SCHEDULING ALGORITHMS AND TIMING ANALYSIS FOR HARD REAL-TIME SYSTEMS

Dissertation

zur Erlangung des Grades eines

DOKTORS DER INGENIEURWISSENSCHAFTEN

der Technischen Universität Dortmund
an der Fakultät für Informatik

von

WEN-HUNG KEVIN HUANG

Dortmund
2017

Tag der mündlichen Prüfung:	18. April 2017
Dekan:	Prof. Dr.-Ing. Gernot A. Fink
Gutachter:	Prof. Dr. Jian-Jia Chen
	Prof. Dr. Jan Reineke

If you put an infinite number of monkeys at typewriters, eventually one will bash out the script for Hamlet.

Infinite Monkey Theorem

ABSTRACT

Real-time systems are designed for applications in which response time is critical. As timing is a major property of such systems, proving timing correctness is of utter importance. To achieve this, a two-fold approach of timing analysis is traditionally involved: (*i*) worst-case execution time (WCET) analysis, which computes an upper bound on the execution time of a single job of a task running in isolation; and (*ii*) schedulability analysis using the WCET as the input, which determines whether multiple tasks are guaranteed to meet their deadlines.

Formal models used for representing recurrent real-time tasks have traditionally been characterized by a collection of independent jobs that are released periodically. However, such a modeling may result in resource under-utilization in systems whose behaviors are not entirely periodic or independent. Examples are (*i*) multicore platforms where tasks share a communication fabric, like bus, for accesses to a shared memory beside processors; (*ii*) tasks with synchronization, where no two concurrent access to one shared resource are allowed to be in their critical section at the same time; and (*iii*) automotive systems, where tasks are linked to rotation (e.g., of the crankshaft, gears, or wheels). There, their activation rate is proportional to the angular velocity of a specific device.

This dissertation presents multiple approaches towards designing scheduling algorithms and schedulability analysis for a variety of real-time systems with different characteristics. Specifically, we look at those design problems from the perspective of speedup factor — a metric that quantifies both the pessimism of the analysis and the non-optimality of the scheduling algorithm. The proposed solutions are shown promising by means of not only speedup factor but also extensive evaluations.

ACKNOWLEDGMENTS

First of all, I would like to express my gratitude to my advisor Prof. Dr. Jian-Jia Chen for giving me the opportunity to work in his group. I am truly grateful for the freedom and support of research he gave. He never hesitated to guide me through my troublesome times of doing research. I thank for all the interesting discussion we had altogether from different perspectives. I would also like to thank Prof. Dr. Jan Reineke for his time and commitment to review this dissertation. Furthermore, special thanks go to Prof. Dr. Heinrich Müller and Prof. Dr. Kristian Kersting for being in my doctoral committee.

I cherish the opportunity to work in a nice group: I thank Waqaas Munawar for being my officemate during my first year in Karlsruhe. I will yearn those inspiring off-topic talks. I thank Matthias Freier for being my peer and hospitable during my visit to Bosch. Many thanks to Timon Kelter for proofreading my drafts and papers and being my officemate in Dortmund. Part of the work presented in this dissertation has been done closely in collaboration with Georg von der Brueggen, Husheng Zhou, Maolin Yang, Kuan Hsun Chen, and Prof. Dr. Cong Liu. Thanks to all you guys. We did a good job.

Without all the kindness of everyone who had profoundly influenced me and were instrumental in getting me to this point, this work would not and could not have been done. I extend my deepest gratitude to all. Last, I also like to thank my friends and family for their unconditional love and support during these years.

This work was supported by DFG, as part of the Collaborative Research Center SFB876 (http://sfb876.tu-dortmund.de/).

PUBLICATIONS

The following publications are included in parts or in an extended version in this thesis:

- Wen-Hung Huang, Jian-Jia Chen, Husheng Zhou, and Cong Liu. "PASS: Priority Assignment of Real-Time Tasks with Dynamic Suspending Behavior under Fixed-Priority Scheduling." In: *Design Automation Conference (DAC), San Francisco, CA, USA*. 2015.
- Wen-Hung Huang and Jian-Jia Chen. "Techniques for Schedulability Analysis in Mode Change Systems under Fixed-Priority Scheduling." In: *IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*. (Best Paper Award). Hong Kong, 2015.
- Wen-Hung Huang and Jian-Jia Chen. "Response Time Bounds for Sporadic Arbitrary-Deadline Tasks under Global Fixed-Priority Scheduling on Multiprocessors." In: *International Conference on Real-Time Networks and Systems (RTNS)*. Lille, France, 2015.
- Wen-Hung Huang and Jian-Jia Chen. "Self-Suspension Real-Time Tasks under Fixed-Relative-Deadline Fixed-Priority Scheduling." In: *Design, Automation and Test in Europe (DATE)*. Dresden, Germany, 2016.
- Wen-Hung Huang and Jian-Jia Chen. "Utilization Bounds on Allocating Rate-Monotonic Scheduled Multi-Mode Tasks on Multiprocessor Systems." In: *Design Automation Conference (DAC)*. Austin, TX, USA, 2016.
- Wen-Hung Huang, Jian-Jia Chen, and Jan Reineke. "MIRROR: Symmetric Timing Analysis for Real-Time Tasks on Multicore Platforms with Shared Resources." In: *Design Automation Conference (DAC)*. Austin, TX, USA, 2016.
- Wen-Hung Huang, Maolin Yang, and Jian-Jia Chen. "Resource-Oriented Partitioned Scheduling in Multiprocessor Systems: How to Partition and How to Share?" In: *Real-Time Systems Symposium (RTSS)*. (Outstanding Paper Award). Porto, Portugal, 2016.

Furthermore, the following publications, which are part of my PhD research, are however not covered in this dissertation. The topics of these publications are outside of the scope of the material covered here:

- Jian-Jia Chen, Wen-Hung Huang, and Cong Liu. "k2U: A General Framework from k-Point Effective Schedulability Analysis to Utilization-Based Tests." In: *Real-Time Systems Symposium (RTSS)*. 2015.

- Konstantinos Bletsas, Neil Audsley, Wen-Hung Huang, Jian-Jia Chen, and Geoffrey Nelissen. *Errata for three papers (2004-05) on fixed-priority scheduling with self-suspensions*. Tech. rep. CISTER-TR-150713. CISTER, 2015.

- Georg von der Bruggen, Jian-Jia Chen, and Wen-Hung Huang. "Schedulability and Optimization Analysis for Non-preemptive Static Priority Scheduling Based on Task Utilization and Blocking Factors." In: *27th Euromicro Conference on Real-Time Systems, ECRTS 2015, Lund, Sweden, July 8-10, 2015*. 2015, pp. 90–101. DOI: 10.1109/ECRTS.2015.16. URL: http://dx.doi.org/10.1109/ECRTS.2015.16.

- Jian-Jia Chen et al. *Many Suspensions, Many Problems: A Review of Self-Suspending Tasks in Real-Time Systems*. Tech. rep. 854. (Status: Preprint). Department of Computer Science, TU Dortmund, 2016. URL: http://ls12-www.cs.tu-dortmund.de/daes/media/documents/publications/downloads/2016-chen-techreport-854.pdf.

- Jian-Jia Chen, Geoffrey Nelissen, and Wen-Hung Kevin Huang. "A Unifying Response Time Analysis Framework for Dynamic Self-Suspending Tasks." In: *Euromicro Conference on Real-Time Systems (ECRTS)*. Toulouse, France, 2016.

- Georg von der Brüggen, Jian-Jia Chen, Robert I. Davis, and Wen-Hung Kevin Huang. "Exact Speedup Factors for Linear-Time Schedulability Tests for Fixed-Priority Preemptive and Non-preemptive Scheduling." In: *Information Processing Letters (IPL)* (2016). URL: doi:10.1016/j.ipl.2016.08.001.

- Georg von der Brüggen, Wen-Hung Huang, Jian-Jia Chen, and Cong Liu. "Uniprocessor Scheduling Strategies for Self-Suspending Task Systems." In: *Proceedings of the 24th International Conference on Real-Time Networks and Systems, RTNS 2016, Brest, France, October 19-21, 2016*. 2016, pp. 119–128. DOI: 10.1145/2997465.2997497. URL: http://doi.acm.org/10.1145/2997465.2997497.

- Georg von der Brüggen, Kuan-Hsun Chen, Wen-Hung Huang, and Jian-Jia Chen. "Systems with Dynamic Real-Time Guarantees in Uncertain and Faulty Execution Environments." In: *Real-Time Systems Symposium (RTSS)*. Porto, Portugal, 2016.

- Jian-Jia Chen, Wen-Hung Huang, and Cong Liu. "k2Q: A Quadratic-Form Response Time and Schedulability Analysis Framework for Utilization-Based Analysis." In: *Real-Time Systems Symposium (RTSS)*. Porto, Portugal, 2016.

CONTENTS

1

INTRODUCTION

Today, real-time computing plays a crucial role in our society, since an increasing number of complex systems rely, in part or completely, on computer control. Real-time systems are defined as those systems in which the correctness of the system behavior depends not only on the logical results of the computations, but also on the physical instant at which these results are produced. Timing correctness requirements in a real-time system arise because of the physical impact of the controlling system's activities upon its environment. This raises the questions of what happens when timing correctness properties of the systems are not satisfied.

On September 20, 2007, the 2005 Toyota Camry driven by Jean Bookout crashed on an Oklahoma highway off-ramp, injuring her and killing her passenger, Barbara Schwarz. The blame for the accident is laid on Electronic Throttle Control System (ETCS) responsible for controlling air flow to the engine based on driver and vehicle condition, causing the vehicle to accelerate out of control. In a spite of no smoking gun founded during the Toyota investigations, Michael Barr, a well-respected embedded software specialist, after spending more than 20 months reviewing Toyota's source code, testified at trial: the death of tasks within a CPU due to not getting to use the CPU can cause the vehicle misbehavior, leading to loss of throttle control [Koo14].

1.1 CONCEPTS AND DEFINITIONS OF REAL-TIME SYSTEMS

Defining requirements to establish specifications is the first stage in the deign flow of a predictable real-time system. In establishing requirements, the specification would include technical information about specific design aspects. For example, consider a Lane Keeping Assist System (LKAS) that monitors the carriageway markings – usually using camera systems fitted behind the windscreen – to support the driver in staying within a lane through electronic assistance with the steering force. In such a system, only a few 100 milliseconds can be taken in determining if a vehicle is drifting off lane, and whether to steer the vehicle back towards the lane center. This timing requirement is written down on the specification to be satisfied.

A typical timing requirement on a task is called *deadline*, which represents the time before which a task should complete its execution without causing any damage to the system. Real-time systems/tasks are classified by the consequence of missing a deadline:

- *Hard real-time*: every deadline must be strictly met. Any deadline misses are deemed a total system failure. Some examples are nuclear systems, automotive

systems, medical applications, such as pacemakers, a large number of defense applications, and avionics.

- *Firm/soft real-time*: deadline misses can be tolerated. Performance will degrade if too many are missed. In firm real-time systems, the usefulness of a result is zero after its deadline, whereas in soft real-time systems, the usefulness of a result degrades after its deadline. Examples of soft real-time systems can be multimedia and telephony. Each audio or video frame must be rendered it time, or else it has to be skipped; If one misses a few bits, no big deal, but if missing too many, eventually the performance of the system degrades.

1.1.1 *Task model*

Most feedback control systems are essentially periodic, where the inputs (reading on sensors) and the outputs (posting on actuators) of the controller are sampled at a fixed rate. Here comes a real-time system example: Anti-lock Brake System (ABS). ABS is an automobile safety system that allows the wheels on a motor vehicle to maintain tractive contact with the road surface according to driver inputs while braking, preventing the wheels from locking up (ceasing rotation) and avoiding uncontrolled skidding. When the system detects sudden braking by comparing the sampled sensor data with the reference input, it will do computation to release braking pressure for a moment. Then, the optimum braking pressure is computed to provide to each wheel via actuator. Such a process is activated periodically, smoothing steering control during sudden stops. It has to be deadline-oriented, meaning that the computation must be finished within a deadline. A typical real-time control system and its respective software are shown in Figure 1.1 and Algorithm 1. This gives rise to the sporadic task model. Each task is characterized by a a three-tuple (C_i, T_i, D_i): C_i denotes the Worst-Case Execution Time (WCET) of each job of τ_i, T_i denotes the minimum inter-arrival time (also known as period) of τ_i; and D_i denotes the relative deadline.

Algorithm 1: Real-Time Control System

set timer to interrupt periodically with period T_i;
do
  perform analog-to-digital conversion $y(t)$ to get input y_k;
  compute control output u_k based on reference r_k and y_k;
  do digital-to-analog conversion of u_k to get output $u(t)$;
while *at each timer interrupt*;

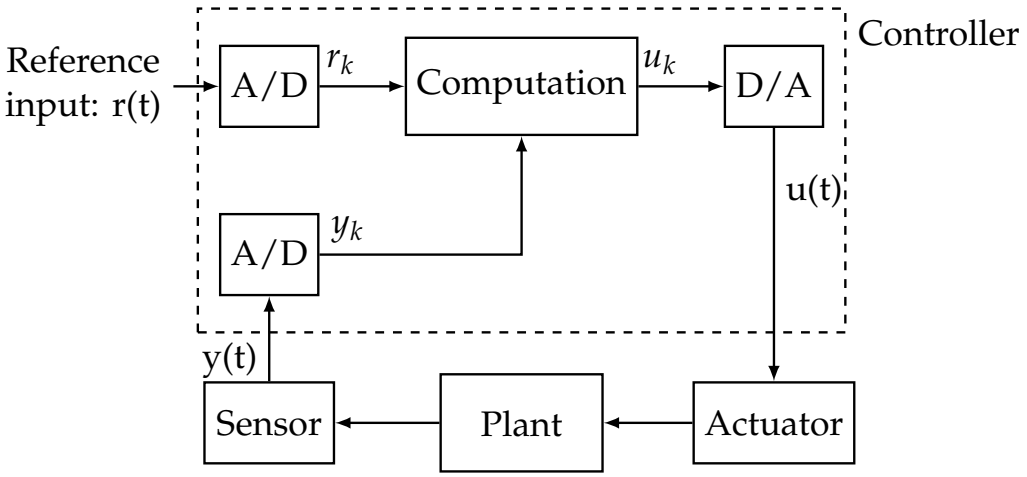


Figure 1.1: Real-Time Control System

1.1.2 *Deadline model*

Relative deadlines of tasks may be classified as:

- *Implicit-deadline*: a task's deadline is equal to its minimum inter-arrival time/period;
- *Constrained-deadline*: a task's deadline is no more than its minimum inter-arrival time/period; or
- *Arbitrary-deadline*: a task's deadline is not restrictive.

1.1.3 *Feasibility and Optimality*

Predictability is one of the most important properties that a hard real-time system should have. We call a system *predictable* if one can guarantee that 100% of all tasks over the entire life of the system will meet their timing requirements.

Definition 1.1 (feasibility)**.** A system τ is said to be *feasible* if there exists a scheduling algorithm that schedules the system without timing constraints violated.

Definition 1.2 (optimality)**.** A scheduling algorithm is said *optimal* with respect to a system and a task model if any task sets that are feasible can also be scheduled by the scheduling algorithm.

1.2 MOTIVATION

Unlike *general-purpose computing systems* in which the design is optimized towards average-case performance, real-time systems must provide the predictable low-latency [SR90].

Testing is not Enough to Establish Safety

Vehicle level testing is a matter of great importance in designing bomb-proof systems. From calendar year 2005 to 2010 Toyota Motor Corp. reported approximately 11 million hours in module level software testing, and 35 million miles of system level testing, including unit testing, hardware-in-the-loop field testing, and the verification of normal functions of the vehicle engine control system, speed control, and vehicle driving handling [Nan]. One might wonder how come the 2005 Toyota Camry mishap resulting in a major catastrophe can still happen after such testing?

In fact, due to system complexity, despite a considerable time invested in exploring potential failures, it is still impracticable to test everything at the vehicle level: too many possible operating conditions, timing sequences, and possible fault types. Butler and Finelli, senior research engineers at the National Aeronautics and Space Administration (NASA) Langley Research Center, stated in [BF93]

> *Life-testing of ultrareliable software is infeasible.*

It would take more than 10^8 h ($\sim$11,415 years) of testing to quantify 10^{-8}/h failure rate. Since life-testing of ultrareliable software is infeasible, timing analysis has been developed for predictable real-time systems.

1.3 ORGANIZATION OF THIS THESIS

The remaining parts of this dissertation is organized as follows:

- In Chapter 2, we introduce a basic concept of timing analysis required for the understanding of this dissertation.
- Chapter 3 first provides an overview of the well-adopted sporadic task model. In this chapter, we study the problem of scheduling arbitrary-deadline real-time sporadic task sets on a multiprocessor system under global fixed-priority scheduling. Since this model is essentially restrictive, we introduce in the following chapters more expressive models that better characterize the behavior of real-time systems in timing analysis.
- In Chapter 4, we study the problem of scheduling real-time tasks on multicore platforms with shared resources. We assume that we have a multicore platform comprised of a set of identical cores connected by a shared resource, e.g., a communication fabric (bus) for accesses to a shared memory.

- In Chapter 5, we study the problem of real-time tasks with synchronizations on multiprocessor systems: no two concurrent accesses to one (logical) shared resource are in their critical sections at the same time.

- In Chapter 6, we study a model that is a generalization of the periodic task model, called *multi-mode* task model: a task has several modes specified with different execution times and periods to switch during runtime, independent of other tasks. Moreover, we also study the problem of allocating a set of multi-mode tasks on a homogeneous multiprocessor system.

1.4 CONTRIBUTIONS OF THIS WORK

The following overview lists the contribution of the author on the presented results for each chapter:

- In **Chapter** 3, there are two main contributions made here. First, it has been shown that the existing response time analysis in multiprocessor arbitrary-deadline systems is flawed: the response time may be larger than the derived bound. This paper provides a revised analysis resolving the problems with the original approach, and then propose a corresponding schedulability test. Secondly, we derive a linear-time upper bound on the response time of arbitrary-deadline tasks in multiprocessor systems. *To the best of our knowledge, this is the first work presenting a linear-time response time bound on global scheduled multiprocessor systems.*

- In **Chapter** 4, we develop a pseudo-polynomial-time schedulability test and response-time analysis for constrained-deadline sporadic tasks scheduled on a multicore platform connected by a shared resource. We also propose our task allocation algorithm. We show that this algorithm offers non-trivial quantitative guarantees, including a speedup factor of 7 in polynomial-time complexity. The experimental evaluation demonstrates that our approach can yield good acceptance ratios. *To the best of our knowledge, this is the first result that provides a speedup factor in the context of multicore platforms with shared resources.*

- In **Chapter** 5, we show that the proposed resource-oriented partitioned scheduling using Priority Ceiling Protocol (PCP) combined with a reasonable allocation algorithm can achieve a non-trivial speedup factor guarantee. Specifically, we prove that our task mapping and resource allocation algorithm has a speedup factor $11 - 6/(m + 1)$ on a platform comprising m processors, where a task may request at most one shared resource and the number of requests on any resource by any single job is at most one. Our empirical investigations show that the proposed algorithm is highly effective in terms of task sets deemed schedulable. *Again, to the best of our knowledge, this is the best result studying the problem of resource*

sharing on multiprocessors in terms of non-optimality when partitioned scheduling schemes are concerned.

- In **Chapter 6**, we propose a technique for analyzing schedulability of the system where tasks can undergo mode change under fixed-priority scheduling. We study two scheduling algorithms upon multi-mode task systems: *Fixed-Priority Task-level (FPT)* and *Fixed-Priority Mode-level (FPM)*. We further show that a utilization of $2 - \sqrt{2} \approx 0.5857$ can be guaranteed in implicit-deadline multi-mode uniprocessor systems if each mode is prioritized according to rate-monotonic policy. *To the best of our knowledge, this is the first work studying the mode change systems under FPM scheduling.* Moreover, we present a scheduling algorithm using any Reasonable Allocation Decreasing (RAD) algorithm for task allocations for scheduling multi-mode tasks on multiprocessor systems. We prove that this algorithm achieves 38% utilization for implicit-deadline Rate-Monotonic (RM) scheduled multi-mode tasks on multiprocessor systems. *To the best of our knowledge, this is the first result for multi-mode tasks on multiprocessor systems.*

2

TIMING ANALYSIS CONCEPTS

Traditionally, timing analysis consists of two separate steps: (*i*) WCET analysis, which computes an upper bound on the execution time of a single job of a task running in isolation and (*ii*) schedulability analysis, which determines whether multiple tasks are guaranteed to meet their deadlines, while sharing a processor. In this chapter, we introduce the WCET analysis in Section 2.1, and definitions and terminologies on real-time scheduling in Section 2.2. Terminologies and design issues towards real-time multiprocessor systems are introduced in Section 2.3. This chapter only serves as a basic introduction required for the understanding of this dissertation.

2.1 WORST-CASE EXECUTION TIME

First of all, the WCET of each tasks in real-time systems must be known in order to guarantee feasibility. In event-triggered or periodic systems it is required for schedulability analysis, in time-triggered systems (e.g. *Time-Triggered Protocol (TTP)* [KG93] and *FlexRay* [Con+05]) it is required for determining a static schedule.

Two classes of methods to analyze the WCET can be distinguished:

- *Measurement-based* approach: running real programs with sets of representative inputs to measure the longest program execution time.
- *Static analysis* approach: counting assembler instructions for each function, loop etc. and then combining them.

2.1.1 *Measurement-Based Approach*

With the availability of the hardware under development and the respective simulators, measurement-based approaches are the approach that has been widely put into practice in industry.

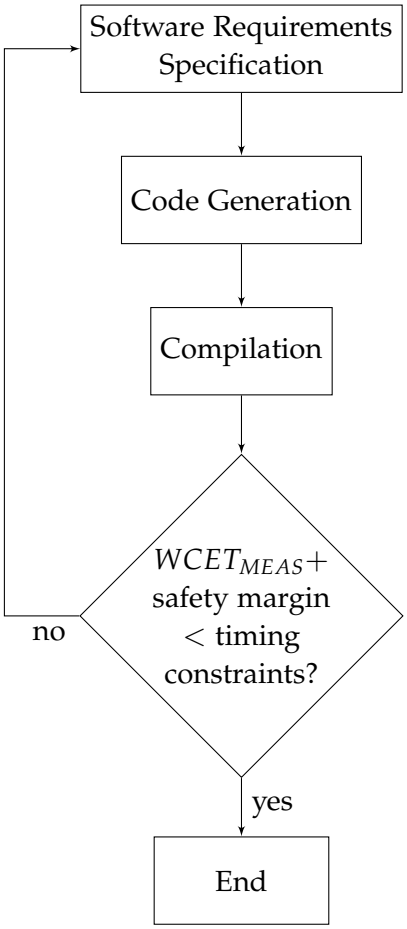


Figure 2.1: Trial-and-error based design process [LM11]

With the increasing complexity of software and hardware, there is a need for tool support in developing a complex system. Code generator tools like ASCET [ETA] are widely used to optimize for resource-limited systems and to provide high-performance with low overheads. The code generated by these tools in a high-level language with strong abstraction from the details of the system is later translated into machine code by a complier, which is utilized for WCET measurements. Then, the program runs with sets of representative inputs to measure the longest *end-to-end* program execution time of a task or task parts. After that, it is common practice in industry to use a *safety margin* for error to account for untested code, hardware performance approximations

or mistakes (cf. Figure 2.1). The accumulated value is used for the verification of timing constraints. In case of violated constraints, the designers have to revisit the requirements and specifications to fix the timing problem, starting the next system design cycle. The design requirements process ends when all the necessary requirements are met.

Issues of this approach

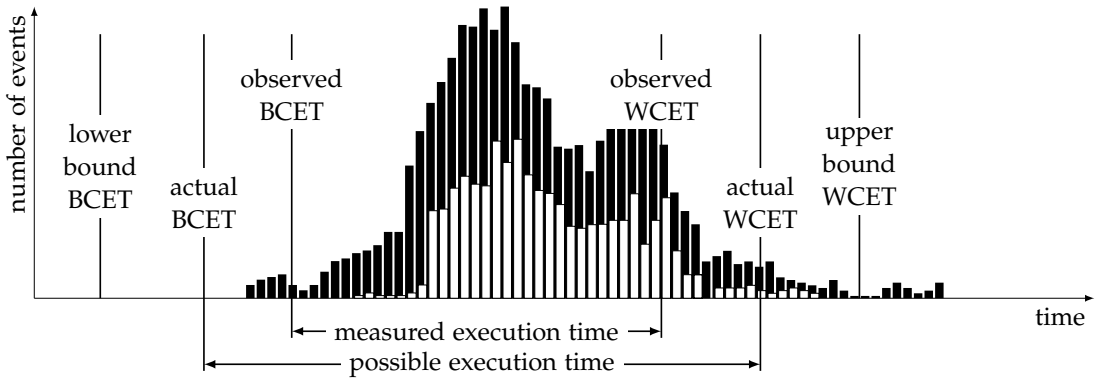


Figure 2.2: Distribution of execution time. The white bars represent the times of measured executions. The black bars represent the times of all possible executions.

This approach has the advantage that execution times are determined from real measurements, avoiding modeling errors; however, it has one major shortcoming: it is not *safe*. Those representative inputs possibly may not cover the worst case *input*. Figure 2.2 depicts the distribution of execution time under different approaches. The shortest execution time is called the Best-Case Execution Time (BCET), the longest time is called the WCET. Practically, it is impossible measuring for *all* inputs. As a result, the measurement-based approach in general will underestimate the actual WCET. Despite that a safety margin, typically of 20%, is often used for accounting for underestimate, there is very little justification for being safe; on the other hand, a large margin for error can result in under-utilization of a real-time system. In summary, there is little science and mostly ad hoc with this approach.

2.1.2 *Static Approach*

Static Approach emphasizes *safety* by producing bounds on the execution time, guaranteeing that the execution time will not exceed these bounds. The bounds allow safe schedulability analysis of hard real-time systems. Figure 2.3 depicts the workflow of most WCET analyzers such as aiT WCET Analyzers[1], the most well-known

1 https://www.absint.com/ait/

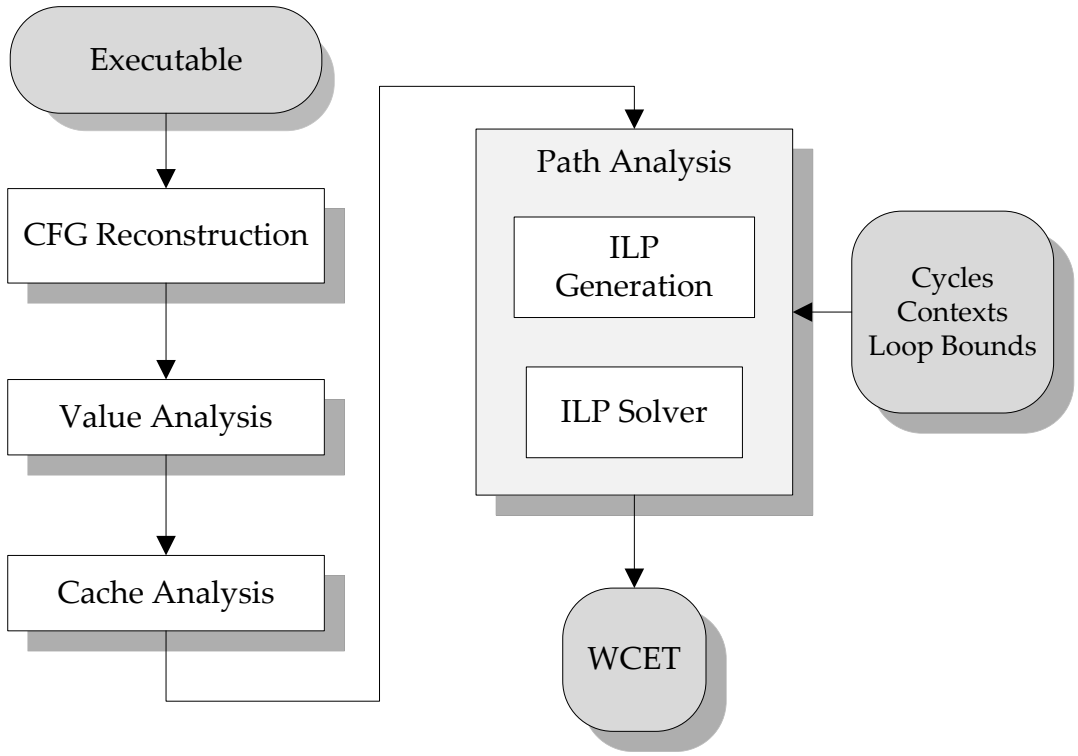


Figure 2.3: Workflow for most static WCET analyzers

and industrially used WCET analyzer. The workflow is composed of several different components, starting from reading a (binary) executable:

- *Control Flow Graph (CFG) reconstruction*: extracts an interprocedural CFG that reflects all possible transitions of control flow among instructions from the given executable.
- *Value analysis*: Knowledge about the memory addresses that are accessed by a memory reference is required in order to compute a precise bound on the WCET. Value analysis determines ranges (safe approximations) for values in registers by means of the annotated CFG served as the input. In some cases, there are addresses of a memory reference that may not be able to determine. Then the cache analysis must conservatively account for all possible addresses, which may be manually refined by the user later on.
- *Cache analysis*: cache analysis would be not needed if all access data could ideally be cached (*cache hit*). However, to be cost-effective and to enable efficient use of data, caches are relatively small, meaning that the requested data may not reside

in a cache but elsewhere (*cache miss*). On the other hand, the suppression of cache (assuming always cache misses) may result in a pessimistically large WCET owing to the high variance of delays: a cache miss may take 100 times longer than a cache hit. Cache analysis classifies the accesses to main memory, i.e. whether or not the needed date resides in the cache. The classification requires *may* and *must* information. *May* and *must* caches are upper and lower approximations, respectively, to the contents of all concrete caches that will occur whenever program execution reaches a program point [Rei09]. *Must* cache information is used to derive safe information about cache *hits*; *may* cache information is used to safely predict cache *misses*. In the context of preemptive scheduling, Cache-Related Preemption Delay (CRPD) must be taken into account, whereby the so-called Useful–Cache-Block (UCB) analysis is used [AB09; AB11].

- *Path analysis*: path analysis is also called *Bound Calculation* in the literature [Wil+08]. It finally computes the WCET result. Intuitively speaking, the upper bound of WCET for a task is determined by finding the overall feasible path with the longest execution time. To achieve this, it must know upper bounds on the execution frequency of all cyclic paths, i.e., loops and recursions. For simple cases, the these bounds are obtained by value analysis, whereas for complex loops the user has to provide a bound. There are three main classes of methods for bound calculation in literature: *structure-based* [PK89; CP00; CB02], *path-based* [HAMWH99], and *Implicit Path Enumeration Technique* (IPET) [LM95; LM97]. The de-facto standard one is IPET. The control flow of tasks is translated into Integer Linear Programming (ILP) by reflecting the structure of the task and possible flows. An upper bound is determined by maximizing the sum of products of the execution counts and times such that the execution count variables are subject to the constraints of bounds on loops and recursions.

2.2 REAL-TIME SCHEDULING

In this section, we introduce definitions and terminologies common to the whole spectrum of research on real-time scheduling.

2.2.1 *The Taxonomy of Scheduling Algorithms*

Scheduling algorithms for real-time systems can be classified according to when changes to priority can be made [Car+04]:

- *Fixed task priority*: each task is associated with an unique priority applied to all of its jobs. It is the scheduling algorithm most commonly used in real-time systems,

also known as *fixed-priority scheduling*[2]. With fixed task priority scheduling, the scheduler ensures that at any given time, the processor executes the highest priority task of all those tasks that are currently ready to execute.

- *Fixed job priority*: each job of a task has a single priority; however, different jobs of the same task may have different priorities. One example of this is Earliest-Deadline-First (EDF) scheduling: the task with the earliest absolute deadline is always executed first.
- *Dynamic priority*: Each job may have different priorities during runtime, for instance, Least-Laxity-First (LLF) scheduling: the task with the least laxity is always executed first.

There are also other commonly used techniques to achieve predictability, for example, *Time Division Multiple Access (TDMA)* [SCT10b]. With a TDMA system, each task in the system is assigned a particular time window, called *TDMA slot*, to use for execution.

Determining when control is transferred from one running task to another., we can identify the following two classes of scheduling algorithms:

- *Preemptive*: with preemptive scheduling, once a job/task has executed, it can be taken away from the CPU at any time, according to a scheduling policy.
- *Non-preemptive*: with non-preemptive scheduling, once a job/task has started to execute, it cannot be preempted until it finishes the execution. The scheduler is only invoked at the end of a job (assuming the interrupts are disabled).

In the context of preemptive scheduling [LL73], whereby EDF scheduling is shown an optimal scheduling, the cost of preemption is considered negligible, and therefore assumed to be zero. But the fact is that *context switching* is in general computationally intensive, typically in single-digit milliseconds [LDS07], depending on what precisely it entails and which architecture it is operated on. On the other hand, with non-preemptive scheduling, context switches are more predictable and much less frequent. However, it is known that non-preemptive scheduling suffers from long non-preemptible region blocking [Sho10], thereby resulting in longer response times. The question whether preemptive scheduling is better than non-preemptive one has been long debated in the real-time community. Recently, *limited preemption* scheduling as a viable alternative to non-preemptive and fully preemptive scheduling has been introduced, to reduce the runtime overhead due to preemptions while avoiding long non-preemptiable region blocking [BBY13]. In the context of limited preemption scheduling, several approaches have been proposed in the literature: (*i*) *preemption thresholds scheduling*: preemptions are allowed to take place if the priority of arriving tasks is over the specified *preemption threshold* [WS99]; (*ii*) *deferred preemptions scheduling*:

2 We interchangeably use fixed task priority and fixed-priority in this dissertation.

preemptions are maximally deferred by a given amount of time [Bar05]; and (*iii*) *fixed preemption points*: preemptions are allowed only at selected points of the code of each task [Bur93]. Please find a more in-depth review in [BBY13].

2.2.2 *Schedulability and Response-Time Analysis*

When developing multitasking real-time systems, schedulability tests using the WCET as input are used to formally prove that a given task set under a scheduling algorithm will meet its deadlines:

Definition 2.1 (schedulability test)**.** *Schedulability test* of a scheduling algorithm is to verify whether the task system is feasible under the given algorithm.

Logically speaking, there are three types of tests in schedulability analysis:

Definition 2.2 (sufficient test)**.** A schedulability test is said *sufficient*, with respect to a scheduling algorithm and a system if all of the task sets that are deemed schedulable according to the test are in fact schedulable.

Definition 2.3 (necessary test)**.** A schedulability test is said *necessary* if all of the task sets that are deemed unschedulable according to the test are in fact unschedulable.

Definition 2.4 (exact test)**.** A schedulability test that is both sufficient and necessary is referred to as *exact*.

An example of necessary tests with respect to any scheduling algorithms is Demand Bound Function (DBF):

Definition 2.5 (demand bound function)**.** For any interval length t, the DBF $dbf_i(t)$ of a sporadic task τ_i bounds the maximum cumulative execution requirement by jobs of τ_i that both arrive in, and have deadlines within, any interval of length t.

In other words, it is necessary for a task set being schedulable (by any scheduling policies) that the total jobs for each task by the DBF must be executed within any interval of length t. It has been shown in [BMR90] that the DBF of a sporadic task τ_i can be defined as

$$dbf_i(t) = \max\left(0, \left\lfloor \frac{t + (T_i - D_i)}{T_i} \right\rfloor C_i\right) \tag{2.1}$$

For any sporadic task uniprocessors system τ that is feasible it must be that $\forall t > 0$

$$\sum_{\tau_i \in \tau} dbf_i(t) \leq t \tag{2.2}$$

Necessary test is a necessary condition for all of the task sets being schedulable. So we cannot say that a system that passes a necessary test is schedulable. Despite the

fact that schedulability cannot be fulfilled by a necessary condition, this condition is however useful in quantifying the optimality of a schedulability test associated with a scheduling algorithm [LL73; BB08; DTGDP15; CL14; HCZL15; HC16a; BCDH16; HCR16; HYC16].

To verify whether or not a given task under a scheduling algorithm can meet it deadline, it suffices to consider only critical instants:

Definition 2.6 (critical instant)**.** A *critical instant* of a task is defined to be an (hypothetical) instant at which the task will have the longest response time.

Typically, such an instant is described as a point in time with particular properties. As an example, a critical instant for tasks under fixed-priority preemptive scheduling is given by a point in time for which all tasks have a simultaneous release [LL73].

Definition 2.7 (time-demand analysis)**.** *Time-Demand Analysis (TDA)* computes the total demand for processor time by a job released at a critical instant of the task and by all the higher-priority tasks, and checks whether this demand can be met before the deadline of the job.

TDA is typically applied to produce a schedulability test for fixed task priority scheduling that ensures that each job of every task completes before the next job of that task is released [LSD89].

An alternative approach for checking the feasibility of a system is to perform Response-Time Analysis (RTA), whereby the Worst-Case Response Time (WCRT) of each task is calculated to compare with its relative deadline.

Definition 2.8 (worst-case response time)**.** The *WCRT* of a task is defined as the largest response time of any of its jobs.

Definition 2.9 (response-time analysis)**.** *RTA* determines the worst-case response time of a task.

It is worth noting that TDA produces a boolean decision: feasibility or infeasibility, whereas RTA calculates a numerical value: the WCRT.

2.2.3 *Metrics for Quantifying Scheduling Algorithms*

Ideally, an exact test associated with an optimal scheduling algorithm is preferred. However, it is often the case that an optimal scheduling is unavailable and/or scheduling algorithms associated with schedulability tests are too expensive due to either the runtime overhead or the complexity of tests themselves. As a result, one settles for (pseudo)polynomial-time, near-optimal solutions, called *approximation algorithm*. Unlike heuristics, which usually find reasonably good solutions in the vast majority of cases, approximation algorithms ensure that solutions with provable quality are far from being optimal up to a constant factor, even for specific pathological inputs.

In mathematics, finding *approximation/competitive ratio* of an algorithm is arguably the most mathematically satisfying approach to pursue ideas other than optimal solutions. The approximation ratio (or approximation factor) is the ratio between the result obtained by the algorithm and the optimal cost or profit. Unlike approximation ratio, in the real-time system community, resource augmentation factors (in speed or processors) are preferably, widely used to quantify the quality of scheduling algorithms associated with the corresponding schedulability tests, first introduced in [KP95]. Specifically, the formal definition of speedup factor is as follows:

Definition 2.10. (speedup factor) A scheduling algorithm associated with some sufficient schedulability test $\mathcal{A}$ is said to have a speedup factor of α if for any task system the task system that is not deemed schedulable by the test is also not schedulable by any scheduling algorithms upon a platform in which each processor is $\frac{1}{\alpha}$ times as fast.

The smaller the processor speedup bound, the better the sufficient schedulability test. In addition, a processor speedup bound of one implies that the test is an exact one and that the scheduling algorithm is optimal.

Alternatively, another metric commonly used is utilization bounds:

Definition 2.11 (utilization bound)**.** A task set is schedulable if its total utilization is less than a specific bound.

Two classic examples of utilization bounds are EDF and RM scheduling algorithm, one example of fixed task priority scheduling. Roughly speaking, with periodic tasks that have deadlines equal to their periods, EDF scheduling has a utilization bound of 100%, whereas RM scheduling can meet all of the deadlines if CPU utilization is less than 69.32% [LL73]. Despite that EDF scheduling is known as an optimal scheduling algorithm for sporadic implicit-deadline tasks [LL73], it entails higher runtime overhead of implementations than RM due to dynamic priority changes during runtime. Therefore, with the quality guarantee, using RM scheduling algorithm would be a pragmatically good trade-off between performance and efficiency.

But the utilization bound metric fails to justify many systems and scheduling algorithms, such as non-preemptive scheduling, constrained-deadline systems, and so on: non-preemptive scheduling can trivially be unfeasible at a very small utilization, whereby the longest execution time of one task that is non-preemptible exceeds the shortest deadline of another [Sho10]. As a result, instead of using utilization bounds, one may have to refer to the processor speedup factor as an indicator to distinguish those non-preemptive scheduling algorithms [DTGDP15].

2.3 MULTIPROCESSOR SYSTEMS

Due to the issue of power consumption and excessive heat dissipation in increasing high processor clock speeds, there has been a move towards using multiprocessor platforms.

Such a move also takes place in real-time system domains due to the evolution and integration of features and consequently the need for more processing power. In the context of multiprocessor platforms, several dimensions are added on to real-time system designs.

2.3.1 *Multiprocessor Scheduling Algorithms*

To schedule real-time tasks on multiprocessor platforms, there are two widely adopted approaches: *partitioned* and *global* scheduling algorithms. In global scheduling, a global queue is implemented for all instances of tasks, namely jobs, and the jobs can be arbitrarily assigned to any processor, depending upon a global scheduling scheme. Hence, different jobs of the same task may execute upon different processors. In partitioned scheduling, contrary to global scheduling, each task is assigned statically to one processor and cannot migrate to the others. As a result, all jobs generated by a task only execute on the processor where the task is assigned.

In addition to the pure partitioned and global ones, *semi-partitioned* scheduling algorithms have recently received considerable attention. They combine characteristics from partitioned and global scheduling by allowing a small number of tasks or a proportion of jobs to migrate, thereby improving schedulability.

