 5$ $g^{size} = 5$ $N^{max} = 15$ $HAAP = T$



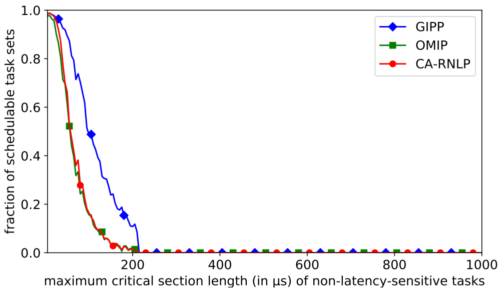
$m = 8$ $U = 4.80$ $n = 16$ $n^{ls} = 0$ $q = 8$ $g^{size} = 4$ $N^{max} = 5$ $HAAP = T$



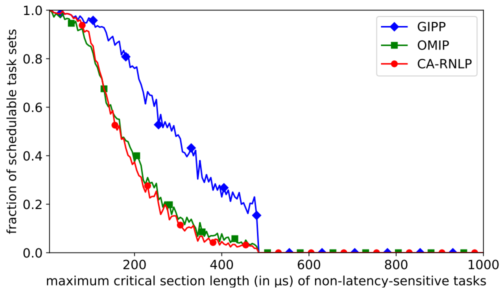
$m = 8$ $U = 4.80$ $n = 16$ $n^{ls} = 0$ $q = 8$ $g^{size} = 4$ $N^{max} = 10$ $HAAP = T$

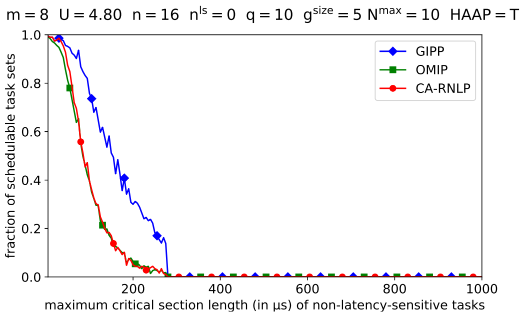


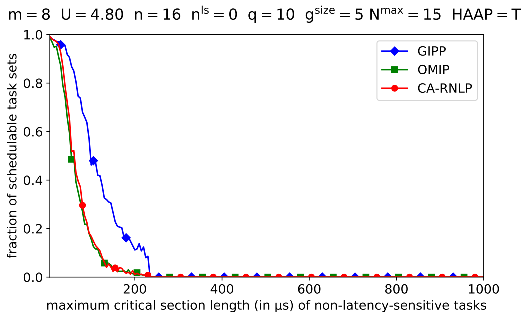
$m = 8$ $U = 4.80$ $n = 16$ $n^{ls} = 0$ $q = 8$ $g^{size} = 4$ $N^{max} = 15$ $HAAP = T$

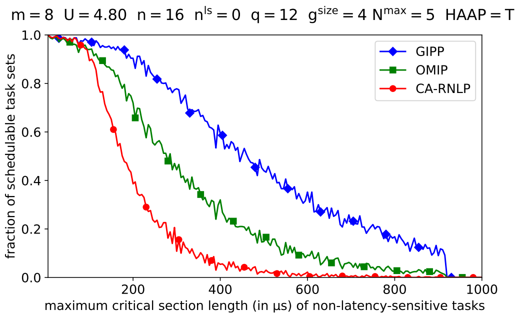


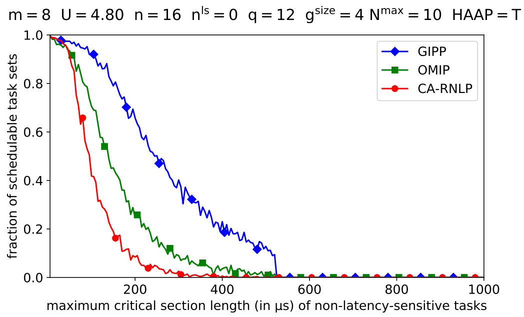
$m = 8$ $U = 4.80$ $n = 16$ $n^{ls} = 0$ $q = 10$ $g^{size} = 5$ $N^{max} = 5$ $HAAP = T$

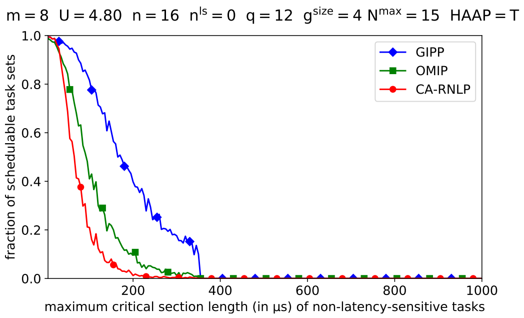


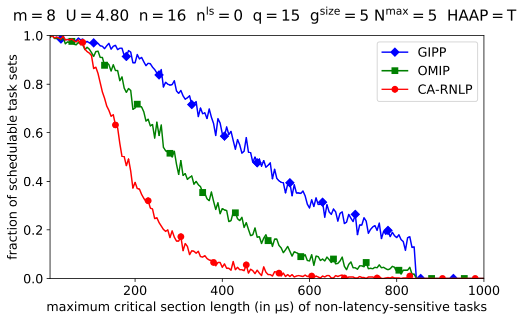

m = 8 U = 4.80 n = 16 n^{ls} = 0 q = 10 g^{size} = 5 N^{max} = 10 HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

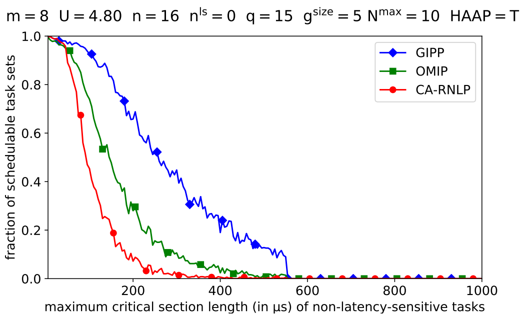

m = 8 U = 4.80 n = 16 n^{ls} = 0 q = 10 g^{size} = 5 N^{max} = 15 HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

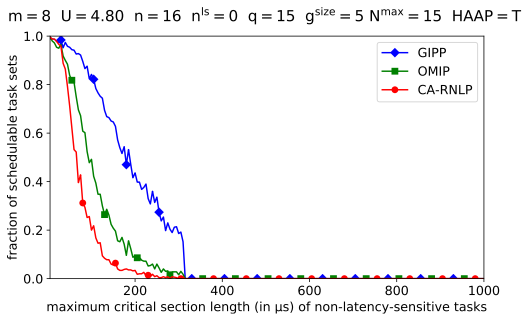

m = 8 U = 4.80 n = 16 n^{ls} = 0 q = 12 g^{size} = 4 N^{max} = 5 HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

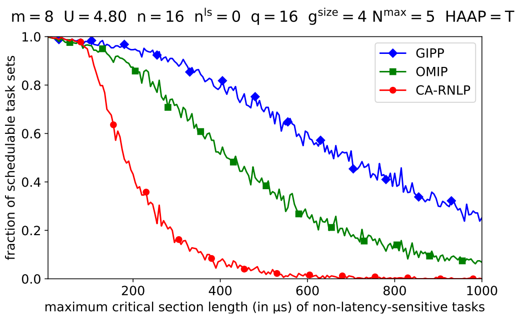

m = 8 U = 4.80 n = 16 n^{ls} = 0 q = 12 g^{size} = 4 N^{max} = 10 HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

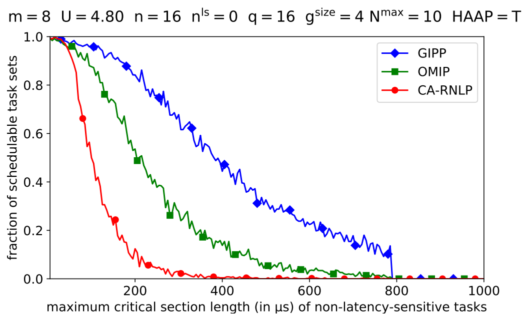

m = 8 U = 4.80 n = 16 n^{ls} = 0 q = 12 g^{size} = 4 N^{max} = 15 HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

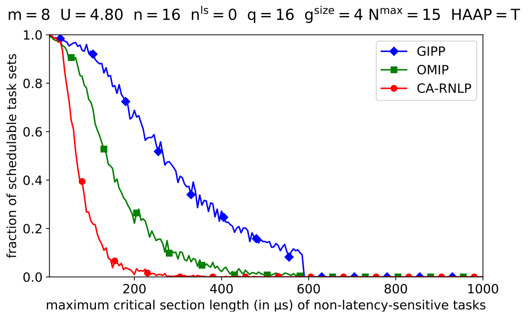

m = 8 U = 4.80 n = 16 n^{ls} = 0 q = 15 g^{size} = 5 N^{max} = 5 HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

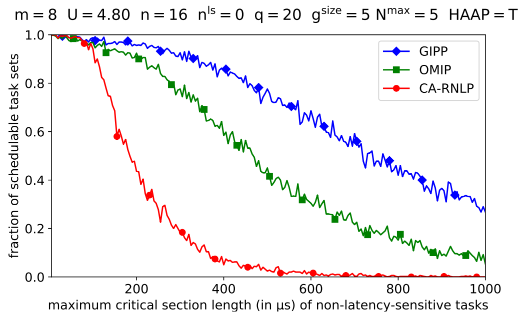

m = 8 U = 4.80 n = 16 n^{ls} = 0 q = 15 g^{size} = 5 N^{max} = 10 HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