2.3.2 *Task-to-Core Mapping*

Partitioned scheduling policy is the scheduling policy supported by the AUTOSAR standard for automotive systems [Aut], as well as most commercial Real-Time Operating System (RTOS), including VxWorks, QNX, LynxOS, and all POSIX-compliant ones. Under partitioned scheduling policy the performance of a system is highly dependent of *task-to-core mapping*. The task allocation under partitioned scheduling is analogous to the well-known *bin packing problem* [GJ79]. In the bin packing problem, the objective is to pack a set of items into a number of bins such that the volume of the packed items in every bin does not exceed that of the bin and the number of used bins is minimized. As task allocation is a bin-packing-like problem, which is NP-hard [Joh73], rather than finding the optimal solution, the goal of optimization problems is usual to find a good solution quickly. For this reason, several *reasonable allocation algorithm* has been long developed for the bin-packing problem. Reasonable allocation algorithm is the algorithm that declares failures only if there is no bin having sufficient capacity to hold the pending item. Examples of reasonable allocation algorithm include:

- *First-Fit (FF)* algorithm: it places the item in the first bin that can accommodate the item. If no bin is found, it opens a new bin and puts the item to the new bin. Every time it starts searching for the first bin.

- *Next-Fit (NF)* algorithm: Same as FF, except every time it starts searching for the next bin from the last bin it allots rather than the first one.
- *Best-Fit (BF)* algorithm: it places each item to the bin with the *lowest* remaining capacity among all the bins with sufficient capacity to accommodate the item.
- *Worst-Fit (WF)* algorithm: it places each item to the bin with the *largest* remaining capacity among all the bins with sufficient capacity to accommodate the item.

3

SPORADIC TASK MODEL

In this chapter, we study the problem of scheduling arbitrary-deadline real-time sporadic task sets on a multiprocessor system under *global fixed-priority* scheduling. Two contributions are made here. First, it has been shown that the existing response time analysis in arbitrary-deadline systems is flawed: the response time may be larger than the derived bound. This dissertation provides a revised analysis resolving the problems with the original approach, and then propose a corresponding schedulability test. Secondly, we derive a linear-time upper bound on the response time of arbitrary-deadline tasks in multiprocessor systems. *To the best of our knowledge, this is the first work presenting a linear-time response time bound on arbitrary-deadline multiprocessors.*

The presentation is organized as follows: We first review some related work in Chapter 3.1, where the state-of-the-art analysis is shown to be flawed. Some preliminary results and definitions are provided in Chapter 3.2. In Chapter 3.3, we propose a revisited response time analysis by using a proper annotation of the workload function. In Chapter 3.4, we derive the first known linear-time response time bound on arbitrary-deadline multiprocessor systems. Finally, we evaluate our results in Section 3.5 by comparing to the state-of-the-art results for both constrained-deadline systems [BCL05b; BBMS10; Bak06; GSYY09] and arbitrary-deadline systems [Bak06; BF08].

3.1 RELATED WORK

The origin of using this model dates back to the seminal work by Liu and Layland [LL73]. For uniprocessor fixed-priority scheduling in arbitrary-deadline task systems, the exact schedulability tests and the (tight) worst-case response time using *time-demand analysis* (TDA) are proposed by Lehoczky [Leh90; TBW94]. Such results may suffer from their high time-complexity because of the explosion of *busy interval* in arbitrary-deadline systems. From the designers' perspective, a fast test is more applicable during design space exploration, even at a price of accuracy. Several approaches have been proposed for reducing the computational complexity of thes TDA [SH98; BB04]. Lehoczky proposes a utilization upper bound for a set of periodic arbitrary-deadline tasks under fixed-priority scheduling [Leh90]. Bini proposes a quadratic bound by considering some information about the task periods [Bin15]. The linear-time response-time bound for fixed-priority systems has been first proposed by Davis and Burns [DB08]. This bound has been later improved by Bini et al. [BNRB09].

To schedule real-time tasks on multiprocessor platforms, there are three widely adopted paradigms: partitioned, global, and semi-partitioned scheduling[DB11a]. In this chapter, we consider *global scheduling*, in which a job can migrate from one processor to another processor during its executions. But, a job cannot be simultaneously executed on more than one processor.

Unfortunately, deriving exact schedulability tests under multiprocessor global scheduling is extremely harder than in uniprocessor due to the non-existence of *critical instant*. The first brute force approach regarding the exact response time is proposed by Baker and Cirinei [BC07a]. The analysis consists in solving *reachability* problem in a finite state machine that represents all possible combinations of arrival times and execution sequences for a task set. In addition, some results are recently proposed to reduce the number of states to be analyzed in the reachability analysis [GGL13; SL14]. Generally speaking, these approaches are so complex that only task sets with very small integer periods can be tractably solved. Alternatively, most of the research in the literature focus on finding approximate upper bounds to the response time. The response time analysis for implicit-deadline sporadic task systems has been studied by Andersson et al. [ABJ01]. The fixed-priority scheduling of constrained-deadline sporadic task systems has been studied in the literature [Bak03; BCL05b; BBMS10].

Regarding arbitrary-deadline task systems, several results have been proposed in the literature [Bak06; BF08; GSYY09; SLGY14]. Baker [Bak06] designs a test based on some deep insights, in that if a task τ_k misses its deadline, then the load in the analyzed interval, called *problem window*, must satisfy some necessary conditions on the parameters of all the tasks. Baruah and Fisher [BF08] use another annotation to extend the analysis window and derive a corresponding pseudo-polynomial-time schedulability test. The first worst-case response-time analysis for arbitrary-deadline task systems is proposed by Guan et al. [GSYY09], in which the authors use the insight

proposed by Baruah [Bar07] to limit the number of carry-in tasks, and then apply the workload function proposed by Bertogna et al. [BCL05b] to quantify the requested demand of higher-priority tasks. Unfortunately, it has recently been shown in [SLGY14] that the analysis by Guan et al. [GSYY09] is optimistic for arbitrary-deadline tasks. The assumption that each carry-in task has only one carry-in job in [GSYY09] is in fact problematic, as a carry-in task can be shown to carry more than one carry-in job [SLGY14]. Sun et al. [SLGY14] derive a complex carry-in workload function for the response time analysis where all possible combinations of carry-in and non-carry-in functions have to be explicitly enumerated. However, such a method is computationally intractable since the time complexity is exponential.

3.2 PRELIMINARY RESULTS AND DEFINITIONS

This section reviews some results in the literature with respect to the worst-case response-time analysis. We will first explain the level-k busy interval (period) concept by Lehoczky [Leh90] in uniprocessor systems and extend the concept for multiprocessor systems. As our analysis also exploits the workload function, as defined in [BCL05a; BNRB09], we also provide its definition at the end of this section.

3.2.1 *Level-k Busy Interval for Task τ_k*

For the special case with uniprocessor systems when $m = 1$, the response-time analysis for sporadic arbitrary-deadline task systems under fixed-priority scheduling has been developed in [Leh90]. The analysis utilizes a *level-k busy interval* concept to evaluate the worst-case response time. For such a case, we release all the higher-priority tasks in $hp(k)$ together with task τ_k at time 0 and all the subsequent jobs are released as early as possible by respecting to the minimum inter-arrival time. The level-k busy interval finishes when a job of task τ_k finishes before the next release of a job of task τ_k.

For the h-th job of task τ_k in the level-k busy interval, the finishing time $R_{k,h}$ is the minimum t such that

$$hC_k + \sum_{\tau_i \in hp(k)} \left\lceil \frac{t}{T_i} \right\rceil C_i \leq t,$$

and, hence, its response time is $R_{k,h} - (h-1)T_k$. The level-k busy interval of task τ_k finishes at the h-th job if $R_{k,h} \leq hT_k$.

The above analysis works on one processor. However, it requires quite some annotations for the general cases when $m \geq 2$. There are some difficulties for multiprocessor systems. In general, we are unaware of the worst-case release pattern of the higher-priority tasks in $hp(k)$ when $m \geq 2$. The level-k busy interval of task τ_k may have different arrival patterns to create the worst-case response time for different hs.

As a result, all possible cases with some unfinished jobs before the arrival of a job have to be considered!

Towards this, we need more precise definitions to capture all the busy intervals for task τ_k and use them to safely bound the worst-case response time. The first definition defines whether a task is busy at time θ in a legal schedule.

Definition 3.1 (Task Busy)**.** A task is said busy at time instant θ in a legal schedule if at least one job of the task that arrives earlier than θ has not yet completed its execution at time instant θ.

Note that, by the definition, even if a task is busy at time θ, it may not be executed at time θ. From Definition 3.2 to Definition 3.3, we will look at a specific time interval starting from time θ_a in a legal schedule of a legal sequence of jobs of τ.

Definition 3.2 (A Level-k Busy Interval)**.** A level-k busy interval of task τ_k in a legal schedule is an interval $[\theta_a, \theta_a + t)$ of time in which, in the legal schedule, (1) task τ_k is busy at all the time points in the interval, and (2) task τ_k is not busy right prior to θ_a.

Definition 3.3 (An h-Incomplete Busy Interval)**.** A level-k busy interval $[\theta_a, \theta_a + t)$ in a legal schedule is an h-incomplete busy interval if the hth-job of task τ_k released in the busy interval $[\theta_a, \theta_a + t)$ has not completed yet at time $\theta_a + t$.

The above definitions are general forms of the level-k busy interval in [Leh90], i.e., θ_a is 0 for the case when m is 1. If θ_a is given, the maximum h-incomplete busy interval $[\theta_a, \theta_a + t^*)$ (under the assumption to start from θ_a) in a legal schedule happens when task τ_k is not busy at time $\theta_a + t^*$ or task τ_k is $(h+1)$-incomplete at time $\theta_a + t^*$. Moreover, according to the above definition, for a given θ_a as a fixed left point in the busy intervals, an $(h+1)$-incomplete busy interval (if it exists) completely covers an h-incomplete busy interval.

Definition 3.4 (Maximum h-Incomplete Length)**.** The maximum h-incomplete length $R_{k,h}$ is the maximum interval length among all the h-incomplete busy intervals in the legal schedules of all the legal sequences of the jobs of task system τ.

According to the above definitions, we can reach the following lemma regarding the worst-case response time $RT_{k,h}$ of the hth-job released in a level-k busy interval.

Lemma 3.1. *Suppose that $R_{k,h}$ is known. The worst-case response time $RT_{k,h}$ of the hth-job released in an h-incomplete busy interval is at most $R_{k,h} - (h-1)T_k$, i.e.,*

$$RT_{k,h} \leq R_{k,h} - (h-1)T_k$$

Proof. According to Definition 3.4, we know that the length of any h-incomplete busy interval of task τ_k is at most $R_{k,h}$. For an h-incomplete busy interval, starting at time θ_a, this implies that the finishing time of the h-th job released no earlier than θ_a is at most $\theta_a + R_{k,h}$. In a legal sequence of jobs of task system τ, we also know that the arrival time of the h-th job released no earlier than θ_a is at least $\theta_a + (h-1)T_k$. Therefore, we reach the conclusion. □

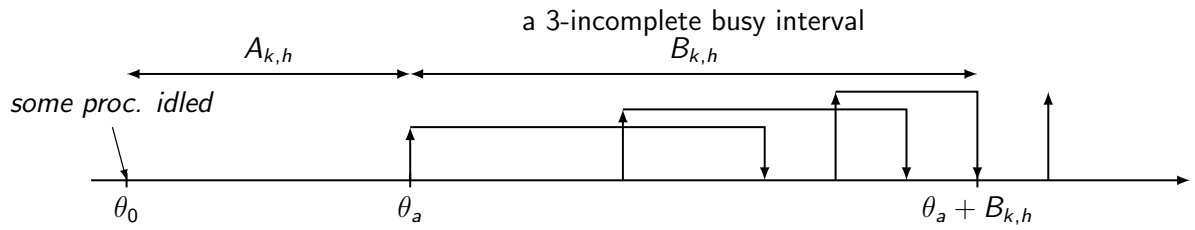


Figure 3.1: An example of an h-incomplete busy interval and the downward extension to θ_0.

Based on Lemma 3.1, we can now obtain an upper bound on the worst-case response time of task τ_k by evaluating all possible $RT_{k,h}$:

$$RT_k \leq max_{h=1,2,...}\{RT_{k,h}\}$$

3.2.2 *Workload Functions*

The concept of *workload function* has been widely used in real-time schedulability analysis [BCL05a; BNRB09]. Briefly, for a given interval length t, the *workload function* of task τ_i is defined to be the largest accumulative execution time of the legal sequence of the jobs of task τ_i over an interval of length t that could be *legally* executed within the interval. We provide the formal definition as follows:

Definition 3.5 (Workload Function [BCL05a; BNRB09])**.** For any interval length t, the workload function $W_i(t)$ of a sporadic task τ_i bounds the maximum cumulative execution requirement by jobs of τ_i that are released and may execute within any interval of length t.

$$W_i(t) = \left\lfloor \frac{t}{T_i} \right\rfloor C_i + min(t \ mod \ T_i, C_i) \tag{3.1}$$

It has been shown in [BNRB09] that the workload function is upper-bounded by the following linear function:

$$W_i(t) \leq U_i t + C_i(1 - U_i) \tag{3.2}$$

3.3 RESPONSE TIME ANALYSIS

Based on the observations in Section 3.2, we will first analyze a safe upper bound of $R_{k,h}$ for a given h. Then, the worst-case response time of task τ_k can be obtained by evaluating all possible values of h. Although the analysis by Guan et al. [GSYY09] is optimistic for arbitrary-deadline task systems, some observations can still be applied. Throughout the section, we will implicitly only analyze a specific task τ_k under the

assumption that the worst-case response time $RT_i \leq D_i$ of every higher-priority task τ_i in $hp(k)$ has been already analyzed.

3.3.1 *Safe Bound of $R_{k,h}$ for Given h*

We will first shortly review the analysis by Guan et al. [GSYY09] and then explain how we plan to conquer the problem to safely derive $R_{k,h}$. Suppose that an h-incomplete busy interval of task τ_k begins at time-instant θ_a. Suppose that this h-incomplete busy interval finishes at time $\theta_a + B_{k,h}$. That is, at time $\theta_a + B_{k,h}$, the hth-job in this h-incomplete busy interval finishes. Throughout this section, we only focus on this interval $[\theta_a, \theta_a + B_{k,h})$ under a legal schedule of a legal sequence of jobs of task system τ.

We first remove some unnecessary jobs in the legal sequence of jobs of task system τ. From this legal sequence of jobs, we first discard all those jobs that have priority lower than task τ_k. Since those jobs with priority lower than τ_k have no effect on the scheduling of the jobs generated by $hp(k)$ and task τ_k, this removal does not change the scheduling for tasks τ_k and $hp(k)$ in the interval $[\theta_a, \theta_a + B_{k,h})$. Moreover, we also remove the jobs of task τ_k that are released strictly before θ_a. Similarly, due to the definition of the h-incomplete busy interval, these jobs also have no influence on the scheduling of the jobs generated by $hp(k)$ and task τ_k in the interval $[\theta_a, \theta_a + B_{k,h})$. For the rest of our analysis in this subsection, we will consider only the above *reduced* legal sequence of jobs.

3.3.1.1 *Extending the h-Incomplete Busy Interval*

The technique for extending the h-incomplete busy interval is needed for precisely bounding the amount of work resulting from higher-priority tasks τ_i in $hp(k)$. Here, we will use the techniques proposed by Baruah [Bar07] and also adopted by Guan et al. [GSYY09]. Let θ_0 denote the latest time-instant $\leq \theta_a$ at which at least one processor is idle in the legal schedule of the reduced legal sequence of jobs. For notational brevity, let $A_{k,h} = \theta_a - \theta_0$.

Example: Figure 3.1 illustrates a 3-incomplete busy interval $[\theta_a, \theta_a + B_{k,h})$ and a downward extension of the interval to time θ_0. Task τ_k is not *busy* prior to the arrival of the first of these 3 jobs, the first job completes its execution after the second job arrives, and the second job completes its execution after the third job arrives. Thus, the task is continuously busy after the arrival of the first job shown, and θ_a is hence set equal to the arrival time of this job.

We then have the following observations:

- θ_a is the arrival time of some job of task τ_k.
- Over $[\theta_0, \theta_a)$, all the processor must be busily executing jobs having priority higher than τ_k.

The shifting of task τ_k to θ_0. Whenever we shift the release time of the job arriving at θ_a to θ_0, this job cannot be executed over $[\theta_0, \theta_a)$, and this task that is being h-incomplete at $\theta_a + B_{k,h}$ previously is still being h-incomplete at $\theta_a + B_{k,h}$ afterwards. In other words, we can consider the more pessimistic case where the release time of the job arriving at θ_a is shifted such that $\theta'_a = \theta_0$, without decreasing the response time of task τ_k. Meanwhile, the amount of work executing prior to θ_0 can still be more precisely quantified than that without extending the busy interval. Thus, we assume $\theta_a = \theta_0$ in the rest of this paper.

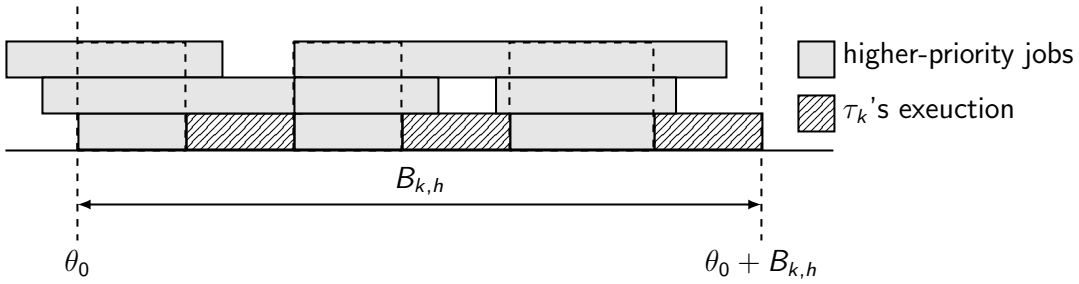


Figure 3.2: All processors are executing other higher-priority tasks whenever τ_k is not executing. Let $\theta_a = \theta_0$.

Another insight on multiprocessors is the lower bound on the workload of the higher-priority tasks in $hp(k)$ in the h-incomplete busy interval, as used in most of analysis in the literature [Bako6; Baro7; GSYY09]: Due to the h-incomplete busy interval of task τ_k in the interval $[\theta_0, \theta_0 + B_{k,h})$ and the definition of θ_0, for any $0 < t < B_{k,h}$, the sum of the length in interval $[\theta_0, \theta_0 + t)$ in which τ_k does not execute must be strictly larger than $t - hC_k$. The situation is illustrated for $m = 3$ processors in Figure 3.2. The striped rectangles indicate the execution of task τ_k during the h-incomplete busy period. The dashed rectangle indicates the intervals in which all m processors execute higher-priority jobs.

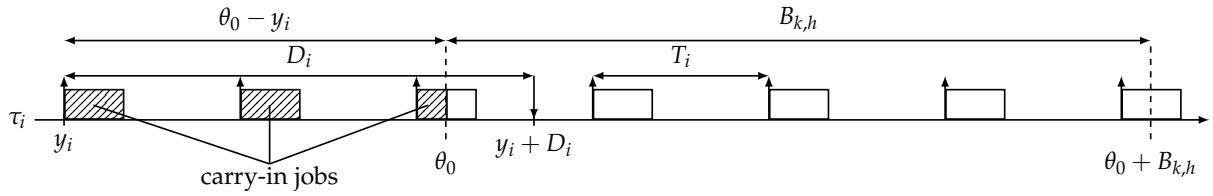


Figure 3.3: An illustration of releases of a higher-priority task τ_i over $[y_i, \theta_0 + B_{k,h})$. The release time of the first job of τ_i released before θ_0 and having an absolute deadline (or worst-case completion time) after θ_0 is denoted as y_i.

Let us denote by Γ_k a collection of intervals, not necessarily contiguous, of cumulative length $B_{k,h} - hC_k$ over $[\theta_0, \theta_0 + B_{k,h})$, during which all m processors are executing jobs other than τ_k's jobs in the legal schedule.

Definition 3.6 (Higher-Priority Interference)**.** We denote $\Omega(t)$ as the maximum total amount of execution times of the higher-priority interference from the tasks in $hp(k)$ in the time interval $[\theta_0, \theta_0 + t)$.

From the above observations, an h-incomplete busy interval $[\theta_0, \theta_0 + B_{k,h})$ of task τ_k requires that

$$\forall 0 < t < B_{k,h}, \quad \Omega(t) > m \times (t - hC_k) \tag{3.3}$$

Therefore, we can reach the following lemma.

Lemma 3.2. *For a given h, the maximum h-incomplete length $R_{k,h}$ is upper bounded by the minimum value $B_{k,h}$ satisfying the following equation*

$$\Omega(B_{k,h}) \leq m \times (B_{k,h} - hC_k) \tag{3.4}$$

Note that Lemma 2 can be thought of as a generalized result of Theorem 2 in [Baro7], where $h = 1$ and $B_{k,h} = D_k$.

3.3.1.2 *Calculating Higher-Priority Interference*

The treatment up to here is identical to the analysis by Guan et al. in [GSYY09]. As long as we can safely bound $\Omega(B_{k,h})$ for an h-incomplete busy interval of task τ_k defined above, we can use Lemma 3.2 to get $R_{k,h}$. However, calculating $\Omega(B_{k,h})$ is not trivial. The derivation in [GSYY09] was not safe enough.

To derive the contribution of task τ_i in $hp(k)$ to the interference $\Omega(B_{k,h})$, the contribution from higher-priority task τ_i can be considered by two parts in the interval $[\theta_0, \theta_0 + B_{k,h})$ (see Figure 3.3): (*i*) carry-in jobs: the jobs (portion) from task τ_i that arrive prior to θ_0 and have not yet completed their execution by θ_0, and (*ii*) body jobs: the jobs of τ_i that have the release time within the interval $[\theta_0, \theta_0 + B_{k,h})$.

By the definition of θ_0, at most $(m - 1)$ tasks are with carry-in jobs at time instant θ_0. Consequently, Lemma 3.3 follows immediately:

Lemma 3.3 (Guan et al. [GSYY09])**.** *There are at most $(m - 1)$ tasks having carry-in jobs in the interval $[\theta_0, \theta_0 + B_{k,h}]$.*

A higher-priority task τ_i in $hp(k)$ can be considered as either with or without carry-in jobs. We now introduce some notations by considering two cases for the amount of execution time executed for a higher-priority task τ_i in the time interval $[\theta_0, \theta_0 + B_{k,h}]$ in the legal schedule:

1. the maximum work done by task τ_i without carry-in jobs over $[\theta_0, \theta_0 + B_{k,h})$ is denoted by $I_i^1(B_{k,h})$, and
2. the maximum work done by task τ_i with carry-in jobs[1] over $[\theta_0, \theta_0 + B_{k,h})$ is denoted by $I_i^2(B_{k,h})$.

Then we explain how to compute a safe interference upper bound on these two functions $I_i^1(B_{k,h})$ and $I_i^2(B_{k,h})$, provided that h and $B_{k,h}$ are given.

Computing $I_i^1(B_{k,h})$. First, if a task τ_i contributes no carry-in jobs, then its contribution to the work over $[\theta_0, \theta_0 + B_{k,h})$ must arrive no earlier than θ_0 and no later than $\theta_0 + B_{k,h}$, and the total amount of work is bounded from above by the workload function with an interval of length $B_{k,h}$. Consequently, I_i^1 can be sufficiently computed by $W_i(B_{k,h})$, as defined in Section 3.2.2.

Secondly, the total amount of executed work cannot exceed the total length of the interval in Γ_k. The reason behind this is that any work that exceeds the total length of the interval in Γ_k cannot be *parallelly* executed with the other work from τ_i that have contributed to Γ_k, in which the total interference by all tasks busily executes on m processors. We formally state this with the following lemma:

Lemma 3.4. *The contribution to Γ_k from any task is at most $B_{k,h} - hC_k + 1$.*

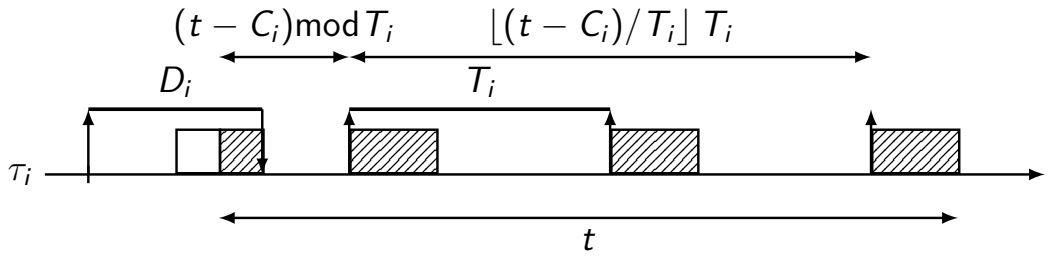
Since both are safe upper bounds, we can safely choose the minimum one between them as the maximum interference. Notice that the total length of Γ_k is equal to $(B_{k,h} - hC_k)$ and "+1" models the minimum interference that prevents τ_k from executing in the integer time-domain [BC07b; GSYY09]. Consequently,

$$I_i^1(B_{k,h}) \stackrel{\text{def}}{=} min\left(W_i(B_{k,h}), max(0, B_{k,h} - hC_k + 1)\right) \tag{3.5}$$

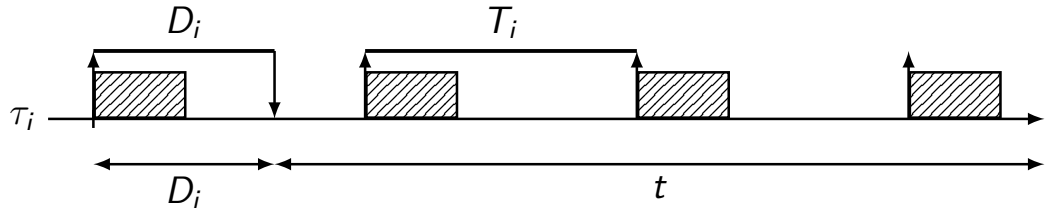
Computing $I_i^2(B_{k,h})$. Now, consider the case that a higher-priority task τ_i contributes carry-in jobs over $[\theta_0, \theta_0 + B_{k,h})$. Let y_i denote the arrival time of the first job of τ_i released before θ_0 and having an absolute deadline (or worst-case completion time) after θ_0. Under the assumption that a higher-priority task τ_i has worst-case response time $RT_i \leq D_i$, the work of task τ_i executed in the interval $[\theta_0, \theta_0 + B_{k,h})$ must arrive no earlier than y_i and no later than $\theta_0 + B_{k,h}$. Consequently, the total amount of work $I_i^2(B_{k,h})$ is bounded from above by the workload function with an interval length of $\theta_0 - y_i + B_{k,h}$ (see Figure 3.3). Since $\theta_0 - y_i \leq RT_i \leq D_i$, consequently, $I_i^2(B_{k,h})$ can be sufficiently computed by $W_i(D_i + B_{k,h})$. Together with the bounded length of the interval in Γ_k, we therefore have that

$$I_i^2(B_{k,h}) \stackrel{\text{def}}{=} min\left(W_i(D_i + B_{k,h}), max(0, B_{k,h} - hC_k + 1)\right) \tag{3.6}$$

1 The assumption by [GSYY09] that at most one carry-in job of a task contributes to the interval $[\theta_0, \theta_0 + B_{k,h})$ is optimistic, as shown in [SLGY14].



(a) The workload function in [BC07b; GSYY09]



(b) Our workload function

Figure 3.4: Comparison between the workload function in [BC07b; GSYY09] and ours for a carry-in task, assumed $R_i = D_i$

Moreover, it is not difficult to see that given an upper bound on the worst-case response time RT_i, the workload function for carry-in task τ_i can be more precisely bounded from above by $W_i(RT_i + B_{k,h})$. Hence, a tighter $I_i^2(B_{k,h})$ can be computed as follows:

$$\hat{I}_i^2(B_{k,h}) \stackrel{\text{def}}{=} min\left(W_i(RT_i + B_{k,h}), max(0, B_{k,h} - hC_k + 1)\right) \tag{3.7}$$

For simplifying the analysis, we only use $I_i^2(B_{k,h})$ as the maximum work of a higher-priority task τ_i with carry-in jobs in the following presentation.

From the discussion above, we have seen how to sufficiently quantify the interference from these two types of higher-priority tasks. By Lemma 3.3, there are at most $(m-1)$ tasks with carry-in jobs at time instant θ_0. Hence, there are at most $(m-1)$ tasks contributing to $I_i^2(B_{k,h})$, and the remaining $(k-m)$ tasks in $hp(k)$ must contribute to $I_i^1(B_{k,h})$.

Computing $\Omega(B_{k,h})$. With the above settings of $I_i^1(B_{k,h})$ and $I_i^2(B_{k,h})$, we can now compute $\Omega(B_{k,h})$. Let us denote by $I_i^{DIFF}(B_{k,h})$ the difference between $I_i^2(B_{k,h})$ and $I_i^1(B_{k,h})$:

$$I_i^{DIFF}(B_{k,h}) \stackrel{\text{def}}{=} I_i^2(B_{k,h}) - I_i^1(B_{k,h})$$

Therefore, by Lemma 3.3, we know that

$$\Omega(B_{k,h}) \stackrel{\text{def}}{=} \sum_{\tau_i \in hp(k)} I_i^1(B_{k,h}) + \sum_{\substack{\text{the } (m-1) \text{ largest} \\ \tau_i \in hp(k)}} I_i^{DIFF}(B_{k,h}) \tag{3.8}$$

where $hp(k)$ denotes the *set* of tasks that are assigned higher priority than τ_k. Note that Eq. (3.8) can be computed in linear time by using linear-time selection [BFPRT73] when $B_{k,h}$ is given.

3.3.2 *Worst-Case Response Time of Task τ_k*

Once having the upper bound on the interferences from higher-priority tasks $\Omega(B_{k,h})$, we can compute the maximum h-incomplete length $R_{k,h}$ by using Time Demand Analysis.

Compute $R_{k,h}$ using TDA. By Lemma 3.2 and the derived $\Omega(B_{k,h})$, $R_{k,h}$ is upper bounded by the smallest t satisfying the following inequality:

$$\Omega(t) \leq m \times (t - hC_k) \tag{3.9}$$

By the above analysis, we can now derive the worst-case response time of task τ_k by the following theorem.

Theorem 3.7. *The worst-case response time RT_k of task τ_k is at most:*

$$RT_k \leq RT_k^* = max_{h \in \{1,\ldots,H\}} \{R_{k,h}^* - (h-1)T_k\}$$

where $H = min\{h \geq 1 | \frac{\Omega(hT_k)}{m} + hC_k \leq hT_k\}$, and $R_{k,h}^*$ is defined as the minimum t satisfying Eq. (3.9).

Proof. Roughly speaking, H represents the maximum h to have h-incomplete busy intervals for task τ_k. Therefore, by Lemmas 3.1 and 3.2, we reach the conclusion. □

Even though H may become infinite, the TDA can be terminated whenever a deadline miss occurs:

$$\frac{\Omega((h-1)T_k + D_k)}{m} + hC_k > (h-1)T_k + D_k \tag{3.10}$$

We than state the schedulability test in the following corollary. In the later section, we will discuss how to upper bound such an H to be tested.

Corollary 3.1 (TDA). *An arbitrary-deadline sporadic task system τ is schedulable on m identical processors under fixed-priority scheduling if for all tasks $\tau_k \in \tau$,*

$$RT_k^* \leq D_k \tag{3.11}$$

where RT_k^* is as defined in Theorem 3.7.

Comparison between carry-in workload functions.

The carry-in workload function derived in [BC07b; GSYY09] shows that the worst-case scenario in constrained-deadline systems occurs when some job of τ_i executing C_i units precedes the end of a contiguous interval of length t and the earliest job having absolute deadline within the interval finishes its execution at the end of its deadline (also see Figure 3.4a). This scenario may accord with the work executing in the schedule as long as the worst-case response time is equal to its relative deadline. On the other hand, our carry-in workload function overly counts the carry-in job that is assumed to arrive D_i time-units prior to t (also see Figure 3.4b), even if such a job cannot effectively execute over the interval. Hence, our carry-in workload function for constrained-deadline tasks is inferior to that in [BC07b; GSYY09]: at most C_i may be overestimated. Nevertheless, empirical results show that the difference is insignificant in terms of schedulability, as will be seen later. On the other hand, the assumption that there is at most one carry-in job of a carry-in arbitrary-deadline task is problematic for the carry-in workload function derived in [BC07b; GSYY09]. The corresponding counterexample can be found in [SLGY14]. Hence, the revised workload function is the first one proved correct for arbitrary-deadline tasks.

3.3.3 *Improved Carry-in Workload Function*

A similar concept from [Bar07] can be conceived here while considering multiple carry-in jobs. Some job finishes its execution right before the end of the interval of interest. From this job on, all its previous jobs are reversely released as soon as possible, as shown in Figure 3.5. Suppose the job released at time y_i is the first job having absolute deadline within the interval of our interest. To respect timing constraints, this job cannot start its execution later than $y_i + D_i - C_i$. The amount of execution time from such a job satisfying the release pattern is maximally $\min(C_i, \max(0, (t - C_i) \bmod T_i - (T_i - D_i \bmod T_i)))$. In addition, there are exactly $(\left\lfloor \frac{t-C_i}{T_i} \right\rfloor + 1)$ complete jobs from body and $\left\lfloor \frac{D_i}{T_i} \right\rfloor$ complete jobs from carry-ins, yielding the following expression for $W_i'(t)$:

$$W_i'(t) = \left(\left\lfloor \frac{t - C_i}{T_i} \right\rfloor + \left\lfloor \frac{D_i}{T_i} \right\rfloor + 1 \right) C_i + \min(C_i, \max(0, (t - C_i) \bmod T_i - (T_i - D_i \bmod T_i))) \tag{3.12}$$

3.4 LINEAR-TIME UPPER BOUND UNDER FP

In this section we present a linear-time response time analysis. This test determines the upper bound on the response time under global fixed-priority scheduling upon the

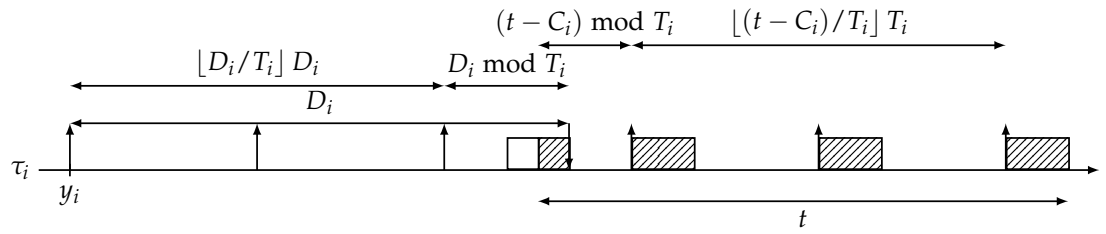


Figure 3.5: Improved carry-in workload function, aligning the very end of the interval with the end of execution of some job and completing the earliest job having absolute deadline within the interval, just in time.

values of the tasks' parameters, i.e., the worst-case execution time, the relative deadline, and the period.

We first linearize the upper interference function $\Omega(t)$ by using Eq. (3.2) in the following lemma:

Lemma 3.5. *For all* $t > 0$

$$\Omega(t) \leq \left(Z_\Sigma + \sum_{\tau_i \in hp(k)} (tU_i + C_i(1 - U_i)) \right) \tag{3.13}$$

where Z_Σ *denotes the sum of the* $(m-1)$ *largest* $U_i D_i$*'s among the tasks in* $hp(k)$*.*

Proof. Let S denote the *set* of the $(m-1)$ largest $I_i^{DIFF}(t)$.

$$\begin{aligned}
\Omega(t) &= \sum_{\tau_i \in hp(k)} I_i^1(t) + \sum_{\text{the } (m-1) \text{ largest}} I_i^{DIFF}(t) \\
&= \sum_{\tau_i \in hp(k) \setminus S} I_i^1(t) + \sum_{\tau_i \in S} I_i^2(t) \\
&\overset{(3.5) \text{ and } (3.6)}{\leq} \sum_{\tau_i \in hp(k) \setminus S} W_i(t) + \sum_{\tau_i \in S} W_i(t + D_i) \\
&\overset{(3.2)}{\leq} Z_\Sigma + \sum_{\tau_i \in hp(k)} (tU_i + C_i(1 - U_i))
\end{aligned}$$

□

Lemma 3.6.

$$R_k \leq R_k^\dagger := \frac{mC_k + Z_\Sigma + \sum_{\tau_i \in hp(k)} C_i(1 - U_i)}{m - mU_k - \sum_{\tau_i \in hp(k)} U_i} \tag{3.14}$$

where Z_Σ *denotes the sum of the* $(m-1)$ *largest* $D_i U_i$*'s.*

Proof. It is necessary for task τ_k being busy over an interval of length t:

$$\begin{aligned} & mW_k(t) + \Omega(t) > mt \\ \equiv & m \left\lceil \frac{t}{T_k} \right\rceil C_k + \Omega(t) > mt \\ \Rightarrow & m\left(tU_k + C_k\right) + \Omega(t) > mt \end{aligned} \tag{3.15}$$

With the same logic, substituting $\Omega(t)$ in the above inequality by the one in Lemma 3.5, the resulting condition is still the one for task τ_k being busy over an interval of length t. The negation of this condition is sufficient for task τ_k being complete. After reformulation, we can obtain Eq (3.14). □

From Lemma 3.6, we know the H to be checked in Theorem 3.7 must be finite if

$$mU_k - \sum_{\tau_i \in hp(k)} U_i < m \tag{3.16}$$

Subsequently, the following lemma provides an upper bound on the maximum length of the h-incomplete busy periods of task τ_k.

Lemma 3.7.

$$R_{k,h} \leq R^{\dagger}_{k,h} := \frac{hmC_k + Z_\Sigma + \sum_{\tau_i \in hp(k)} C_i(1 - U_i)}{m - \sum_{\tau_i \in hp(k)} U_i} \tag{3.17}$$

where Z_Σ denotes the sum of the $(m-1)$ largest D_iU_i's.

Proof. Substituting $\Omega(t)$ in the recurrence Eq. (3.9) by the one in Lemma 3.5, we can find the point t of the intersection, which leads to the statement. □

By Lemmas 3.1 and 3.7, $RT_{k,h}$ is bounded from above by

$$RT_{k,h} \leq RT^{\dagger}_{k,h} := R^{\dagger}_{k,h} - (h-1)T_k \tag{3.18}$$

where $R^{\dagger}_{k,h}$ is defined in Lemma 3.7. Intuitively, one may examine all possible h in the above equation to ensure that the upper bound on the worst-case response time. Our observation is that $RT^{\dagger}_{k,h}$ defined by Eq. (3.18) is monotonically decreasing with respect to h, under certain condition. We state this with the following lemma:

Lemma 3.8. *Suppose that*

$$mU_k + \sum_{\tau_i \in hp(k)} U_i < m \tag{3.19}$$

Then, for any integer $h \geq 1$,

$$RT^{\dagger}_{k,1} \geq RT^{\dagger}_{k,h}$$

where $RT^{\dagger}_{k,h}$ is as defined in Eq. (3.18).

Proof. We now prove that $R^{\dagger}_{k,h} - (h-1)T_k$ is maximized when h is 1, under Condition (3.19). Differentiating $R^{\dagger}_{k,h} - (h-1)T_k$ with respect to h, we have that

$$\begin{aligned}\frac{\partial}{\partial h}\left(R^{\dagger}_{k,h} - (h-1)T_k\right) =& \frac{mC_k}{m - \sum_{\tau_i \in hp(k)} U_i} - T_k \\ =& T_k \left(\frac{mU_k}{m - \sum_{\tau_i \in hp(k)} U_i} - 1\right) \\ =& T_k \left(\frac{mU_k + \sum_{\tau_i \in hp(k)} U_i - m}{m - \sum_{\tau_i \in hp(k)} U_i}\right)\end{aligned}$$

which is negative due to our assumption $mU_k + \sum_{\tau_i \in hp(k)} U_i < m$. Since $R^{\dagger}_{k,h} - (h-1)T_k$ is decreasing with respect to h, the maximum occurs when $h = 1$. □

Putting these pieces together, Theorem 3.8 provides the response time upper bound of arbitrary-deadline tasks on multiprocessor systems.

Theorem 3.8. *The worst-case response time RT_k of task τ_k is at most*

$$RT_k \leq RT^{\dagger}_k = \begin{cases} R^{up}_k & \text{if } mU_k + \sum_{\tau_i \in hp(k)} U_i < m, \\ \infty & \text{otherwise.} \end{cases}$$

where $R^{up}_k = \frac{mC_k + Z_\Sigma + \sum_{\tau_i \in hp(k)} C_i(1-U_i)}{m - \sum_{\tau_i \in hp(k)} U_i}$ and Z_Σ denotes the sum of the $(m-1)$ largest D_iU_i's among the tasks in $hp(k)$.