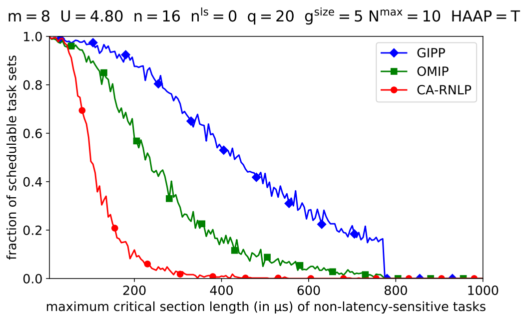

m = 8 U = 4.80 n = 16 n^{ls} = 0 q = 15 g^{size} = 5 N^{max} = 15 HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

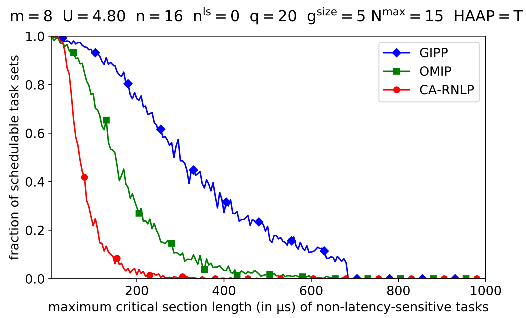

m = 8 U = 4.80 n = 16 n^{ls} = 0 q = 16 g^{size} = 4 N^{max} = 5 HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

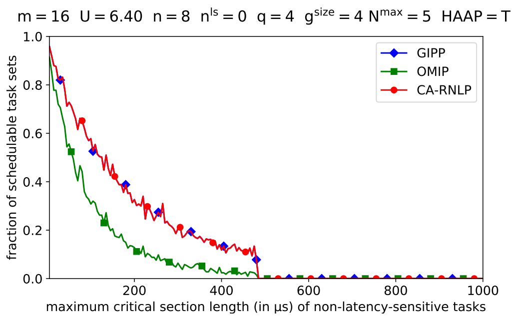

m = 8 U = 4.80 n = 16 n^{ls} = 0 q = 16 g^{size} = 4 N^{max} = 10 HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

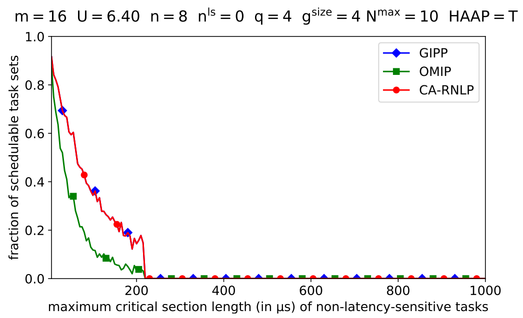

m = 8 U = 4.80 n = 16 n^{ls} = 0 q = 16 g^{size} = 4 N^{max} = 15 HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

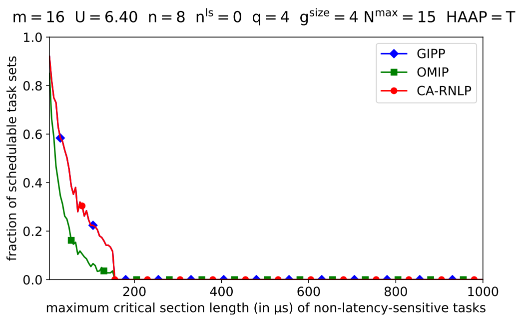

m = 8 U = 4.80 n = 16 n^{ls} = 0 q = 20 g^{size} = 5 N^{max} = 5 HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

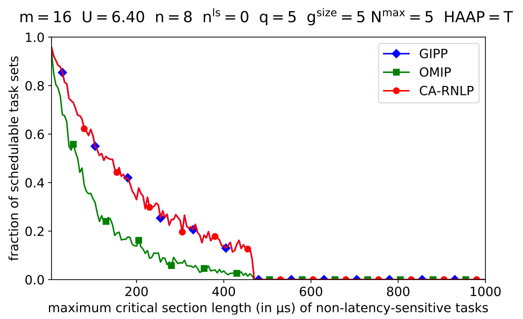

m = 8 U = 4.80 n = 16 n^{ls} = 0 q = 20 g^{size} = 5 N^{max} = 10 HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

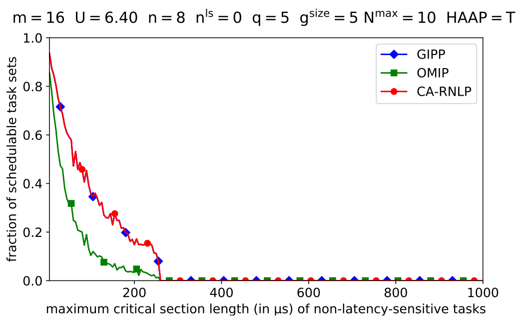

m = 8 U = 4.80 n = 16 n^{ls} = 0 q = 20 g^{size} = 5 N^{max} = 15 HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

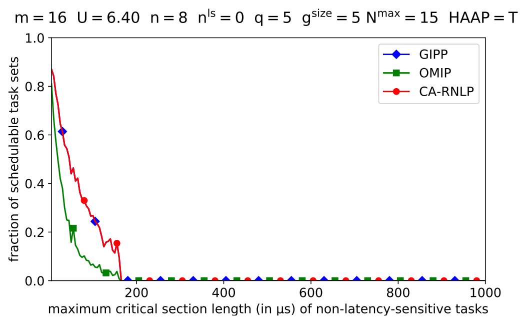

m = 16 U = 6.40 n = 8 n^{ls} = 0 q = 4 g^{size} = 4 N^{max} = 5 HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

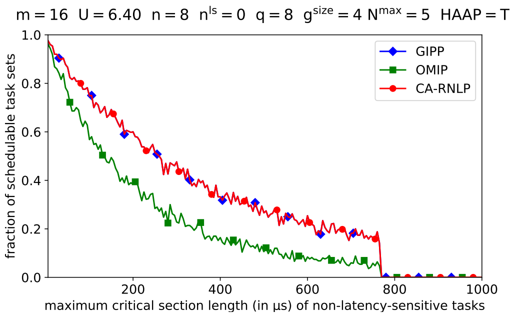

m = 16 U = 6.40 n = 8 n^{ls} = 0 q = 4 g^{size} = 4 N^{max} = 10 HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

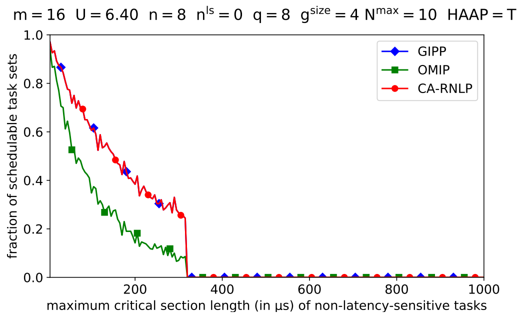

m = 16 U = 6.40 n = 8 n^{ls} = 0 q = 4 g^{size} = 4 N^{max} = 15 HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

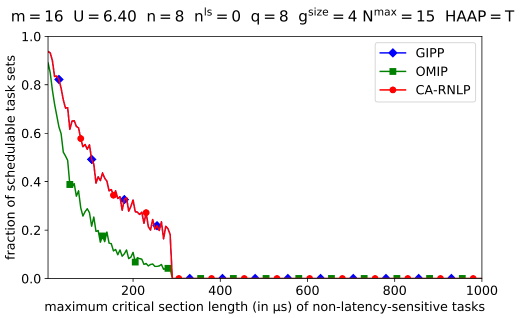

m = 16 U = 6.40 n = 8 n^{ls} = 0 q = 5 g^{size} = 5 N^{max} = 5 HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

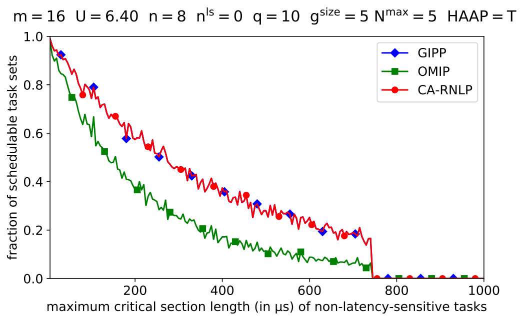

m = 16 U = 6.40 n = 8 n^{ls} = 0 q = 5 g^{size} = 5 N^{max} = 10 HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

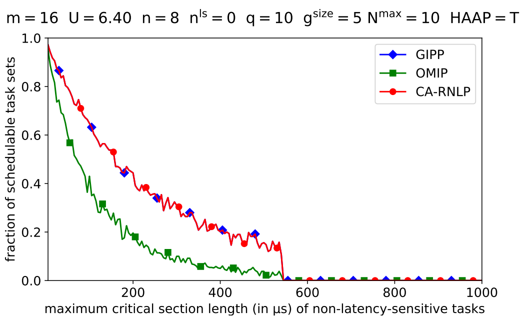

m = 16 U = 6.40 n = 8 n^{ls} = 0 q = 5 g^{size} = 5 N^{max} = 15 HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