Proof. By Lemmas 3.7 and 3.8, we can conclude this theorem by setting $h = 1$ in Eq. (3.18). □

Subsequently, we provide our linear-time upper bound (LTUB) on the worst-case response time in the follow corollary:

Corollary 3.2 (LTUB). *An arbitrary-deadline sporadic task system τ is schedulable on m identical processors under fixed-priority scheduling if for all tasks $\tau_k \in \tau$,*

$$RT^{\dagger}_k \leq D_k \tag{3.20}$$

where $RT^{\dagger}_k$ is as defined in Theorem 3.8.

3.5 EXPERIMENTS

In this section, we conduct extensive experiments using synthesized task sets for evaluating the proposed tests. The metric to compare results is to measure the *acceptance ratio* of the above tests with respect to a given goal of task set utilization. We generate 100 task sets for each utilization level, from 0.05 to 1, in steps of 0.05. The acceptance ratio of a level is said to be the number of task sets that are schedulable divided by the number of task sets for this level, i.e., 100.

3.5.1 *Simulation Environment*

We first generated a set of sporadic tasks. The cardinality of the task set was 5 times the number of processors. In each utilization step, the `Randfixedsum` method [ESD10] is adopted to generate a set of utilization values with the given goal. We use the approach suggested by Emberson et al. [ESD10] to generate the task periods according to the *logarithm distribution*. The order of magnitude p to control the period values between largest and smallest periods is parameterized in evaluations, e.g., $1-10ms$ for $p=1$ and $1-100ms$ for $p=2$.

We evaluate these tests in 8-core processor systems with $p \in [1,2,3]$. Similar results can be seen in different numbers of processors. The priority of tasks is assigned according to deadline-monotonic (DM) scheduling: the smaller the relative deadline, the higher the priority level. We note that deadline-monotonic (DM) scheduling is not *optimal* on multiprocessor systems. Alternatively, one can use the *laxity-monotonic* (LM) scheduling: the smaller the laxity $D_i - C_i$, the higher the priority level [AJ00]. Also, it is not difficult to see that the proposed TDA analysis and the linear-time response bound comply with the required conditions for the optimal priority assignment (OPA) compatibility provided in [DB11b]. Hence, our proposed tests are compatible with OPA: if there exists feasible priority assignments by our test, OPA returns one of them. It is also worth noting that Davis and Burns have suggested that being OPA-compatible is as important as being effective since priority assignment is an important factor in determining the schedulability of tasksets under global fixed priority pre-emptive scheduling [DB11b]. Consequently, applying the tighter test but not OPA-compatible, e.g. maximum work function $\hat{I}_i^2(B_{k,h})$, by using DM, may not yield better performance than the less effective but OPA-compatible test using OPA priority assignment. Nevertheless, we here focus on showing the effectiveness of the tests themselves, and similar results can be seen under different priority assignment policies.

3.5.2 *Constrained-Deadline Systems & Results*

The execution time was set accordingly, i.e., $C_i = T_i U_i$. Task relative deadlines were uniformly drawn from the interval $[0.8T_i, T_i]$.

The tests evaluated are shown as follows:

- *BCL*: the polynomial-time test in Theorem 4 in [BCL05b].
- *FF*: the force-forward (FF) analysis in Eq. (5) in [BBMS10].
- *BAK*: the $O(n^3)$ test in Theorem 11 in [Bak06].
- *Guan*: the TDA analysis by Guan [GSYY09].
- *TDA*: Corollary 3.1 in this paper using the carry-in function by Eq. (3.6).
- *TDA+*: Corollary 3.1 in this paper using the improved carry-in function by Eq. (3.12).

- *LTUB*: Corollary 3.2 in this paper.

Result I. Figure 3.6 presents the evaluation result in constrained-deadline systems. We first notice that the response time analysis with limited carry-in tasks, namely *Guan* and *TDA*, can admit the most number of task sets. Overall, these two tests achieve similar performance, even though our *TDA* is inferior to *Guan* due to the over-approximate workload function, as mentioned earlier. As shown in Figure 3.6, *BCL*, *FF*, and *BAK* perform poorer than *LTUB*, even though their computational complexity is higher than *LTUB*. (pseudo-polynomial time for *FF* and $O(n^3)$ for *BAK*). It is noticeable that *TDA* is only slightly advantageous to *LTUB* in case of $p = 2$ and $p = 3$ (Figure 3.6(b) and (c)) where the TDA essentially suffers from its high computational complexity. Hence, in practice, one would expect that the TDA be adopted in the low order of magnitude of task periods and the linear-time response time bound in the high order of the magnitude.

3.5.3 *Arbitrary-Deadline Systems & Results*

The execution time was set accordingly, i.e., $C_i = T_i U_i$. Task relative deadlines were uniformly drawn from the interval $[0.8T_i, 2T_i]$.

The tests evaluated are shown as follows:

- *LOAD*: the load-based analysis [BFo8].
- *BAK*: the $O(n^3)$ test in Theorem 11 [Bako6].
- *LTUB*: Corollary 3.2 in this paper.
- *TDA*: Corollary 3.1 in this paper.

Result II. Figure 3.7 presents the result in arbitrary-deadline systems where $\frac{D_i}{T_i} \in [0.8, 2]$. Due to the overly optimistic workload function, *Guan*'s analysis becomes unsafe for arbitrary-deadline tasks. The proposed TDA is the best TDA with limited carry-in interferences in arbitrary-deadline systems, as can be seen in Figure 3.7. Also, in terms of schedulability and time complexity, the proposed linear-time upper bound *LTUB* is superior to *LOAD* and *BAK*, where the time complexity is pseudo-polynomial for *LOAD* and $O(n^3)$ for *BAK*.

3.6 SUMMARY

In this chapter, we have proposed the response time analysis for sporadic arbitrary-deadline tasks on multiprocessor systems under fixed-priority scheduling. We also derived a linear-time upper bound on the response time of arbitrary-deadline tasks in multiprocessor systems. This bound is capable of leveraging the number of task sets that are deemed schedulable while keeping task sets solvable within polynomial time.

Today, the problem of scheduling the sporadic task model is well-understood, and many sound scheduling algorithms are also available. Nevertheless, several assumptions on this model must be assured to make use of those algorithms:

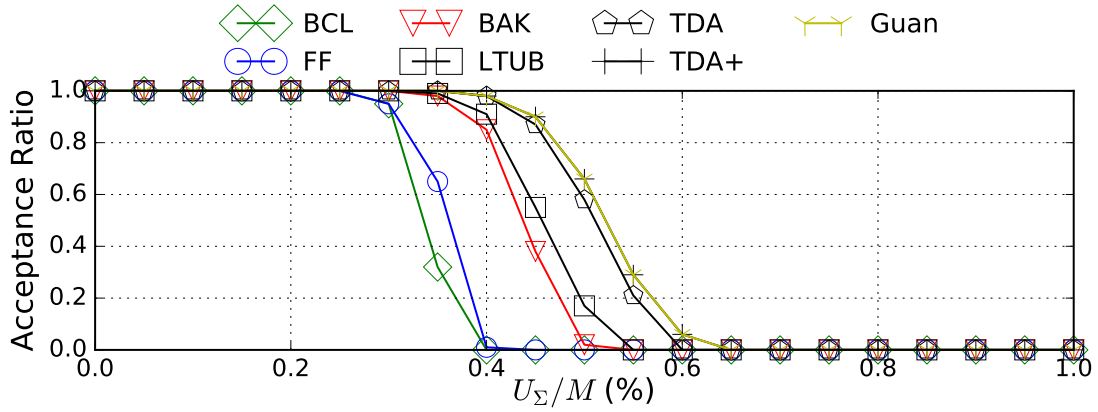
1. All tasks are assumed to be independent; there is neither precedence relationships nor resource constraints among tasks.

2. Tasks cannot suspend themselves. Once a job is released, it becomes idle only when its execution is finished.

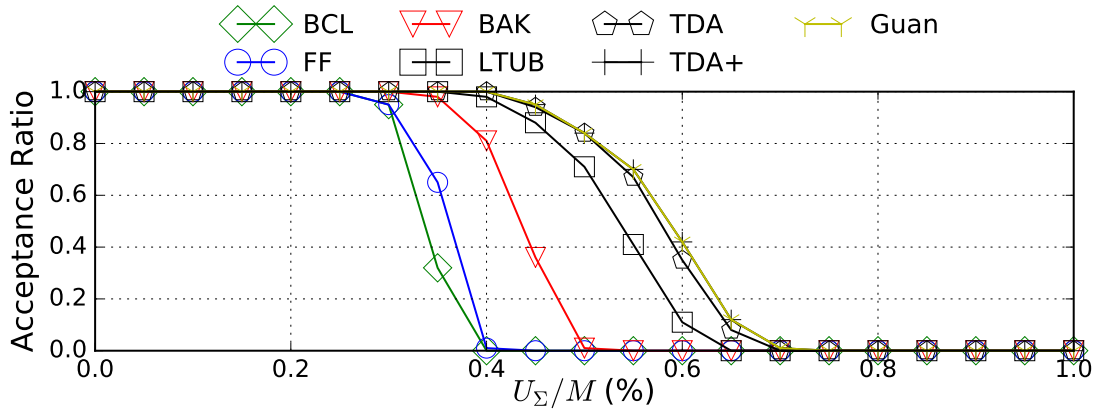
3. Tasks are periodically/sporadically activated; and all jobs of a task have the same worst-case execution time.

However, this is often not the case in practice. For example, if no data information is locally available, I/O operations must be involved during the execution of a task, typically making the task suspended on CPUs, yielding the CPU to other tasks. To cope with such assumptions while benefiting from those well-developed scheduling algorithms for the sporadic task model, one may choose to properly account for the unfavorable effect of relaxing the assumptions. For instance, the times spent on I/O operations of a task would be subsumed into its WCET, figuratively assuming the task running without suspensions. This, however, may result in underutilization of real-time systems.

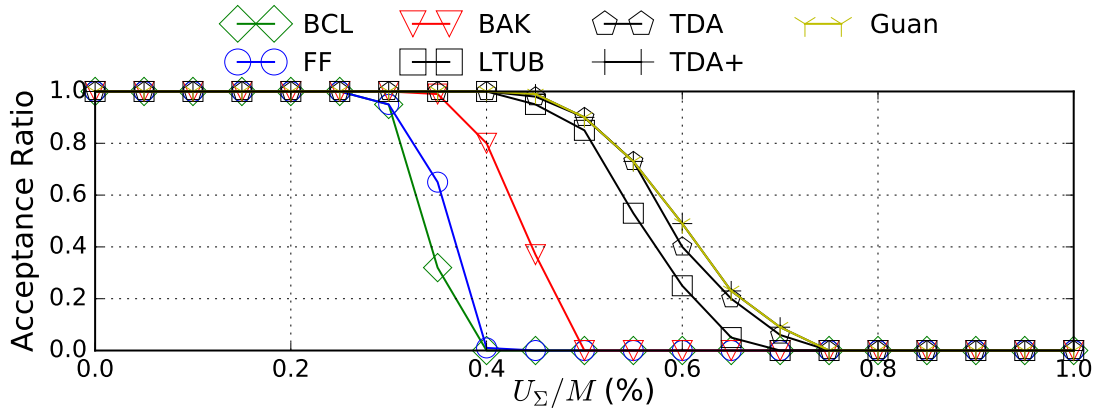
More preferably, one may adopt more expressive models that better characterize the behavior of real-time systems in timing analysis. In the following chapters, we discuss the problems and solutions related to the above assumptions not valid.



(a) $p = 1$

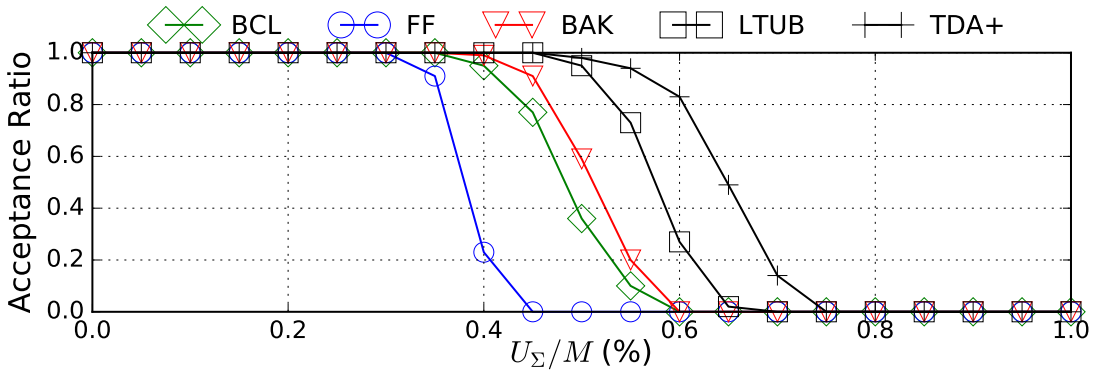


(b) $p = 2$

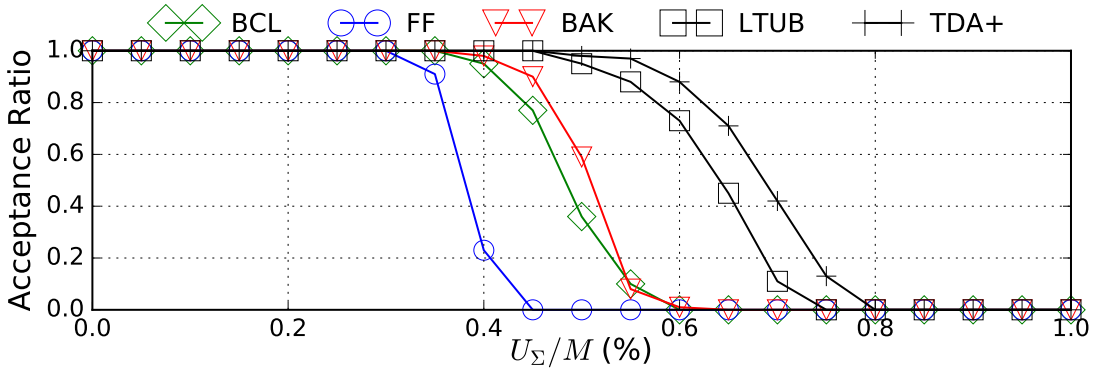


(c) $p = 3$

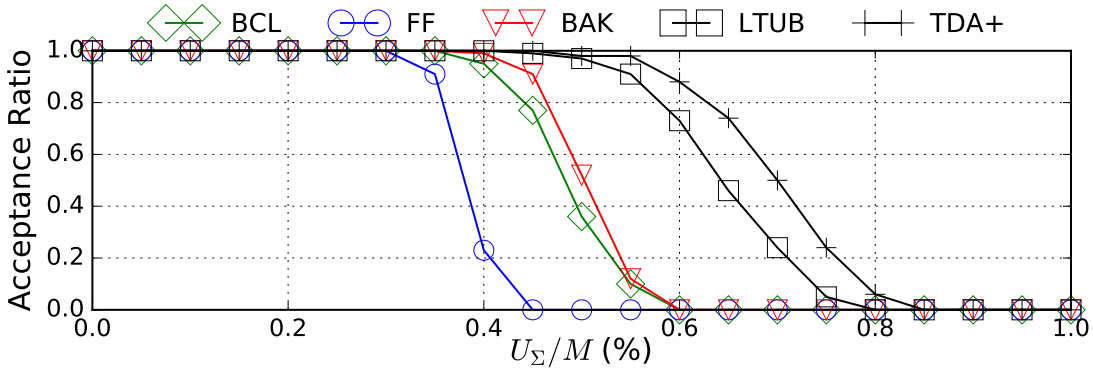
Figure 3.6: Acceptance ratio comparison with different p on 8-multiprocessor, constrained-deadline systems where $\frac{D_i}{T_i} \in [0.8, 1]$.



(a) $p = 1$



(b) $p = 2$



(c) $p = 3$

Figure 3.7: Acceptance ratio comparison with different p on 8-multiprocessor, arbitrary-deadline systems where $\frac{D_i}{T_i} \in [0.8, 2]$.

4

RESOURCE ACCESS TASK MODELS

Traditionally, timing analysis consists of two separate steps: (*i*) worst-case execution time (WCET) analysis, which computes an upper bound on the execution time of a single job of a task running in isolation, and (*ii*) schedulability analysis, which determines whether multiple tasks are guaranteed to meet their deadlines, while sharing a processor.

As far as resource access is concerned, the delay of access must be taken into account of calculating the WCET. Traditional approaches assume that each memory access takes constant time. For instance, SymTA/P assumes that each memory access takes constant time. This approach is sensible only if resource access time is relatively short. However, in practice each resource access can range from a few microseconds (e.g., a write operation on a flash drive) to a few hundreds of milliseconds (e.g., offloading computation to Graph Processing Units (GPUs)). As a result, subsuming resource consumptions into WCET can result in a huge underutilization of systems.

In this chapter, we propose an approach that characterizes each task by not only its local computation times but also resource access times to avoid unnecessary utilization loss in abstract interpretation. The main contribution is that we propose a task partitioning algorithm that achieves a speedup factor of 7, compared to the optimal schedule. This constitutes the first result in this research line with a speedup factor guarantee.

The presentation is organized as follows: In Chapter 4.1, we first introduce the system and task model studied here, followed by a review of related work. In Chapter 4.2,

we develop the corresponding schedulability analysis. In Chapter 4.3, we propose our task allocation algorithm. Later on in Chapter 4.4, we show that this algorithm offers non-trivial quantitative guarantees, including a speedup factor of 7 in polynomial-time complexity. In Chapter 4.5, the experimental evaluation demonstrates that our approach can yield good acceptance ratios. We also show that our analysis can utilize the structured information in the task models to provide tighter schedulability analysis. Finally, we summarize our results in Chapter 4.6.

4.1 SYSTEM MODEL

4.1.1 *Multicore Architecture Model*

We assume that we have a multicore platform comprised of m identical cores connected by a shared resource, e.g., a communication fabric (bus) for accesses to a shared memory. We assume the following properties:

- Each data request (after it is granted) has a processing time (upper bound) of B.
- Each resource access request is atomic (non-split transaction). However, a segment of resource accesses with several consecutive requests can be preempted at any point at which a request finishes its transfer. This is illustrated in Figure 4.1 where task τ_1 requesting the shared resource at the beginning of clock cycle 1 experiences blocking of 4 clock cycles from task τ_2 granted at time 0, i.e., B is 5 clock cycles in this example, whereas the second request from τ_2 is preempted by τ_1 having highest-priority at time 5.
- We consider a fixed-priority resource arbiter in this dissertation: the arbiter of the shared resource always grants the request that is assigned the highest priority.
- We assume that data transfers from the local to the shared memory and vice versa are handled by direct memory access (DMA) units, so that the computation on the core may carry on during such activities. Such a feature is found in PTARM [RLPKL11].

4.1.2 *Task Model*

We consider a real-time system to execute a set of n independent, preemptive, *resource access sporadic* (RAS), real-time tasks $\tau = \{\tau_1, \tau_2, ..., \tau_n\}$. Each task can release an infinite number of jobs under a given minimum inter-arrival time constraint. An RAS real-time task τ_i is characterized by a 5-tuple $(C_i, A_i, T_i, D_i, \sigma_i)$: T_i denotes the minimum inter-arrival time (also known as period) of τ_i; each job of τ_i has a relative deadline D_i; C_i denotes an upper bound on total execution time of each job of τ_i on a computing core

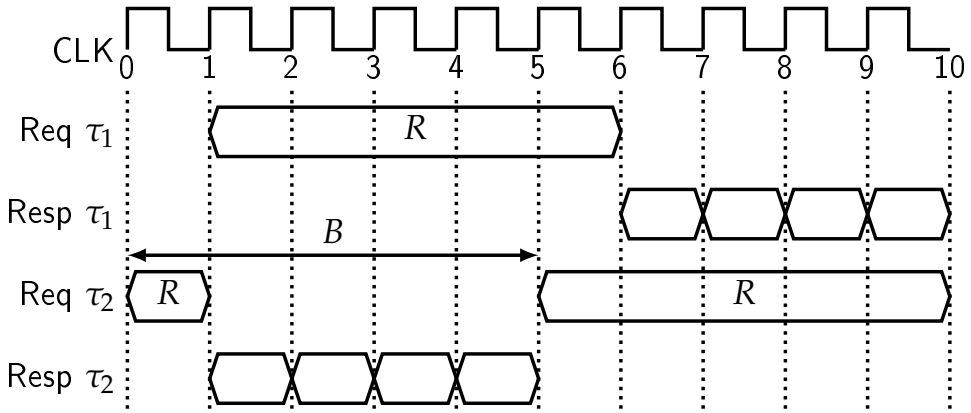


Figure 4.1: Two tasks access the share resource on a non-split-transaction bus.

(ignoring the latency due to shared resource accesses); A_i denotes an upper bound on the amount of execution time of shared resource accesses of τ_i; and σ_i denotes the maximum number of resource access segments, each of them may consist of several consecutive *data requests* to access the shared resource. That is, if the platform allows multiple requests to be directly processed without involving the core, a task can have multiple requests to the shared resource within a resource access segment, e.g., fetching (or writing back in the other direction) a collection of data and instructions from the shared main memory to the local scratchpad memory.

That is, if there is no interference on the shared resource accesses, the worst-case execution time of task τ_i is at most $C_i + A_i$. Note that, the derivation of C_i assumes that the shared resource accesses incur no timing cost. Moreover, the derivation of A_i considers only the worst-case shared resource access time of task τ_i. Figure 4.2 provides an example. Each resource access segment may consist of several requests on the shared resource. For example, given that $B = 1$ time units, there are twenty resource accesses in resource access segment (block) E in Figure 4.2. Note that A_i is assumed to be $w_i B$ where w_i is an integer. Similar to the models in the literature, [PSCCT10; PSCCT10; Alt+15], we consider that the multicore platform is timing-compositional [Wil+09; HRW15]. Therefore, we can use these parameters to bound the worst-case behaviour safely.

We assume that $C_i + A_i \leq D_i$ for any task $\tau_i \in \tau$. The utilizations of task τ_i on a core and on shared resource are defined as $U_i^C = C_i/T_i$ and $U_i^A = A_i/T_i$, respectively. We further assume that $U_\Sigma^C = \sum_{i=1}^n U_i^C \leq m$ and $U_\Sigma^A = \sum_{i=1}^n U_i^A \leq 1$. Otherwise, it cannot be feasibly scheduled.

In this dissertation, we consider *partitioned scheduling*: each task is statically assigned onto one core, and we assume that all the tasks allocated to a core are preemptively scheduled using fixed priority scheduling, for which each task is associated with a unique priority level for scheduling. Since each shared resource request is assumed *non-*

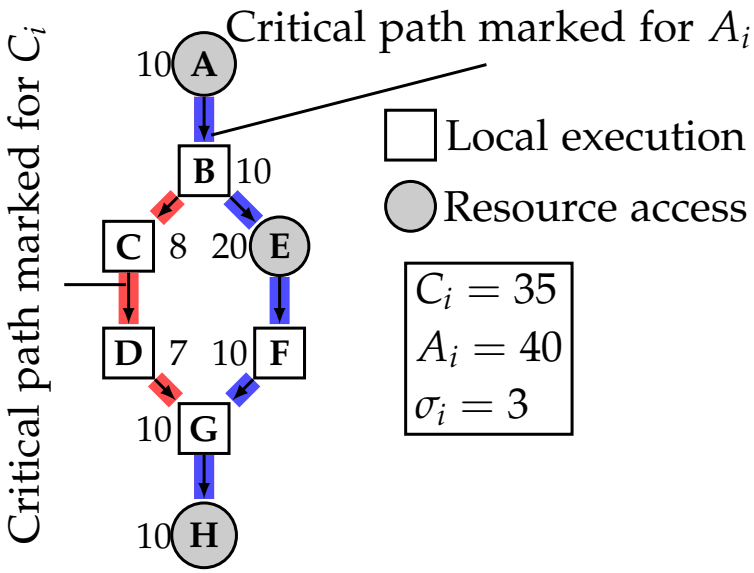


Figure 4.2: Two tasks access the share resource on a non-split-transaction bus.

preemptive, if task τ_i starts a resource access segment, it has to wait for a lower priority task to finish the access of the shared resource for at most B time units. Therefore, we know that the maximum blocking time of task τ_i due to non-preemptive blocking on the shared resource is at most $\sigma_i B$. We assume that $\sigma_i B \leq A_i$.

For any RAS task τ_i and any real number $t \geq 0$, the demand bound function of task τ_i on the core for an interval of length t is $dbf_i^C(t) = max(0, (\left\lfloor \frac{t-D_i}{T_i} \right\rfloor + 1) \times C_i)$ and that on the shared resource is $dbf_i^A(t) = max(0, (\left\lfloor \frac{t-D_i}{T_i} \right\rfloor + 1) \times A_i)$.

The terminology and notation to be used are list in Table 5.1.

4.1.3 *Related Work.*

For a more in-depth treatment of literature on the impact of resource sharing on performance and worst-case timing analysis please consult the survey in [Abe+13]. One line of work in timing analysis for multicores is to assume structured execution models, e.g., [LGPWS14; SPCTC10; PSCCT10; MBBMB15]. For example, the superblock execution model, proposed by Pellizzoni et al. [PSCCT10], classifies the execution of a superblock into three phases: data acquisition, local execution, and data replication phases. This is also standardized in IEC 61131-3. For this structured model, the state-of-the-art results are restricted to the sequential execution of superblocks without any preemptions [LGPWS14; SPCTC10; PSCCT10]. Recently, Melani et al. [MBBMB15] have studied the problem of scheduling tasks consisting of one memory phase and one execution phase, called M/C (memory-computation) tasks, providing an exact

SYMBOL	MEANING
C_i	an upper bound on total execution time of each job of τ_i on a computing core (ignoring the latency due to shared resource accesses)
A_i	an upper bound on the amount of execution time of shared resource accesses of τ_i
σ_i	the maximum number of resource access segments, each of which may consist of several consecutive *data requests* to access the shared bus
U_i^A	the utilizations of task τ_i on the shared bus; $U_i^A = A_i/T_i$
U_i^C	the utilizations of task τ_i on a core; $U_i^C = C_i/T_i$
U_i	the utilization of task τ_i; $U_i = (C_i + A_i)/T_i$
B	the (upper bound) processing time of each data request (after it is granted)
$dbf_i(t)$	the demand bound function of task τ_i over an interval of length t; $dbf_i(t) = \max\left(0, (\left\lfloor \frac{t-D_i}{T_i} \right\rfloor + 1) \times (A_i + C_i)\right)$
$dbf_i^C(t)$	the demand bound functions of task τ_i for local computing execution; $dbf_i^C(t) = max\left(0, (\left\lfloor \frac{t-D_i}{T_i} \right\rfloor + 1) \times C_i\right)$
$dbf_i^A(t)$	the demand bound functions of task τ_i for accesses to the shared bus; $dbf_i^{R_q}(t) = \max\left(0, (\left\lfloor \frac{t-D_i}{T_i} \right\rfloor + 1) \times A_i\right)$

Table 4.1: Resource access: notation and terminology

response-time analysis under fixed-priority scheduling. Another direction is to consider structured resource arbitration methods, e.g., Time Division Multiple Access (TDMA) or Round-Robin (RR) arbitration [KFMCR14; SPCTC10]. Since the resource sharing problem in multicore systems does not admit tight analytical solutions, approaches adopting timed automata, e.g., [PSCCT10; LYGY10], have been also reported. Altmeyer et. al [Alt+15] present a framework to decouple response-time analysis from a reliance on context independent WCET values by separately analyzing the timing contribution of each resource to a task's response time. They implicitly assume that tasks spin while awaiting the response from a shared resource, rather than suspending.

In summary, existing schemes either make very strong assumptions about the tasks and their execution model [PSCCT10; SPCTC10; PSCCT10; MBBMB15] or the analysis complexity is intractably high (the state explosion problem) [PSCCT10; LYGY10]. Further, mathematical guarantees about the quality of the scheduling policy and the schedulability analysis are usually not provided [PSCCT10; SPCTC10; KFMCR14; Alt+15], in contrast to classical results for single-core scheduling. In this work, we introduce a new task model that is more permissive than the structured models proposed earlier, while at the same time enabling the derivation of hard mathematical guarantees in the form of a speedup factor.

4.2 SCHEDULABILITY ANALYSIS

We present our response-time and schedulability analyses under the following conditions:

- We assume that the priority levels are assigned *a priori*.
- Each task τ_i is assigned to a core. Let Γ_p denote the *set* of tasks that are allocated on core p, on which task τ_k (that is under analysis) is allocated.
- We *only* test the schedulability of task τ_k assuming that all tasks with higher priority than task τ_k are already guaranteed to meet their deadlines.

The last condition also implies that we have to perform schedulability tests by considering all the tasks one by one. The task set $hp(k)$ consists of the tasks with higher priority than task τ_k. Therefore, the tasks in $hp(k)$ can interfere with task τ_k on the shared resource and the tasks in $hp(k) \cap \Gamma_p$ can interfere with task τ_k on core p. Throughout this section, we assume that we know an upper bound R_i on the worst-case response time of any higher-priority task τ_i in $hp(k)$, which can be derived in the previous iterations. By the last condition above and the assumption of constrained-deadline task systems, we have $R_i \leq D_i \leq T_i$.

4.2.1 *Our Strategies and Preliminaries*

Consider the simplest case in which two tasks τ_1 and τ_2 are assigned on core 1 and core 2, respectively. That is, there is no multitasking on each core. We assume that τ_1 has higher priority than τ_2 and we are now analyzing task τ_2 in all the examples. The schedule of accessing a shared resource by these two tasks is indicated in Figure 4.3:

- At time t_1, tasks τ_1 and τ_2 start their computation.
- At time t_2, tasks τ_1 and τ_2 both attempt to access the shared resource. The request from task τ_1 is granted, while task τ_2 suspends itself on core 2.
- At time t_3, task τ_1 finishes its access to the shared resource and resumes its local computation. At the same time, task τ_2 starts to access the shared resource.
- At time t_4, task τ_1 again attempts to access the shared resource. Due to the minimum non-preemptive region, this request from task τ_1 is blocked by task τ_2.
- At time t_5, after leaving the non-preemptive region, task τ_2 on the shared resource is preempted by task τ_1.
- At time t_6, task τ_1 finishes its access to the shared resource and resumes its local computation. At the same time, task τ_2 continues to access the shared resource.
- At time t_7, task τ_2 finishes its access to the shared resource and resumes its local computation.
- At time t_8, task τ_2 finishes its execution.

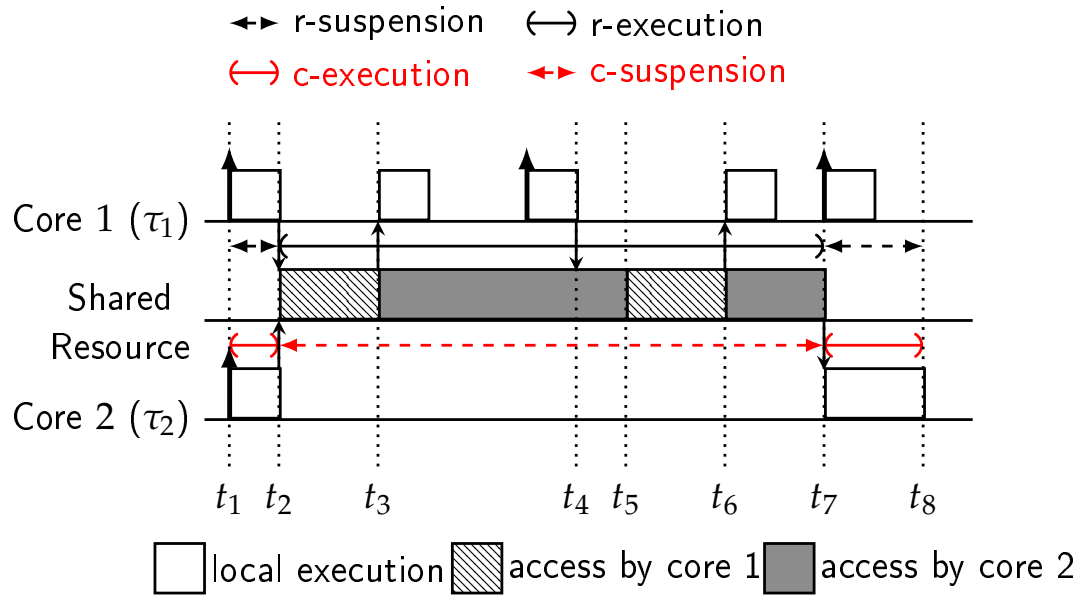


Figure 4.3: An example of accessing a shared resource by two cores, on which τ_1 and τ_2 are allocated separately.

From the point of view of the shared resource, task τ_2 suspends its resource accesses in time intervals $[t_1, t_2)$ and $[t_7, t_8)$ due to local computations, both of which are called *r-suspension* intervals in this dissertation. Moreover, task τ_2 *executes* on the shared resource in time intervals $[t_3, t_4)$ and $[t_6, t_7)$ and *awaits* (in the queue) to access the shared resource in time intervals $[t_2, t_3)$ and $[t_5, t_6)$, all of which are called *r-execution* intervals in this dissertation. Symmetrically, from the point of view of core 2, task τ_2 executes/suspends its computation on the core, called *c-execution*/*c-suspension*, in time intervals $[t_1, t_2)$ and $[t_7, t_8)$/ time intervals $[t_2, t_7)$. In either perspective, the feasibility of τ_2 can be examined by answering whether the summation of r-execution time and r-suspension time (or c-execution time and c-suspension time) of a job of task τ_2 is never more than its relative deadline.

Our analysis strategy to safely bound R_k of task τ_k is as follows: Suppose that task τ_k releases a job at time t_0. We check whether this job of task τ_k is guaranteed to be finished by $t_0 + t$. Therefore, we have to determine the cumulative amount of time in the time interval $(t_0, t_0 + t]$ for r-execution and r-suspension while executing this job of task τ_k. They are defined as *r-execution time* and *r-suspension time* of task τ_k in an interval of length t, respectively.

Without multitasking, the total time of resource access suspension from the bus (r-suspension) is trivially upper bounded by the execution time due to local computation. However, with multitasking, this time increases due to interference from other tasks

allocated on the same core p. To calculate an upper bound on the r-execution and r-suspension time of task τ_k, we first provide two lemmas:

Lemma 4.1. *The cumulative resource access time of task $\tau_i \in hp(k)$ on the shared resource within an interval of length t is upper bounded by:*

$$E_i(t) = \left\lceil \frac{t + R_i - A_i}{T_i} \right\rceil A_i \tag{4.1}$$

Proof. In the presence of suspension, we need to consider the *carry-in* effect, as pointed out in [LC14; HCZL15]. Nevertheless, the interference due to such an effect can be quantified as *jitter*, and thereafter the access times can be upper bounded by releasing all the accesses from the first job (that arrives before t_0 and has an absolute deadline after t_0) as late as possible and the following accesses as soon as possible. □

Lemma 4.2. *The cumulative execution time of task $\tau_i \in hp(k)$ on the core, on which τ_i is allocated, within an interval of length t is upper bounded by:*

$$W_i(t) = \left\lceil \frac{t + R_i - C_i}{T_i} \right\rceil C_i \tag{4.2}$$

Proof. This is due to the same argument in Lemma 4.1. □

The rest of this section is organized as follows: (1) We first derive a bound on the r-execution time (denoted as $X_k(t)$) and r-suspension time (denoted as $S_k(t)$) in Section 4.2.2 under the assumption that the worst-case response time of task τ_k is no more than t for some $0 < t \leq T_k$. (2) We then derive a bound on the r-execution time (denoted as X_k^*) and r-suspension time (denoted as S_k^*) that is *independent* of the task's response time in Section 4.2.3 by referring to the number of resource access segments. (3) Section 4.2.4 provides the schedulability test, combining the results from (1) and (2).

4.2.2 *Calculating $S_k(t)$ and $X_k(t)$*

To evaluate the maximum r-suspension time, provided that the interval length of interest is $t \leq T_k$, we can simply sum over the computation workload $W_i(t)$ from all the higher-priority tasks that are allocated on core p plus C_k. The proofs are relatively straightforward.

Lemma 4.3. *The r-suspension time of task τ_k due to its executions on core p in an interval of length t is at most*

$$S_k(t) = C_k + \sum_{\tau_i \in hp(k) \cap \Gamma_p} W_i(t) \tag{4.3}$$

Similarly, we have the corresponding lemma for the maximum r-execution time as follows.

Lemma 4.4. *The r-execution time for task τ_k in an interval of length t is at most*

$$X_k(t) = A_k + \sigma_k B + \sum_{\tau_i \in hp(k)} E_i(t) \tag{4.4}$$

4.2.3 *Calculating X_k^* and S_k^**

We now show how to bound the r-execution time and the r-suspension time, *independently* of the task's overall response time. To this end, we use σ_k to calculate X_k^* and S_k^*.

Lemma 4.5. *X_k^* for task τ_k is the smallest t satisfying the following inequality:*

$$A_k + \sigma_k B + \sum_{\tau_i \in hp(k)} \left(\sigma_k - 1 + \left\lceil \frac{t + \sigma_k \cdot (R_i - A_i)}{T_i} \right\rceil \right) A_i \leq t \tag{4.5}$$

Proof. We consider any arbitrary program path with $z \leq \sigma_k$ shared resource access segments. W.l.o.g., we only have to consider z as an integer and ≥ 1. Let A_k^j denote the amount of execution times on the jth resource access segment from τ_k in the program path. By definition, $\sum_{j=1}^{z} A_k^j \leq A_k$. Let r_j be the smallest t satisfying $A_k^j + B + \sum_{\tau_i \in hp(k)} E_i(t) \leq t$. In other words, we know that the condition $\forall 0 \leq t < r_j, A_k^j + B + \sum_{\tau_i \in hp(k)} E_i(t) > t$ holds for all $j = 1, \ldots, z$. By adopting Lemma 4.1, the worst-case response of A_k^j can be easily proved to be upper bounded by r_j for all $j = 1, \ldots, z$.

Thus, the maximum r-execution time of this program path for task τ_k is upper bounded by $r_1 + r_2 + \cdots + r_z$. For a specified $0 \leq \theta < r_1 + r_2 + \cdots + r_z$, there always exists a combination of $t_1, t_2, \ldots, t_z$ with $0 \leq t_j < r_j$ such that $\theta = t_1 + t_2 + \cdots + t_z$. Therefore, we know that

$$(\sum_{j=1}^{z} A_k^j) + zB + \sum_{\tau_i \in hp(k)} \sum_{j=1}^{z} E_i(t_j) > \sum_{j=1}^{z} t_j = \theta. \tag{4.6}$$

By the fact that $\lceil x + y \rceil + 1 \geq \lceil x \rceil + \lceil y \rceil$, we have

$$\begin{aligned} \sum_{j=1}^{z} E_i(t_j) &= \sum_{j=1}^{z} (\left\lceil \frac{t_j + R_i - A_i}{T_i} \right\rceil) A_i \\ &\leq (z - 1 + \left\lceil \frac{z \cdot (R_i - A_i) + \sum_{j=1}^{z} t_j}{T_i} \right\rceil) A_i \\ &= (z - 1 + \left\lceil \frac{z \cdot (R_i - A_i) + \theta}{T_i} \right\rceil) A_i \end{aligned} \tag{4.7}$$

By substituting $\sum_{j=1}^{z} E_i(t_j)$ in Eq. (4.6) with the above inequality, the left hand side in Eq. (4.6) is at most

$$A_k + \sigma_k B + \sum_{\tau_i \in hp(k)} \left(\sigma_k - 1 + \left\lceil \frac{\theta + \sigma_k \cdot (R_i - A_i)}{T_i} \right\rceil \right) A_i \tag{4.8}$$

That is, the smallest t solved by Eq. (4.5) is at least $r_1 + r_2 + \cdots + r_z$. □

Similarly, we have the following lemma:

Lemma 4.6. *S_k^{r*} for task τ_k is the smallest t satisfying the following inequality:*

$$C_k + \sum_{\tau_i \in hp(k) \cap \Gamma_p} \left(\sigma_k + \left\lceil \frac{t + (\sigma_k + 1) \cdot (R_i - C_i)}{T_i} \right\rceil \right) C_i \leq t \tag{4.9}$$

Proof. Since the number of shared resource access segments of τ_k is at most σ_k, the maximum number of execution segments of τ_k on core p will be potentially $\sigma_k + 1$, interleaved the τ_k resource access segments. The rest of the proof for Eq. (4.9) is similar to Lemma 4.5. □

Note that it is also possible that resource accesses start both at the beginning and at the end of a program. Hence, there are at most $\sigma_k - 1$ execution segments of τ_k in this case, which in turn reduces pessimism, to be discussed in Section 4.5.

4.2.4 *Response-Time Analysis*

The following theorem concludes the schedulability test and response-time analysis directly from Lemmas 4.3 to 4.6.

Theorem 4.1. *The smallest t satisfying:*

$$\min\{X_k(t), X_k^*\} + \min\{S_k(t), S_k^*\} \leq t \tag{4.10}$$

is a safe upper bound of R_k if t is no more than T_k.

Proof. We prove this by contraposition. Suppose that task τ_k releases a job at time t_0. (Since we assume constrained-deadline task systems, we can safely consider that there is no job of task τ_k in the ready queue at time t_0 if $R_k \leq D_k \leq T_k$.) If this job has not yet finished by time $t_0 + t$, then, at any point in the time interval $(t_0, t_0 + t]$, task τ_k either executes (or is queued) on the shared resource or suspends on the shared resource. Therefore, if the job of task τ_k is not finished at time $t_0 + t$, a necessary condition is that its r-execution time plus its r-suspension time from t_0 to time $t_0 + t$ has been strictly larger than t. Such a condition implies $\min\{X_k(t), X_k^*\} + \min\{S_k(t), S_k^*\} > t$, which contradicts the assumption that t satisfies Eq. (4.10). □

Corollary 4.1. *An RAS task τ_k allocated on core p is schedulable in fixed-priority partitioned scheduling if there exists $0 \leq t \leq D_k$ s.t. Eq. (4.10) holds.*

The response-time analysis in Theorem 4.1 and the schedulability test in Corollary 4.1 require pseudo-polynomial time complexity. Note that we can also simply test $t = D_k$ requiring only polynomial time complexity at the expense of potentially less precise results.

RESPONSE-TIME ANALYSIS: SPINNING VERSUS SUSPENSION One might assume that Lemmas 4.3 and 4.4 also apply to a scenario in which tasks spin rather than suspending upon resource accesses. Unfortunately, this is not true, as it may underestimate delays due to lower-priority blocking: In a spinning scenario the response time of task τ_k is also affected by lower-priority blocking of higher-priority task that have preempted τ_k. To safely account for such delays, one would have to adapt Lemma 4.1 to account for lower-priority blocking of each resource access segment of task τ_i. A response-time analysis obtained in this manner roughly corresponds to the analysis described by Altmeyer et al. [Alt+15] adapted to the setting explored in this dissertation. For the special case that B is 0, employing Lemmas 4.3 and 4.4 without adaptation is correct for a model in which tasks spin upon resource accesses:

Proposition 4.2. *For $B = 0$, the smallest t satisfying:*

$$X_k(t) + S_k(t) \leq t \tag{4.11}$$

is a safe upper bound of R_k if t is no more than T_k, even if tasks spin upon resource accesses rather than suspending.

4.3 TASK ALLOCATION

We now present a task allocation algorithm that is compatible with the schedulability test in Corollary 4.1. Our strategy is to first sort the tasks according to their relative deadlines such that $D_1 \leq D_2 \leq \cdots \leq D_n$. Then, we allocate the tasks by using a simple heuristic reasonable allocation (RA) algorithm [GJ79], i.e., either First-Fit (FF), Best-Fit (BF), or Worst-Fit (WF). The First-Fit algorithm places the task in the first core that can accommodate the task. If no core is found, it opens a new core and places the task in the new core. The Best-Fit (Worst-Fit, respectively) algorithm places each task in the core with the *smallest* (*largest*, respectively) *remaining capacity* among all the core with sufficient capacity to accommodate the item. The *remaining capacity* is defined as the relative deadline minus the upper bound on the worst-case response time by Theorem 4.1. Algorithm 2 presents the pseudocode for the first-fit packing. Due to Corollary 4.1, if Algorithm 2 returns a feasible allocation, this results in a feasible schedule under the deadline-monotonic strategy (a task with a shorter relative deadline has higher priority).

Algorithm 2: MIRROR First-Fit Deadline-Monotonic

input : A set τ of RAS tasks and m identical cores
output: Task allocations Γ_j and the feasibility of system τ
sort the given n tasks in τ s.t. $D_1 \leq D_2 \leq \cdots \leq D_n$;
$\Gamma_j \leftarrow \emptyset, \forall j = 1, 2, \ldots, m$;
for $k = 1, 2, \ldots, n$ **do**
  for $j = 1, 2, \ldots, m$ **do**
    if *task* τ_k *is schedulable according to Corollary 4.1* **then**
      $\Gamma_j \leftarrow \Gamma_j \cup \{\tau_k\}$; `// assign` τ_k `to core` j
      break (continue the outer loop) ;
  return "infeasible allocation";
return "feasible allocation";

4.4 SPEEDUP FACTOR

In this section, we derive a speedup factor for Algorithm 2.

Lemma 4.7. *Any constrained-deadline RAS task system τ that is feasible upon a multicore platform comprised of m cores and a shared resource must satisfy*

$$\max \left\{ \max_{t>0} \frac{\sum_{\tau_i \in \tau} dbf_i^C(t)}{mt}, \max_{t>0} \frac{\sum_{\tau_i \in \tau} dbf_i^A(t)}{t}, \max_{\tau_i \in \tau} \frac{C_i + A_i}{D_i} \right\} \leq 1 \tag{4.12}$$

Proof. These conditions come from the definition of the demand bound functions, as also used in traditional multiprocessor scheduling [CC11] (without considering resource sharing).

First, if τ_i generates jobs with execution time exactly $C_i + A_i$, it is necessary for meeting all the deadlines of task τ_i that $C_i + A_i \leq D_i$.

Secondly, all the demand for accesses to the shared resource, i.e., $dbf_i^A(t)$, must be sequentially executed on a single computing unit, i.e., shared bus.

Last, the demand coming from the all tasks must be proceed on the platform with the maximum processing capacity of m, within any interval of length t. □

Theorem 4.3. *Algorithm 2 has a speedup factor of 7.*

Proof. Suppose that Algorithm 2 fails to obtain a partition for τ: there exists task τ_k which cannot be mapped to any core. Note that due to the sorting of the tasks in Algorithm 2, all the tasks before task τ_k mapped onto cores have been ensured $R_i \leq D_i$ for $i = 1, 2, \ldots, k-1$. Since τ_k fails the test of Corollary 4.1 on each of the m cores, for each core $p = 1, 2, \ldots, m$, we have $S_k(D_k) + X_k(D_k) > D_k$. By Lemma 4.3 and Lemma 4.4, we have

$$C_k + A_k + \sigma_k B + \sum_{\tau_i \in hp(k)} E_i(D_k) + \sum_{\tau_i \in hp(k) \cap \Gamma_p} W_i(D_k) > D_k$$

Summing over all m such cores, we obtain

$$m\left(C_k + A_k + \sigma_k B + \sum_{\tau_i \in hp(k)} E_i(D_k)\right) + \sum_{\tau_i \in hp(k)} W_i(D_k) > mD_k \tag{4.13}$$

By definition, $R_i \leq D_i \leq D_k$ for $\tau_i \in hp(k)$. To conclude the speedup factor, we need to prove $E_i(D_k) \leq 3dbf_i^A(D_k)$. If $T_i > D_k$, then

$$\begin{aligned} E_i(D_k) &\leq \left(\left\lceil \frac{D_k}{T_i} \right\rceil + 1\right) A_i \\ &\leq 2A_i = 2dbf_i^A(D_i) \\ &\leq 2dbf_i^A(D_k) \end{aligned} \tag{4.14}$$

Otherwise, if $T_i \leq D_k$, then

$$\begin{aligned} E_i(D_k) &\leq \left(\left\lceil \frac{D_k}{T_i} \right\rceil + 1\right) A_i \\ &\leq \left(\left\lfloor \frac{D_k}{T_i} \right\rfloor + 2\right) A_i \\ &\leq 3\left\lfloor \frac{D_k}{T_i} \right\rfloor A_i \\ &\leq 3dbf_i^A(D_k) \end{aligned} \tag{4.15}$$

Similarly, $W_i(D_k) \leq 3dbf_i^C(D_k)$. Dividing both sides by mD_k in Eq. (4.13), together with our assumption $\sigma_k B \leq A_k$ and the facts $E_i(D_k) \leq 3dfb_i^A(D_k)$ and $W_i(D_k) \leq 3dbf_i^C(D_k)$, we have

$$\frac{C_k}{D_k} + \frac{2A_k + 3\sum_{\tau_i \in hp(k)} dbf_i^A(D_k)}{D_k} + \frac{3\sum_{\tau_i \in hp(k)} dbf_i^C(D_k)}{mD_k} > 1. \tag{4.16}$$

Recall that $\sum_{\tau_i \in \tau} dbf_i^A(D_k) \geq A_k + \sum_{\tau_i \in hp(k)} dbf_i^A(D_k)$. Therefore, we have

$$\frac{C_k}{D_k} + \frac{3\sum_{\tau_i \in \tau} dbf_i^C(D_k)}{mD_k} + \frac{3\sum_{\tau_i \in \tau} dbf_i^A(D_k)}{D_k} > 1. \tag{4.17}$$

Assume for a contradiction that the task set is feasible on a multicore with speed $\frac{1}{7}$. Then, considering the adjusted speed, we know by Lemma 4.7 that $\frac{C_k}{D_k} \leq \frac{1}{7}$, $\frac{\sum_{\tau_i \in \tau} dbf_i^C(D_k)}{mD_k} \leq \frac{1}{7}$, and $\frac{\sum_{\tau_i \in \tau} dbf_i^A(D_k)}{D_k} \leq \frac{1}{7}$. This, however, clearly contradicts Eq. (4.17), implying a speedup factor of 7. □

We would like emphasize that the factor 7 in Theorem 4.3 can already be obtained in polynomial time if we only test t at D_k in Corollary 4.1. Moreover, for implicit-deadline, *harmonic-period* task systems, i.e., $T_i = D_i$ and $\frac{D_k}{T_i}$ is an integer for $\tau_i \in hp(k)$, the speedup factor is 5, since $E_i(D_k) \leq 2dbf_i^A(D_k)$ and $W_i(D_k) \leq 2dbf_i^C(D_k)$ in such cases.

4.5 EXPERIMENTAL RESULTS

In this section, we conduct extensive experiments using synthesized task sets. We evaluate these tests on a 4-core system, i.e., $m = 4$. We generate 100 task sets for each utilization level, from $0.05m$ to m, in steps of $0.05m$. The metric to compare the results is to measure the *acceptance ratio*. The acceptance ratio of a level is said to be the number of task sets that are deemed schedulable by the test divided by the number of task sets for this level, i.e., 100.

The cardinality of the task set is 5 times the number of processors, e.g. 20 tasks on 4 cores. In each utilization step, the `Randfixedsum` method [ESD10] is adopted to generate a set of utilization values with the given goal. We use the approach suggested by Emberson et al. [ESD10] to generate the task periods according to the *exponential distribution*. The distribution of periods is within two orders of magnitude, i.e., 10ms-1000ms. The execution time is set accordingly, i.e., $C_i = T_i U_i$. Task relative deadlines are implicit, i.e., $D_i = T_i$. We then generate a set of access utilization values U_i^A for tasks with the given U_Σ^A, according to the uniform distribution, generated by the UUniFast method. We consider task sets with total access utilization U_Σ^A of 40% and 70%. We ensure that for every task τ_i, $U_i^A + U_i^C \leq 1$. Then, the upper bound on the access time to shared resources A_i was set accordingly, i.e., $A_i = T_i U_i^A$.

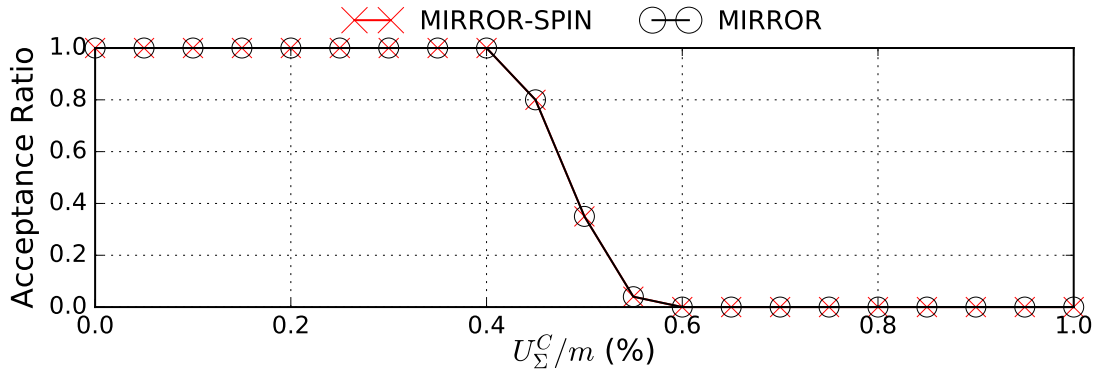
The maximum number of resource access segments σ_i is set depending on the following types of access: 1 (rare access, type=R), 2 (moderate access, type=M), and 10 (frequent access, type=F). The evaluated tests are listed as follows:

- MIRROR: the test proposed in Corollary 4.1.
- MIRROR-SPIN: the test following Proposition 4.2 using only $S_k(t)$ and $X_k(t)$, which applies also to tasks that spin while awaiting access to the shared resource. This resembles the test from [Alt+15] in our model when B is 0.
- exact-MC: the exact test proposed in [MBBMB15] for M/C tasks, which is the special case of the RAS task model.

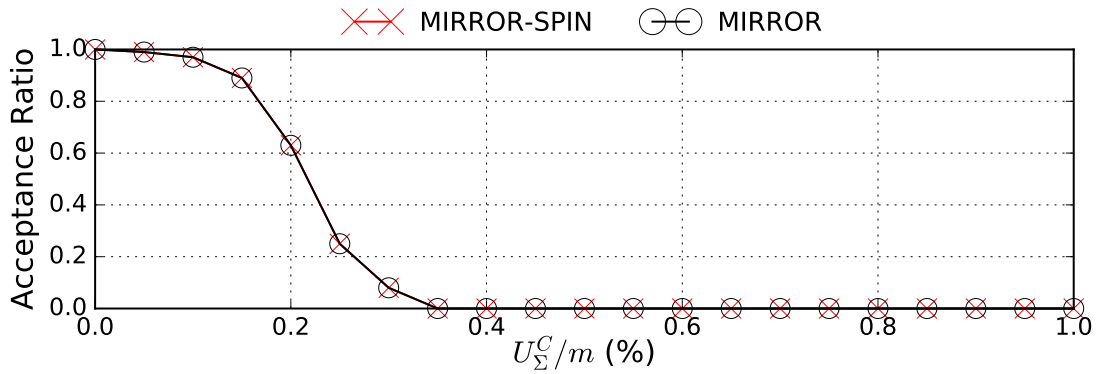
To fairly compare the results with respect to [Alt+15; MBBMB15], we assume that $B = 0$ in all the tests; otherwise, our tests can benefit from the short blocking time, and the other results can be significantly impacted by their considerations of blocking time as explained in Section 4.2.4.

Result I. Figures 4.4 (a) and (b) depict the results (by using the FF allocation) with frequent access (F) for 40% and 70% bus utilizations. As shown in Figures 4.4 (a) and (b), the performance by MIRROR-SPIN and MIRROR are identical in the case of frequent accesses. The reason behind this is that Lemma 4.5 and 4.6, by which MIRROR are advantageous to MIRROR-SPIN, become ineffective when the number of resource access segments is large.

Result II. In Figures 4.5 (a) and (b) we consider performance differences between different reasonable allocations: FF, BF, and WF, denoted by MIRROR-FF, MIRROR-BF, and MIRROR-WF, respectively. The number of resource access segments is 2 (type=M),



(a) $U_\sigma^A = 40\%$ and type=F

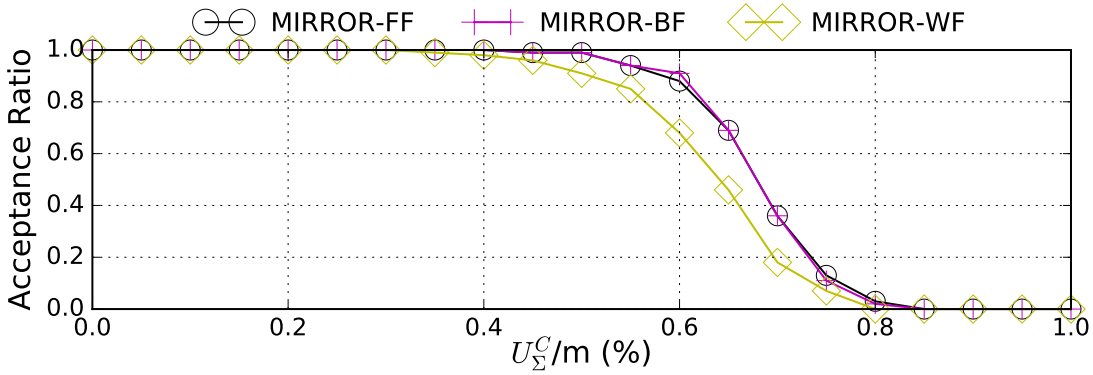


(b) $U_\sigma^A = 70\%$ and type=F

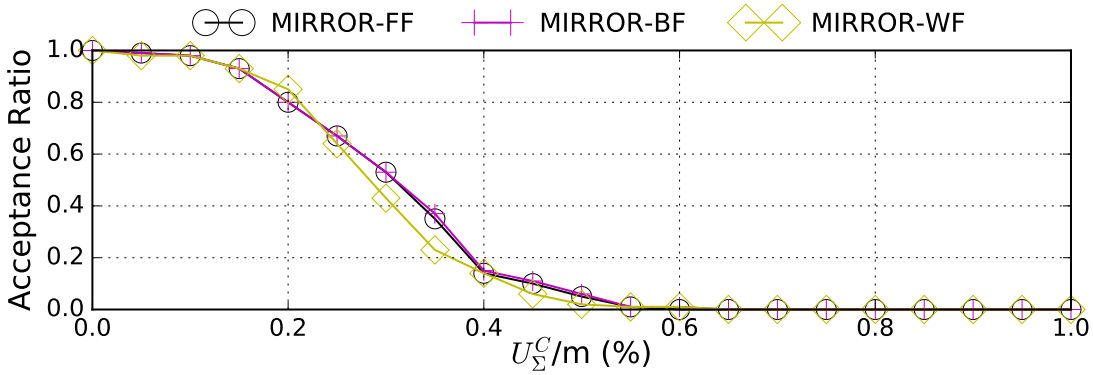
Figure 4.4: Comparison with different access utilizations U_Σ^A under frequent access (F).

which is aligned with the superblock model used in the literature [LGPWS14; PSCCT10] and the IEC61131-3 standard. This models the execution of a job into three phases, i.e., read, execution, and write. Therefore, we only consider one computation segment in Lemma 4.6. The performance of FF is identical to that of BF, and noticeably better than that of WF. In such a case, the effectiveness of MIRROR is sustainable for the case of 40% bus utilization, smoothly dropping down at around 50% core utilization.

Result III. In Figures 4.6 (a) and (b), we show the effectiveness by our proposed MIRROR (by using the FF allocation) for M/C tasks, in which each task has only one computation segment and one resource access segment [MBBMB15]. We denote the proposed approach for such tasks as MIRROR-MC, distinct from MIRROR-RAS for RAS tasks which needs two segments in Lemma 4.6. Note that in the presence of the M/C task model, we do not have to consider the arrival jitter on the interference due to memory accesses, quantified in Lemma 4.1. We first notice that without knowing the program structure, both MIRROR-SPIN and MIRROR-RAS perform relatively poorly,



(a) $U_\sigma^A = 40\%$ and type=M



(b) $U_\sigma^A = 70\%$ and type=M

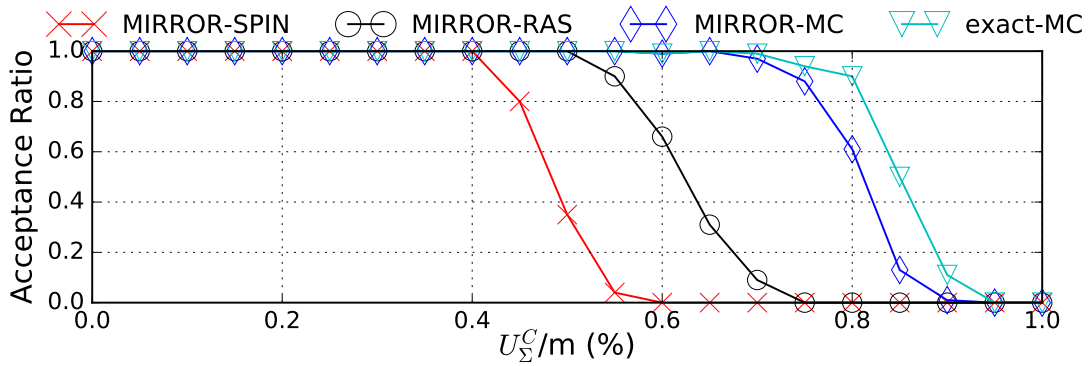
Figure 4.5: Comparison with different access utilizations U_Σ^A under moderate access (M).

even though MIRROR still outperforms MIRROR-SPIN. In the presence of M/C tasks, the proposed MIRROR achieves high schedulability, compared to exact-MC.

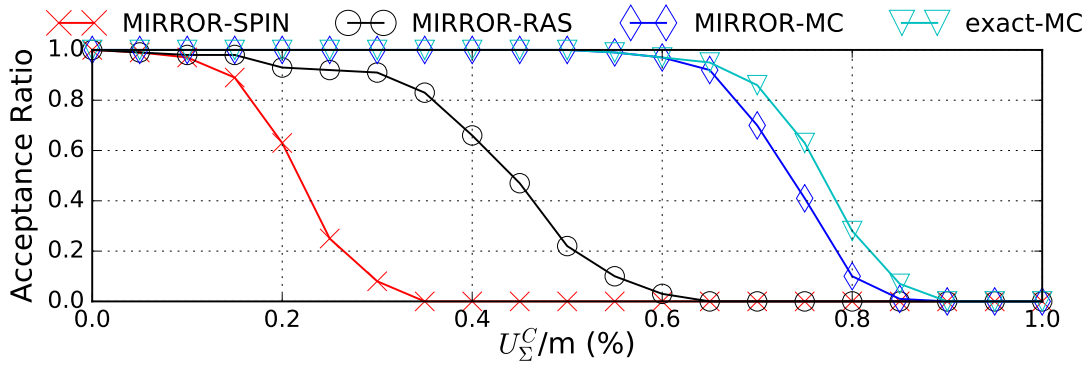
More importantly, the more the number of resource access segments, the lower the schedulability, for instance, as can be seen from MIRROR-RAS in Figure 4.6 (a) to MIRROR in Figure 4.4 (a).

4.6 SUMMARY

This chapter presents a symmetric approach to analyze the schedulability of a set of resource access sporadic real-time tasks. Our key observation is to consider *suspension-aware* response-time analysis in both core-centric and shared-resource-centric perspectives. Our schedulability analysis can also be seamlessly adopted for shared resource models, e.g., Time Division Multiple Access (TDMA), First-Come First-Serve



(a) $U_\sigma^A = 40\%$ and type=R



(b) $U_\sigma^A = 70\%$ and type=R

Figure 4.6: Comparison with different access utilizations U_Σ^A under rare access (R).

(FCFS), and Round-Robin (RR) protocols. Together with reasonable task allocations, we show that the speedup factor of our approach using fixed-priority arbitration, compared to the optimal schedule, is at most 7. This is the first result that provides a speedup factor for the scheduling problem in the presence of shared resources.

5

SYNCHRONIZATION

Classic fixed-priority scheduling algorithms for scheduling a set of periodic real-time tasks are based on the assumption that tasks are independent, i.e., the tasks do not share any other resource except the processing platform. However, this assumption does not hold and problems begin when tasks share resources.

Mars pathfinder is the first mission of NASA's Discovery program for investigating the atmosphere and other factors of Mars. Although the project was successful from a scientific point of view, its path to success was not smooth at the beginning. The system suffered from the priority inversion problem – a real-time synchronization problem – soon after it was landed on Mars on July 4, 1997. The spacecraft was resetting itself due to the priority inversion problem and caused significant delay in collecting scientific data from the surface of the red planet [Dur98].

The source of the problem was due to *priority inversion* which subsequently caused a deadline-miss of a critical task, which was identified by a watchdog timer, and finally, the action in such faulty scenario was to reset the spacecraft.

Why is Synchronization Needed and Priority Inversion Problem

The technique for access management is called *locking* or *mutual exclusion*—making sure that no two concurrent accesses to one shared resource can execute at the same time. To this end, we can set up *critical section*: code that can be executed by only one task at any given time, typically guarded by a binary semaphore Sem_k. Each critical section begins with a $wait(Sem_k)$ primitive and ends with a $signal(Sem_k)$, as shown in Figure 5.1.

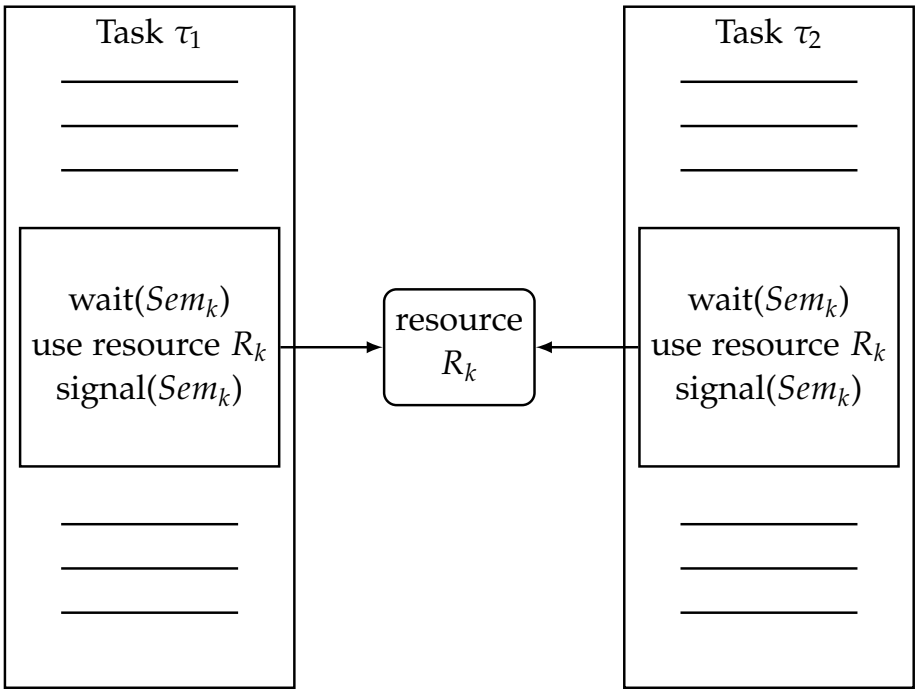


Figure 5.1: Critical section

Without furnishing a mechanism specifically dealing with resource access, a task can be arbitrarily long blocked by lower-priority tasks—called priority inversion problem. It is unavoidable under mutual exclusion that task τ_1 may be blocked by task τ_2's critical section holding resource R_k while requesting it. Since it is a direct consequence of protecting the shared resource from having concurrent accesses of tasks, such a blocking is called *direct blocking*. However, with classic scheduling algorithms, e.g., RM algorithm, a task may additionally be blocked by those lower-priority tasks that access no shared resources.

Consider a set of three sporadic tasks where task τ_1 and τ_2 share a resource; and task τ_m has only normal execution. Here, task τ_1, τ_m, and τ_2 have decreasing priorities:

- At time t_1, task τ_2 enters the critical section while it is held by no one.
- At time t_2, task τ_m is released, and preempts the execution of τ_2 due to having higher priority.

- At time t_3, tasks τ_1 is released and attempts to enter the critical section. The request from task τ_1 is not granted, as the resource is hold by task τ_2, being preempted by τ_m.
- At time t_4, task τ_m finishes its execution, giving up its time on the CPU. At the same time, task τ_2 continues to execute its critical section.
- At time t_5, after leaving the critical section, task τ_2 exits the critical section waited by task τ_1. Meanwhile, task τ_1 having highest priority starts to enter the critical section.
- At time t_6, task τ_1 finishes its execution.

Figure 5.2 illustrates the schedule. In this example, *priority inversion* occurs in the interval $[t_2, t_4]$, since highest-priority is waiting for the execution of lower-priority task τ_m having no access to shared resources. Generally speaking, such a duration can be relatively long, depending on intermediate-priority tasks, easily causing task τ_1's deadline misses.

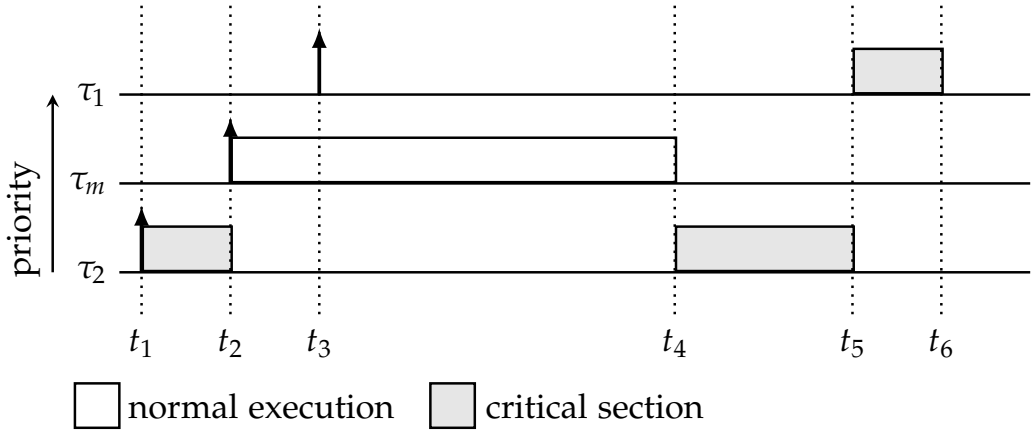


Figure 5.2: Priority Inversion

In this chapter, we study the problem of scheduling real-time tasks with synchronization. The presentation is organized as follows: In Chapter 5.1, we first introduce the task model and shared resources studied here. In Chapter 5.2, we review those consolidated techniques long developed in uniprocessor systems. In Chapter 5.3, we then move on to multiprocessor systems, where we propose resource-oriented algorithms as a backbone to schedule task with synchronization. In Chapter 5.3.2, we provide a schedulability test for a given resource-oriented task partition. Combined with the proposed schedulability test, we then propose an algorithm for deriving resource-oriented task partitioning. In Chapter 5.4.1, we show that our algorithm using PCP (after resource and task partitioning) can achieve a speedup factor $11 - 6/(m+1)$ if a task may request at most one shared resource and each job of a task may request the resource at most once during its execution, where $m \geq 2$ is the number of processors. The effectiveness of our proposed algorithm is highly supported by extensive evaluations, in Chapter 5.7.

Finally, we summarize our results in Chapter 5.8. To the best of our knowledge, this is the best result studying the problem of resource sharing on multiprocessors in terms of non-optimality The terminology and notation to be used are list in Table 5.1.

SYMBOL	MEANING
C_i	the upper bound on the amount of *non-critical-section* execution time
R_q	shared resource q
$\mathcal{RS}_i$	the set of all resources accessed by jobs of task τ_i; $\mathcal{RS}_i \subseteq \{R_1, R_2, ..., R_r\}$
$N_{i,q}$	the maximum number of requests on resource R_q by any single job of τ_i
$V_{i,q}$	the maximum (worst-case) resource usage time among all requests on resource R_q by jobs of τ_i
$A_{i,q}$	an upper bound on the total resource usage time for resource R_q by any single job of τ_i
A_i	the total resource usage time of task τ_i; $A_i = \sum_{R_q \in \mathcal{RS}_i} A_{i,q}$
U_i^A	the total resource utilization of task τ_i; $U_i^A = A_i / T_i$
U_i	the utilization of task τ_i; $U_i = (C_i + A_i)/T_i$
U_i^C	the utilization of task τ_i with non-resource execution; $U_i^C = C_i/T_i$
U^C	the total utilization of non-critical-section; $U^C = \sum_{\tau_i \in \tau} U_i^C$
$U_i^{R_q}$	the utilization of resource R_q from task τ_i; $U_i^{R_q} = A_{i,q}/T_i$
U^{R_q}	the total utilization of resource R_q; $U^{R_q} = \sum_{\tau_i \in \tau} U_i^{R_q}$
$U^{\mathcal{RS}}$	the total utilization of shared resources; $U^{\mathcal{RS}} = \sum_{R_q \in \mathcal{RS}} U^{R_q}$
$\pi(\tau_i)$	the base priority of task τ_i
$\mathcal{C}(R_q)$	the ceiling priority of resource R_q
$\wp_p$	processor p
Θ_p	the set of shared resources that are bound to the synchronization processor $\wp_p$
$dbf_i(t)$	the demand bound function of task τ_i over an interval of length t; $dbf_i(t) = \max\left(0, (\left\lfloor \frac{t-D_i}{T_i} \right\rfloor + 1) \times (A_i + C_i)\right)$
$dbf_i^C(t)$	the demand bound functions of task τ_i for non-critical-section execution; $dbf_i^C(t) = \max\left(0, (\left\lfloor \frac{t-D_i}{T_i} \right\rfloor + 1) \times C_i\right)$
$dbf_i^{R_q}(t)$	the demand bound functions of task τ_i for accesses to resource R_q; $dbf_i^{R_q}(t) = \max\left(0, (\left\lfloor \frac{t-D_i}{T_i} \right\rfloor + 1) \times A_{i,q}\right)$

Table 5.1: Synchronization: notation and terminology

5.1 TASK MODEL AND SHARED RESOURCES

We assume in this chapter that we have a set of sporadic tasks $\tau = \{\tau_1, \tau_2, ..., \tau_n\}$ sharing a set of resources $\mathcal{RS} = (R_1, R_2, ..., R_r)$.

5.1.1 *Task Model*

Each sporadic task is characterized as $\tau_i = (C_i, A_i, T_i, D_i)$, where C_i the upper bound on the amount of *non-critical-section* execution time; A_i the upper bound on the amount of *critical-section* execution time; T_i denotes the minimum inter-arrival time; and D_i the relative deadline. We note that the total worst-case execution time (WCET) of task τ_i, including critical-sections and non-critical-sections, is equal to $WCET_i = C_i + A_i$. Each job of task τ_i requires $WCET_i$ units of processing capacity within D_i time units from its release, and this processing capacity must be supplied sequentially, i.e., the job cannot be scheduled on more than one processor at any given time instant. Further, any two successive jobs of this task must be released at least T_i time units apart.

5.1.2 *Shared Resources*

A shared resource can be in-memory data, such as a set of variables, and external objects, such as files, database connections, and network connections. To prevent *race conditions*, resources shared between sets of tasks must be accessed under *mutual exclusion*: no two concurrent accesses to one shared resource are in their critical sections at the same time. We note that in this work we focus ourselves on *logical* shared resources: a piece of code to be executed on processors. Hence, no shared resources are processor-specific.

The jobs of any task can issue requests for exclusive access to shared resources $R_1, R_2, ..., R_r$. A job of task τ_i could request resource R_q on multiple occasions during its execution, and we denote as $N_{i,q} \leq N$ the maximum number of such requests by any single job of τ_i where N is an integer. Associated with these requests is the worst-case duration of time for which a job uses resource R_q. We do not put any assumption on the access patterns of resource requests, dependent on different execution paths with different execution times. Resource requests cannot be *nested*. We denote by $V_{i,q}$ the maximum (worst-case) resource usage time among all requests for resource R_q by jobs of τ_i. We denote by $A_{i,q}$ an upper bound on the total resource usage time for resource R_q by any single job of τ_i (sum of resource usage times over all requests for resource R_q). Clearly, $A_{i,q}/N_{i,q} \leq V_{i,q}$. Further, we denote by $\mathcal{RS}_i \subseteq \{R_1, R_2, ..., R_r\}$ the set of all resources accessed by jobs of τ_i, and the cardinality of this set is at most Q, i.e. $|\mathcal{RS}_i| \leq Q \leq r$.

We denote the total resource usage time of task τ_i as $A_i = \sum_{R_q \in \mathcal{RS}_i} A_{i,q}$. We assume that $C_i + A_i \leq D_i$ for any task $\tau_i \in \tau$. The utilization of resource R_q from task τ_i

is denoted by $U_i^{R_q} = A_{i,q}/T_i$. The total resource utilization of task τ_i is denoted by $U_i^A = A_i/T_i$. We denote the utilization of task τ_i as $U_i = (C_i + A_i)/T_i$. The utilization of task τ_i with non-resource execution is defined as $U_i^C = C_i/T_i$. We further assume that $U_\Sigma = \sum_{i=1}^{n} U_i \leq m$. Otherwise, it cannot be feasibly scheduled. The total utilization of resource R_q is denoted by $U^{R_q} = \sum_{\tau_i \in \tau} U_i^{R_q}$. The total utilization of non-critical-section is denoted by $U^C = \sum_{\tau_i \in \tau} U_i^C$; and the total utilization of shared resources is denoted by $U^{\mathcal{RS}} = \sum_{R_q \in \mathcal{RS}} U^{R_q}$.

In this dissertation, we focus on preemptive fixed-priority scheduling, in which each task τ_i is associated with a unique priority level, called *base priority* $\pi(\tau_i)$: $\pi(\tau_i) > \pi(\tau_j)$ if task τ_i has a higher base priority than task τ_j. We restrict our attention to *implicit-deadline* task systems.

5.2 UNIPROCESSOR SYNCHRONIZATION PROTOCOLS

To avoid such priority inversion, several approaches have been proposed to deal with the problem of scheduling tasks accessing shared resources on uniprocessors. We briefly review those approaches in the following section.

5.2.1 *Non-Preemptive Protocol*

The Non-Preemptive Protocol (NPP) is characterized by the fact that once a critical section has started to execute, it cannot be preempted until it finishes the section. This has several advantages:

- The implementation of a non-preemptive scheduling is simpler because the scheduler is inactive during the execution of a non-preemptible section.
- A set of tasks may need to share resources that must be accessed under *mutual exclusion*. It implies that once a job enters a critical section, it cannot be preempted by any access to this critical section until it finishes the critical section. This condition is automatically satisfied under non-preemptive scheduling.

The schedule of two tasks accessing a shared resource by NPP is illustrated in Figure 5.3:

- At time t_1, tasks τ_2 enters the critical section while it is hold by no one and the CPU is idle.
- At time t_2, task τ_m is released, but the request from τ_m is not granted due to τ_2's non-preemptive execution.
- At time t_3, tasks τ_1 is released, and attempts to enter the critical section. The request from task τ_1 is not granted due to τ_2's non-preemptive execution.
- At time t_4, task τ_2 finishes its execution, giving up its time on the CPU. At the same time, task τ_1 having highest priority starts to execute, entering the critical section.
- At time t_5, task τ_1 finishes its execution. Meanwhile, task τ_m starts to execute.

- At time t_6. Task τ_m finishes its execution.

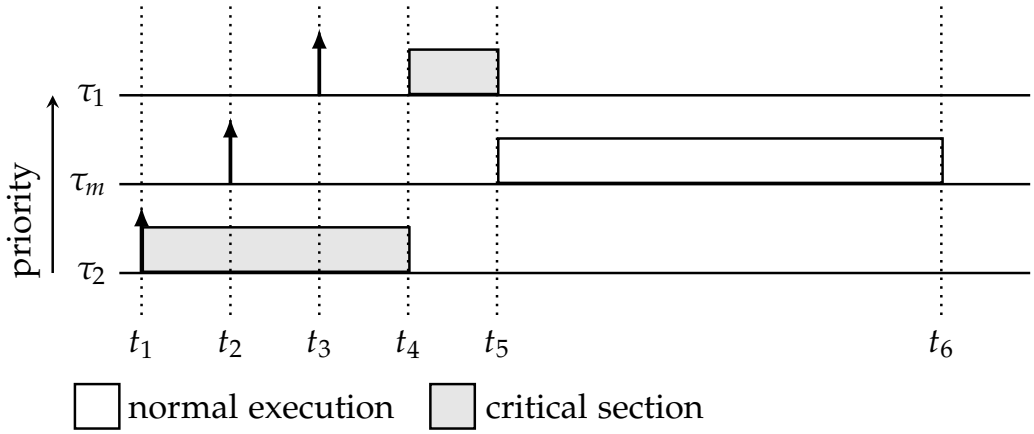


Figure 5.3: Non-preemptive protocol

Let $\pi(\tau_i)$ denote the base priority of task τ_i. It is shown in [But11] that under NPP a task may incur blocking due to any lower-priority task that accesses resources. Under the NPP, the blocking for a task τ_k being analyzed is

$$B_k = \max_{i,q}\{V_{i,q} | \pi(\tau_i) < \pi(\tau_k)\} \tag{5.1}$$

where $V_{i,q}$ is the maximum (worst-case) resource usage time among all requests for resource R_q by jobs of τ_i.

5.2.2 *Priority Ceiling Protocol*

Despite some advantages, the NPP is only appropriate for task sets with short critical sections, but falls short of avoiding some unnecessary blocking incurred by lower-priority tasks [But11]. For example, the length of critical section of lower-priority tasks, as in the interval $[t_1, t_4]$, can be arbitrarily long, unfavorably blocking task τ_m from execution.

Recall that the problem of priority inversion is induced by the grant for intermediate-priority task's execution while a higher-priority task tries to lock a shared resource that is held by the lower-priority task to be preempted. Intuitively, one can avoid such a blocking by preventing the intermediate-priority task from preempting the lower-priority task. This is the idea behind the Priority Inheritance Protocol (PIP) [SRL90]. Nevertheless, the PIP incurs another substantial blocking duration because of *chained blocking*, even if *nesting* is disallowed.