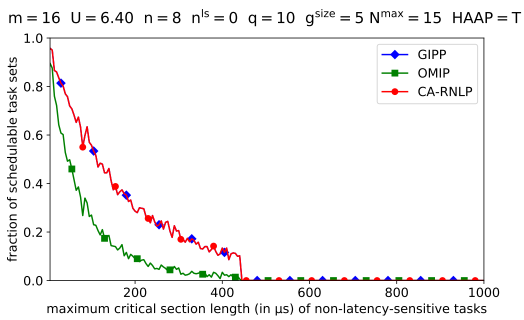

m = 16 U = 6.40 n = 8 n^{ls} = 0 q = 8 g^{size} = 4 N^{max} = 5 HAAP = T
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

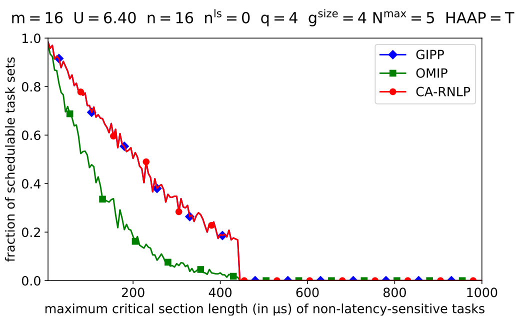

m = 16 U = 6.40 n = 8 n^{ls} = 0 q = 8 g^{size} = 4 N^{max} = 10 HAAP = T
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

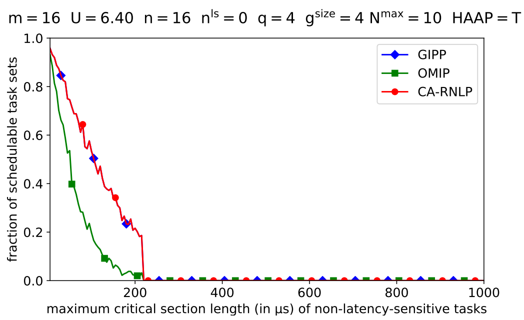

m = 16 U = 6.40 n = 8 n^{ls} = 0 q = 8 g^{size} = 4 N^{max} = 15 HAAP = T
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

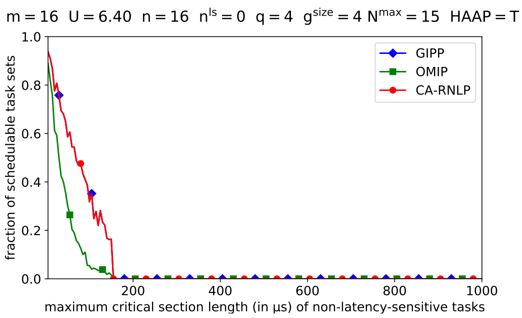

m = 16 U = 6.40 n = 8 n^{ls} = 0 q = 10 g^{size} = 5 N^{max} = 5 HAAP = T
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

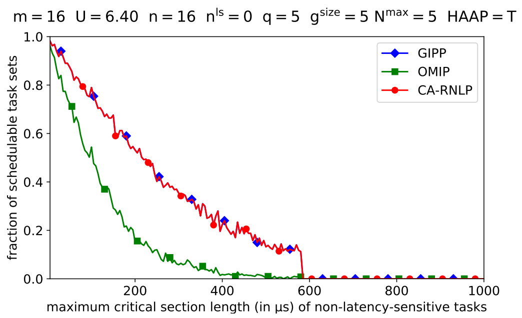

m = 16 U = 6.40 n = 8 n^{ls} = 0 q = 10 g^{size} = 5 N^{max} = 10 HAAP = T
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

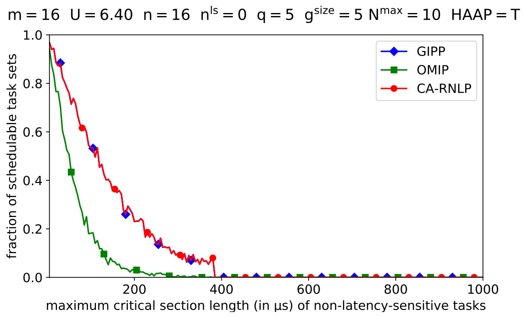

m = 16 U = 6.40 n = 8 n^{ls} = 0 q = 10 g^{size} = 5 N^{max} = 15 HAAP = T
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

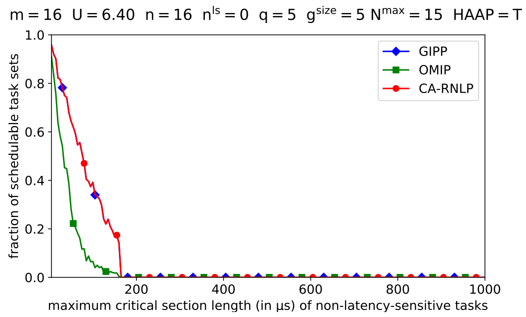

m = 16 U = 6.40 n = 16 n^{ls} = 0 q = 4 g^{size} = 4 N^{max} = 5 HAAP = T
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

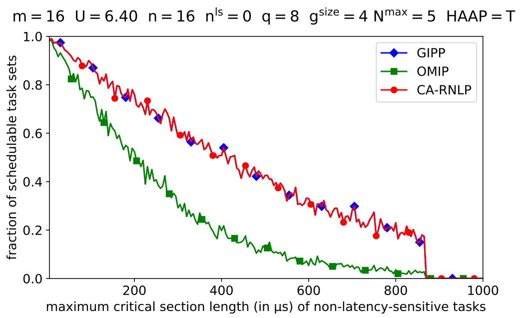

m = 16 U = 6.40 n = 16 n^{ls} = 0 q = 4 g^{size} = 4 N^{max} = 10 HAAP = T
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

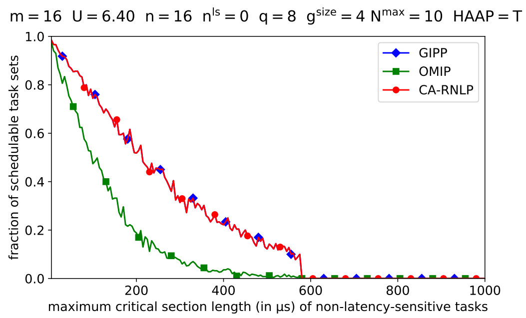

m = 16 U = 6.40 n = 16 n^{ls} = 0 q = 4 g^{size} = 4 N^{max} = 15 HAAP = T
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

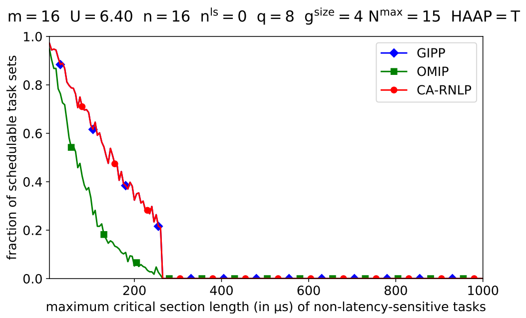

m = 16 U = 6.40 n = 16 n^{ls} = 0 q = 5 g^{size} = 5 N^{max} = 5 HAAP = T
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

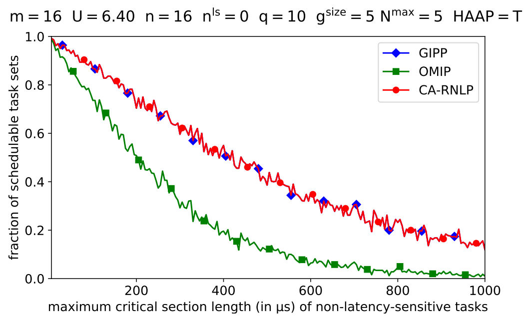

$m = 16$ $U = 6.40$ $n = 16$ $n^{ls} = 0$ $q = 5$ $g^{size} = 5$ $N^{max} = 10$ HAAP = T
fraction of schedulable task sets
1.0
0.8
0.6
0.4
0.2
0.0
200
400
600
800
1000
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

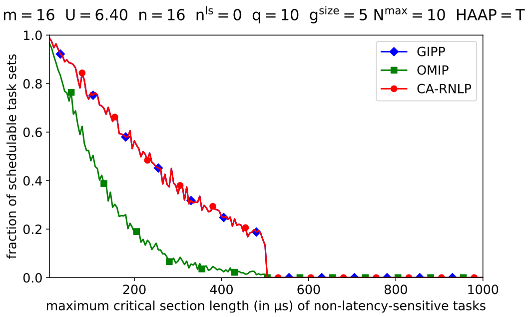

$m = 16$ $U = 6.40$ $n = 16$ $n^{ls} = 0$ $q = 5$ $g^{size} = 5$ $N^{max} = 15$ HAAP = T
fraction of schedulable task sets
1.0
0.8
0.6
0.4
0.2
0.0
200
400
600
800
1000
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

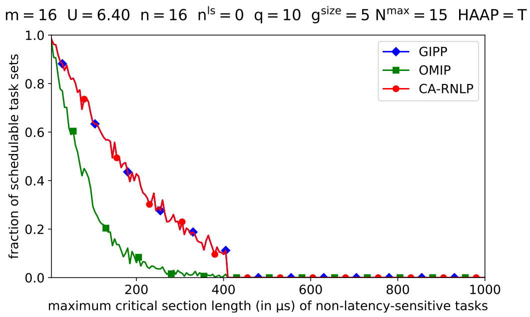

$m = 16$ $U = 6.40$ $n = 16$ $n^{ls} = 0$ $q = 8$ $g^{size} = 4$ $N^{max} = 5$ HAAP = T
fraction of schedulable task sets
1.0
0.8
0.6
0.4
0.2
0.0
200
400
600
800
1000
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