Consider a set of three sporadic tasks where task τ_1 and τ_3 share one resource; τ_2 and τ_3 share another resource. Here, task τ_1, τ_2, and τ_3 have decreasing priorities:

The schedule of the three tasks by PIP is illustrated in Figure 5.4:

- At time t_1, tasks τ_3 enters critical section A while it is hold by no one and the CPU is idle.
- At time t_2, task τ_2 is released and preempts the execution of τ_3 due to having higher priority.
- At time t_3, tasks τ_1 is released, and preempts the execution of τ_2 due to having higher priority.
- At time t_4, tasks τ_1 attempts to enter critical section A. The request from task τ_1 is not granted because critical section A is currently locked by τ_3. At the same time, τ_3 inherits the priority of τ_1 and resumes its execution.
- At time t_5, task τ_3 finishes its execution, leaving critical section A. At the same time, task τ_1 having highest priority starts to execute, entering critical section A.
- At time t_6, task τ_1 exits critical section A and attempts to enter critical section B. The request from task τ_1 is not granted because critical section A is currently locked by τ_2. At the same time, τ_2 inherits the priority of τ_1 and resumes its execution.
- At time t_7, task τ_2 exits critical section B, whereas τ_1 enters the critical section.
- At time t_8, task τ_1 finishes its execution, blocked by lower-priority critical sections twice, in the intervals $[t_4, t_5]$ and $[t_6, t_7]$.

Extended from PIP, the PCP is introduced by Sha, et. al [SRL90] to remove the unnecessary blocking while preventing deadlocks and chained blocking. The idea behind this protocol to prevent chained blocking is that a job is not allowed to enter a critical section if there is potential that the critical section can further block any higher-priority tasks that may request on the semaphores being locked. In other words, a critical section to be executed must be the one with priority strictly higher than all the tasks that may enter the critical sections being locked. Such an idea can be furnished through a priority ceiling table. Each semaphore is assigned with a ceiling priority which is the priority of highest-priority tasks that may request the semaphore. From the above exemplary task set, semaphores are assigned the following priority ceilings:

Table 5.2: Example of Ceiling Priority Table

Semaphore	$\mathcal{C}(R_q)$
R_A	1
R_B	1

The schedule by PCP is illustrated in Figure 5.5:

- At time t_1, tasks τ_3 enters critical section A while it is hold by no one and the CPU is idle.
- At time t_2, task τ_2 is released and attempts to enter critical section B. However, task τ_2 is blocked by the protocol because its priority is not higher than $\mathcal{C}(R_A)$ so as to enter critical section B. At the same time, task τ_3 is running with τ_2's priority, preventing any intermediate blocking.

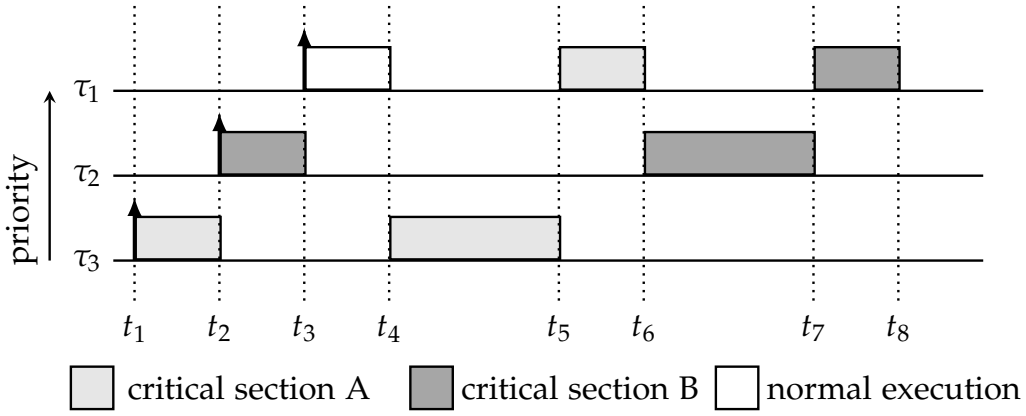


Figure 5.4: Chained blocking under PIP

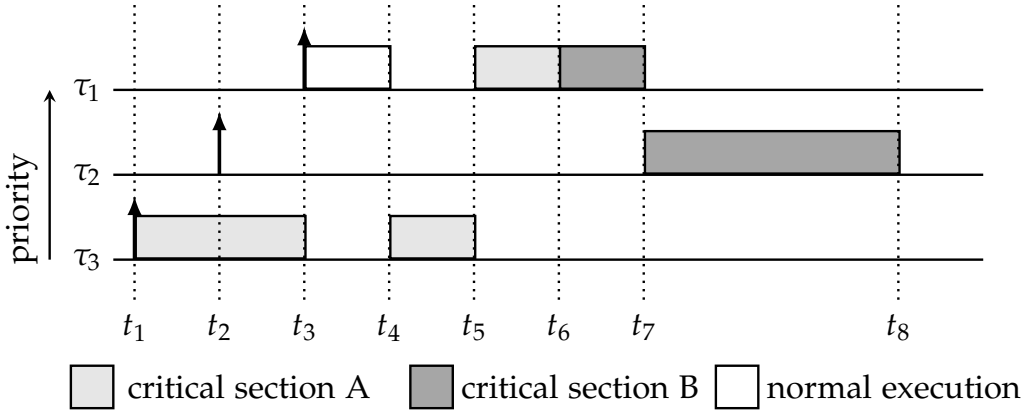


Figure 5.5: Example of PCP

- At time t_3, tasks τ_1 is released, and preempts the execution of τ_3 due to having higher priority.
- At time t_4, tasks τ_1 attempts to enter critical section A. The request from task τ_1 is not granted because critical section A is currently locked by τ_3. At the same time, τ_3 inherits the priority of τ_1 and resumes its execution.
- At time t_5, task τ_3 finishes its execution, leaving critical section A. At the same time, task τ_1 having highest priority starts to execute, entering critical section A.
- At time t_6, task τ_1 exits critical section A and enters critical section B.
- At time t_7, task τ_1 finishes its execution, blocked for the duration of critical section A of task τ_3. At the same time, τ_2 enters critical section B.
- At time t_8, task τ_2 finishes its execution.

In fact, under PCP a job can be blocked for at most the duration of the longest critical section among those that can block the job [SRL90]. Formally, the blocking under PCP can be computed as follows:

$$B_k = \max_{i,q}\{V_{i,q} | \pi(\tau_i) < \pi(\tau_k), \mathcal{C}(R_q) \geq \pi(\tau_k)\} \tag{5.2}$$

where $\mathcal{C}(R_q)$ is the *ceiling* priority of shared resource R_q under PCP.

In the following theorem, we show that PCP associated with the TDA test by [ABRTW93] under RM scheduling offers non-trivial quantitative guarantees:

Theorem 5.1. *The speedup factor of the PCP associated with the TDA test by [ABRTW93] under RM scheduling in uniprocessor systems is* 2.

Proof. We prove this theorem by showing that a task set deemed schedulable by the priority ceiling protocol (PCP) using TDA by [ABRTW93] under RM scheduling is also not schedulable by any scheduling polices on a speed-1/2 uniprocessor.

Suppose task τ_k is the task that misses its relative deadline. Let $B_k \neq 0$ be the longest critical section of accessing resource R_b of lower-priority tasks τ_{lp} that blocks task τ_k. Note that if $B = 0$, we can conclude the utilization is at least 69% by the Liu and Layland bound, which by taking its multiplicative inverse implies a speedup factor 1.44. Recall that under the PCP a task can only be blocked by lower-priority tasks' critical sections that are accessed by a task with higher priority than τ_k. Thus, under RM scheduling there must exist a task τ_a with a relative deadline $D_a \leq D_k$ that may request on R_b.

We now release task τ_{lp} alone, at time $-\epsilon$, right before time 0, starting to execute the critical section B_k of accessing R_b. We then release all the other tasks at time 0 and let the following jobs execute in their worst case and release as soon as possible. To complete τ_a's execution under mutual exclusion, it is necessary for the task set being schedulable that B_k together with all the necessary demands are finished at τ_a's relative deadline D_a and afterwards. This means that by Definition 2.5 and Eq. (2.2), it must be

$$\forall t \geq D_a, \quad B_k + \sum_{\tau_i \in \tau \setminus \{\tau_{lp}\}} dbf_i(t) \leq t \tag{5.3}$$

The failure of RTA by [ABRTW93] implies that

$$\forall 0 < t \leq D_k, \qquad B_k + \sum_{\tau_i \in hep(k)} \left\lceil \frac{t}{T_i} \right\rceil (C_i + A_i) > t. \tag{5.4}$$

where $hep(k)$ is the set of task τ_k together with tasks having higher-priority than τ_k. Let us instantiate this inequality for $t \leftarrow D_k$:

$$
\begin{aligned}
& B_k + C_k + A_k + \sum_{\tau_i \in hep(k)} \left\lceil \frac{D_k}{T_i} \right\rceil (C_i + A_i) > D_k \\
& \{\text{Thanks to RM scheduling, } hep(k) = \{\tau_i | D_i \leq D_k\}\} \\
\Rightarrow & B_k + \sum_{\tau_i : D_i \leq D_k} \left\lceil \frac{D_k}{T_i} \right\rceil (C_i + A_i) > D_k \\
& \{\forall \tau_i \text{: } D_i \leq D_k, 2dbf_i(D_k) \geq \left\lceil \frac{D_k}{T_i} \right\rceil (C_i + A_i)\} \\
\Rightarrow & B_k + 2 \sum_{\tau_i : D_i \leq D_k} dbf_i(D_k) > D_k \\
\Rightarrow & 2(B_k + \sum_{\tau_i \in \tau \setminus \{\tau_{lp}\}} dbf_i(D_k)) > D_k
\end{aligned}
$$

which implies that by Eq. (5.3) and $D_k \geq D_a$ this task set is not schedulable by any scheduling policies on a speed-1/2 uniprocessor. Hence, we here conclude this theorem. □

5.3 MULTIPROCESSORS

To handle the synchronization problem when tasks share resources in real-time systems, a wide variety of real-time locking protocols have been developed. Briefly speaking, a real-time locking protocol is used to limit priority inversions [SRL90; BA10], such that higher-priority tasks incur priority inversions only if lower-priority tasks execute in critical sections. Recent analysis and comparisons of semaphore protocols may be found in [Bra13] for partitioned scheduling (e.g., the DPCP [RSL88] and the MPCP [Raj90].) and in [YWB15] for global scheduling (e.g., the global PIP). To support nested critical sections, Ward and Anderson [WA12; WA13] recently introduced the Real-time Nested Locking Protocol (RNLP) [WA12], which adds supports for fine-grained nested locking on top of non-nested protocols.

Further, to ensure mutual exclusive access to shared resources, tasks either self-suspend (under semaphore protocols such as the DPCP [RSL88] and the MPCP [Raj90]) or busy wait (under spin-based protocols such as the MSRP [GLN01]) when blocked on shared resources. Busy waiting has been shown to be efficient when critical sections are short [Bra11; HZNLD14], but the resulting loss of processor service must be accounted for. More recently, Burns and Wellings [BW13] proposed a variant of the MSRP, the Multiprocessor resource sharing Protocol (MrsP) [BW13], to exploit the spinning cycles of one task to help other tasks make progress. Wieder and Brandenburg [WB13b] presented a fine-grained analysis for several types of spin locks. In contrast, self-suspensions are more efficient for long critical sections [Bra11; LNR09]. Moreover, in

some distributed-configured scheduling systems, such as the designated [RSL88] or dedicated [HLK11] synchronization frameworks, jobs self-suspend on host processors, waiting for resource service on remote processors, is a natural fit for the scheduling strategy. In this work, we adopt the suspension-based methodology for resource sharing.

As far as task partitioning is concerned, Lakshmanan et al. [LNR09] presented a synchronization-aware partitioned (SAP) heuristic tailored to the MPCP [Raj90], which organizes tasks sharing common resources into groups and attempts to assign each group of tasks to the same processor. Following the same principle, Nemati et al. [NNB10] presented a blocking-aware partitioning method that uses advanced cost heuristic to split task group when an entire group fails to be assigned on one processor. In subsequent work, Hsiu et al. [HLK11] proposed a dedicated-core framework to separate the execution of critical sections and normal sections, and employed an priority-based RPC-like mechanism for resource sharing, such that each higher-priority request can be blocked by at most one lower-priority request. More recently, Wieder and Brandenburg [WB13a] use integer linear programming to solve the partitioning problem in the presence of spin locks.

From an algorithmic optimality point of view, Brandenburg and Anderson [BA10] are first to study the multiprocessor real-time locking problem. It is shown that FIFO-based locking protocols, such as the Flexible Multiprocessor Locking Protocol (FMLP) [BLBA07], the Generalized FIFO Multiprocessor Locking Protocol (FMLP$^+$) [Bra14a], and the Distributed FIFO Locking Protocol (DFLP) [Bra14b], are asymptotically optimal in terms of maximum priority-inversion blocking. Andersson and Easwaran [AE10] presented an virtualization-based G-EDF scheduling with shared resources, which guarantees a $12(1 + 3r/(4m))$ competitive ratio on a platform comprising m identical processors, where each task uses at most one of the r shared resources and each job may request the resource at most once, whereas the presented algorithm in this work achieves a constant speedup factor 11, without using any virtualization.

5.3.1 *Resource-Oriented Algorithm*

It is known that $\Omega(m)$ pi-blocking is unavoidable on multiprocessor systems, provided that any task partition is given, as shown in [BA10]. Therefore, as can be seen in the literature [Raj90; BW13; LNR09] in which task partitioning is given, a higher-priority job under MPCP or MrsP may suffer from the so-called *chained blocking*: a higher-priority job can be blocked for the duration of either m, for MrsP, or n, for MPCP, critical sections. The chained blocking has significant influence on the schedulability analyses.

The spirit of resource-oriented partitioned scheduling is to position the resource accesses as the first-class citizens in the system when we consider task and resource

partitioning so as to keep the blocking time as short as possible. In principle, the resource-oriented algorithm works as follows:

- Each shared resource should be assigned on one processor, and the critical sections guarded by this shared resource are executed on the designated processor.
- The non-critical sections of a task are executed on its designated processor (that can be different from the processors executing the critical sections of the task).

Note that, in general, we still follow the partitioned scheduling policy for the non-critical sections, but the critical sections are proposed to be executed on the designated synchronization processors. A similar concept to execute the critical sections guarded by one shared resource on a designated processor can also be found in Distributed Priority Ceiling Protocols (DPCP) developed by Rajkumar et al. [RSL88]. But, there has been no further elaboration in the literature to provide evidence on how to assign tasks and resources among multiple processors.

A clear advantage of resource-oriented scheduling by bounding all accesses to each shared resource to the same processor is that it avoids chained blocking: each access to a shared resource of a task can be only blocked by those low-priority accesses that are assigned on the same processor.

Runtime overhead. The additional overheads in resource-oriented partitioned scheduling are just identical to semi-partitioned scheduling (which is less than global scheduling) due to the critical-section migration, since we only specifically migrate critical sections. The migration can be managed directly from the operating systems, in which a request to lock a shared resource can be handled directly (by the resource manager residing) on the designated processor. The resource-oriented partitioned scheduling has more overhead than the original partitioned scheduling since the critical sections have to be executed on the remote processors. Even though this may seem to be a drawback, we should also keep in mind the implementations of MPCP and MrsP also require an arbiter to decide which critical section is granted to be executed, which is also not free.

Compared to traditional partitioned scheduling, resource-oriented partitioned scheduling has additional overheads. In our approach, the execution of a task in the resource-oriented scheduling is split into more than one processor, similar to *semi-partitioned* scheduling. From the implementation's point of view, the resource-oriented partitioned scheduling can benefit from the pre-planned nature of *push-migrations*: the jobs to be scheduled on the next processor are statically determined (more details can be found in [BBA11]). This gives us several advantages:

- As which critical-section executions will migrate and also among which processors are known beforehand, cache-related preemption and migration delay (CPMD) accounting is task-specific and hence less pessimistic.
- Furthermore, since push-migrations can be implemented with mostly-local state, migrations of the resource-oriented scheduling entail less overhead and are easier to implement.

However, for example, MrsP requires a lock holder to progress within the critical section of a task waiting for a resource already locked by a preempted task executing on a different processor. Therefore, the migration of jobs across partitions must be furnished, and the next processor is dynamically determined at runtime, the so-called *pull-migrations*. Such migrations imply much higher overheads than push-migrations at runtime.

In addition, based on resource-oriented partitioned scheduling, we actually break down the problem of scheduling tasks with shared resources on multiprocessors into smaller uniprocessor sub-problems, on which standard consolidated techniques can still be applied (on each synchronization processor), e.g., NPP and PCP in uniprocessor systems, denoted by R-NPP and the R-PCP for the rest of this paper, respectively. Moreover, unlike uniprocessor systems, the unnecessary blocking incurred by the NPP (compared to PCP) could be removed by partitioning tasks properly on multiprocessor systems. This might in turns enable us to use the R-NPP due to its simplicity of implementations, where maintaining a list of currently locked semaphores and priority ceiling orders during runtime is not needed.

Inspired by these, in this work we aim at obtaining a better understanding of this seemingly promising scheduling along with task and resource partitioning.

5.3.2 *Schedulability Analysis*

In this section, we present the response time analysis and the schedulability test for a specific task τ_k. We assume that the priority assignment for fixed-priority preemptive scheduling is already specified and the tasks are already mapped onto the processors. Note that as mentioned in Section 3.2, we will focus on the case that $N = 1$ and $Q = 1$ in this section. In Section 5.3.2.1, we begin with a simpler case in which the synchronization processors are used only for executing critical sections and application processors are used only for executing non-critical sections. We will consider running non-critical sections on some synchronization processors in Section 5.3.2.2.

We implicitly assume that the higher-priority tasks already meet their deadlines while analyzing task τ_k. Since we use partitioned scheduling for the non-critical sections, we define $\mathbf{hpl}(k)$ as the set of the higher-priority tasks that are assigned on the same processor where task τ_k is assigned to run its non-critical sections. Suppose that a job of task τ_k arrives at time t_0 and has an absolute deadline at time $t_0 + D_k$. To analyze whether we can finish the job in time, we need to analyze the interference from the higher-priority tasks $\mathbf{hpl}(k)$ on the *local* application processor and that due to resource accesses on the *remote* synchronization processors in the time interval $[t_0, t_0 + D_k)$. Without loss of generality, we can set t_0 to 0.

5.3.2.1 *Response Time Analysis*

Here, in this subsection, suppose that the synchronization processors are used only for executing critical sections and application processors are used only for executing non-critical sections. Under resource-oriented partitioned scheduling, every time a task requests a shared resource mapped onto a remote processor, the task suspends itself on the local processor until the request is complete. With the suspension-based scheduling, no *critical instant theorem*, at which the worst-case behavior in analyzing a task is concretely captured, has been yet established. To cover the worst-case behavior for the non-existence of critical instant, accounting for the so-called *carry-in* jobs, that may be carried into the interval of our interest, i.e., $[t_0, t_0 + t)$, is commonly used in schedulability analysis, as shown in [LC14; HCZL15; BAHCN15; CNH16]. Moreover, the interference due to such jobs can be more precisely quantified as *jitter*. Importantly, such jitter has to be carefully incorporated. Several misconceptions with incorrect quantifications of jitter for self-suspending task systems were reported in a recent survey paper by Chen et al. [Che+16]. Nevertheless, it has been reported in [BAHCN15; CNH16] that using $RT_i - C_i$ as the jitter is safe, where RT_i is the worst-case response time of task τ_i, like global scheduling in multiprocessor systems [BC07b].

Lemma 5.1. *The cumulative non-critical-section execution time of a task τ_i in $\mathbf{hpl}(k)$ in the time interval $[t_0, t_0 + t)$ is upper bounded by:*

$$W_i(t) = \left\lceil \frac{t + RT_i - C_i}{T_i} \right\rceil C_i \tag{5.5}$$

Lemma 5.2. *The cumulative execution time of a task τ_i for accesses to resource R_q in the time interval $[t_0, t_0 + t)$ is upper bounded by:*

$$E_{i,q}(t) = \left\lceil \frac{t + RT_i - A_{i,q}}{T_i} \right\rceil A_{i,q} \tag{5.6}$$

where RT_i (in the above two lemmas) is the worst-case response time of task τ_i and $RT_i \leq T_i$.

Proof. The proofs are omitted since they are identical to the proofs in the literature. See [BAHCN15; CNH16] for details. □

Under multiprocessor partitioned scheduling for sporadic tasks without resource sharing, all the jobs generated by a task are constrained to execute only upon the processor to which the task is assigned. It thus follows that only those tasks that are executed on the same processor have to be taken into consideration when we analyze the schedulability of a task. However, in the presence of resource sharing, the time a

task waits for executing itself is determined by not only the interference caused by local tasks but also resource accesses to the processors on which the task may request for shared resources.

Let $S(t)$ be the upper bound on the demand in time interval $[t_0, t_0 + t)$ on the processor at which task τ_k executes its *critical section*. Let $X(t)$ be the upper bound on the demand in time interval $[t_0, t_0 + t)$ on the processor at which task τ_k executes its *non-critical section*.

Then, if the job of task τ_k released at time t_0 cannot finish its execution at time $t_0 + t$, it must be the case that $S(t) + X(t) > t$. Equivalently, the negation of this condition is sufficient to upper bound the response time of a task. As a result, the classic TDA can be extended as follows:

Theorem 5.2. *Suppose that $S(t)$ and $X(t)$ are both safe upper bounds. The smallest t satisfying the following*

$$S(t) + X(t) \leq t \tag{5.7}$$

is a safe upper bound on the response time of task τ_k if $t \leq T_k$.

We now explain how to compute $X(t)$ and $S(t)$.

Computing $X(t)$. The demand on the local processor is only dependent on the execution of non-critical-section codes of tasks bound to the same processor, which can be elaborated as follows:

$$X(t) = I^{nRes}(t) + C_k \tag{5.8}$$

where the terms are as described below:

- C_k denotes an upper bound on the amount of *non-critical-section* execution from task τ_k itself.
- $I^{nRes}(t)$ denotes an upper bound on the amount of interferences from higher-priority non-critical-section execution with an interval of length t on the same processor.

From Lemma 5.1, it is safe to set

$$I^{nRes}(t) = \sum_{\tau_i \in \mathbf{hpl}(k)} W_i(t) \tag{5.9}$$

where $\mathbf{hpl}(k)$ is the set of the tasks with priority higher than τ_k on the local application processor where τ_k runs its non-critical sections.

Computing $S(t)$. The demand on the synchronization processor that runs the critical section of task τ_k can be elaborated as follows:

$$S(t) = A_k + B_k + I_k^{rRes}(t) \tag{5.10}$$

where the terms are as described below:

- A_k denotes an upper bound on the amount of *critical-section* execution from task τ_k itself.
- B_k denotes an upper bound on the amount of blocking from lower-priority critical-section accesses bound on the same synchronization processor.
- $I_k^{rRes}(t)$ denotes an upper bound on the amount of interferences from higher-priority synchronization execution with an interval of length t.

Under the resource-oriented scheduling, the demand on a synchronization processor in time interval $[t_0, t_0 + t)$ is contributed by all the resources that are bound to the same synchronization processor. Let Θ_r be the set of shared resources that are bound to the synchronization processor $\wp_r$ on which task τ_k executes its critical section.

The B_k parameter is similar to the one in uniprocessor systems, but dependent on only those low-priority accesses to the same synchronization processor: under the R-NPP

$$B_k = \max_{i,q}\{V_{i,q} | \pi(\tau_i) < \pi(\tau_k), R_q \in \Theta_r\} \tag{5.11}$$

and under the R-PCP

$$B_k = \max_{i,q}\{V_{i,q} | \pi(\tau_i) < \pi(\tau_k), \mathcal{C}(R_q) \geq \pi(k), R_q \in \Theta_r\} \tag{5.12}$$

From Lemma 5.2, it follows that

$$I_k^{rRes}(t) = \sum_{R_q \in \Theta_r} \sum_{\tau_i \in hp(k)} E_{i,q}(t) \tag{5.13}$$

5.3.2.2 *Non-Critical Sections on Synchronization Processors*

A synchronization processor can also execute non-critical sections. In our design, if there are non-critical sections assigned to be executed on a synchronization processor, the execution of the non-critical-sections has the priority *lower* than any of the critical-sections. Suppose that task τ_k is assigned to a synchronization processor. In this case, when calculating $X(t)$, we also need to consider the interference from critical-section executions running at higher priority:

$$X(t) = I^{nRes}(t) + I^{lRes}(t) + F_k \tag{5.14}$$

where $I_k^{lRes}(t)$ denotes an upper bound on the amount of interferences from local critical-section executions of tasks with their *base* priority higher than τ_k with an interval of length t, i.e. $\pi(\tau_i) > \pi(\tau_k)$; and F_k denotes an upper bound on the amount of interferences from local critical-section executions of tasks, denoted by a set $lp(k)$, with their *base* priority lower than τ_k, i.e. $\pi(\tau_i) < \pi(\tau_k)$, i.e., $lp(k) = \{\tau_i \in \tau | \pi(\tau_i) < \pi(\tau_k)\}$.

Let Θ_l be the set of shared resources that are bound to the processor $\wp_l$ on which task τ_k executes its non-critical-section codes. We have

$$I^{lRes}(t) = \sum_{R_q \in \Theta_l} \sum_{\tau_i \in hp(k)} E_{i,q}(t) \tag{5.15}$$

Algorithm 3: Linear Search

input : A set of n tasks τ, m processors $\wp$, and r resources $\mathcal{RS}$
output: Resources allocations Θ, task allocations Γ and the feasibility of system τ
for $m^R = 1, ..., min(m, r)$ **do**
  if *WFD* (Q, m^R) *returns "feasible allocation"* **then**
    $\Theta \leftarrow$WFD (Q, m^R);
  else
    continue;
  if *FFRM*(τ,Θ) *returns "feasible allocation";*
  then
    return "feasible system";
return "infeasible system";

To calculate F_k, we can again use Lemma 5.2 directly if we are aware of the worst-case response time of the lower-priority task under the assumption that $RT_i \leq T_i$ for each $\tau_i \in lp(k)$. Later in Section 5.4, we will not be able to know RT_i of a lower-priority task τ_i in $lp(k)$ when testing the schedulability of task τ_k. However, it can be safely assumed that $RT_i \leq T_i$ and this predicate $RT_i \leq T_i$ for $\tau_i \in lp(k)$ will be verified later when we test the schedulability of task τ_i. Therefore, we have

$$F_k = \sum_{R_q \in \Theta_l} \sum_{\tau_i \in lp(k)} \left\lceil \frac{t + T_i - A_{i,q}}{T_i} \right\rceil A_{i,q} \tag{5.16}$$

As $S(t)$ and $X(t)$ have been computed, we present our schedulability in the following theorem:

Theorem 5.3. *For each task τ_k in τ, if there exists $0 \leq t \leq D_k$ s.t. Eq. (5.7) holds, then task τ_k with resource sharing is schedulable in fixed-priority partitioned scheduling under a given task partitioning.*

5.4 TASK AND RESOURCE ALLOCATIONS

In this section we present our algorithm that determines a set of synchronization processors and that allocates both shared resources and tasks onto processors. The intuition underlying the proposed algorithm is that under the resource-oriented scheduling, once shared resources are mapped onto the synchronization processors, the blocking time incurred by lower-priority tasks can be determined, irrespective of their task allocations. Hence, by initially sorting the tasks in an order of decreasing priorities (non-decreasing order of relative deadlines), any task being assigned will not

Algorithm 4: Worst-Fit Decreasing (WFD)

input : A set resource $\mathcal{RS}$ and m^R identical processors for synchronization
output: Resources allocations Θ
$\Theta_j \leftarrow \emptyset, \forall j = 1, 2, \ldots, m$;
sort the r shared resources $\mathcal{RS}$ with non-increasing U^{R_q};
for $R_q \in \mathcal{RS}$ **do**
    `// put these processors furthest`
    for $h = m - m^R + 1, \ldots, m$ **do**
        calculating the load $\sum_{R_j \in \Theta_h} U^{R_j}$;
    `// least utilization first`
    assign R_q to the m^R processor $\wp_h$ with the *minimum* load;
    if $U^{R_q} + \sum_{R_j \in \Theta_h} U^{R_j} > 1$ **then**
        return "infeasible allocation";
    else
        $\Theta_h \leftarrow \Theta_h \cup \{R_q\}$;
return "feasible allocation",Θ;

jeopardize the schedulability of the tasks that have been successfully assigned onto processors:

1. First, we iteratively determine a *configuration* of initializing a set of processors to be used as synchronization processors. From a schedulability point of view, the reduction in the number of the synchronization processors is a tradeoff between
    - an increase on the time spent on the execution of critical sections on the synchronization processors, and
    - a reduction on the time spent on the execution of non-critical sections on the application processors.

    As each resource is bound to one processor, at most $min(m, r)$ configurations of processors need to be checked until either a feasible system is found or does not exist. In each configuration of processors, resources and tasks are respectively allocated by *Worst-Fit Decreasing* (WFD) algorithm and *First-Fit Rate-Monotonic* (FFRM) algorithm, presented later.

2. The resources are ordered in a list in a non-increasing order of their utilization. The algorithm attempts to allocate each resource onto the synchronization processor with the *least* load, called WFD. We note that this is a well-known strategy for the bin-packing problem. The intuition underlying the WFD is that by distributing resources evenly, it is sensible to reduce the time spent by tasks waiting for resource accesses remotely. These synchronization processors are

preferably put as far as possible from the processors on which tasks will be allocated later.

3. Tasks are prioritized and sorted in the order of non-decreasing relative deadlines, i.e., $D_1 \leq D_2 \leq ... \leq D_n$. That is, for implicit-deadline task systems, we use the well-known rate-monotonic (RM) policy for assigning the base priorities of the tasks. Then, the algorithm considers to assign (the non-critical sections of) the tasks to processors from the highest base priority to the lowest base priority. Our algorithm here, called First-Fit Rate-Monotonic (FFRM), places the task in the first processor that can accommodate the task according to Theorem 5.3. In addition, the algorithm heuristically places the task in the processors to which no shared resources are bound. If no such processors can accommodate this task, then the algorithm will also check the feasibility of putting it in those processors initialized to be synchronization processors.

We denote our algorithm as Algorithm ROP-PCP when PCP is used and Algorithm ROP-NPP when NPP is adopted.

Runtime complexity. In attempting to find a configuration for synchronization processors, we need at most m rounds. We note that the overall sorting time of Algorithm 4 and Algorithm 5 in all rounds can be amortized to $\mathcal{O}(r \log r + n \log n)$ by using appropriate data structures. In each round, Algorithm 4 runs in time complexity $\mathcal{O}(r \log m)$ by maintaining the processor utilization with a heap data structure, and Algorithm 5 requires $\mathcal{O}(mnD_n)$ for checking whether a task can fit into one processor according to Theorem 5.3, where D_n is the longest relative deadline among tasks. Overall, our algorithm runs in $\mathcal{O}(r \log r + n \log n + m(r \log m + mnD_n))$, which is in pseudo-polynomial time complexity.

5.4.1 *Speedup Factor under RM*

In this section, we obtain a speedup factor for Algorithm ROP-PCP under rate-monotonic (RM) scheduling, in which $D_i = T_i$ for every task $\tau_i \in \tau$. The approach is as follow. We identify the smallest value of $x \geq 1$ for which we can prove that any task set that is feasible upon a platform comprising m unispeed processors is deemed to be schedulable by the RMFF upon a platform in which each processor is at least x times as fast. Consequently, we can conclude that the value x is a processor speedup bound.

In the following lemma, we first provide necessary conditions for any optimal scheduling, which are based upon the concept of demand bound functions (dbf).

Lemma 5.3. *Any implicit-deadline task system τ that is feasible upon a platform comprised of m processors must satisfy*

$$U^C + U^{\mathcal{RS}} \leq m \text{ and } \forall \tau_i \in \tau, \quad U_i \leq 1 \tag{5.17}$$

Algorithm 5: First-Fit Rate-Monotonic (FFRM)

input : A set τ of tasks, m^C identical processors for non-synchronization, resources allocation Θ
output: Task allocations Γ_j and the feasibility of system τ

```
sort the given n tasks in τ s.t. D_1 ≤ D_2 ≤ ··· ≤ D_n;
Γ_j ← ∅, ∀j = 1, 2, ..., m;
for k = 1, 2, ..., n do
    for p = 1, ..., ..., m do
        if task τ_k is schedulable according to Theorem 5.3 then
            Γ_p ← Γ_p ∪ {τ_k}; // assign τ_k to processor p
            break ;
    if τ_k cannot fit any processor in the above loop then
        return "infeasible allocation";
return "feasible allocation";
```

and $\forall \tau_k \in \tau$, $\forall R_q \in \mathcal{RS}_k$

$$\frac{\max_{\tau_i:D_i>D_k} V_{i,q} + \sum_{\tau_i:D_i\leq D_k} dbf_i^{R^q}(D_k)}{D_k} \leq 1 \tag{5.18}$$

Proof. If τ_i generates jobs with execution time exactly $C_i + A_i$, it is necessary for meeting all the deadlines of task τ_i that $(C_i + A_i)/D_i \leq 1$. Moreover, in order for the task system to be schedulable by any algorithm upon a platform comprised of m processors, it is necessary that

$$U^C + U^{\mathcal{RS}} \leq m \tag{5.19}$$

Suppose that task τ_{l_p} is the task having a relative deadline larger than D_k with the longest critical section of accessing R_q. Let task τ_{l_p} be released alone at time $-\epsilon$, right before time 0, starting to execute this longest critical section of accessing R_q. Suppose that all the tasks having relative deadlines $D_i \leq D_k$ that may request on R_q along with τ_k are released at time 0, and each task τ_i generates jobs *only* requesting on R_q and the jobs are released as soon as possible. Recall these executions are subject to exclusion constraints, and therefore must be serialized. To complete τ_k's execution under mutual exclusion feasibly, it is necessary to finish τ_{l_p}'s critical section together with all the necessary demands at relative deadline D_k:

$$\max_{\tau_i:D_i>D_k} V_{i,q} + \sum_{\tau_i:D_i\leq D_k} dbf_i^{R^q}(D_k) \leq D_k \tag{5.20}$$

Hence, we here conclude this lemma. □

We will derive a speedup factor for the RMFF. Before that, we first need the following two lemmas:

Lemma 5.4. *For $t \geq D_i$,*

$$3dbf_i^C(t) \geq W_i(t) \tag{5.21}$$

and

$$3dbf_i^{R^q}(t) \geq E_{i,q}(t) \tag{5.22}$$

Proof. The proof is similar to that of Theorem 4.3. We here omit the details. □

Lemma 5.5. *Suppose that $U_i \leq 1$ for every task $\tau_i \in \tau$. Given m^R synchronization processors, the utilization of shared resources on each processor under WFD is at most*

$$1 + \frac{U^{\mathcal{RS}} - 1}{m^R} \tag{5.23}$$

Proof. The proof is similar to the scheduling algorithms of the *makespan* problem. Let L^* be the maximum utilization among the m^R processors to schedule shared resources under WFD. Let u_ℓ be the utilization of the last resource mapped onto L^*. The WFD algorithm assigns each resource onto the processor with the least load. It follows that

$$L^* - u_\ell \leq \frac{U^{\mathcal{RS}} - u_\ell}{m^R} \Rightarrow L^* \leq \frac{U^{\mathcal{RS}} - u_\ell}{m^R} + u_\ell \leq \frac{U^{\mathcal{RS}} - 1}{m^R} + 1 \tag{5.24}$$

where the last inequality is due to the condition $u_\ell \leq 1$. □

In the following theorem, we show that the speedup factor of our algorithm is $11 - 6/(m+1)$ when $N = 1$ and $Q = 1$, irrespective of how many shared resources, r, are present.

Theorem 5.4. *The speedup factor of the proposed resource-oriented partitioned scheduling algorithm ROP-PCP is $11 - \frac{6}{m+1}$ when $m \geq 2$, $N = 1$, and $Q = 1$ under the resource-oriented scheduling using PCP in Section 5.3.1.*

Proof. We prove this theorem by showing that any task set that is feasible upon a platform comprising m unispeed processors is deemed to be schedulable by Algorithm ROP-PCP upon a platform in which each processor is at least $11 - 6/(m+1)$ times as fast.

Suppose that Algorithm ROP-PCP fails to obtain an allocation for τ: there exists task τ_k which cannot be mapped on to any processor by Algorithm FFRM. Note that due to the sorting of the tasks in Algorithm ROP-PCP, all the tasks before task τ_k mapped onto processor have been ensured $RT_i \leq D_i = T_i$ for $i = 1, 2, \ldots, k-1$. Since τ_k fails the test of Theorem 5.3, it must be the case that for every processor

$$S(D_k) + X(D_k) > D_k \tag{5.25}$$

The failure of Algorithm ROP-PCP implies that τ_k also fails the test of Theorem 5.3 on each of the $m^C = m - m^R$ application processors.

Summing over all m^C such processors and after reformulation, we obtain

$$\frac{C_k + A_k}{D_k} + \frac{B_k}{D_k} + \frac{I_k^{rRes}(D_k)}{D_k} + \frac{\sum_{\tau_i \in hp(k)} W_i(D_k)}{m^C D_k} > 1 \tag{5.26}$$

Let $B_k \neq 0$ be the longest critical section of accessing resource R_b of tasks having relative deadline larger than D_k that blocks task τ_k's critical-section execution on the designated processor. We now consider two separate cases:

- τ_k **may request on** R_b. From Lemma 5.3, it is necessary for task τ_k that $\left(B_k + \sum_{\tau_i : D_i \leq D_k} dbf_i^{R_b}(D_k)\right) / D_k \leq 1$. Clearly, it follows that $B_k \leq D_k$.
- τ_k **does not request on** R_b**.** Recall that under PCP a task can only be blocked by lower-priory tasks' critical sections that are accessed by a task with an equal or higher priority than τ_k. Thus, under RM scheduling there must exist a task τ_a with a relative deadline $D_a \leq D_k$ that may request on R_b. As $D_a \leq D_k$, the execution time from tasks having relative deadlines larger than D_a that has to be serialized must be at least B_k. From Lemma 5.3, it is necessary for task τ_a being schedulable that $\left(B_k + \sum_{\tau_i : D_i \leq D_a} dbf_i^{R_b}(D_a)\right) / D_a \leq 1$. It then follows that $B_k \leq D_a \leq D_k$.

In either case, we can see that $B_k \leq D_k$.

If each processor is at least x times as fast, and by Lemma 5.3 and the above discussions, we know $(C_k + A_k)/D_k \leq 1/x$, $B_k/D_k \leq 1/x$, and

$$U^C \leq \frac{m - U^{\mathcal{RS}}}{x} \tag{5.27}$$

By Lemma 5.5, if each processor is at least x times as fast, it must be the case that

$$\frac{\sum_{R_q \in \Theta_h} \sum_{\tau_i \in hp(k)} dbf_i^{A_q}(D_k)}{D_k} \overset{\text{Eq. (5.23)}}{\leq} \frac{1 + \frac{U^{\mathcal{RS}} - 1}{m^R}}{x} \tag{5.28}$$

Putting the pieces together, at processors with speed x we have

$$\begin{aligned} \frac{I_k^{rRes}(D_k)}{D_k} &= \frac{\sum_{\tau_i \in hp(k)} \sum_{R_q \in \Theta_h} X_{i,q}(D_k)}{D_k} \\ &\overset{\text{Eq. (5.22)}}{\leq} \frac{3 \sum_{\tau_i \in hp(k)} \sum_{R_q \in \Theta_h} dbf_i^{A_q}(D_k)}{D_k} \\ &\overset{\text{Eq. (5.28)}}{\leq} \frac{3(1 + \frac{U^{\mathcal{RS}} - 1}{m^R})}{x} \end{aligned} \tag{5.29}$$

and

$$\frac{\sum_{\tau_i \in hp(k)} W_i(D_k)}{m^C D_k} \overset{\text{Eq. (5.21)}}{\leq} \frac{3\frac{\sum_{\tau_i \in \tau} dbf_i^C(D_k)}{D_k}}{m^C} \leq \frac{3U^C}{m^C} \overset{\text{Eq. (5.27)}}{\leq} \frac{3(m - U^{\mathcal{RS}})}{x(m - m^R)} \tag{5.30}$$

Summing over the corresponding terms and to contradict to Eq. (5.26), we need to set

$$x \geq 5 + \frac{3(U^{\mathcal{RS}} - 1)}{m^R} + \frac{3(m - U^{\mathcal{RS}})}{m - m^R} \tag{5.31}$$

Let $f(m, U^{\mathcal{RS}}) = 5 + \frac{3(U^{\mathcal{RS}}-1)}{m^R} + \frac{3(m-U^{\mathcal{RS}})}{m-m^R}$. In the following proof, we show that $f(m, U^{\mathcal{RS}})$ is upper bounded by $11 - 6/(m+1)$. We consider two separate cases:

- m is even. Let m^R be $\frac{m}{2}$. Thus, we have

$$\begin{aligned} f(m, U^{\mathcal{RS}}) &= 5 + \frac{3(U^{\mathcal{RS}} - 1)}{m/2} + \frac{3(m - U^{\mathcal{RS}})}{m/2} \\ &= 5 + 6\frac{m-1}{m} \\ &= 11 - \frac{6}{m} \end{aligned} \tag{5.32}$$

- m is odd. Due to our assumption, $m \geq 3$; therefore, $(m-1)/2 \geq 1$. In this case, we further consider two subcases:
  - $U^{\mathcal{RS}} \geq \frac{m+1}{2}$. Let m^R be $\frac{m+1}{2}$.

$$\begin{aligned} f(m, U^{\mathcal{RS}}) &= 5 + \frac{3(U^{\mathcal{RS}} - 1)}{(m+1)/2} + \frac{3(m - U^{\mathcal{RS}})}{(m-1)/2} \\ &= 5 + 6\left(\frac{(m^2-1) + 2 - 2U^{\mathcal{RS}}}{m^2 - 1}\right) \\ &\leq^1 11 - \frac{6}{m+1} \end{aligned} \tag{5.33}$$

    where $\leq^1$ is due to the assumption that $U^{\mathcal{RS}} \geq \frac{m+1}{2}$.
  - $U^{\mathcal{RS}} < \frac{m+1}{2}$. Let m^R be $\frac{m-1}{2}$.

$$\begin{aligned} f(m, U^{\mathcal{RS}}) &= 5 + \frac{3(U^{\mathcal{RS}} - 1)}{(m-1)/2} + \frac{3(m - U^{\mathcal{RS}})}{(m+1)/2} \\ &= 5 + 6\left(\frac{(m^2-1) - 2m + 2U^{\mathcal{RS}}}{m^2 - 1}\right) \\ &\leq^1 11 - \frac{6}{m+1} \end{aligned} \tag{5.34}$$

    where $\leq^1$ is due to the assumption that $U^{\mathcal{RS}} < \frac{m+1}{2}$.

In either case, we can see that $f(m, U^{\mathcal{RS}})$ is upper bounded by $11 - 6/(m+1)$.

Note that our analysis in this proof greedily sets m^R instead of searching m^R sequentially as in Algorithm 3. It is possible that m^R in our greedy setting is larger than r, the number of shared resources. However, this does not create any problem for the above analysis of the speedup factor. In this case, $m^R - r$ processors are completely unused and wasted in the above proof since they are not used under WFD in Algorithm 4. Therefore, the above analysis is more pessimistic, and we here conclude this theorem. □

5.5 MULTIPLE RESOURCE ACCESSES

In this section, we extend the schedulability analyses presented in Section 5.3.2 to include multiple resource accesses during a job's execution and multiple occurrences on one resource request, i.e., $N \geq 2$ or $Q \geq 2$.

5.5.1 *Multiple Occurrences on Each Resource*

Eq. (5.2) is based on the assumption that once a job begins execution, it does not suspend itself until completion. However, the execution on the synchronization processor under the resource-oriented scheduling may suspend itself for executing the non-critical-section codes on the application processor; Let $\wp_h$ be the processor h task τ_k may execute *remotely* and $\gamma_{k,h}$ be the number of blocking of task τ_k subject to $\wp_h$. It is clear that the number of blocking of a task having multiple occurrences on resource requests can be up to the total number of accesses to those critical sections on $\wp_h$:

$$\gamma_{k,h} = \sum_{R_q \in \mathcal{RS}_k \cap \Theta_h} N_{k,q} \tag{5.35}$$

where Θ_h is the set of shared resources that are bound to the synchronization processor $\wp_h$. Moreover, each access may suffer from the longest duration of one critical section of lower priority jobs on $\wp_h$. Hence, the maximum blocking time of a job of task τ_k subject to $\wp_h$ is upper bounded by

$$B_{k,h} = \gamma_{k,h} \times B_{k,h}^{Uni} \tag{5.36}$$

where $\gamma_{k,h}$ is defined as in Eq. (5.35), and $B_{k,h}^{Uni}$ is similar to the one in uniprocessor systems, as defined in Eqs. (5.2) or (5.1), determined by the used protocol.

5.5.2 *Multiple Resource Accesses*

The resource-oriented scheduling assumes *partitioned fixed-priority* scheduling; the computation has to be executed locally in accordance with the designated processors.

With multiple resource accesses, there may be more than one processor on which task τ_k may execute *remotely*, dependent of resource allocations. Similar to Section 5.3.2.1, at any time, a task being busy must execute/wait on one of the processors that the task may execute. Hence, the $S(t)$ on synchronization processors can be generalized as follows: Let Φ_k be the *set* of processors on which task τ_k may execute *remotely*.

$$S(t) = \sum_{\wp_h \in \Phi_k} S_h(t) \tag{5.37}$$

Elaborating the time spent on each processor gives us:

$$S_h(t) = A_{k,h} + B_{k,h} + I_{k,h}^{rRes}(t) \tag{5.38}$$

where the terms are as described below:

- $A_{k,h}$ denotes an upper bound on the amount of critical section execution from task τ_k itself on processor $\wp_h$.
- $B_{k,h}$ denotes an upper bound on the amount of blocking from lower-priority tasks on processor $\wp_h$, as defined in Eq. (5.36).
- $I_{k,h}^{rRes}(t)$ denotes an upper bound on the amount of interferences from higher-priority synchronization execution on processor $\wp_h$ with an interval of length t.

With the above definition of $S(t)$ in Eq. (5.37) and B_k in Eq. (5.36), we can again apply Theorem 5.2 and Theorem 5.3 for schedulability test.

5.6 IMPROVED RESPONSE TIME ANALYSIS

We have shown that the analysis presented in Section 5.3.2.1 combined with our allocation algorithm can achieve a non-trivial resource augmentation factor. Despite this, the TDA in Section 5.3.2.1 using Theorem 5.2 may be deficient in some aspect: the analyzed task being suspended or executing is indistinguishable to the calculation of the work imposed either by execution or by resource accesses. To put it more precisely, the interference is calculated only based on the interval of the analyzed task being busy, regardless of executing or being suspended. By doing so, it is safe but there may have to some degree pessimism over analysis. This is because any executions on the core during resource accesses, and vice versa, of the analyzed task have no impact on this analyzed task. In this section we propose an approach, similar to the one in Section 4.2.4, to improve the analysis in Section 5.3.2.1.

Let S^* be the upper bound on the cumulative times, not necessarily contiguous, of a job of task τ_k that *executes* or *waits in the waiting queue to be granted to enter* the critical section. Similarly, suppose X^* is the upper bound on the cumulative times, necessarily contiguous, of a job of task τ_k that *executes* or *waits in the waiting queue to execute* its non-critical-section. Unlike $S(t)$ and $X(t)$, the upper bound on demands generated over an interval, S^* and X^* are the times of task τ_k spent on some particular

processing element, independent of any interval of interest. By the definition of S^* and X^*, they are the times by which a job of task τ_k will certainly finish its execution on the synchronization processor and the application, respectively. Taking such upper bounds into consideration, we can tighten the analysis in Theorem 5.2 as follows:

Theorem 5.5. *The smallest t satisfying the following*

$$min\{S^*, S(t)\} + min\{X^*, X(t)\} \leq t \tag{5.39}$$

is a safe upper bound on the response time of task τ_k if $t \leq T_k$.

Consider multiple requests with at most σ times from a task on same processing element. The work by a higher-priority task τ_i that may prevent such requests from execution over an interval of length t is upper bounded by:

$$E_{i,q}(t, \sigma) = \left(\sigma - 1 + \left\lceil \frac{t + \sigma \cdot (RT_i - A_{i,q})}{T_i} \right\rceil\right) A_{i,q} \tag{5.40}$$

Intuitively speaking, the $\sigma - 1$ extra complete jobs here counted are the price we have to pay for having arbitrary requests.

Similarly, we have the workload function for a higher-priority task preventing σ normal executions over an interval of length t:

$$W_i(t, \sigma) = \left(\sigma - 1 + \left\lceil \frac{t + \sigma \cdot (RT_i - C_i)}{T_i} \right\rceil\right) C_i \tag{5.41}$$

Putting these pieces together, we have the following two lemmas to bound the response times, S^* and X^*:

Lemma 5.6. *S^* for task τ_k is the smallest t satisfying the following inequality:*

$$S(t) = A_k + \gamma_k B_k + I_k^{rRes}(t, \gamma_k) \tag{5.42}$$

where γ_k is defined as in Eq. (5.35) and

$$I^{lRes}(t, \sigma) = \sum_{R_q \in \Theta_l} \sum_{\tau_i \in hp(k)} E_{i,q}(t, \sigma) \tag{5.43}$$

Notice that with σ external requests, there are at most $\sigma + 1$ normal executions within a job.

Lemma 5.7. *X^* for task τ_k is the smallest t satisfying the following inequality:*

$$X(t) = I^{nRes}(t, \gamma_k + 1) + C_k \tag{5.44}$$

where γ_k is defined as in Eq. (5.35) and

$$I^{nRes}(t, \sigma) = \sum_{\tau_i \in \mathbf{hpl}(k)} W_i(t, \sigma) \tag{5.45}$$

5.7 EXPERIMENTAL RESULTS

In this section, we conduct extensive experiments using synthesized task sets. We evaluate these tests on m-processor platforms. We generate 100 task sets for each utilization level, from $0.05m$ to m, in steps of $0.05m$. The metric to compare the results is to measure the *acceptance ratio*. The acceptance ratio of a level is said to be the number of task sets that are deemed schedulable by the test divided by the number of task sets for this level, i.e., 100.

The cardinality of the task set is 10 times the number of processors, e.g. 40 tasks on 4 processors. We use the approach suggested by Emberson et al. [ESD10] to generate the task periods according to the *exponential distribution*. The distribution of periods is within two orders of magnitude, i.e., 10ms-1000ms. Task relative deadlines are implicit, i.e., $D_i = T_i$.

We vary the ratio of non-critical-sections to critical-sections $\alpha \in \{20, 5\}$ to evaluate the effect of resource sharing: the smaller the α, the more the critical-sections. For example, if $\alpha = 5$ and the utilization level $U_\Sigma = 120\%$, we have $U^{\mathcal{RS}} = 120\% \times \frac{1}{5+1} = 20\%$ and $U^C = 120\% \times \frac{5}{5+1} = 100\%$ in the system. In each utilization step, the `Randfixedsum` method [ESD10] is adopted twice to generate two sets of utilization values with the given goals of critical-sections and non-critical-sections. We ensure that for every task τ_i, $U_i^A + U_i^C \leq 1$. The worst-case execution time of a task for its non-critical-sections and critical-sections is set accordingly, i.e., $C_i = T_i U_i^C$ and $A_i = T_i U_i^A$.

We assume there are 5, 8, and 16 shared resources in multiprocessor systems comprised of 4, 8, and 16 processors, respectively. We vary the number of shared resources that a task requests $N \in \{1, 3\}$. For each task, we then again use the `Randfixedsum` method to generate a set of vectors which are evenly distributed in the Q resource accesses and the total access utilization sums to its critical-section utilization. The critical section of accessing resource R_q of task τ_i is set accordingly, i.e. $A_{i,q} = U_{i,q} T_i$. We set the number of requests on each resource during execution N to either 1 or 3. The maximum total resource usage time $V_{i,q}$ for resource R_q by any single job of τ_i is drawn uniformly from $[A_{i,q}/N, A_{i,q}]$.

The global/partitioned RM scheduling is applied by default in the system, unless otherwise stated. The evaluated tests are listed as follows:

- PIP: the Priority Inheritance Protocol (PIP) [EA09], which is a fixed-priority global scheduling algorithm.
- MPCP: the Multiprocessor Priority Ceiling Protocol (MPCP) [Raj90] along with the Synchronization-Aware Partitioning Algorithm (SPA) [LNR09]. Informally speaking, contrary to the proposed approach, the MPCP can be thought of as the conservative approach wherein task bodies are not split.
- MrsP: the Multiprocessor resource sharing Protocol (MrsP) [BW13] along with the SPA.

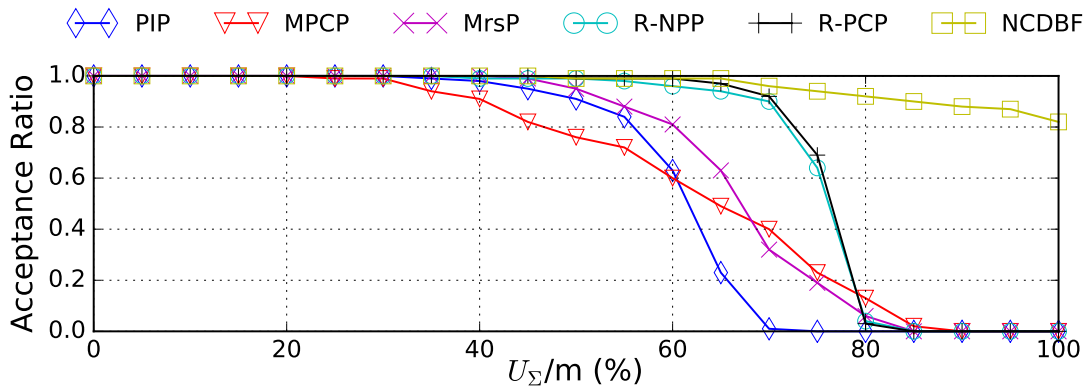


Figure 5.6: Effectiveness by different algorithms with $m = 4$, $\alpha = 20$, $r = 5$, $Q = 1$, $N = 1$.

- R-NPP: the proposed resource-oriented partitioned scheduling using ROP-NPP by Algorithm 3 where Theorem 5.3 uses the blocking time Eq. (5.11).
- R-PCP: the proposed resource-oriented partitioned scheduling using ROP-PCP by Algorithm 3 where Theorem 5.3 uses the blocking time Eq. (5.12).
- NCDBF: the theoretical upper bound using necessary conditions stated in Lemma 5.3.

To make comparison fair, all the implemented tests have pseudo-polynomial time complexity. We note that the accuracy of the sufficient schedulability tests listed above can be improved by formalizing the problem of finding a response-time bound as a linear optimization problem [YWB15]. However, tests using the LP solver may suffer from their high time complexity, especially, in conjunction with partitioning resources and tasks. Besides, we here focus on showing the effectiveness of protocols themselves, and similar results can be seen under LP formalizations.

Result I. We show the evaluation of the performance by the above scheduling algorithms in terms of task sets deemed schedulable. Figures 5.6 to 5.11 show the effectiveness by different algorithms, varying the number of processors m, the ratio of non-critical-sections to critical-sections α, the number of available resources r, the maximum number of resource accesses Q, and the maximum requests on each resource N. In the first two figures (Figures 5.6 and 5.7), we vary the number of processors $m \in \{4, 8\}$, assuming $\alpha = 20$, $Q = 1$, and $N = 1$. We first notice that both the R-NPP and the R-PCP dominate all the existing approaches, including the PIP, the MPCP, and the MrsP. The later three are neither better than another.

Although the PIP bounds the priority inversion and is shown to be relatively effective [YWB15], the execution for a job still suffers from the chained blocking. MPCP and MrsP using the SPA attempt to assign each of the so-called *macrotasks* (defined in [LNR09]) onto a processor. However, it is often the case that a macrotask is

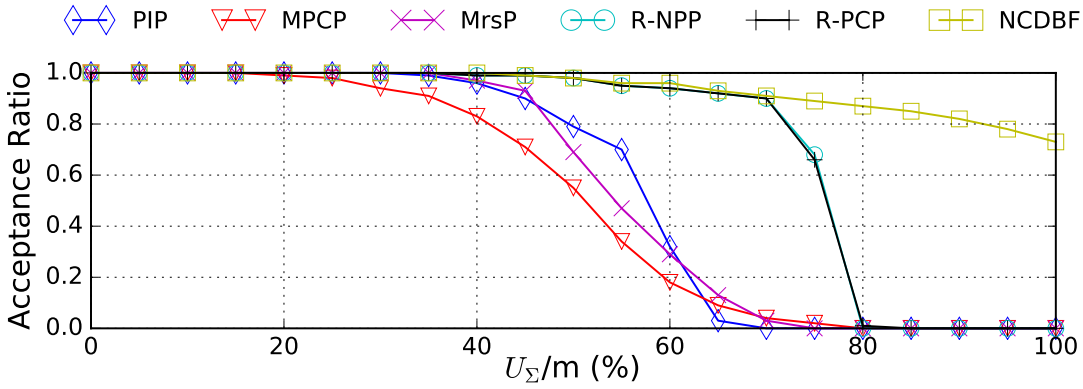


Figure 5.7: Effectiveness by different algorithms with $m = 8$, $\alpha = 20$, $r = 8$, $Q = 1$, $N = 1$.

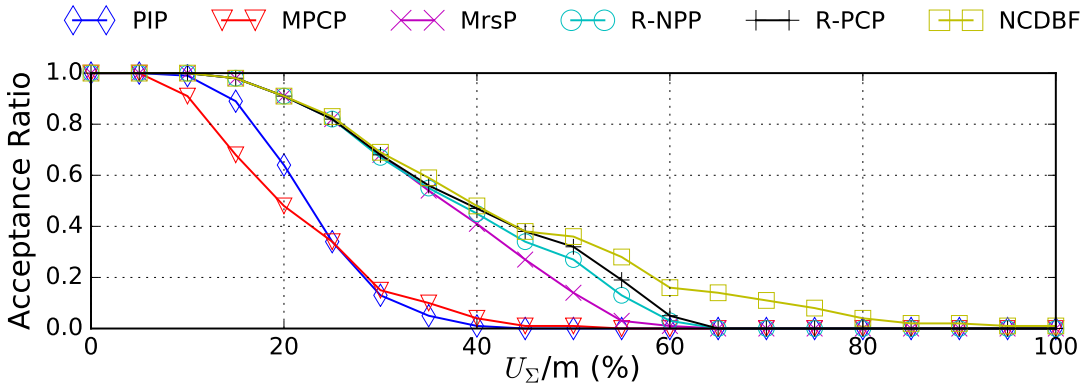


Figure 5.8: Effectiveness by different algorithms with $m = 8$, $\alpha = 5$, $r = 8$, $Q = 1$, $N = 1$.

too heavy to fit into one processor and hence is forced to be split into several processors. After that, any task belonging to the split macrotask suffers from the excessive blocking time incurred by lower-priority tasks on other processors. On the other hand, even though the total utilization of a macrotask can fit into one processor, sometimes it is better to shift some non-critical-section execution to other processor as the utilization of a macrotask may become quite heavy. Hence, not surprisingly, PIP, MrsP, and MPCP perform worse than R-PCP and R-NPP, both minimizing the number of blocking with $Q = 1$ and $N = 1$.

It is remarkable that both R-PCP and R-NPP can still keep pace with the theoretical upper bound, the NCDBF, until utilizations are over 70%. A similar result can also be seen from Figure 5.11 where a 16-processor platform is assumed.

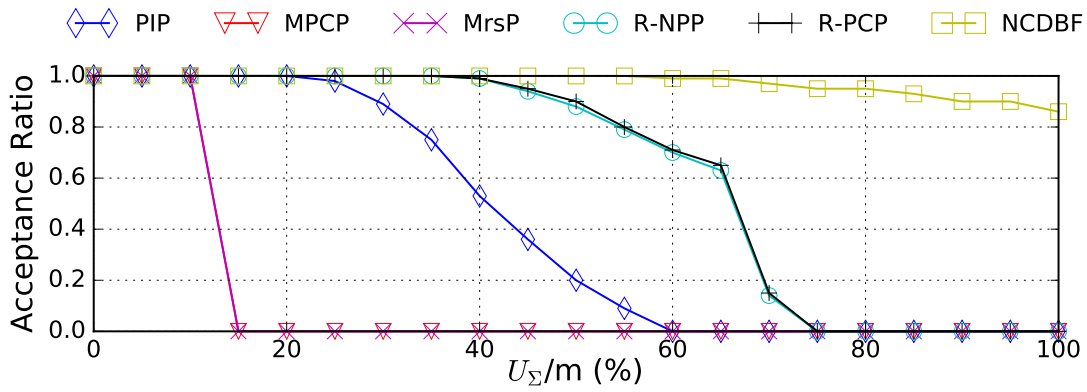


Figure 5.9: Effectiveness by different algorithms with $m = 8$, $\alpha = 20$, $r = 8$, $Q = 3$, $N = 1$.

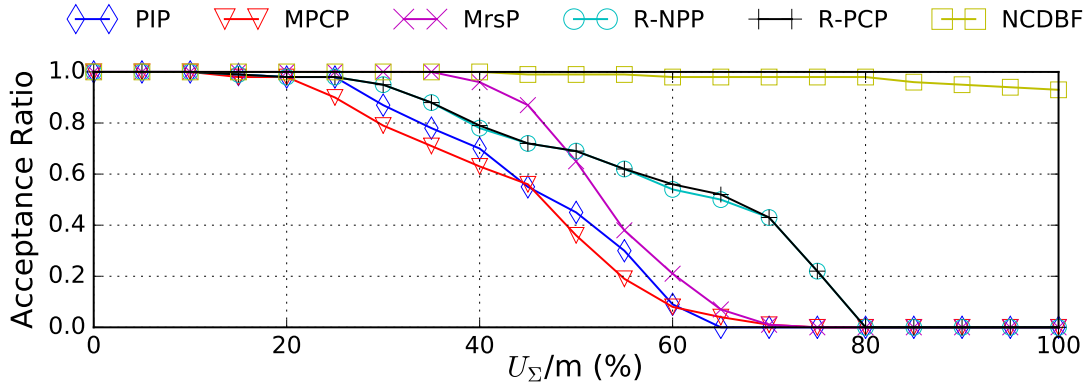


Figure 5.10: Effectiveness by different algorithms with $m = 8$, $\alpha = 20$, $r = 8$, $Q = 1$, $N = 3$.

In Figures 5.7 and 5.8, we vary α in $[20, 5]$ on a 8-processor platform with 8 shared resources where a task requests one resource out of the five, assigned randomly, and each job of a task requests the resource once during its execution. We first notice that NCDBF drops off as α decreases from 20 to 5. R-PCP and R-NPP can still keep pace with the theoretical upper bound, NCDBF, until utilizations are over 50%

In Figures 5.9 and 5.10, we evaluate the effect of accessing multiple resource and requesting multiple times on one resource. Here we can see that despite that R-PCP and R-NPP are still superior to the others, there exists a large gap between the theoretical upper bound and the best available algorithm, dropping down from 40% with multiple resource accesses and from 25% with multiple requests on one resource. We also notice that in Figure 5.10 both MPCP and MrsP drop quickly from 15%. This is due to that the SPA algorithm only considers utilizations to allocate tasks onto processors. With

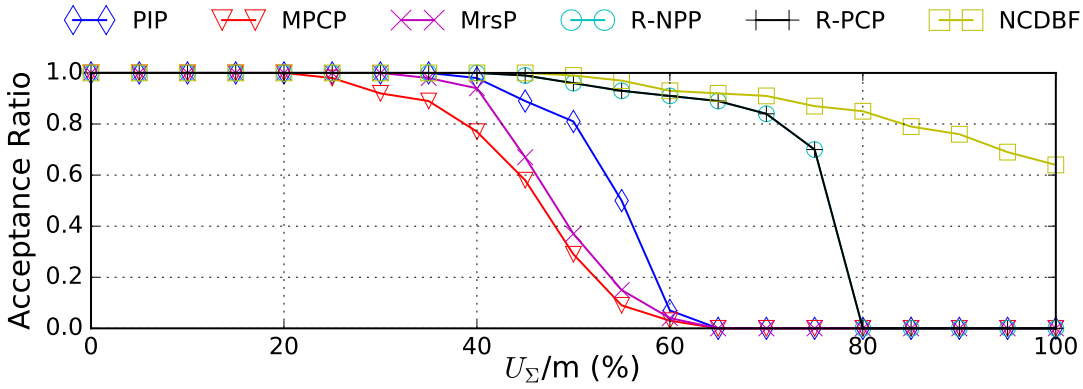


Figure 5.11: Effectiveness by different algorithms with $m = 16$, $\alpha = 20$, $r = 16$, $Q = 1$, $N = 1$.

multiple resource accesses, it is often the case that a *macrotask*, within which tasks directly or indirectly share resources are bundled, becomes extremely heavy. The algorithm allocates the tasks within the macrotask onto one processor as far as possible, so as to reduce the processors used (also remote blocking). As a result, some tasks on this processor can be punished by such excessive interferences: the effectiveness of partitioned protocols is highly dependent on task partitioning. Hence, both designing synchronization protocols and finding task mapping are needed to be considered.

From the perspective of schedulability, using PCP in each individual processor is as good as using the non-preemptive scheduling. Surprisingly, R-PCP and R-NPP have almost the same performance in our evaluations (although invisible, there still have some tasks not deemed schedulable by R-NPP but schedulable by R-PCP). This is because the long blocking incurred by the NPP might be painlessly removed by finding a good task and resource partitioning.

Maintaining a list of currently locked semaphores and priority ceiling orders during runtime may incur a noticeable computational overhead. In practice, one would expect R-NPP to be default, and R-PCP to be used only when the unnecessary blocking from the non-preemptive scheduling can be surely removed. Last but not least, the research result reported in this paper suggests that finding a good task mapping and resource allocation is as important as designing a good resource sharing protocol while ignoring task partitioning.

Result II. Intuitively speaking, task partitioning is tractably done by iteratively ensuring the schedulability of each task, starting from the task with highest priority, until a feasible partitioning is found or no processor can accommodate the task being assigned. The idea behind this is that no tasks that have been successfully assigned can be jeopardized by the task being assigned. Under our proposed resource-oriented algorithm, a synchronization processor may execute non-critical sections. In this case,

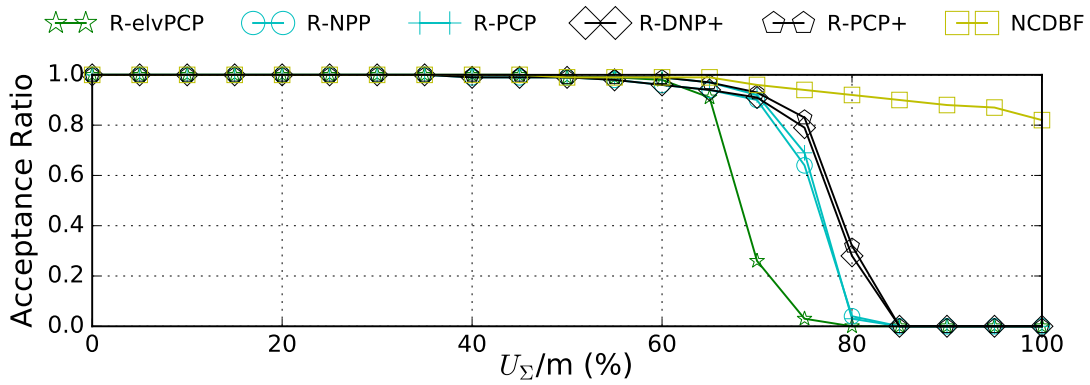


Figure 5.12: Effectiveness by different algorithms with $m = 4$, $\alpha = 20$, $r = 5$, $Q = 1$, $N = 1$.

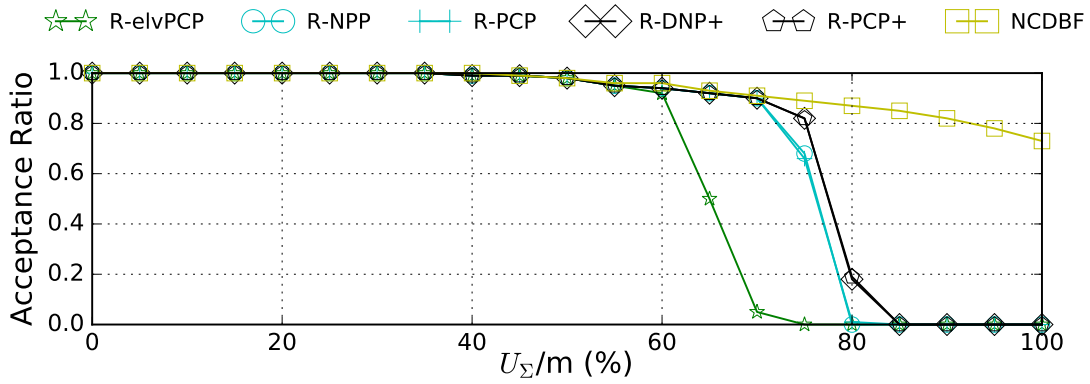


Figure 5.13: Effectiveness by different algorithms with $m = 8$, $\alpha = 20$, $r = 8$, $Q = 1$, $N = 1$.

the above idea is followed if a task being assigned executes its non-critical sections on the synchronization processors at the priority *no higher* than all the tasks that have been successfully assigned. But at which priority should the task execute its non-critical sections? From the schedulability point of view, assigning higher priority for the task executing its non-critical sections may make this task easier schedulable at first, but may result in longer resource access times for the tasks that have not been assigned yet. We discuss two extreme cases in this dissertation:

1. the execution of non-critical-sections at *lower* priority than any of critical-sections.
2. the execution of non-critical-sections at its *base* priority.

Notice that the first case has been preferably studied in this dissertation, as appeared in Section 5.3.2.2. The question is whether or not the second case can outperform the first one. Another question interesting to know is how much the schedulability can be

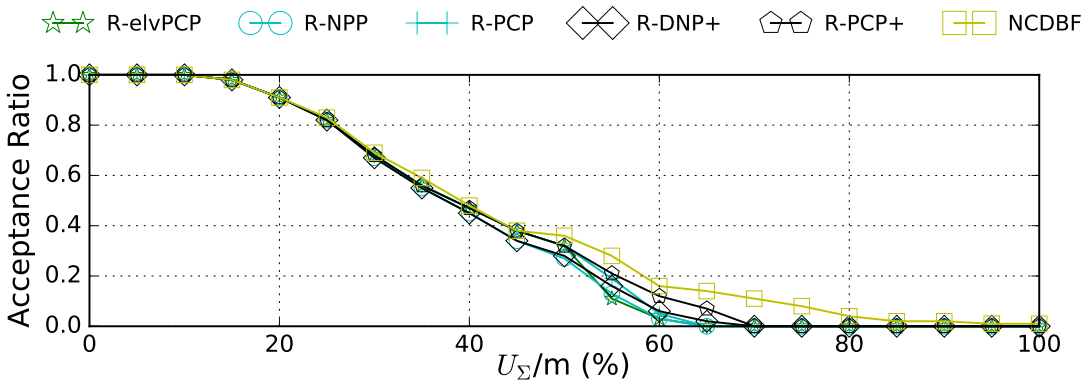


Figure 5.14: Effectiveness by different algorithms with $m = 8$, $\alpha = 5$, $r = 8$, $Q = 1$, $N = 1$.

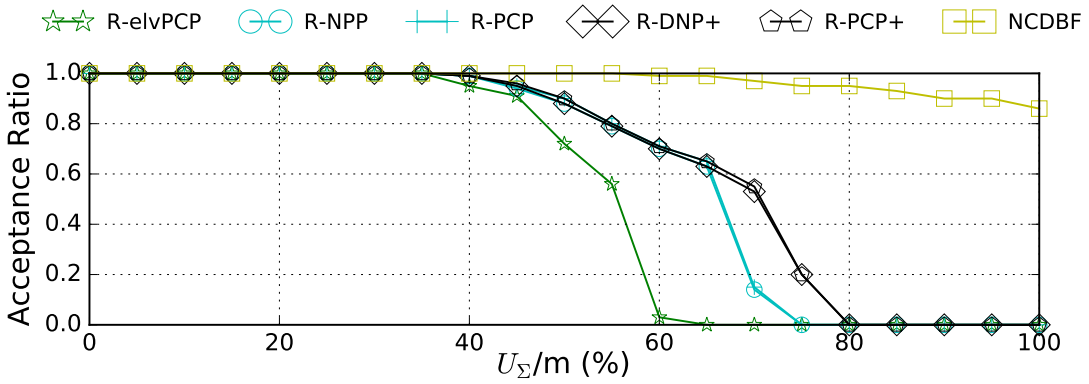


Figure 5.15: Effectiveness by different algorithms with $m = 8$, $\alpha = 20$, $r = 8$, $Q = 3$, $N = 1$.

improved by using the improved response time analysis proposed in Section 5.6. The evaluated tests are listed as follows:

- R-elvPCP: the execution of non-critical-sections on each synchronization processor at its *base* priority.
- R-NPP: the proposed resource-oriented partitioned scheduling using ROP-NPP by Algorithm 3 where Theorem 5.3 uses the blocking time Eq. (5.11).
- R-PCP: the proposed resource-oriented partitioned scheduling using ROP-PCP by Algorithm 3 where Theorem 5.3 uses the blocking time Eq. (5.12).
- R-NPP+: the improved ROP-NPP using Theorem 5.5.
- R-PCP+: the improved ROP-PCP using Theorem 5.5.

Figures 5.12 to 5.17 show the effectiveness by the above algorithms. In all the evaluated settings we notice that the performance by R-elvPCP is not as good as one

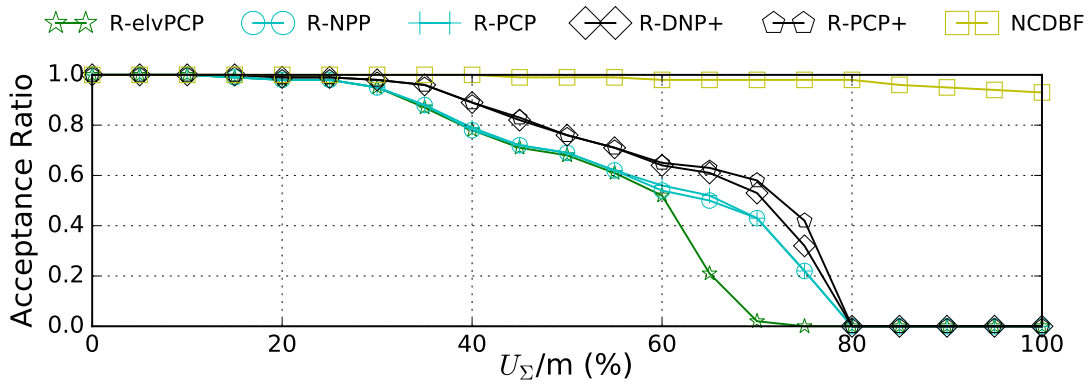


Figure 5.16: Effectiveness by different algorithms with $m = 8$, $\alpha = 20$, $r = 8$, $Q = 1$, $N = 3$.

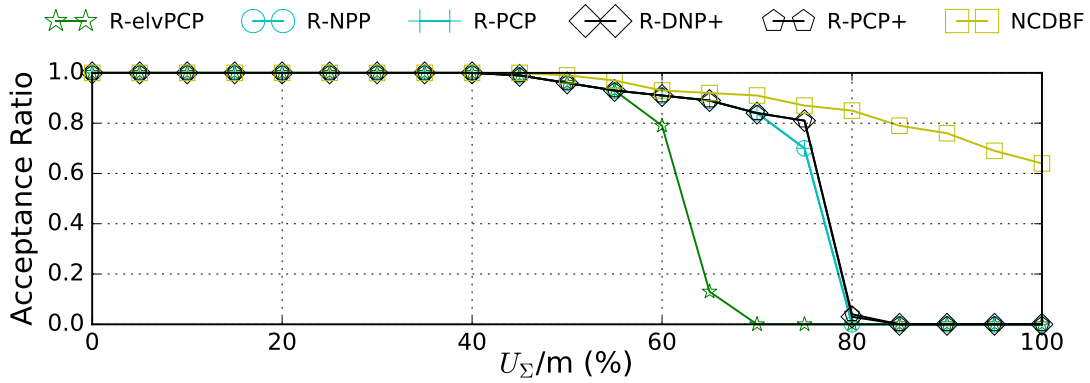


Figure 5.17: Effectiveness by different algorithms with $m = 16$, $\alpha = 20$, $r = 16$, $Q = 1$, $N = 1$.

would expect: there is a noticeable gap between R-elvPCP and R-PCP. On the other hand, the approaches using the improved response time analysis, R-PCP+ and R-NPP+, can additionally admit quite a number of task sets that are not deemed schedulable by R-PCP and R-NPP.

5.8 SUMMARY

In this chapter, we study the problem of scheduling real-time tasks with synchronization. We show that the proposed resource-oriented scheduling using PCP combined with a reasonable allocation algorithm can achieve a non-trivial speedup factor guarantee. Furthermore, our empirical investigations indicate that the resource-oriented scheduling

algorithm using NPP is pragmatically good in the sense of high performance in terms of schedulability and low runtime overheads.

6

MULTI-MODE TASK MODEL

Formal models used for representing recurrent real-time processes have traditionally been characterized by a collection of jobs that are released periodically. However, such a modeling may result in resource under-utilization in systems whose behaviors are not entirely periodic. For instance, tasks in Cyber-Physical System (CPS) may change their service levels, e.g., periods and/or execution times, to adapt to the changes of environments.

In CPS, the characteristics of a real-time task may be able to change over time, e.g., the computational demand or the resource allocation. Such behavior is referred to as *mode changes*. The importance of mode changes for real-time systems has been pointed out in many perspectives, for instance, aircraft control systems, automotive Electronic Control Units (ECU) [RC04; DFPS14], energy management [SCT10a], and server-based systems [SBB11].

Specifically, the engine control in automotive systems has several computational demands reacted to different angular rotations. In each periodic interval, the engine control software calculates the engine speed and position to determine when to fire the next spark signal and evaluates the acceleration/deceleration commands from the driver to adjust the settings of fuel flow [DFPS14]. Such a control has the nature of computation mode changes according to the physical environment. Besides, the stringent timing requirement has to be met to inject and to deliver fuel to each cylinder at every revolution.

In automotive applications, several tasks are linked to rotation (e.g., of the crankshaft, gears, or wheels). Thus their activation rate is proportional to the angular velocity of a specific device. In such a system a common practice is to design a rate-dependent

task, called Variable Rate-dependent Behavior (VRB) task model [KLR12; DFPS14; BBB14; BMMNB14]. Typically, at lower rotation some functions that minimize fuel consumption and emissions have to be executed, whereas they are shed at higher rotation speeds to reduce the processor utilization. Table 6.1 illustrates an example of a task with four levels of functionality, specified for different speed intervals.

Table 6.1: An example of variable-rate behavior task with four types of execution modes dependent on the rotation speed.

rotation (rpm)	functions to be executed
$[0, 2000]$	$f1(); f2(); f3(); f4();$
$(2000, 4000]$	$f1(); f2(); f3();$
$(4000, 6000]$	$f1(); f2();$
$(6000, 8000]$	$f1();$

In this chapter we study a real-time system comprised of n independent, preemptive *multi-mode* tasks $\tau = \{\tau_1, \tau_2, ..., \tau_n\}$ where each task has several execution *modes* to switch during the runtime. The cost of synchronization on mode change within a task is assumed to be subsumed into the worst-case execution time of each mode. A multi-mode task τ_i with H_i modes is denoted by a set of triplets:

$$\begin{aligned}\tau_i = \{&\tau_i^1 = (C_i^1, T_i^1, D_i^1),\\ &\tau_i^2 = (C_i^2, T_i^2, D_i^2), ...,\\ &\tau_i^{H_i} = (C_i^{H_i}, T_i^{H_i}, D_i^{H_i})\}\end{aligned}$$

C_i^m denotes the *WCET* of task τ_i under mode m, T_i^m denotes the *minimum inter-arrival time* of task τ_i under mode m, and D_i^m denotes the *relative deadline*. That is, when a job of mode τ_i^m is released at time t, the next release time of task τ_i is no earlier than $t + T_i^m$, and when a job of mode τ_i^m is released at time t, this job has to be finished no later than its *absolute deadline* at time $t + D_i^m$.

Related Mode Change Models

The studied mode change model is a generalization of the sporadic model [Mok83]. The concept of mode change is distinct from that of system-wide operating modes [TBW92; SRLR89]. This model characterizes the system where different *tasks* may progress through their different execution modes independent of each other.

Even though the studied mode change model essentially differs from the VRB model where different angular sources drive the inter-arrival time, the mode change model is still applicable when considering only those thresholds that determine which level of functionality should be executed, and in fact the worst-case scenario occurs at the particular value of thresholds. Similar concepts have been presented in [DFPS14]. In

general, the mode change model can be thought of as a relaxation model of VRB task model [KLR12; DFPS14] and digraph Digraph Real-Time (DRT) [SEGY11]. From the designer's perspective, the studied mode change model provides an easier way to specify and reason comparing to the more general DRT model.

Specifically, this dissertation studies two scheduling algorithms upon multi-mode task uniprocessor systems: FPT and FPM. In this dissertation, we derive a technique for analyzing the schedulability in uniprocessor multi-mode systems. Based on this technique, we propose tests that leverage the accuracy and the time complexity for both FPT and FPM scheduling in multi-mode uniprocessor systems. Furthermore, we make use of these tests from uniprocessor systems to derive utilization bounds for multiprocessor multi-mode systems.

The presentation is organized as follows: This chapter consists of two main parts: (*i*) uniprocessors, in Chapter 6.1, and (*ii*) multiprocessors, in Chapter 6.2. In Chapter 6.1.1, we first review the known critical instance for multi-mode tasks, and then formally define a multi-mode demand bound function. After putting these pieces together, the schedulability test for uniprocessor multi-mode systems under FPT scheduling is derived. In Chapter 6.1.2, based on the proposed technique for FPT scheduling, we derive schedulability tests for FPM scheduling. From there, we prove that a utilization bound of $2 - \sqrt{2} \approx 0.5857$ can be guaranteed in implicit-deadline multi-mode systems under rate-monotonic (RM) scheduling, one example of FPM scheduling. In Chapter 6.2, following the result from uniprocessor systems, we trivially provide a utilization bound $\sum_i U_i \leq \frac{2-\sqrt{2}}{2} M \approx 0.293 \cdot M$ if $U_i \leq 2 - \sqrt{2}$ for every task τ_i. More favorably, a utilization bound $\sum_i U_i \leq \frac{3-\sqrt{5}}{2} M \approx 0.381 \cdot M$ on multiprocessor systems under RM scheduling is derived in Chapter 6.2.3. In Chapter 6.2.4, we then further improve this bound by considering the upper bound on the utilization for every task. Finally, we summarize our results in Chapter 6.3. The notation and terminology to be used in this chapter are listed in Table 6.2.