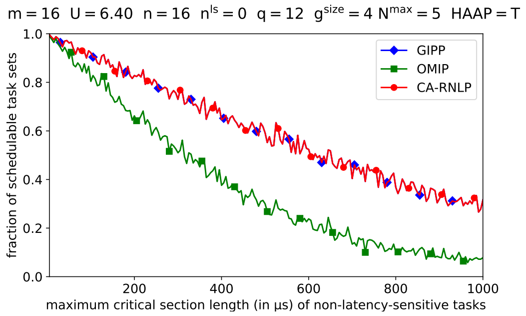

$m = 16$ $U = 6.40$ $n = 16$ $n^{ls} = 0$ $q = 8$ $g^{size} = 4$ $N^{max} = 10$ HAAP = T
fraction of schedulable task sets
1.0
0.8
0.6
0.4
0.2
0.0
200
400
600
800
1000
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

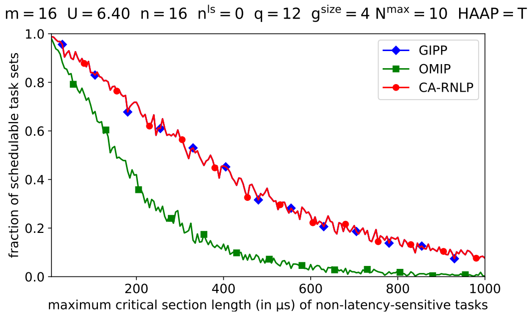

$m = 16$ $U = 6.40$ $n = 16$ $n^{ls} = 0$ $q = 8$ $g^{size} = 4$ $N^{max} = 15$ HAAP = T
fraction of schedulable task sets
1.0
0.8
0.6
0.4
0.2
0.0
200
400
600
800
1000
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

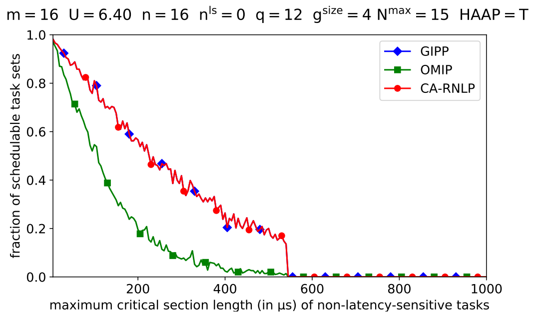

$m = 16$ $U = 6.40$ $n = 16$ $n^{ls} = 0$ $q = 10$ $g^{size} = 5$ $N^{max} = 5$ HAAP = T
fraction of schedulable task sets
1.0
0.8
0.6
0.4
0.2
0.0
200
400
600
800
1000
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

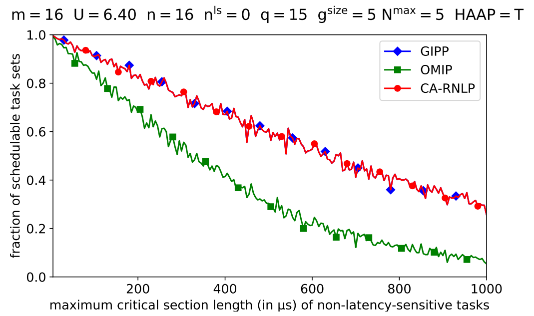

$m = 16$ $U = 6.40$ $n = 16$ $n^{ls} = 0$ $q = 10$ $g^{size} = 5$ $N^{max} = 10$ HAAP = T
fraction of schedulable task sets
1.0
0.8
0.6
0.4
0.2
0.0
200
400
600
800
1000
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

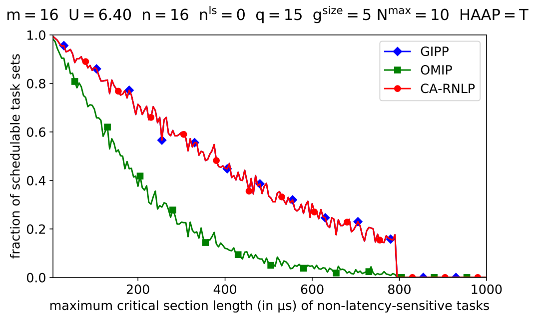

$m = 16$ $U = 6.40$ $n = 16$ $n^{ls} = 0$ $q = 10$ $g^{size} = 5$ $N^{max} = 15$ HAAP = T
fraction of schedulable task sets
1.0
0.8
0.6
0.4
0.2
0.0
200
400
600
800
1000
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

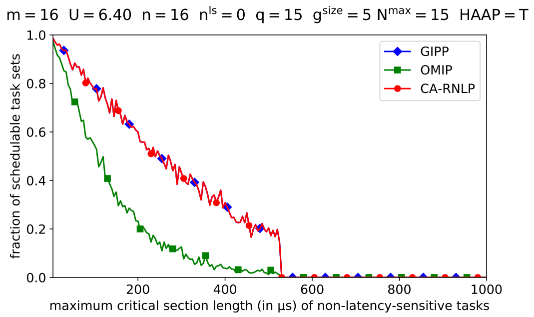

$m = 16$ $U = 6.40$ $n = 16$ $n^{ls} = 0$ $q = 12$ $g^{size} = 4$ $N^{max} = 5$ HAAP = T
fraction of schedulable task sets
1.0
0.8
0.6
0.4
0.2
0.0
200
400
600
800
1000
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

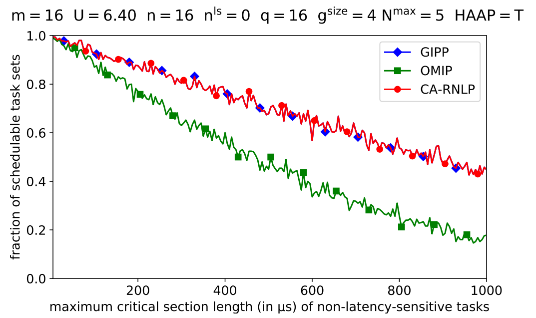

$m = 16$ $U = 6.40$ $n = 16$ $n^{ls} = 0$ $q = 12$ $g^{size} = 4$ $N^{max} = 10$ HAAP = T
fraction of schedulable task sets
1.0
0.8
0.6
0.4
0.2
0.0
200
400
600
800
1000
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

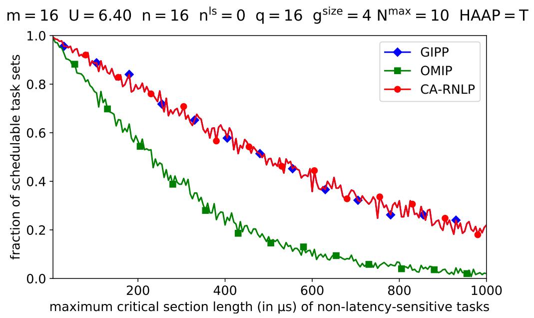

m = 16 U = 6.40 n = 16 n^{ls} = 0 q = 12 g^{size} = 4 N^{max} = 15 HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

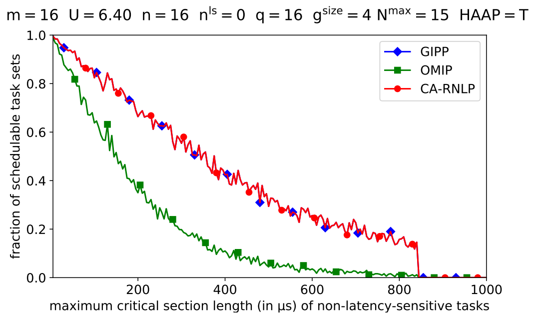

m = 16 U = 6.40 n = 16 n^{ls} = 0 q = 15 g^{size} = 5 N^{max} = 5 HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

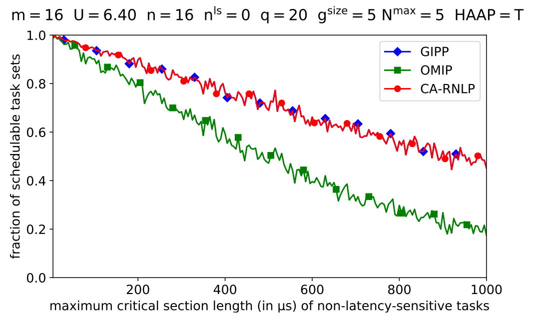

m = 16 U = 6.40 n = 16 n^{ls} = 0 q = 15 g^{size} = 5 N^{max} = 10 HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

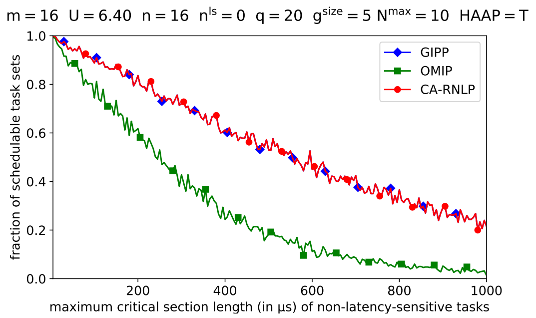

m = 16 U = 6.40 n = 16 n^{ls} = 0 q = 15 g^{size} = 5 N^{max} = 15 HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

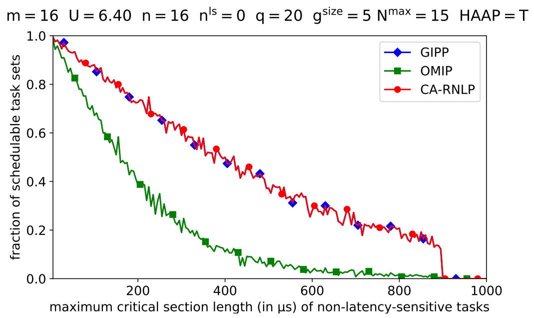

m = 16 U = 6.40 n = 16 n^{ls} = 0 q = 16 g^{size} = 4 N^{max} = 5 HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

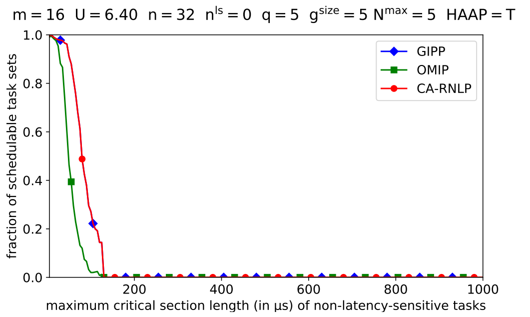

m = 16 U = 6.40 n = 16 n^{ls} = 0 q = 16 g^{size} = 4 N^{max} = 10 HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

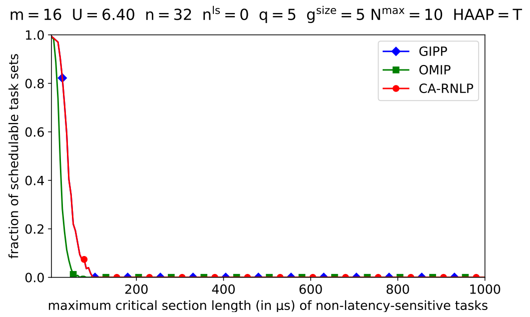

m = 16 U = 6.40 n = 16 n^{ls} = 0 q = 16 g^{size} = 4 N^{max} = 15 HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

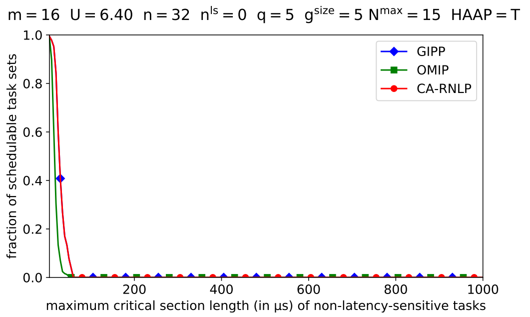

m = 16 U = 6.40 n = 16 n^{ls} = 0 q = 20 g^{size} = 5 N^{max} = 5 HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

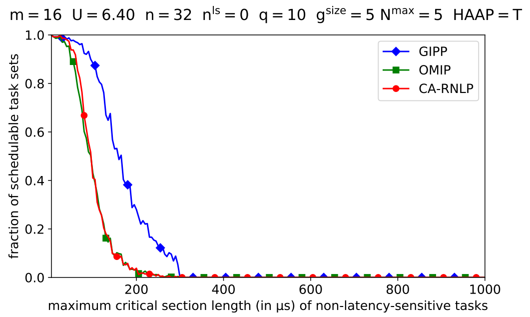

m = 16 U = 6.40 n = 16 n^{ls} = 0 q = 20 g^{size} = 5 N^{max} = 10 HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

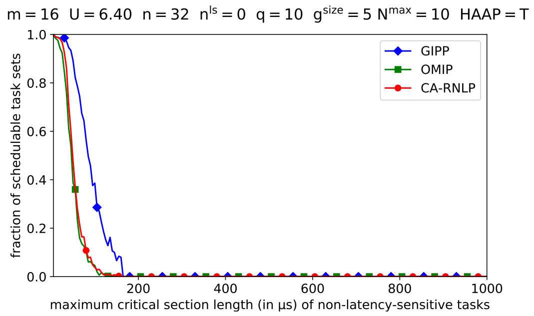

m = 16 U = 6.40 n = 16 n^{ls} = 0 q = 20 g^{size} = 5 N^{max} = 15 HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

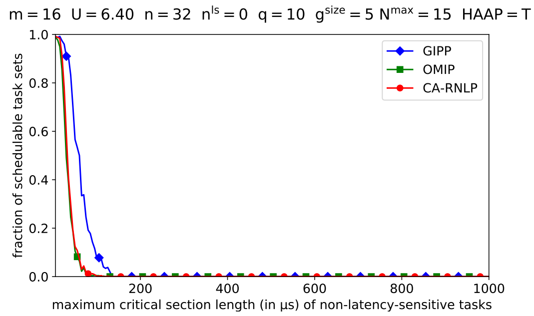

m = 16 U = 6.40 n = 32 n^{ls} = 0 q = 5 g^{size} = 5 N^{max} = 5 HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

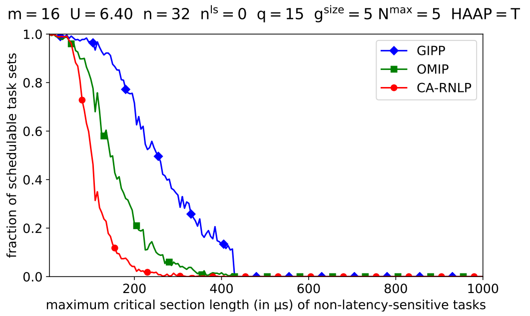

m = 16 U = 6.40 n = 32 n^{ls} = 0 q = 5 g^{size} = 5 N^{max} = 10 HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

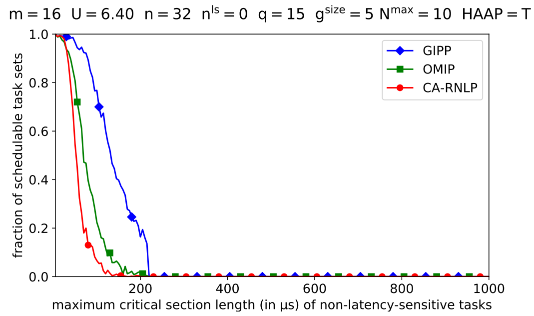

m = 16 U = 6.40 n = 32 n^{ls} = 0 q = 5 g^{size} = 5 N^{max} = 15 HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

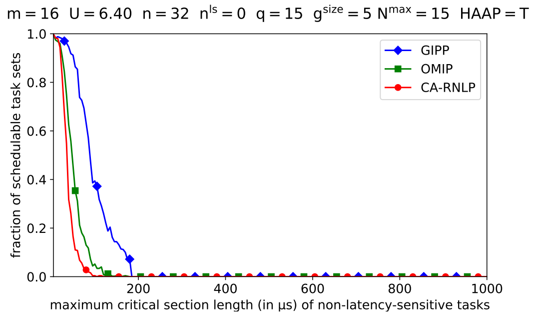

m = 16 U = 6.40 n = 32 n^{ls} = 0 q = 10 g^{size} = 5 N^{max} = 5 HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

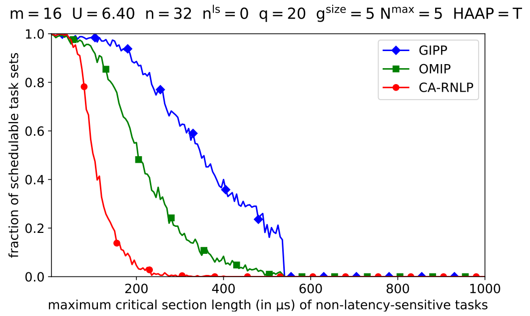

m = 16 U = 6.40 n = 32 n^{ls} = 0 q = 10 g^{size} = 5 N^{max} = 10 HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

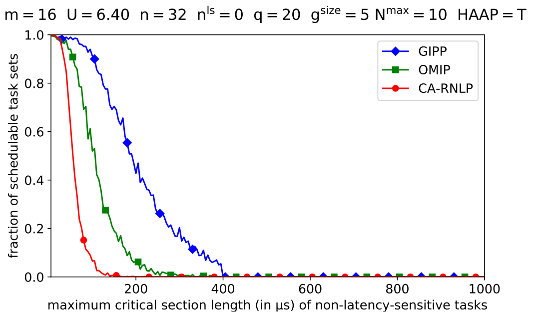

m = 16 U = 6.40 n = 32 n^{ls} = 0 q = 10 g^{size} = 5 N^{max} = 15 HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

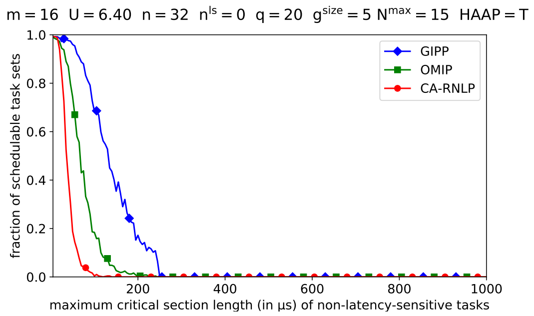

m = 16 U = 6.40 n = 32 n^{ls} = 0 q = 15 g^{size} = 5 N^{max} = 5 HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

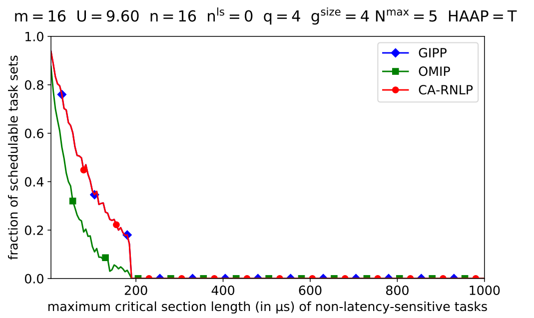

m = 16 U = 6.40 n = 32 n^{ls} = 0 q = 15 g^{size} = 5 N^{max} = 10 HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