6.1 UNIPROCESSORS

The objective of the mode change schedulability analysis is to guarantee that a system is feasible not only in the steady state but also during the mode transition. In mode change systems, during mode transitions, the phenomenon of demand strides may occur and lead to an unfeasible scheduling, even though there is no deadline miss in the steady state. We show this with the following example:

Example. Consider a system with one multi-mode task and one sporadic task: $\tau_1 = \{(2,3,3),(4,8,8)\}$ and $\tau_2 = \{(4,12,12)\}$. Both tasks are schedulable by RM without mode transition. However, task τ_2 misses its deadline at time-instant 12 when task τ_1 switches from mode τ_1^2 to mode τ_1^1 at time-instant 9, as illustrated in Figure 6.1.

On the other hand, if mode τ_1^2 is released at the beginning, followed by mode τ_1^1, then τ_2 becomes schedulable, meaning that different release sequences of modes may

SYMBOL	MEANING
M	the number of processors available
N	the number of tasks
H_i	the total number of modes of task τ_i
C_i^{max}	the maximum worst-case execution times of task τ_i among all the modes to invoke; $C_i^{max} = max_{\tau_i^m \in \tau_i}(C_i^m)$
C_i^h / T_i^h	the utilization factor U_i^h of task τ_i under mode h
U_i	the (maximum) utilization of task τ_i; $U_i = max_{h=1,\ldots,H_i}\{U_i^h\}$
β_i	the ratio of the maximum worst-case execution times among modes to the maximum utilization among modes; $\beta_i = C_i^{max} / U_i^{max}$
α	the upper bound on utilizations for every task; $0 < U_i \leq \alpha \leq 1$
U^{sum}	the total utilization of the system; $U^{sum} = \sum_{i=1}^{N} U_i \leq M$
$DBF_i(t, \tau_k^h)$	the demand bound function of a multi-mode task τ_i over an interval of length t with respect to mode τ_k^h

Table 6.2: Multi-mode systems: notation and terminology

result in different response times. As a result, we have to take combinatorial releases into account so as to identify the worst-case scenario. In addition, the response time of a task depends on not only release sequences of modes but also time instants of mode releases.

We first check whether there exists *critical instant* for multi-mode tasks, in the hopes of providing analysis amenable to the calculation of WCRT. *Critical instant* is the instant at which the execution of a task will have the longest response time [LL73]. For sporadic tasks with constrained deadlines, it is proven that the critical instant for a task occurs when the task is released simultaneously with all higher priority tasks and all the following jobs are released as early as possible [LL73]. As for multi-mode tasks, the critical instant for a multi-mode task under FPT scheduling has been shown in Theorem 1 in [DFPS14].[1] For completeness, we paraphrase this theorem in the following lemma:

Lemma 6.1 (Davis et al. [DFPS14]). *There is a sequence Y of jobs of task τ_i , that τ_i releases the maximum interference $I_i(w)$ in a window $[0, w)$, where*

1. *the offset, from the start of the window, of the first job of τ_i is zero;*
2. *each of the jobs of τ_i released in the window has the minimum period commensurate with its particular execution mode;*
3. *the last job has the largest WCET for any execution mode.*

1 Although the focus in [DFPS14] was for the VRB task model, the critical instant theorem in [DFPS14] works for the mode change task model in this dissertation as well. The proof to create the maximum interference is exactly the same.

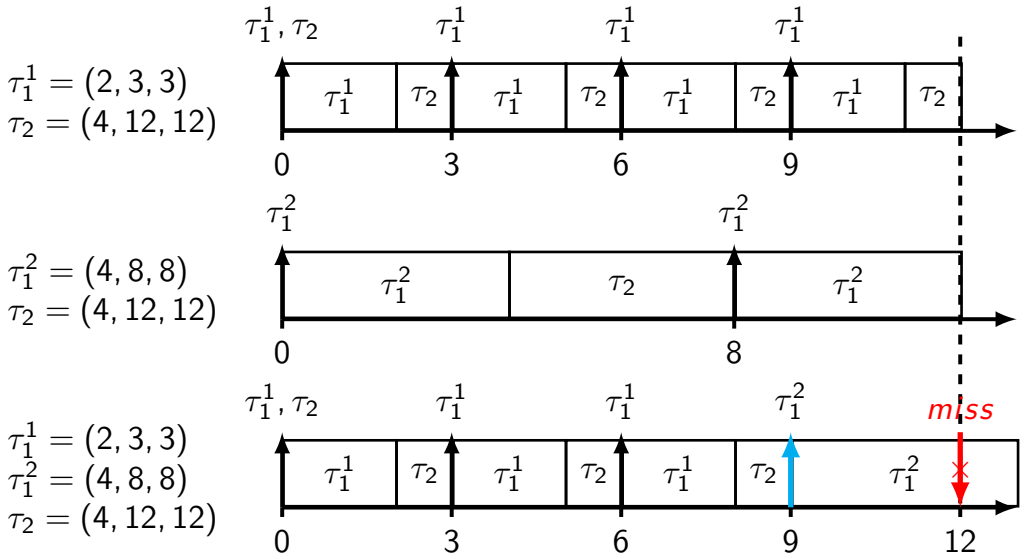


Figure 6.1: The missed deadline during mode transition

Nevertheless, unlike for the sporadic task, the critical instant provided in Lemma 6.1 is only necessary for creating the maximum interference. The calculation of the *exact* worst-case interference from all tasks involves enumerating all possible release sequences that satisfy the above conditions, but is, however, computationally intractable since the time complexity is $O(W_1 \times W_2 \times ...W_{n-1})$ where W_i is the possible release sequences within the input period and is $\geq M_i$.

As an amenable solution, instead of enumerating possible release sequences of all the tasks, one can count the interferences from tasks one by one, by enumerating each task's possible release modes over the interval of interest, which is computable in pseudo-polynomial time. Such an approach has been proposed in the literature. For instance, a method using integer linear programming (ILP) solvers for deriving the maximum demand has been proposed in [DFPS14]. Also, a dynamic programming method has been reported in [SEGY11] for the DRT model, which is a generalization of the studied mode change model. However, both approaches may lead to an over-approximation of the exact response time, due to non-concrete traces.

6.1.1 *Schedulability Analysis under FPT Scheduling*

In this section, we first introduce the concept of the multi-mode demand bound function, and then make use of this function to derive a sufficient schedulability test for multi-mode task systems under a given FPT assignment, in Theorem 6.4.

By Lemma 6.1, we can see that no matter how the sequence of task τ_i releases prior to the arrival time of the last release, one can replace the last release mode in $[0, w)$ by the mode with the largest WCET among all modes without decreasing the interference. Based on this observation, we can decompose the interference from a multi-mode task into two parts: (*i*) the execution time from the last release that has the largest WCET among modes; and (*ii*) the total demand prior to the arrival time of the last release. We formally define the last release time and multi-mode demand bound function as follows:

Definition 6.1 (Last Release Time)**.** For a given sequence of releases of task τ_i in the interval $[t_0, t_0 + t)$, we define t_i as the time of the last release of task τ_i upon the sequence.

Definition 6.2 (Multi-mode demand bound function)**.** For any interval length of t, the demand bound function $DBF_i(t, \tau_k^h)$ of a multi-mode task τ_i is defined as the maximum cumulative execution requirement by jobs of τ_i that are assigned with higher priority than mode τ_k^h and have both arrive time and next release time within an interval length of t.

Multi-Mode Demand Bound Function

The DBF bounds the maximum cumulative execution requirement by jobs of task τ_i that both arrive in and have absolute deadlines within any interval of length t, as mentioned earlier in Definition 2.5. This function for sporadic tasks, shown in Eq. (2.1), is computable in constant time. As far as multi-mode tasks are concerned, the multi-mode demand bound function however involves combinatorial releases.

In fact, calculating the multi-mode demand bound function is equivalent to the well-known *unbounded knapsack problem* (UKP) [Van93]. The *unbounded knapsack problem* is to determine the number of each item to include in a collection of items so that total weight (execution time) of the selected items is less than or equal to a given limit (interval length) (called knapsack) and total value (cumulative executions) of the selected items is maximized.

Given that the UKP can be optimally solved by using *dynamic programming* [Van93], the multi-mode demand bound function is solvable in pseudo-polynomial time. Furthermore, it has been shown in [Van93] that an upper bound B for UKP is

$$B = \left\lfloor c \frac{p_1}{w_1} \right\rfloor \tag{6.1}$$

where p_i denotes the *profit* of an item of type i, w_i denotes the *weight* of an item of type i, c is the limit of the knapsack, and the item types are ordered so that

$$\frac{p_1}{w_1} \geq \frac{p_2}{w_2} \geq ... \geq \frac{p_n}{w_n} \tag{6.2}$$

Notice that $\frac{p_1}{w_1}$ corresponds to the maximum utilization among the eligible modes, i.e., $U_i^{max}(\tau_k^h)$. Lemma 6.2 follows immediately:

Lemma 6.2. *For any τ_i and a higher-priority mode τ_k^h, for $t > 0$*

$$DBF_i(t, \tau_k^h) \leq t \cdot U_i^{max}(\tau_k^h)$$

Proof. Notice that $\frac{p_1}{w_1} = U_i^{max}(\tau_k^h)$.

$$\begin{aligned} \text{By Eq. (6.1)} \quad DBF_i(t, \tau_k^h) &\leq \left\lfloor t U_i^{max}(\tau_k^h) \right\rfloor \\ \Rightarrow DBF_i(t, \tau_k^h) &\leq t \cdot U_i^{max}(\tau_k^h) \end{aligned}$$

□

It is worth noting that by definition it is possible that the mode belonging to $U_i^{max}(\tau_k^h)$ has a period $> t$ and thus cannot both arrive in and have next release time within interval length of t. Nevertheless, the upper bound still holds. We will use the upper bound on the multi-mode demand function to derive a schedulability technique that requires the *continuity* of the upper bound with the interval length of t. The mode having period larger than t can be removed only if the interval of interest is known, e.g., under FPT $0 < t \leq D_k^h$.

6.1.1.1 *Sufficient Test*

In this section, we use the concept of the interference decomposition to derive a technique for analyzing schedulability of real-time systems represented using the mode change model.

Consider any legal sequence of jobs of task system τ, on which a deadline miss occurs. Suppose that a mode m of the kth- highest priority task is the one to first miss a deadline and that the mode arrives at time-instant t_a and this deadline miss occurs at time-instant $t_a + D_k^h$.

Without loss of generality, by Lemma 6.1, we set $t_a = 0$. We first assume that the tasks assigned with higher priority than mode τ_k^h are indexed according to the last release time ordering π, upon which the deadline miss occurs.

Definition 6.3 (Last Release Time Ordering)**.** Let π be the assignment of a last release time ordering as a bijective function $\pi : \tau \to \{1, 2, \ldots, k-1\}$ to define the last release time ordering of task $\tau_i \in hp(\tau_k^h)$. The ordering of last releases is numbered from 1 to $k-1$ where 1 is the earliest and $k-1$ the latest.

We now derive a necessary condition for a deadline miss to occur with a last release time ordering π in the following lemma:

Lemma 6.3. *If task mode τ_k^h misses its deadline under FPT upon a last release assignment π, then there exists an assignment $t_1, t_2, \ldots, t_{k-1} \in [0, D_k^h)$ of the last release of task τ_i such that $\forall i \in \{1, ..., k-1, k\}$*

$$\sum_{j=1}^{k-1} DBF_j(t_j, \tau_k^h) + \sum_{j=1}^{i-1} C_j^{max} + C_k^h > t_i \tag{6.3}$$

where $t_k \equiv D_k^h$.

Proof. By Lemma 6.1 and the assumption of the deadline miss of mode τ_k^h, the released pattern described in Lemma 6.1 will result in a deadline miss for the job released by mode τ_k^h at time $t_a = 0$. Let π be such a last release ordering, and define the last release time t_i of task τ_i for $i = 1, 2, \ldots, k-1$ accordingly with $t_i \leq t_{i+1}$. The executed workload of the job released by mode τ_k^h must be strictly less than C_k^h amount of execution time over $[0, D_k^h]$. We observe the followings:

- At time point t_i for any $i = 1, 2, \ldots, k-1$, the requested higher-priority execution time prior to t_i plus C_k^h is larger than t_i.
- Up to time t_i, a task τ_j with $j = 1, 2, \ldots, i-1$ has requested $DBF_j(t_j, \tau_k^h) + C_j^{max}$ amount of execution time to be executed.
- Up to time t_i, a task τ_j with $j = i, i+1, \ldots, k-1$ has requested $DBF_j(t_i, \tau_k^h)$ amount of execution time to be executed.

The above observation results in $\forall i = 1, 2, \ldots, k$,

$$C_k^h + \sum_{j=1}^{i-1} (DBF_j(t_j, \tau_k^h) + C_j^{max}) + \sum_{j=i}^{k-1} DBF_j(t_i, \tau_k^h) > t_i$$

By the fact that $DBF_j(t, \tau_k^h)$ is monotonically non-decreasing with respect to the interval length t, we know that $DBF_j(t_i, \tau_k^h) \leq DBF_j(t_j, \tau_k^h)$ when $j \geq i$. (Due to the last release ordering π, we have $t_j \geq t_i$ for such cases.) As a result, we reach the conclusion in Eq. (6.3). □

Without knowing the exact last release times t_i, all possible last release time-instants of t_i have to be enumerated by combinatorial releases as shown in [DFPS14]. In contrast, in this work we are aiming at obtaining tests that leverage the accuracy and the time complexity (in polynomial). In the following lemma we provide a necessary condition for a deadline miss by maximizing the higher-priority tasks' interference in Eq. (6.3), depending on the assignment of last release times:

Lemma 6.4. *If mode τ_k^h misses its deadline upon a last release time assignment π, it must be either the case that*

$$C_k^h + \sum_{i=1}^{k-1} C_i^{max} > D_k^h \tag{6.4}$$

or

$$C_k^h > D_k^h - \sum_{i=1}^{k-1} \left(U_i^{max} \cdot \left(D_k^h - \sum_{j=i}^{k-1} C_j^{max} \right) \right) - \sum_{i=1}^{k-1} C_i^{max} \tag{6.5}$$

Proof. In the case of Eq. (6.4), it is clear that there will be a deadline miss.

We now consider the case where Eq. (6.4) does not hold. From Lemma 6.3, we know that it is necessary for a deadline miss to occur: there exists an assignment $t_1, t_2, \ldots, t_{k-1} \in [0, D_k^h)$ of the last release of task τ_i, $\forall i \in \{1, ..., k-1, k\}$

$$\sum_{j=1}^{k-1} DBF_j(t_j, \tau_k^h) + \sum_{j=1}^{i-1} C_j^{max} + C_k^h > t_i$$

$$\text{(By Lemma 6.2)} \Rightarrow \sum_{j=1}^{k-1} U_j^{max} t_j + \sum_{j=1}^{i-1} C_j^{max} + C_k^h > t_i$$

where $t_k \equiv D_k^h$. Our objective is to find the infimum C_k^h such that the above constraints always hold. In fact, this is equivalent to the following linear programming (LP):

$$\begin{aligned}
\inf \quad & C^* \\
\text{s.t.} \quad & \sum_{j=1}^{k-1} U_j^{max} t_j + \sum_{j=1}^{i-1} C_j^{max} + C^* > t_i, && \forall 1 \le i \le k && \text{(6.6a)} \\
& t_i \le t_{i+1}, && \forall 1 \le i \le k-1 && \text{(6.6b)} \\
& t_i \ge 0, && \forall 1 \le i \le k-1 && \text{(6.6c)}
\end{aligned}$$

where Eq. (6.6c) and Eq. (6.6b) come from the definition of t_i and $t_k \equiv D_k^h$ for notational brevity. We now replace $>$ with $\ge$ in Eq. (6.6a) as infimum and minimum are the same if $\ge$ is used.

From Eq. (6.6a) when $i = k$, we get $C^* \ge t_k - \sum_{j=1}^{k-1} U_j^{max} t_j - \sum_{j=1}^{k-1} C_j^{max}$. Then, we can use this inequality by adding a slack variable $s \ge 0$ into its RHS to replace C^* in our objective, and thus finding the minimum C_k^h is equivalent to finding the maximum $\sum_{j=1}^{k-1} U_j^{max} t_j - s$ as t_k and $\sum_{j=1}^{k-1} C_j^{max}$ are constant. Additionally, by replacing C^* in Eq. (6.6a), we get

$$\begin{aligned}
& \sum_{j=1}^{k-1} U_j^{max} t_j + \sum_{j=1}^{i-1} C_j^{max} + t_k - \sum_{j=1}^{k-1} U_j^{max} t_j - \sum_{j=1}^{k-1} C_j^{max} + s \ge t_i \\
\equiv & t_k - \sum_{j=i}^{k-1} C_j^{max} + s \ge t_i, \quad \forall 1 \le i \le k-1
\end{aligned}$$

After reformulation, we have the following LP:

$$\begin{aligned}
\max \quad & \sum_{j=1}^{k-1} U_j^{max} t_j - s \\
\text{s.t.} \quad & t_k - \sum_{j=i}^{k-1} C_j^{max} + s \geq t_i, && \forall 1 \leq i \leq k-1 && \text{(6.7a)} \\
& t_i \leq t_{i+1}, && \forall 1 \leq i \leq k-1 && \text{(6.7b)} \\
& s, t_i \geq 0, && \forall 1 \leq i \leq k-1 && \text{(6.7c)}
\end{aligned}$$

The objective function is maximized when t_i is maximized and s is minimized if all the constraints in Eq. (6.7a), Eq. (6.7b) and Eq. (6.7c) are feasible. From Eq. (6.7a), any increase Δ on s will increase the feasible range of t_i by at most Δ, but, at the same time, the objective function decreases due to the assumption that $\sum_{i=1}^{n} U_i \leq 1$, i.e., $\Delta \sum_{i=1}^{n} U_i - \Delta \leq 0$. Thus, the objective function is maximized when $s = 0$. From Eq. (6.7a), t_i is upper bounded by $t_k - \sum_{j=i}^{k-1} C_j^{max}$, by assuming $s = 0$. In other words, if there are no constraints violated by setting $t_i = t_k - \sum_{j=i}^{k-1} C_j^{max}$ and $s = 0$, then the objective function is maximized.

By setting $t_i = t_k - \sum_{j=i}^{k-1} C_j^{max}$ and $s = 0$, the given condition that Eq. (6.4) does not hold implies t_i's non-negativity by Eq. (6.7c), i.e., $t_k - \sum_{j=i}^{k-1} C_j^{max} \geq D_k^h - \sum_{j=1}^{k-1} C_j^{max} \geq C_k^h > 0$. In addition, this setting assures that $t_i \leq t_{i+1}$ by Eq. (6.7b). This immediately follows that the maximum $\sum_{j=1}^{k-1} U_j^{max} t_j - s$ occurs when all the constraints in Eq. (6.7a) are active and $s = 0$, and thus we have that for $1 \leq i \leq k-1$

$$t_i = t_k - \sum_{j=i}^{k-1} C_j^{max} \tag{6.8}$$

Replacing t_i in $C^* = t_k - \sum_{j=1}^{k-1} U_j^{max} t_j - \sum_{j=1}^{k-1} C_j^{max} - s$ by the above equalities and $s = 0$, we obtain the minimum C^*, as represented in the RHS of Eq. (6.5). Thus, we conclude that if mode τ_k^h misses its deadline upon a last release time assignment π and Eq. (6.4) does not hold, it must the case that $C_k^h > C^*$. Hence, this lemma is proven. □

However, the necessary condition above for a deadline miss is established upon a given last release ordering π. Without knowing the exact ordering observed in the schedule, intuitively, one may relax the ordering assumption by examining all the possible permutations. However, this is computationally intractable as $(k-1)!$ permutations have to be necessarily checked. Fortunately, the following lemma shows that $D_k^h - \sum_{i=1}^{k-1} \left(U_i^{max} \cdot \left(D_k^h - \sum_{j=i}^{k-1} C_j^{max} \right) \right) - \sum_{i=1}^{k-1} C_i^{max}$ is minimized for a specific ordering of π.

Lemma 6.5. *$D_k^h - \sum_{i=1}^{k-1}\left(U_i^{max}\cdot\left(D_k^h - \sum_{j=i}^{k-1} C_j^{max}\right)\right) - \sum_{i=1}^{k-1} C_i^{max}$ is minimized when the last release time ordering π of the $k-1$ higher priority tasks is with a non-increasing order of β_i.*

Proof. Suppose that π^* does not follow a non-increasing order of β_i. That is, there exists ℓ in the ordering π^* such that $\beta_\ell < \beta_{\ell+1}$ for some $\ell = 1, 2, \ldots, k-2$. We can now swap these two tasks, and this results in a new last release time ordering π'. By inspecting the term to be proved, we know that the last release ordering only changes $\sum_{i=1}^{k-1}\left(U_i^{max}\sum_{j=i}^{k-1} C_j^{max}\right)$. Therefore, to prove the lemma, we just have to show that the ordering π' has smaller $\sum_{i=1}^{k-1}\left(U_i^{max}\sum_{j=i}^{k-1} C_j^{max}\right)$ than the ordering π^*. By comparing these two orderings π^* and π', the term $\left(U_i^{max}\sum_{j=i}^{k-1} C_j^{max}\right)$ remains the same in π and π' for any $i \neq \ell$ and $i \neq \ell+1$. Therefore, the difference (the result by using π' minus the result by using π^*) in the term $\sum_{i=1}^{k-1}\left(U_i^{max}\sum_{j=i}^{k-1} C_j^{max}\right)$ is

$$\left(U_{\ell+1}^{max}\sum_{j=\ell}^{k-1} C_j^{max} + U_\ell^{max}\sum_{j=\ell+1}^{k-1} C_j^{max}\right) - \left(U_\ell^{max}\sum_{j=\ell}^{k-1} C_j^{max} + U_{\ell+1}^{max}\sum_{j=\ell+1}^{k-1} C_j^{max}\right)$$

It follows that

$$\begin{aligned} U_{\ell+1}^{max}C_\ell^{max} - U_\ell^{max}C_{\ell+1}^{max} &= U_{\ell+1}^{max}U_\ell^{max}\left(\frac{C_\ell^{max}}{U_\ell^{max}} - \frac{C_{\ell+1}^{max}}{U_{\ell+1}^{max}}\right) \\ &= U_{\ell+1}^{max}U_\ell^{max}(\beta_\ell - \beta_{\ell+1}) <^1 0, \end{aligned}$$

where the last inequality comes from the definition of ℓ and $<^1$ is due to the non-increasing order of β_i.

Therefore, we can keep swapping the last release ordering to reduce $\sum_{i=1}^{k-1}\left(U_i^{max}\sum_{j=i}^{k-1} C_j^{max}\right)$. By adopting the above swapping procedure repeatedly, we reach the conclusion. □

Since Lemma 6.4 is the necessary condition for a deadline miss to occur, equivalently, the negation of Lemma 6.4 is the sufficient condition for a deadline to be met. We can now conclude the following schedulability test by using Lemma 6.4 and Lemma 6.5.

Theorem 6.4 (QT-FPT). *A constrained-deadline multi-mode task τ_k^h is schedulable if*

$$D_k^h - \sum_{i=1}^{k-1} C_i^{max} - C_k^h \geq 0$$

and

$$C_k^h \leq D_k^h - \sum_{i=1}^{k-1}\left(U_i^{max}\cdot\left(D_k^h - \sum_{j=i}^{k-1} C_j^{max}\right)\right) - \sum_{i=1}^{k-1} C_i^{max}$$

where the higher priority tasks τ_i are indexed by non-increasing β_i.

Computational Complexity. There are $(M_1 + M_2 + ... + M_n)$ task modes to be checked for the schedulability. The test for each mode runs in time of $O(n^2)$ without any optimization while higher-priority tasks have to be sorted by non-increasing β_i beforehand, which requires $O(n \cdot \log n)$. Therefore, the proposed test can be computed in polynomial time of $O(n^2 \cdot (M_1 + M_2 + ... + M_n))$.

6.1.2 *Schedulability Analysis under FPM Scheduling*

In this subsection, we derive a schedulability test for multi-mode systems under FPM scheduling by using a similar technique provided in Section 6.1.1. Specifically, we study RM scheduling, one example of FPM, and then provide utilization bounds for implicit-deadline multi-mode systems, in Theorem 6.7 and Theorem 6.8.

Several results have been reported on FPT scheduling [SY13; DFPS14]. We here show that FPT scheduling may perform rather poorly comparing to FPM scheduling.

Example. Consider the set of tasks defined in Table 6.3 where task τ_1 is a sporadic task; τ_2 is a multi-mode task; $0 < \epsilon \leq 0.5$. If τ_1 has higher priority than τ_2, mode τ_2^1 cannot meet its deadline. Similarly, if τ_2 has higher priority than τ_1, task τ_1 cannot meet its deadline due to the preemption resulting from the mode τ_2^2.

Task	Mode	C_i^m	T_i^m	D_i^m
τ_1	1	1	$1/\epsilon$	$1/\epsilon$
τ_2	1	ϵ	1	1
	2	$1/\epsilon$	$(1/\epsilon)^2$	$(1/\epsilon)^2$

Table 6.3: An example of associating fixed priority with the task set

According to the example, when FPT is adopted, a feasible scheduling that will always meet deadlines does not exist, no matter task τ_1 or task τ_2 is assigned to the higher priority for any $\epsilon > 0$. Therefore, the utilization bound for such a case is 0 when ϵ is close to 0. That is, there exist input instances with a very small utilization such that the task set is not schedulable under any FPT assignments.

A naive way to check the schedulability under RM is to pessimistically consider each mode as an individual sporadic task. Subsequently, by inspecting the Liu & Layland bound [LL73], the schedulability of the above example can be guaranteed when $\frac{C_1^1}{T_1^1} + \frac{C_2^1}{T_2^1} + \frac{C_2^2}{T_2^2} \leq 3(2^{\frac{1}{3}} - 1) = 0.779$ and accordingly $\epsilon \leq \frac{0.779}{3}$. Nevertheless, the schedulability bound is inversely proportional to the total number of modes in a system, and it thus cannot be put into practice.

To the best of our knowledge, there is no previous work studying on the mode change system under FPM.

6.1.2.1 *Carry-in Effect under FPM Scheduling*

Unfortunately, the critical instant provided by Lemma 6.1 is no longer satisfied under FPM scheduling due to the so-called *carry-in* effect. We show this with the following example.

Example. Consider a system with two tasks: a sporadic task $\tau_1 = \{(10, 30, 30)\}$ and a multi-mode task $\tau_2 = \{\tau_2^1 = (5, 10, 10), \tau_2^2 = (16, 30, 30)\}$. We assign the highest, the middle, and the lowest priority to task τ_2^1, task τ_1, and task τ_2^2, respectively. As shown in Figure 6.2, five additional time-units jobs from mode τ_1 are carried into the interval of task τ_2^2's releases due to the preemption from higher-priority mode τ_2^1 under FPM scheduling. Hence, task τ_1^2 misses its deadline at time 40, whereas it will meet its deadline if released as the *synchronous arrival sequence*, mentioned in Lemma 6.1.

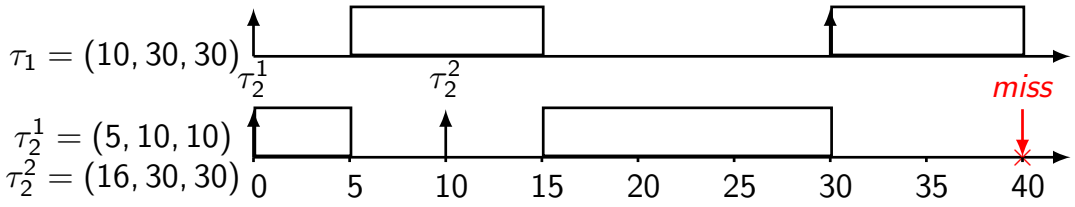


Figure 6.2: An example of carry-in effects under FPM.

6.1.2.2 *Sufficient Test for FPM Scheduling*

In this subsection, we first use the concept of problem window extension to quantify the carry-in job and then reuse the technique derived in the previous section for FPT scheduling to establish schedulability tests for FPM scheduling.

Consider any legal sequence of jobs of task system τ, on which a deadline miss occurs. Without loss of generality, we assume mode τ_k^h is the mode that there exist $k-1$ tasks such that each task has at least one mode that is assigned higher priority than mode τ_k^h. Suppose that mode τ_k^h is the one to first miss a deadline and arrives at time-instant t_a, and this deadline miss occurs at time-instant $t_a + D_k^h$.

We now discard all those jobs that have priority lower than τ_k^h's priority from this sequence. Since those jobs with priority lower than τ_k^h's have no effect on the scheduling of the jobs with priority higher than or equals to task τ_k^h, this schedule will see a deadline miss of mode τ_k^h at time-instant $t_a + D_k^h$, and it will also be the first deadline miss in the schedule. Hence in this section, we consider only such a *reduced* sequence of jobs.

Let t_0 denote the latest time-instant with $t_0 \leq t_a$ at which the processor is idle (or executing some lower-priority jobs before discarding), c.f. Figure 6.3. Clearly, t_0 exists and is well-defined. Let $A_k^h = t_a - t_0$ and $t_d = t_a + D_k^h$. The technique to extend the interval of interest is needed for identifying the necessary condition for a deadline miss where the carry-in effect may occur. We have the following observation:

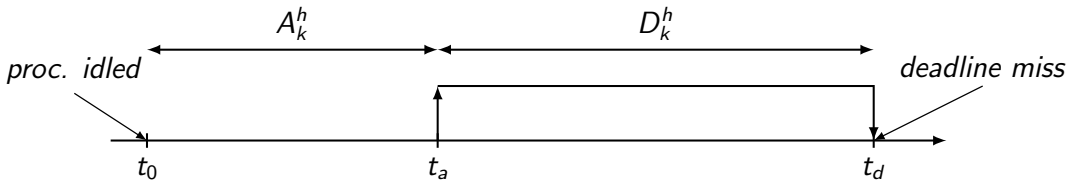


Figure 6.3: Illustration: a mode of task τ_i arrives at t_a and misses its deadline at time-instant t_d. The latest time-instant prior to t_a when the processor is idle is denoted t_0.

- Let $\mathcal{X}$ denote the workload contributed from task τ_k, that is to under anlaysis, over $[t_0, t_d)$. Due to mode τ_k^h's deadline miss, the amount of execution time, executed for mode τ_k^h, is strictly less than C_k^h over $[t_a, t_d)$. By the definition of t_0, the jobs of task τ_k that contribute to the interval $[t_0, t_d)$ must arrive no earlier than t_0. The jobs of task τ_k contributing to $[t_0, t_a)$ is bounded by the multi-mode demand bound function of task τ_i over $[t_0, t_a)$. Hence, we have

$$\mathcal{X} \leq C_k^h + DBF_i(t_a - t_0, \tau_k^h) \tag{6.9}$$

- The higher-priority jobs of any multi-mode task must arrive no earlier than time-instant t_0. Hence, the work executing prior to the last release time t_i is bounded from above by its multi-mode demand bound function with an interval length of $t_i - t_0$.

Consequently, we provide a necessary condition for a deadline to be missed under FPM scheduling in the following lemma:

Lemma 6.6. *If task mode τ_k^h misses its deadline under FPM upon an assignment π, it must be the case that there exist an $A_k^h \geq 0$ and an assignment $t_1, t_2, \ldots, t_{k-1}$ of the last release of task τ_i such that $\forall i \in \{1, ..., k-1, k\}$,*

$$\mathcal{X} + \sum_{\tau_j \in hp(\tau_k^h)} DBF_j(t_j - t_0, \tau_k^h) + \sum_{j: t_j \leq t_i} C_j^{max}(\tau_k^h) > t_i - t_0$$

and for time t_d

$$\mathcal{X} + \sum_{\tau_j \in hp(\tau_k^h)} DBF_j(t_j - t_0, \tau_k^h) + \sum_{j=1}^{k-1} C_j^{max}(\tau_k^h) > A_k^h + D_k^h$$

Proof. This is by the above discussions with a similar proof as in Lemma 6.3. □

Without loss of generality, we can simply set t_0 to 0.

Theorem 6.5 (QT-FPM). *Task mode τ_k^h is schedulable under an FPM scheduling if*

$$D_k^h - \sum_{i=1}^{k-1} C_i^{max}(\tau_k^h) - C_k^h \geq 0 \tag{6.10}$$

and

$$C_k^h \leq D_k^h - \sum_{i=1}^{k-1} \left(U_i^{max}(\tau_k^h) \cdot t_i^* \right) - \sum_{i=1}^{k-1} C_i^{max}(\tau_k^h) \tag{6.11}$$

where

$$t_i^* = D_k^h - \sum_{j=i}^{k-1} C_j^{max}(\tau_k^h)$$

in which higher-priority tasks τ_i are indexed by non-increasing $\beta_i(\tau_k^h)$.

Proof. In case of Eq. (6.10), it is clear that there will be a deadline miss.

By Lemma 6.6 and Eq. (6.9) and using the same concept of Lemma 6.4 and Lemma 6.5, we then have the necessary condition for the deadline miss under FPM:

$$\begin{aligned} C_k^h > D_k^h - \sum_{i=1}^{k-1} \left(U_i^{max}(\tau_k^h) \cdot t_i^* \right) - \sum_{i=1}^{k-1} C_i^{max}(\tau_k^h) \\ + A_k^h \left(1 - \sum_{i=1}^{k} U_i^{max}(\tau_k^h) \right) \end{aligned} \tag{6.12}$$

where

$$t_i^* = D_k^h - \sum_{j=i}^{k-1} C_j^{max}(\tau_k^h)$$

Notice that due to the assumption that $U^{sum} \leq 1$, the RHS of the above inequality is monotonically increasing with the length of interval A_k^h. By substituting $A_k^h = 0$ and taking the negation of Eq. (6.12), this theorem is proven. □

Roughly speaking, the work contributing over interval $[t_0, t_d)$ where $t_0 \neq t_a$ will be no more than that in the case $t_0 = t_a$. One may observe that in order to make the interval $[t_0, t_a)$ busy, there will be a loss of some workloads over $[t_a, t_d)$ due to $U^{sum} \leq 1$ according to our analysis.

6.1.2.3 *Utilization Bounds for Implicit-Deadline Tasks under RM*

Here, we specifically study RM scheduling in implicit-deadline multi-mode systems where the relative deadline of each mode is equal to its period. Under RM scheduling, we first observe that the interference from the last release can be naturally bounded:

Lemma 6.7. *Under RM scheduling, for any task τ_i with respect to mode τ_k^h*

$$C_i^{max}(\tau_k^h) \leq U_i^{max}(\tau_k^h) \cdot T_k^h$$

Proof. Under RM scheduling, only those modes with period $T_a^b \leq T_k^h$ will be assigned higher priority than τ_k^h. By the definition of $C_i^{max}(\tau_k^h)$ we have that

$$C_i^{max}(\tau_k^h) = \max_{\tau_a^b \in hp(\tau_k^h) \cap \tau_i} \{U_a^b \cdot T_a^b\} \leq \qquad\qquad U_i^{max}(\tau_k^h) \cdot T_k^h$$

□

By Lemma 6.7 we now pessimistically inflate each $C_i^{max}(\tau_k^h)$ to $U_i^{max}(\tau_k^h) \cdot T_k^h$. Note that thereafter we can simply relax the ordering specified in Theorem 6.5 since according to Theorem 6.5 for all tasks τ_i

$$\beta_1 = \beta_2 = ... = \beta_{k-1} = \frac{C_i^{max}(\tau_k^h)}{U_i^{max}(\tau_k^h)} = T_k^h$$

after the inflation. Theorem 6.6 immediately follows:

Theorem 6.6. *Implicit-deadline mode τ_k^h is schedulable under RM if*

$$U_k^h \leq 1 - 2\sum_{i=1}^{k-1} U_i^{max}(\tau_k^h) + \frac{1}{2}\left(\sum_{i=1}^{k-1} U_i^{max}(\tau_k^h)\right)^2 + \frac{1}{2}\sum_{i=1}^{k-1}\left(U_i^{max}(\tau_k^h)\right)^2 \tag{6.13}$$

Proof. In case of Eq. (6.10), due to $U^{sum} \leq 1$, we know that $T_k^h \cdot \left(1 - \sum_{i=1}^{k-1} U_i^{max}(\tau_k^h) - U_k^h\right) \geq 0$. For notational simplicity, we denote U_i as $U_i^{max}(\tau_k^h)$ in this proof. In case of Eq. (6.11), substituting C_i^{max} in Eq. (6.11) by $U_i^{max} \cdot T_k^h$ and D_k^h by T_k^h, we have

$$\begin{aligned}
\frac{C_k^h}{T_k^h} &\leq 1 - \sum_{i=1}^{k-1}\left(U_i \cdot \left(1 - \sum_{j=i}^{k-1} U_j\right)\right) - \sum_{i=1}^{k-1} U_i \\
&= 1 - 2\sum_{j=1}^{k-1} U_j + \sum_{i=1}^{k-1}\left(U_i \cdot \left(\sum_{j=i}^{k-1} U_j\right)\right) \\
&= 1 - 2\sum_{i=1}^{k-1} U_i + \frac{1}{2}\left(\sum_{i=1}^{k-1} U_i\right)^2 + \frac{1}{2}\sum_{i=1}^{k-1} U_i^2
\end{aligned}$$

Hence this theorem is proven. □

The schedulability test proposed in the previous section must check each task mode τ_k^h individually. If one is willing to sacrifice some precision, we can further provide two linear-time schedulability tests for a task set, called *quadratic bound* (QB) and *total utilization bound*, in Theorem 6.7 and 6.8, respectively.

Theorem 6.7 (QB-RM). *A multi-mode task set τ with implicit-deadline is schedulable under RM scheduling if*

$$U_a \leq 1 - 2 \sum_{\tau_i \in \tau \setminus \{\tau_a\}} U_i^{max} + \frac{1}{2} \left(\sum_{\tau_i \in \tau \setminus \{\tau_a\}} U_i^{max} \right)^2 + \frac{1}{2} \sum_{\tau_i \in \tau \setminus \{\tau_a\}} (U_i^{max})^2 \tag{6.14}$$

where $U_a = min_{\tau_i \in \tau}\{U_i^{max}\}$.

Proof. It is clear that for any task $\tau_k \in \tau$ if

$$U_k^{max} \leq 1 - 2 \sum_{\tau_i \in \tau \setminus \{\tau_k\}} U_i^{max} + \frac{1}{2} \left(\sum_{\tau_i \in \tau \setminus \{\tau_k\}} U_i^{max} \right)^2 + \frac{1}{2} \sum_{\tau_i \in \tau \setminus \{\tau_k\}} (U_i^{max})^2 \tag{6.15}$$

then Eq. (6.13) must hold for all modes $\tau_k^h \in \tau_k$. We then show that if Eq. (6.15) holds by the choice of $U_k^{max} = min_{\tau_i \in \tau}\{U_i^{max}\}$, then it also holds for all the other cases.