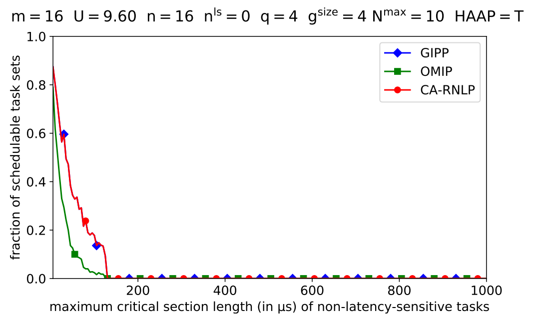

m = 16 U = 6.40 n = 32 n^{ls} = 0 q = 15 g^{size} = 5 N^{max} = 15 HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

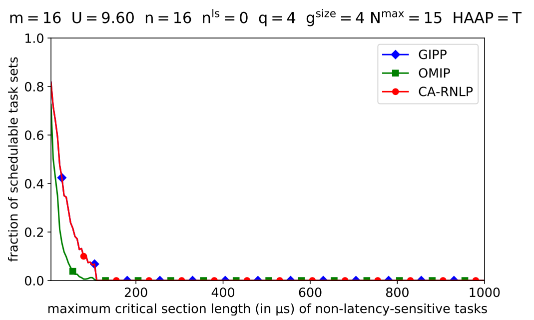

m = 16 U = 6.40 n = 32 n^{ls} = 0 q = 20 g^{size} = 5 N^{max} = 5 HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

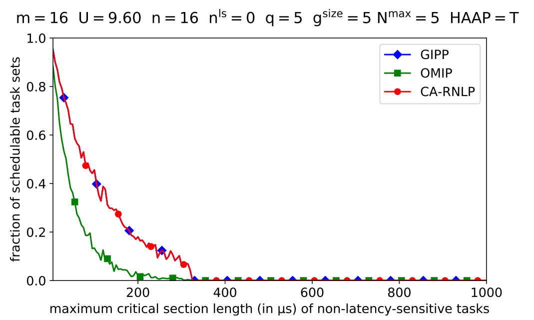

$m = 16$ $U = 6.40$ $n = 32$ $n^{ls} = 0$ $q = 20$ $g^{size} = 5$ $N^{max} = 10$ HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

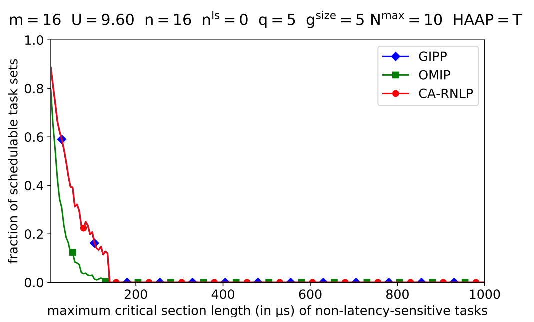

$m = 16$ $U = 6.40$ $n = 32$ $n^{ls} = 0$ $q = 20$ $g^{size} = 5$ $N^{max} = 15$ HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

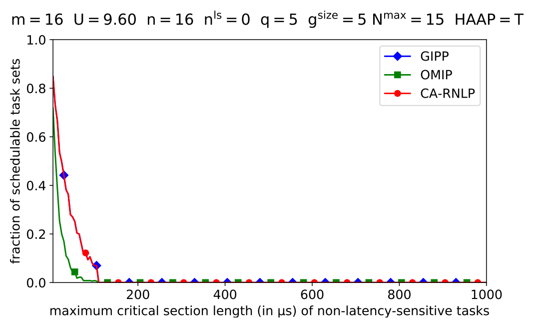

$m = 16$ $U = 9.60$ $n = 16$ $n^{ls} = 0$ $q = 4$ $g^{size} = 4$ $N^{max} = 5$ HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

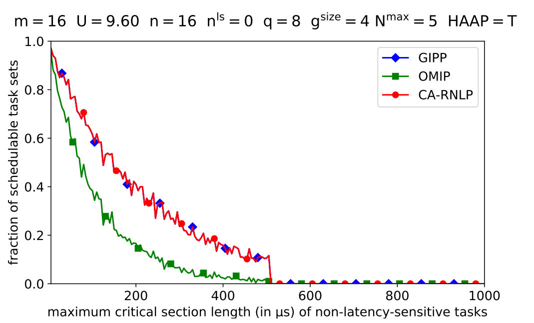

$m = 16$ $U = 9.60$ $n = 16$ $n^{ls} = 0$ $q = 4$ $g^{size} = 4$ $N^{max} = 10$ HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

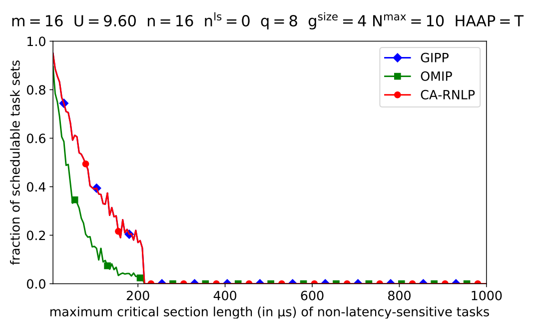

$m = 16$ $U = 9.60$ $n = 16$ $n^{ls} = 0$ $q = 4$ $g^{size} = 4$ $N^{max} = 15$ HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

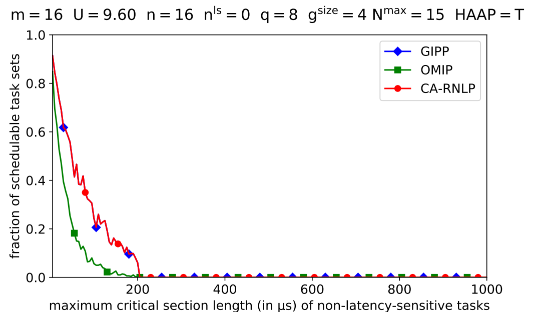

$m = 16$ $U = 9.60$ $n = 16$ $n^{ls} = 0$ $q = 5$ $g^{size} = 5$ $N^{max} = 5$ HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

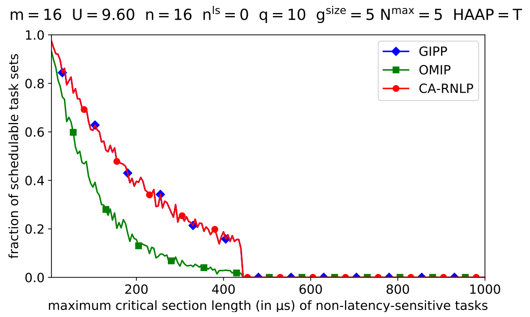

$m = 16$ $U = 9.60$ $n = 16$ $n^{ls} = 0$ $q = 5$ $g^{size} = 5$ $N^{max} = 10$ HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

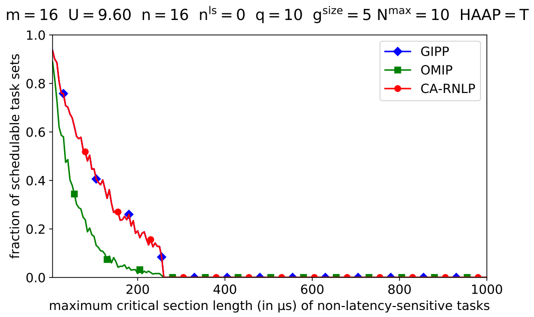

$m = 16$ $U = 9.60$ $n = 16$ $n^{ls} = 0$ $q = 5$ $g^{size} = 5$ $N^{max} = 15$ HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

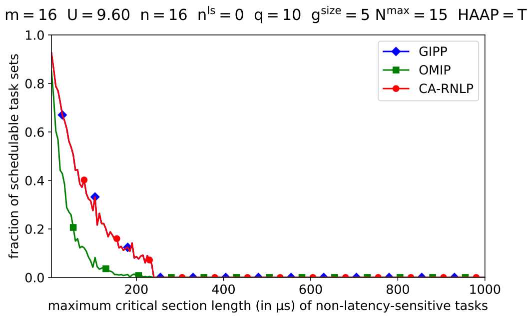

$m = 16$ $U = 9.60$ $n = 16$ $n^{ls} = 0$ $q = 8$ $g^{size} = 4$ $N^{max} = 5$ HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

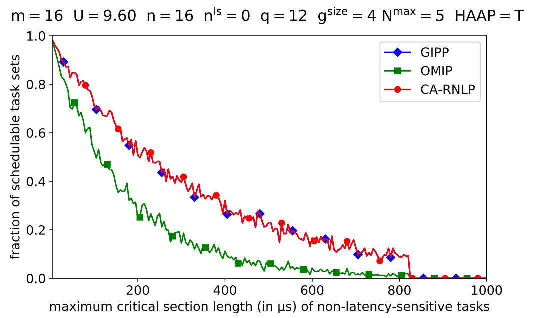

$m = 16$ $U = 9.60$ $n = 16$ $n^{ls} = 0$ $q = 8$ $g^{size} = 4$ $N^{max} = 10$ HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks

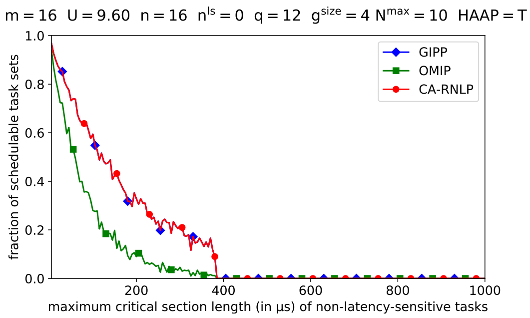

m = 16 U = 9.60 n = 16 n^ls = 0 q = 8 g^size = 4 N^max = 15 HAAP = T
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

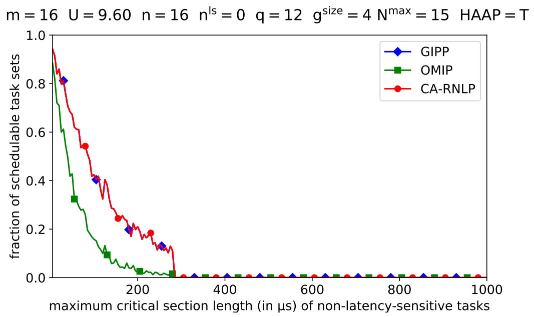

m = 16 U = 9.60 n = 16 n^ls = 0 q = 10 g^size = 5 N^max = 5 HAAP = T
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

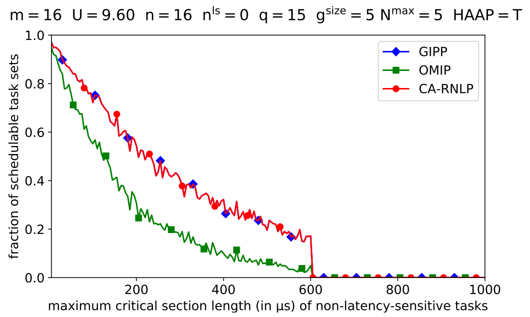

m = 16 U = 9.60 n = 16 n^ls = 0 q = 10 g^size = 5 N^max = 10 HAAP = T
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

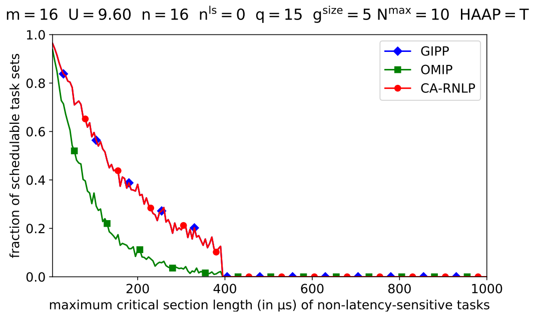

m = 16 U = 9.60 n = 16 n^ls = 0 q = 10 g^size = 5 N^max = 15 HAAP = T
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

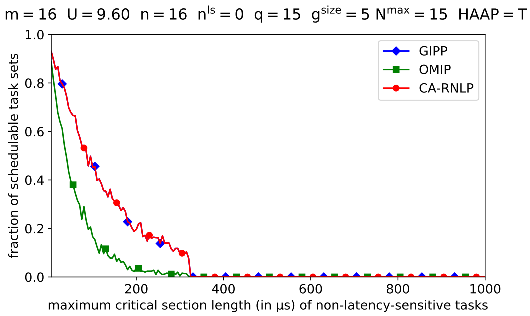

m = 16 U = 9.60 n = 16 n^ls = 0 q = 12 g^size = 4 N^max = 5 HAAP = T
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

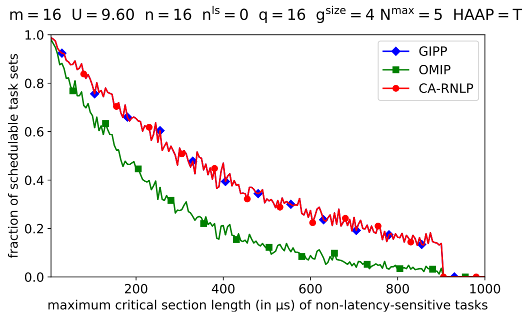

m = 16 U = 9.60 n = 16 n^ls = 0 q = 12 g^size = 4 N^max = 10 HAAP = T
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

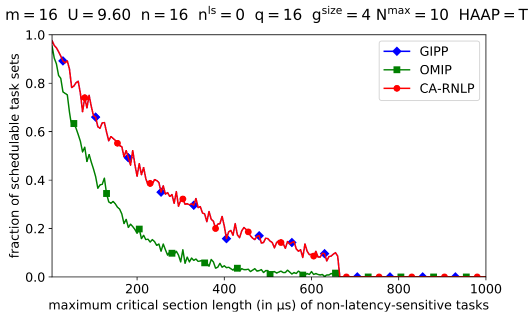

m = 16 U = 9.60 n = 16 n^ls = 0 q = 12 g^size = 4 N^max = 15 HAAP = T
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

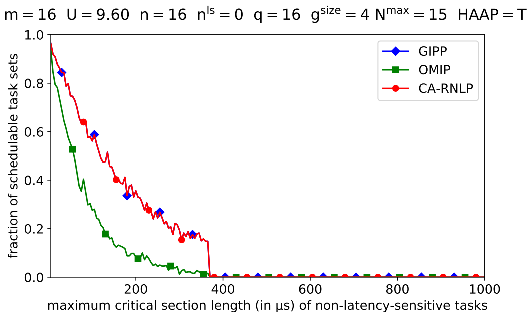

m = 16 U = 9.60 n = 16 n^ls = 0 q = 15 g^size = 5 N^max = 5 HAAP = T
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

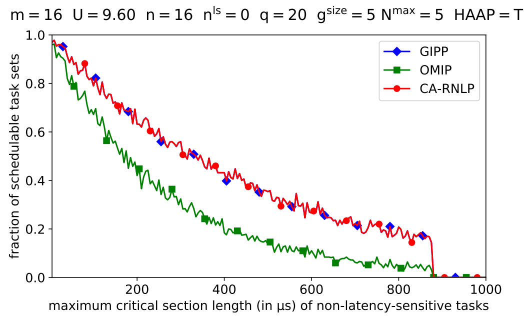

m = 16 U = 9.60 n = 16 n^ls = 0 q = 15 g^size = 5 N^max = 10 HAAP = T
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

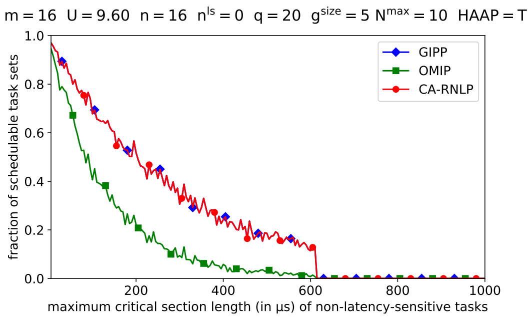

m = 16 U = 9.60 n = 16 n^ls = 0 q = 15 g^size = 5 N^max = 15 HAAP = T
fraction of schedulable task sets
maximum critical section length (in µs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

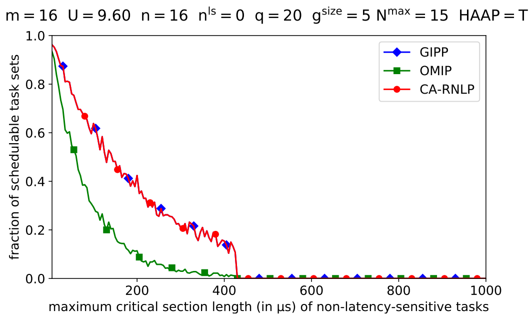

$m = 16$ $U = 9.60$ $n = 16$ $n^{ls} = 0$ $q = 16$ $g^{size} = 4$ $N^{max} = 5$ HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks

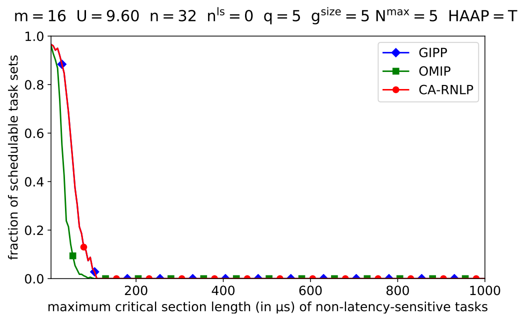

$m = 16$ $U = 9.60$ $n = 16$ $n^{ls} = 0$ $q = 16$ $g^{size} = 4$ $N^{max} = 10$ HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks

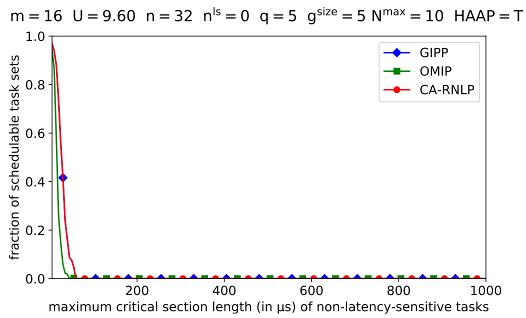

$m = 16$ $U = 9.60$ $n = 16$ $n^{ls} = 0$ $q = 16$ $g^{size} = 4$ $N^{max} = 15$ HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks

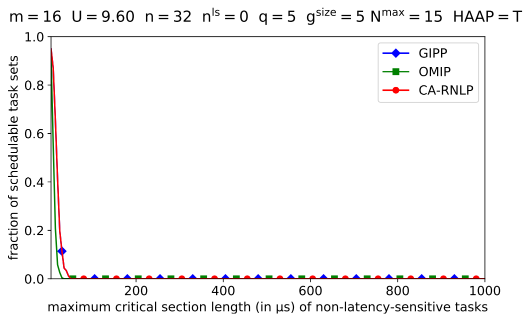

$m = 16$ $U = 9.60$ $n = 16$ $n^{ls} = 0$ $q = 20$ $g^{size} = 5$ $N^{max} = 5$ HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks

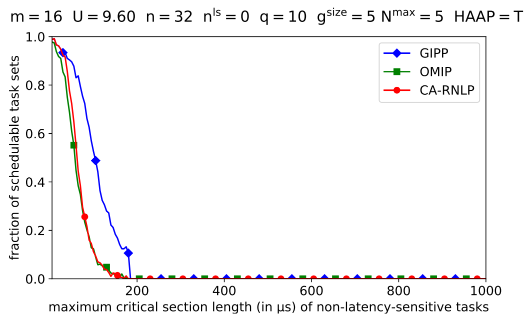

$m = 16$ $U = 9.60$ $n = 16$ $n^{ls} = 0$ $q = 20$ $g^{size} = 5$ $N^{max} = 10$ HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks

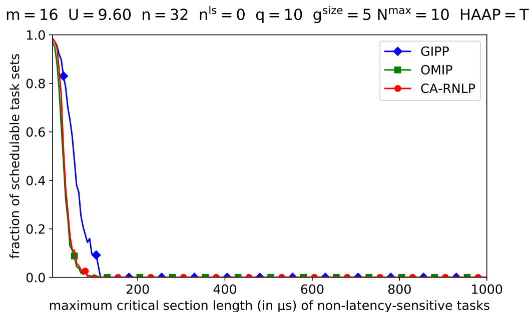

$m = 16$ $U = 9.60$ $n = 16$ $n^{ls} = 0$ $q = 20$ $g^{size} = 5$ $N^{max} = 15$ HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks

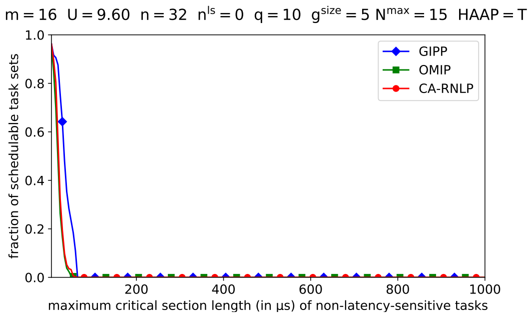

$m = 16$ $U = 9.60$ $n = 32$ $n^{ls} = 0$ $q = 5$ $g^{size} = 5$ $N^{max} = 5$ HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks

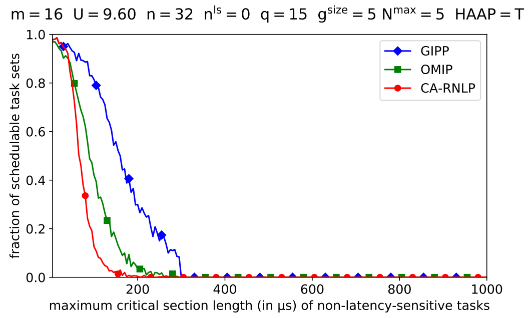

$m = 16$ $U = 9.60$ $n = 32$ $n^{ls} = 0$ $q = 5$ $g^{size} = 5$ $N^{max} = 10$ HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks

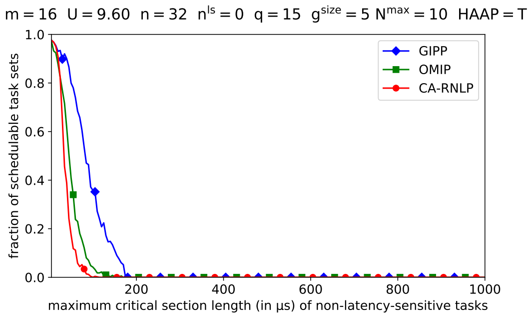

$m = 16$ $U = 9.60$ $n = 32$ $n^{ls} = 0$ $q = 5$ $g^{size} = 5$ $N^{max} = 15$ HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks

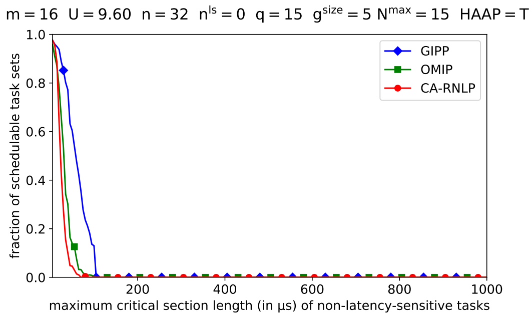

$m = 16$ $U = 9.60$ $n = 32$ $n^{ls} = 0$ $q = 10$ $g^{size} = 5$ $N^{max} = 5$ HAAP = T
GIPP
OMIP
CA-RNLP
fraction of schedulable task sets
maximum critical section length (in μs) of non-latency-sensitive tasks

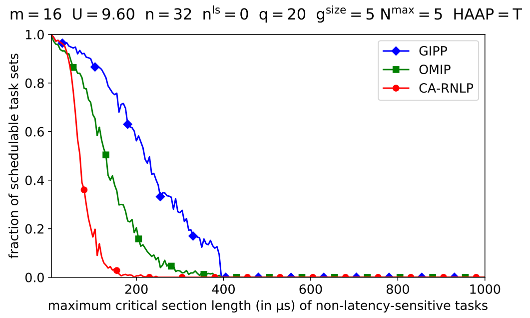

$m = 16$ $U = 9.60$ $n = 32$ $n^{ls} = 0$ $q = 10$ $g^{size} = 5$ $N^{max} = 10$ HAAP = T
fraction of schedulable task sets
0.0
0.2
0.4
0.6
0.8
1.0
200
400
600
800
1000
maximum critical section length (in μs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

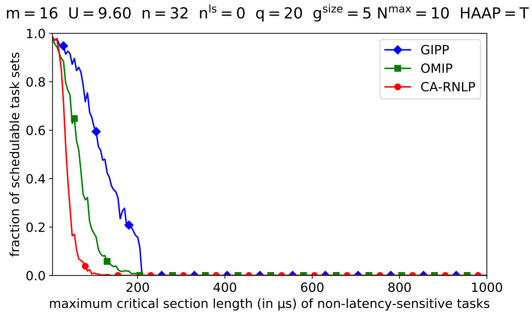

$m = 16$ $U = 9.60$ $n = 32$ $n^{ls} = 0$ $q = 10$ $g^{size} = 5$ $N^{max} = 15$ HAAP = T
fraction of schedulable task sets
0.0
0.2
0.4
0.6
0.8
1.0
200
400
600
800
1000
maximum critical section length (in μs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP

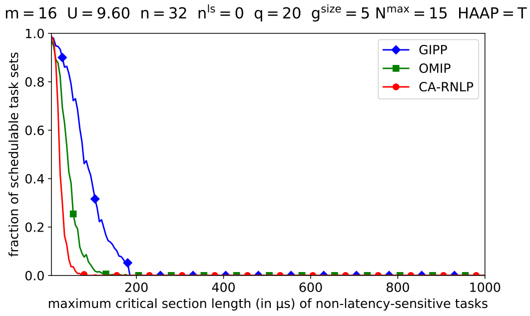

$m = 16$ $U = 9.60$ $n = 32$ $n^{ls} = 0$ $q = 15$ $g^{size} = 5$ $N^{max} = 5$ HAAP = T
fraction of schedulable task sets
0.0
0.2
0.4
0.6
0.8
1.0
200
400
600
800
1000
maximum critical section length (in μs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP


$m = 16$ $U = 9.60$ $n = 32$ $n^{ls} = 0$ $q = 15$ $g^{size} = 5$ $N^{max} = 10$ HAAP = T
fraction of schedulable task sets
0.0
0.2
0.4
0.6
0.8
1.0
200
400
600
800
1000
maximum critical section length (in μs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP


$m = 16$ $U = 9.60$ $n = 32$ $n^{ls} = 0$ $q = 15$ $g^{size} = 5$ $N^{max} = 15$ HAAP = T
fraction of schedulable task sets
0.0
0.2
0.4
0.6
0.8
1.0
200
400
600
800
1000
maximum critical section length (in μs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP


$m = 16$ $U = 9.60$ $n = 32$ $n^{ls} = 0$ $q = 20$ $g^{size} = 5$ $N^{max} = 5$ HAAP = T
fraction of schedulable task sets
0.0
0.2
0.4
0.6
0.8
1.0
200
400
600
800
1000
maximum critical section length (in μs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP


$m = 16$ $U = 9.60$ $n = 32$ $n^{ls} = 0$ $q = 20$ $g^{size} = 5$ $N^{max} = 10$ HAAP = T
fraction of schedulable task sets
0.0
0.2
0.4
0.6
0.8
1.0
200
400
600
800
1000
maximum critical section length (in μs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP


$m = 16$ $U = 9.60$ $n = 32$ $n^{ls} = 0$ $q = 20$ $g^{size} = 5$ $N^{max} = 15$ HAAP = T
fraction of schedulable task sets
0.0
0.2
0.4
0.6
0.8
1.0
200
400
600
800
1000
maximum critical section length (in μs) of non-latency-sensitive tasks
GIPP
OMIP
CA-RNLP