Let τ_a denote the task with minimum U_i^{max} among all tasks. Given that Eq. (6.15) holds for τ_a, we have that

$$U_a^{max} \leq 1 - 2 \sum_{\tau_i \in \tau \setminus \{\tau_a\}} U_i^{max} + \frac{1}{2} \left(\sum_{\tau_i \in \tau \setminus \{\tau_a\}} U_i^{max} \right)^2 + \frac{1}{2} \sum_{\tau_i \in \tau \setminus \{\tau_a\}} (U_i^{max})^2$$

We prove this by contradiction: *suppose for some* $U_b^{max} > U_a^{max}$ *Eq.* (6.14) *does not hold*:

$$U_b^{max} > 1 - 2 \sum_{\tau_i \in \tau \setminus \{\tau_b\}} U_i^{max} + \frac{1}{2} \left(\sum_{\tau_i \in \tau \setminus \{\tau_b\}} U_i^{max} \right)^2 + \frac{1}{2} \sum_{\tau_i \in \tau \setminus \{\tau_b\}} (U_i^{max})^2$$

For notational simplicity, let U_a^{max} and U_b^{max} denote U_a and U_b. Summing the above two inequities we have

$$\begin{aligned}
U_a - U_b &< 2\,(U_a - U_b) + \frac{1}{2}\,(U_b^2 - U_a^2) \\
&\quad + \frac{1}{2}\left(U_a + U_b + 2\sum_{\tau_i \in \tau \setminus \{\tau_a, \tau_b\}} U_i^{max}\right)(U_b - U_a) \\
&\quad \{\text{reorganization}\} \\
&= (U_a - U_b)\left(2 - \sum_{\tau_i \in \tau \setminus \{\tau_a, \tau_b\}} U_i^{max}\right) \\
&\{\text{reorganization}\} \\
&\Rightarrow (U_a - U_b)\,(1 - \sum_{\tau_i \in \tau \setminus \{\tau_a, \tau_b\}} U_i^{max}) > 0 \\
&\{\text{divided by } (U_a - U_b) \text{ and recall } U_b > U_a\} \\
&\Rightarrow 1 - \sum_{\tau_i \in \tau \setminus \{\tau_a, \tau_b\}} U_i^{max} < 0 \\
&\Rightarrow \sum_{\tau_i \in \tau \setminus \{\tau_a, \tau_b\}} U_i^{max} > 1
\end{aligned}$$

which contradicts the assumptions that $U^{sum} \leq 1$ and $U_a, U_b > 0$. Hence this theorem is proven. □

Theorem 6.8. *An implicit-deadline multi-mode tasks system τ is schedulable under RM scheduling if*

$$U^{sum} \leq \begin{cases} \dfrac{2(n-1) - \sqrt{2(n-1)(n-2)}}{n} & \text{if } n \geq 3 \quad (6.16) \\ \dfrac{1}{2} + \dfrac{1}{2n} & \text{if } n = 2 \quad (6.17) \end{cases}$$

Proof. For $n = 2$ the minimization can be done directly by solving the differential equation in two variables. We discuss the case for $n \geq 3$ by using Lagrange Multiplier Method.

$$\begin{aligned}
&\text{minimize} && \sum_{i=1}^{n} U_i \\
&\text{subject to} && U_n = 1 - 2\sum_{i=1}^{n-1} U_i + \frac{1}{2}(\sum_{i=1}^{n-1} U_i)^2 + \frac{1}{2}\sum_{i=1}^{n-1} U_i^2 \quad (6.18a) \\
&&& U_n \geq 0 \quad (6.18b)
\end{aligned}$$

Let λ be the multiplier of the schedulability constraint by Eq. (6.18a) and μ be the multiplier of the constraint $U_n \geq 0$. The Lagrange function is

$$L(U_1, U_2, ..., U_n) = \sum_{i=1}^{n} U_i + \mu(-U_n)$$
$$+\lambda \left(1 - 2 \sum_{i=1}^{n-1} U_i + \frac{1}{2} (\sum_{i=1}^{n-1} U_i)^2 + \frac{1}{2} \sum_{i=1}^{n-1} U_i^2 - U_n \right)$$

with derivatives

$$\frac{\partial L}{\partial U_i} = \begin{cases} 1 - \mu - \lambda, & \text{if } i = n \quad (6.19a) \\ 1 + \lambda \left(-2 + U_i + \sum_{i=1}^{n-1} U_i \right) & \text{otherwise} \quad (6.19b) \end{cases}$$

A necessary condition for the minimum is that the two derivatives of (6.19a) and (6.19b) are zero. Eq. (6.19b) implies that for all $i \neq n$

$$U_1 = U_2 = ... = U_{n-1} \tag{6.20}$$

To solve these equations, we look at several cases:
Case 1: $\mu = 0$

When $\mu = 0$, $\lambda = 1$ by Eq. (6.19a) according to the necessary condition for the minimum. Setting $\lambda = 1$ in Eq. (6.19b) and using the above condition, we obtain optimum values U_i $\forall 1 \leq i \leq n-1$:

$$U_i = \frac{1}{n} \tag{6.21}$$

After solving the condition of (6.18a) by replacing Eq. (6.21), we have

$$U_n = \frac{1}{2} \left(\frac{3}{n} - 1 \right) \tag{6.22}$$

which leads to the violation of the nonnegative constraint $U_n \geq 0$ if $n > 3$. In case of $n = 3$, we have $U_n = 0$, which leads to Case 2.
Case 2: $\mu \neq 0$

When $\mu \neq 0$, it must be the case $U_n = 0$ due to the complementarity.

After solving Eq. (6.18a) with the condition (6.20) and $U_n = 0$, we get for all $i \neq n$

$$U_i = \frac{2(n-1) - \sqrt{2(n-1)(n-2)}}{n} \cdot \frac{1}{n-1} \tag{6.23}$$

It follows that

$$\sum_{i=1}^{n} U_i = \frac{2(n-1) - \sqrt{2(n-1)(n-2)}}{n} \tag{6.24}$$

□

Theorem 6.9. *An implicit-deadline multi-mode tasks system τ is schedulable under RM scheduling if*

$$U^{sum} \leq 2 - \sqrt{2} \approx 0.5857$$

Proof. By taking $n \to \infty$ for $n \geq 3$ in Theorem 6.8, Theorem 6.9 follows immediately. □

It is not difficult to see that QB and the total utilization bound are tight in Theorem 6.7 and in Theorem 6.8, respectively. The idea is that increasing the maximum utilization of any higher-priority task mode τ_i leads to mode τ_n^h's deadline miss. We illustrate this with the following example.

Example. For a given n, there are two modes in τ_i for $i = 1, 2, \ldots, n-1$ with given $U_i^{max} = \frac{2(n-1)-\sqrt{2(n-1)(n-2)}}{n} \cdot \frac{1}{n-1}$, in which T_i^2 is T_n^h, C_i^2 is $T_n^h \cdot U_i^{max}$, T_i^1 is $T_n^h \cdot \left(1 - \sum_{j=i}^{n-1} U_j^{max}\right)$, and C_i^1 is $T_i^1 \cdot U_i^{max}$. Thus, task mode τ_n^h can only be schedulable when its utilization is 0.

Comparison Between Total Utilization Bound and Quadratic Bound

Figure 6.4 illustrates the difference between QB and the total utilization bound in Theorem 6.7 and in Theorem 6.8, respectively, for two tasks. We can clearly see that the feasibility region below QB is larger than that below the total utilization bound, in Figure 6.4a. Moreover, as shown in Figure 6.4b, the summation of utilizations that can be schedulable under QB is larger, especially when U_1 is away from 0.5.

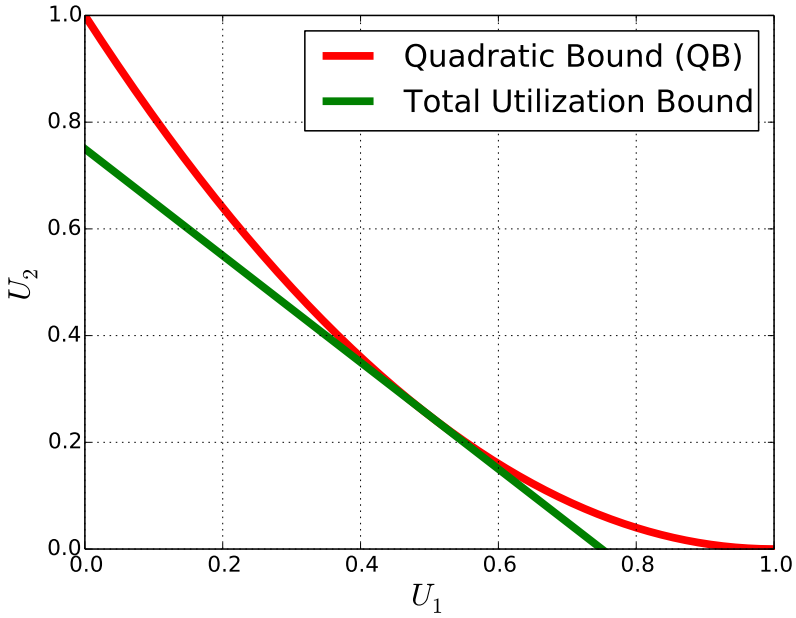
Computational Complexity. Considering all deadlines to be met, QT-RM runs in $O(n^2 \cdot (M_1 + M_2 + ... + M_n) + (M_1 + M_2 + ... + M_n) log(M_1 + M_2 + ... + M_n))$ where n^2 comes from the time complexity of the test for each mode and $(M_1 + M_2 + ... + M_n) log(M_1 + M_2 + ... + M_n)$ is the sorting time for prioritizing the modes at the beginning. Moreover, Theorem 6.6 can be computed in $O(n \cdot (M_1 + M_2 + ... + M_n) + (M_1 + M_2 + ... + M_n) \cdot log(M_1 + M_2 + ... + M_n))$, and QB-RM requires only $O(n + (M_1 + M_2 + ... + M_n))$ where $(M_1 + M_2 + ... + M_n)$ accounts for the time of deciding the maximum utilization among modes.

6.1.2.4 *Optimal Priority Assignment for FPM scheduling*

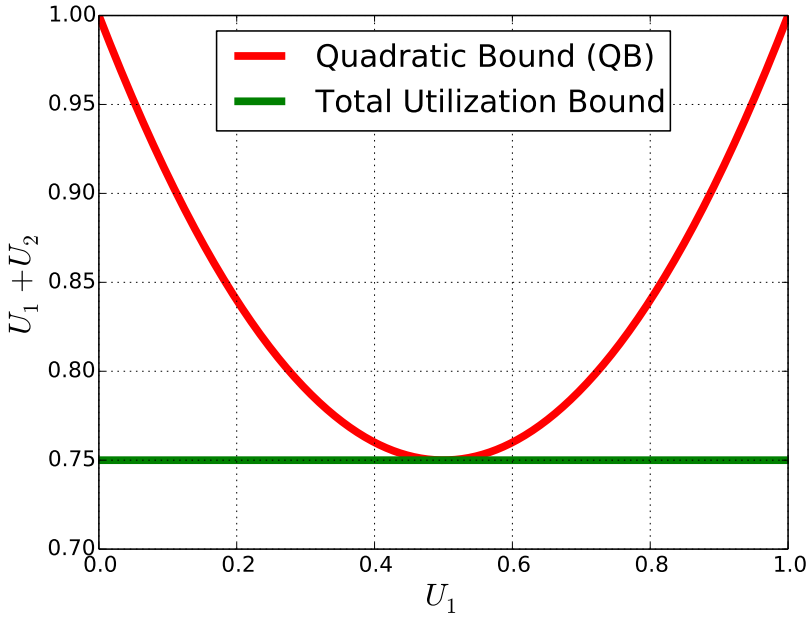
Despite RM scheduling algorithm preserves some properties of being good scheduling algorithms, e.g., utilization bound, RM for multi-mode tasks is however not optimal in the sense of FPM scheduling algorithm. We state this with the following theorem:

Theorem 6.10. *DM/RM scheduling is not an optimal FPM scheduling algorithm.*

Proof. We prove this theorem by showing: there exists a task set not deemed schedulable by DM scheduling algorithm, that is deemed schedulable by another model-level fixed-priority scheduling.



(a) Utilization of U_2 under given U_1



(b) Total utilization $U_1 + U_2$ under given U_1

Figure 6.4: Utilization Bounds for 2 implicit-deadline multi-mode tasks under RM

Table 6.4: An example of showing DM is not optimal under FPM scheduling.

Task	Mode	C_i^m	T_i^m	D_i^m	DM priority		optimal priority	
τ_1	1	4	8	8	1	$\surd$	1	$\surd$
	2	6	12	12	2	$\surd$	3	$\surd$
τ_2	1	5	14	14	3	$\times$	2	$\surd$

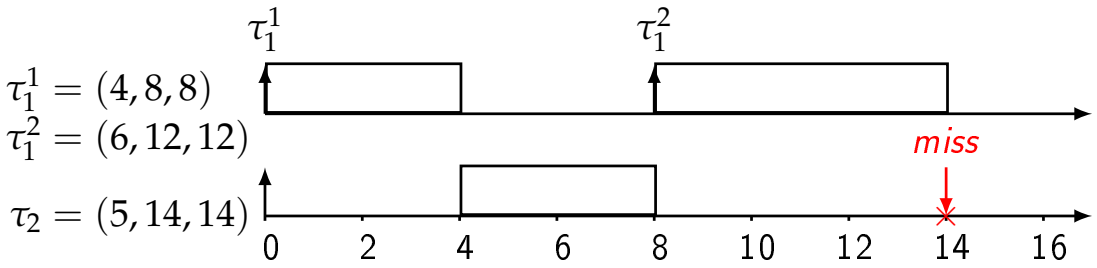


Figure 6.5: DM scheduling algorithm fails to produce a feasible schedule for task τ_2.

Consider a set of implicit-deadline tasks defined in Table 6.4 where task τ_1 is a multi-mode task consisting of two modes and task τ_2 is a sporadic task. Under DM scheduling mode 2 of task τ_1 is assigned the highest priority, whereas task τ_2 the lowest priority. It is evident that highest-priority task τ_1 is by all means schedulable. However, task τ_2 misses its deadline at the synchronous arrival release where task τ_1 releases its mode 1 at time-instant 0 and then mode 2 at time-instant 8, as illustrated in Figure 6.5.

Now, let us consider another mode-level priority assignment by swapping the priority levels of task τ_2 and mode 2 of task τ_1, as shown in Table 6.4 (optimal priority). It is not difficult to see that task τ_2 and mode 1 of task τ_1 are both schedulable.

Recall that the synchronous arrival release is not critical instant under FPM, at which the worst-case behavior in analyzing tasks is concretely captured. As a result, we need to consider the so-called *push-through* effect. We observe that there is a slack of four time units over any release of task τ_1^1, giving the CPU time to task τ_2. In other words, at most one additional execution unit of task τ_2 can be pushed through into the interval of τ_1^2's release, resulting in a response time of 12 for task τ_1^2, which is schedulable. The worst-case scenario is illustrated in Figure 6.6. □

Optimal Priority Assignment. For $\mathcal{M}$ modes, inspecting all possible priority orderings takes $O(\mathcal{M}!)$ time complexity, which is however computationally intractable. It has been shown in [Audo1] that it suffices to examine a polynomial number of priority orderings in systems with n periodic tasks, in total with at most $n(n+1)/2$ schedulability tests, for finding a feasible priority ordering. This is a significant improvement over inspecting all $n!$ possible priority orderings. This method is called *Optimal Priority*

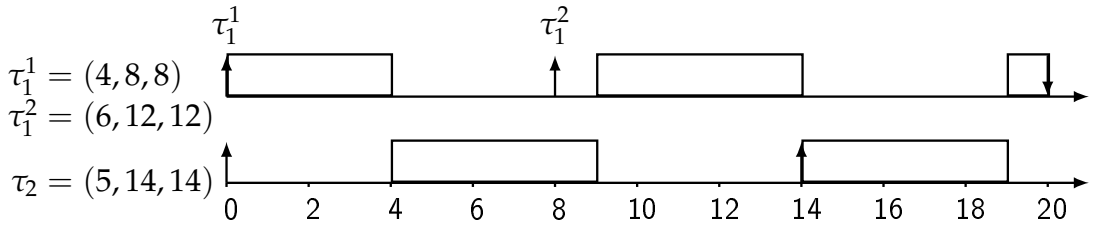


Figure 6.6: A feasible schedule for task τ_1^2 under FPM scheduling algorithm.

Assignment (OPA) [Aud91; Aud01; DB11b]. The OPA algorithm assigns each priority level ℓ to one of unassigned tasks that has no deadline miss along with the other unassigned task assumed to have higher priorities. The iterative priority assignment terminates as soon as either no unassigned task can be assigned at priority level ℓ or all priority levels are assigned. The pseudo code for the OPA algorithm, using some compatible schedulability test S is given in Algorithm 6.

Algorithm 6: Optimal Priority Assignment Algorithm

```
input  : A set of tasks τ
output : Priority assignment π and the feasibility of system τ
π ← ∅;
for each priority ℓ from |τ| to 1 do
    for each unassigned task τ_i do
        if task τ_i is schedulable at priority ℓ with all unassigned tasks (assume them as
        higher-priority tasks) according to schedulability test S then
            π(τ_i) ← ℓ // assign task τ_i to priority ℓ
            break ;
    if no task τ_i can be assigned to priority ℓ in the above loop then
        return "unscheduable";
return "scheduable";
```

In fact, checking the fixed-priority feasibility of a multi-mode task set under FPM scheduling can also be achieved by the OPA. For a schedulability test to be compatible with the OPA algorithm, it must comply with three conditions provided in [DB11b]. For completeness, we state these conditions as follows:

- **Condition 1.** The schedulability of a task τ_i may, according to test S, depend on any independent properties of tasks with priorities higher than ℓ, but not on any properties of those tasks that depend on their relative priority ordering.

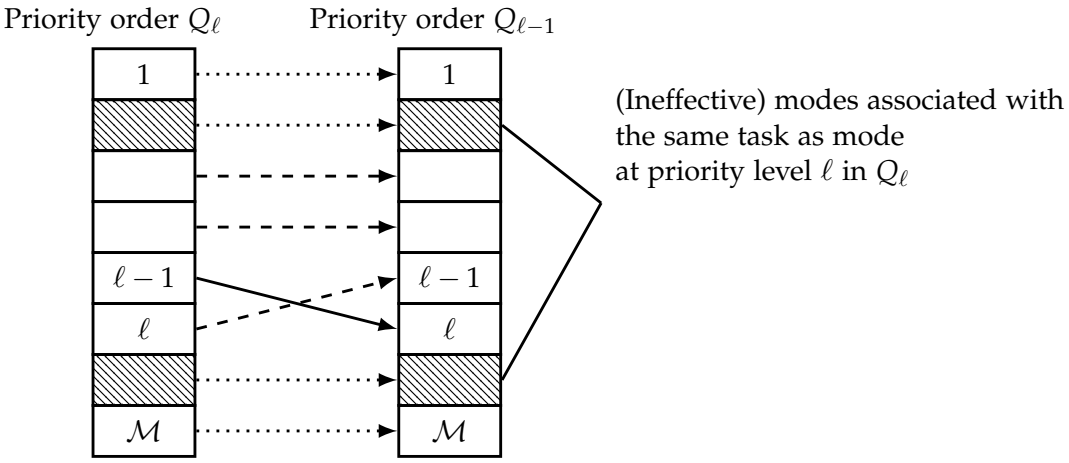


Figure 6.7: Transformation of the priority ordering

- **Condition 2.** The schedulability of a task τ_i may, according to test S, depend on any independent properties of tasks with priorities lower than ℓ, but not on any properties of those tasks that depend on their relative priority ordering.

- **Condition 3.** When the priorities of any two tasks of adjacent priority are swapped, the task being assigned the higher priority cannot become unschedulable according to test S, if it was previously schedulable at the lower priority.

In the following lemma we show that the sufficient schedulability test by Theorem 6.5 complies with the required conditions for the OPA algorithm:

Lemma 6.8. *The sufficient schedulability test by Theorem 6.5 complies with Conditions 1-3.*

Proof. Inspection of Eq. (6.11) shows that the schedulability of τ_k^h depends on the set of higher-priority modes but not on their relative priority ordering, as their associated tasks must be indexed by non-increasing $\beta_i(\tau_k^h)$ in the schedulability test. Hence, Condition 1 holds.

It is obvious that the schedulability of τ_k^h testing by Eq. (6.11) has no dependency on the set of modes with lower priority than τ_k^h, hence Condition 2 holds.

Consider two modes τ_k^h and τ_q^r initially at priorities $\ell - 1$ and ℓ, respectively. Inspection of Eq. (6.11) shows that the schedulability of τ_q^r is independent of the modes associated the same task as mode τ_q^r. If task τ_q^r is schedulable, it is still schedulable when it is shifted up one priority level to priority $\ell - 1$, since the only change of higher-priority mode demand is the removal of mode τ_k^h from the set of modes that have higher priority than mode τ_q^r (cf. Figure 6.7). Hence, Condition 3 holds. □

6.1.3 *Evaluations*

As can be seen, we have established several utilization-based tests for FPT and FPM scheduling. In this section we evaluate the effectiveness of the existing tests and the proposed utilization-based tests in terms of the number of tasksets that are deemed schedulable. First, we recap these tests as follows:

- Demand-based Test under FPT (DT-OPA): the time-demand test approach under FPT presented in Section III.D in [DFPS14] along with the OPA.
- Quadratic Test under FPT along with the OPA (QT-OPA) : Theorem 6.4.
- Quadratic Bound under FPM using RM (QB-RM): Theorem 6.7.
- Quadratic Test under FPM using RM (QT-RM): Theorem 6.5.

The metric to compare the results is to measure the *success ratio* of these tests for a given goal of task set utilization. We generate 100 task sets for each utilization level. The success ratio of a level is said to be the number of task sets that are schedulable divided by the number of task sets for this level.

Task Set Generation.

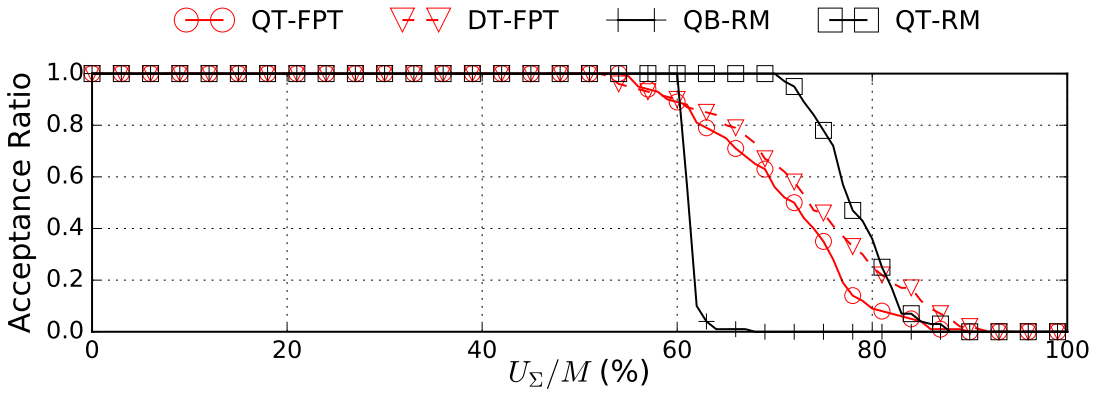
The task set was generated in a similar manner to the method in [DFPS14]. We first generated a set of sporadic tasks. The cardinality of the task set was 10. The UUniFast method [BB05] was adopted to generate a set of utilization values with the given goal. We use the approach suggested by Emberson et al. [ESD10] to generate the task periods according to the *exponential distribution*. The distribution of periods is within two orders of magnitude, i.e., 10ms-1000ms. Task relative deadlines are implicit, i.e., $D_i = T_i$. The worst-case execution time was computed accordingly, i.e. $C_i = T_i U_i$. We converted a proportion p of tasks to multi-mode tasks:

- A multi-mode task has M execution modes
- The generated sporadic task triplet (C_i, T_i, D_i) was assigned to the setting of task mode τ_i^1.
- We use a scaling factor 1.5 to assign the parameters of the other modes[2], i.e., $C_i^{m+1} = 1.5C_i^m$ and $T_i^{m+1} = 1.5T_i^m$.
- We randomly choose a mode to have the largest utilization. The worst-case execution times of the remaining modes were adjusted by multiplying them by uniform random values in the range $[0.75, 1]$.

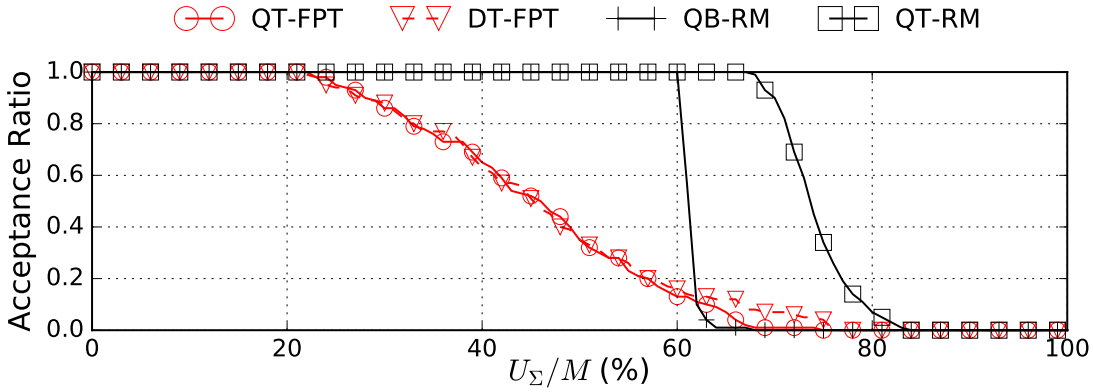
Checking the FPT feasibility of a multi-mode task set was achieved by using the method for sporadic tasks, called *Audsley's Algorithm* [Aud01], since the above tests comply with the required conditions for the compatibility provided in [DB11b]. We report the results evaluating a variety of M and p in the following subsection.

Result I. We first evaluated the impact on different numbers of modes when p was set to 0.5, as illustrated in Figure 6.8. With few number of modes, the schedulability of

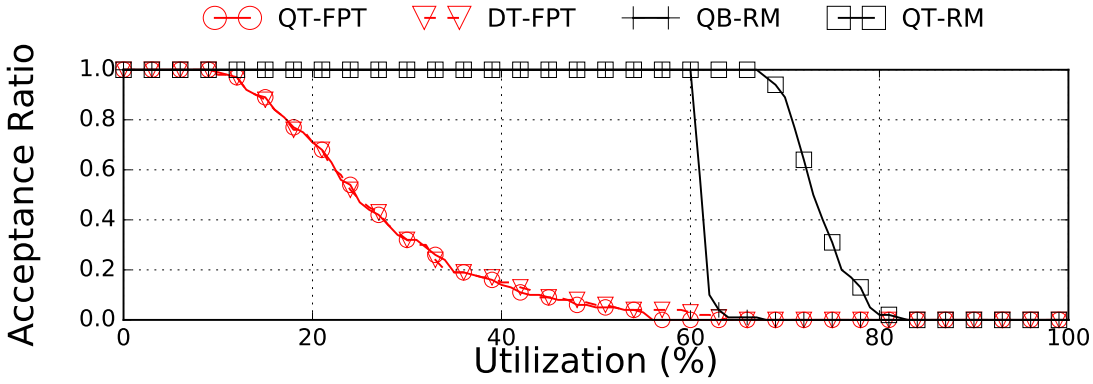
2 We use the dynamic programming approach to implement DT-FPT instead of using the ILP solver for the sake of efficiency. To comply with it, we apply a correction factor $\frac{\lceil T_i \rceil}{T_i}$ on both the generated period and execution time to ensure the discrete time model.



(a) $M = 5$

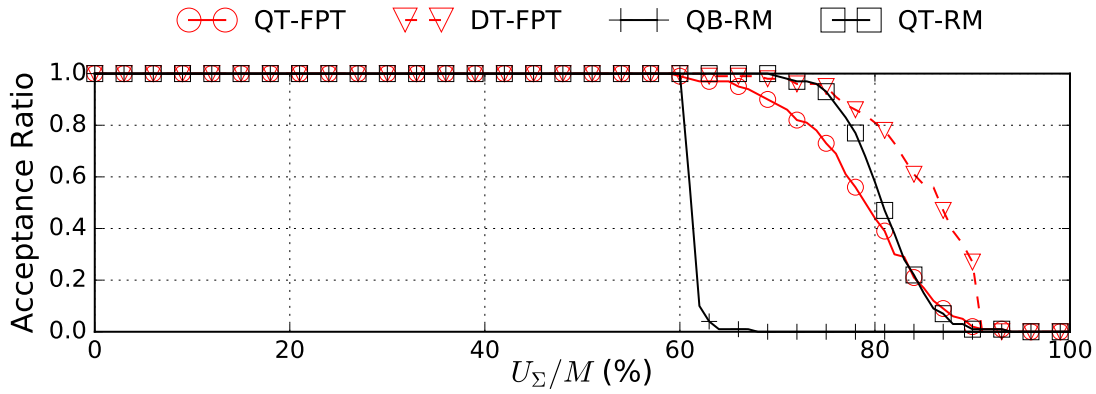


(b) $M = 8$

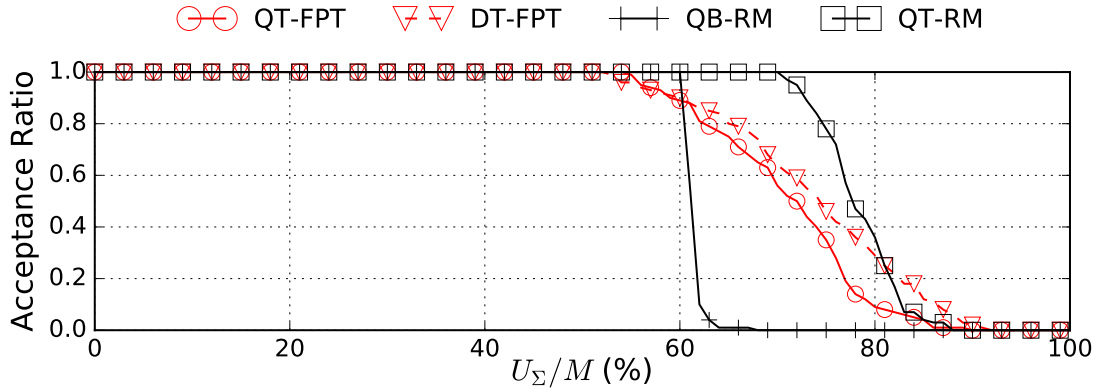


(c) $M = 10$

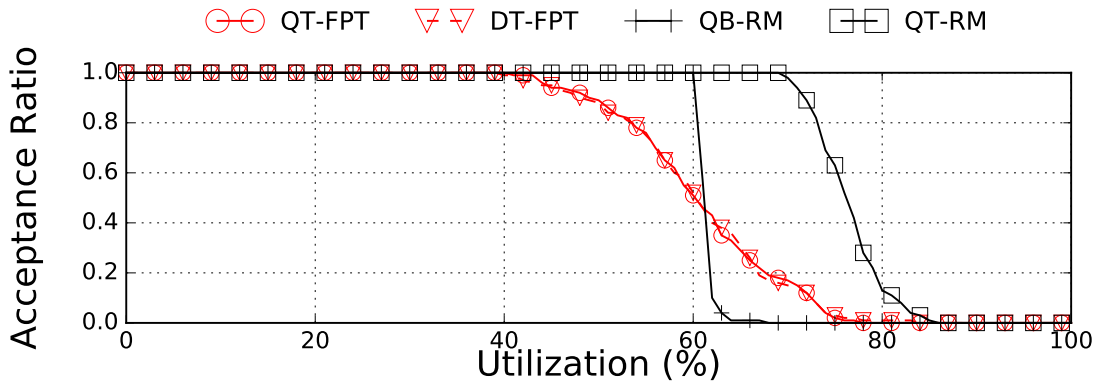
Figure 6.8: Comparison with different numbers of modes M of multi-mode tasks, assuming $p = 50\%$.



(a) $p = 20\%$

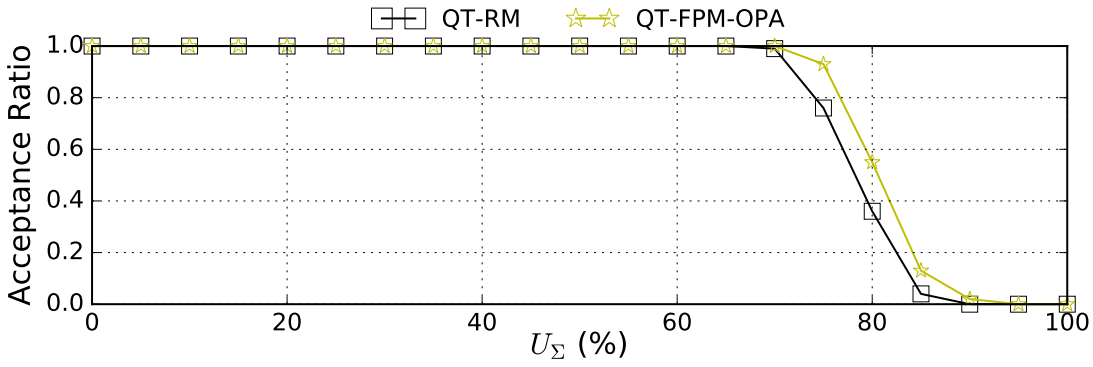


(b) $p = 50\%$

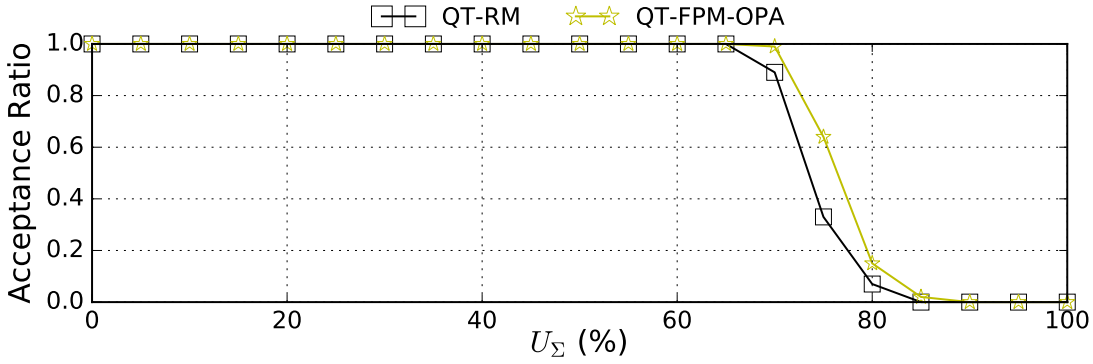


(c) $p = 80\%$

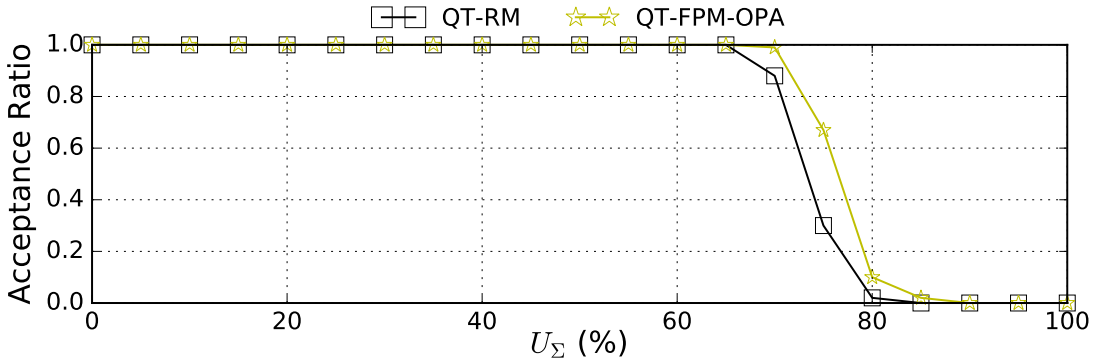
Figure 6.9: Comparison with different proportions p of multi-mode tasks, each of which is with 5 modes.



(a) $M = 5$



(b) $M = 8$



(c) $M = 10$

Figure 6.10: Comparison between the QTs associated with RM and OPA, with different numbers of modes M of multi-mode tasks, assuming $p = 50\%$.

multi-mode tasks under FPT scheduling can be provided with up to 90% of utilization; however, it drops significantly when the number of modes increases. This also accords

to the discussion in previous section: the FPT scheduling may perform rather poorly. In fact, under FPT, some jobs with sizable WCET released by different tasks may dominate each other, and this thus results in poor performance of the schedulability. On the other hand, RM scheduling is expected to be more sustainable due to the guarantee of theoretical qualifications. Moreover, QT-RM can accept the task set with total utilizations of up to 80% compared to only 65% by QB-RM. However, QB-RM takes the advantage on the runtime overhead and can be applied on-line efficiently when long execution times are prohibitive.

As far as the FPT scheduling is concerned, we observe that the proposed QT-FPT is competitive to the pseudo-polynomial-time DT-FPT. The more the number of modes, the more competitive the QT-FPT. One can imagine that the actual workload generated by a substantial modes can be very close to the one we overly approximate.

We further evaluate the impact of proportion of multi-mode tasks on schedulability. Figure 6.9 shows the data for different proportions of multi-mode tasks equal to 20%, 50%, and 80%. Similar results can be observed here.

Result II. In Chapter 6.1.2.4, we have shown that the QT by Theorem 6.5 is compatible with the OPA. It is interesting to see how much improvement over the RM can be brought up by using the OPA, denoted by QT-FPM-OPA. Figure 6.10 shows the comparisons by varying the number of modes in $\{5, 8, 10\}$. From the graphs, we can see that the QT along with the OPA can further admit task sets that are unschedulable by the test with the best performance, i.e., QT-RM. We conclude that the QT along with OPA brings up a great improvement in the number of tasksets deemed schedulable.

6.2 MULTIPROCESSORS

To schedule real-time tasks on multiprocessor platforms, there are two widely adopted approaches: *partitioned* and *global* scheduling. In this dissertation, we consider the partitioned strategy: each task is assigned statically to one processor and cannot migrate to the others.

As for the sporadic task model, when the relative deadline is equal to the period, the task allocation for partitioned scheduling on multiprocessor systems is analogous to the well-known *bin packing problem* [GJ79]. In the bin packing problem, the objective is to pack a set of items into a number of bins such that the volume of the packed items in every bin does not exceed that of the bin and the number of used bins is minimized. The problem of scheduling sporadic real-time tasks on multiprocessors regarding partitioned scheduling has been addressed in a number of studies. The details can be found in [DB11a]. Unfortunately, because the studied mode change model is a generalized model of the sporadic task model, those results are by no means applicable.

6.2.1 *Utilization Bounds in Uniprocessors*

As shown earlier in Theorem 6.9, a uniprocessor system comprised of a set of multi-mode rate-monotonic scheduled tasks is schedulable if

$$U^{sum} \leq 2 - \sqrt{2} \tag{6.25}$$

We could use this bound as the capacity of each bin to allocate tasks under partitioned scheduling; however, if doing so, we may fall short of some accuracy. Instead, we make use of a more precise utilization bound based a quadratic bound (denoted as QB), as shown in Theorem 6.7. For completeness, we state QB in the following lemma:

Lemma 6.9 (Theorem 6.7)**.** *A multi-mode task set Γ with implicit deadline is schedulable on a uniprocessor under RM scheduling if $\sum_{\tau_i \in \Gamma} U_i \leq 1$ and*

$$U_a \leq 1 - 2 \sum_{\tau_i \in \Gamma \setminus \tau_a} U_i + \frac{1}{2} \left(\sum_{\tau_i \in \Gamma \setminus \tau_a} U_i \right)^2 + \frac{1}{2} \sum_{\tau_i \in \Gamma \setminus \tau_a} (U_i)^2 \tag{6.26}$$

where $U_a = min_{\tau_i \in \Gamma} U_i$.

Note that the above lemma requires to test both $\sum_{\tau_i \in \Gamma} U_i \leq 1$ and Eq. (6.26). Testing only Eq. (6.26) is unsafe since the quadratic form in the right-hand side of Eq. (6.26) may become larger than 1 when $\sum_{\tau_i \in \Gamma \setminus \tau_a} U_i$ is sufficiently large. Nevertheless, one can show that the schedulability can be equivalently ensured if the tasks are sequentially tested by the condition Eq. (6.26) alone, which is exactly what task allocations do. We state this in the following theorem:

Theorem 6.11. *Suppose that the tasks in Γ are sorted and indexed such that $U_1 \geq U_2 \geq \cdots \geq U_{|\Gamma|}$. A multi-mode task set Γ with implicit deadline is schedulable on a uniprocessor under RM scheduling if $\forall k = 1, ..., \Gamma$,*

$$U_k \leq 1 - 2 \sum_{i=1}^{k-1} U_i + \frac{1}{2} \left(\sum_{i=1}^{k-1} U_i \right)^2 + \frac{1}{2} \sum_{i=1}^{k-1} (U_i)^2 \tag{6.27}$$

Proof. When k is 1, this holds naturally. In this proof, we show that any feasible allocation by using Eq. (6.27) implies that $\sum_{i=1}^{k} U_i \leq 1$ under the assumption that $\sum_{i=1}^{k-1} U_i \leq 1$ for $k = 2, 3, \ldots, |\Gamma|$.

Due to the evidence that $\left(\sum_{i=1}^{k-1} U_i \right)^2 \geq \sum_{i=1}^{k-1} (U_i)^2$, the satisfaction of Eq. (6.27) implies that

$$U_k \leq 1 - 2 \sum_{i=1}^{k-1} U_i + \frac{1}{2} \left(\sum_{i=1}^{k-1} U_i \right)^2 + \frac{1}{2} \left(\sum_{i=1}^{k-1} U_i \right)^2 \tag{6.28}$$

Let z denote $\sum_{i=1}^{k-1} U_i$. Replacing $\sum_{i=1}^{k-1} U_i$ by z, we have $U_k \leq 1 - 2z + z^2$. It follows that

$$U_k + z \leq 1 - z + z^2 \tag{6.29}$$

Hence, the total utilization $U_k + z$ of the k tasks to pass Eq. (6.27) is upper bounded by the right-hand side of Eq. (6.29), which is upper bounded by 1 if $0 \leq z \leq 1$. By the assumption that $U_i \leq 1$ for every task $\tau_i \in \Gamma$ and $0 \leq z \leq 1$, if Eq. (6.27) holds for every $k = 1, 2, \ldots, |\Gamma|$, the task set Γ can also pass the test in Lemma 6.9. □

6.2.2 *Reasonable Allocation Decreasing Algorithm*

In this section, we first introduce some simple heuristic allocation algorithms that have been developed for the bin-packing problem [GJ79]. An allocation algorithm is said to be *reasonable* if the tasks are allocated sequentially and the allocation algorithm fails to allocate a task to the previously allocated processors only if there is no processor having sufficient *capacity* to hold the pending task. There are three simple heuristic reasonable allocation (RA) algorithms, known as First-Fit (FF), Best-Fit (BF), and Worst-Fit (WF). The First-Fit algorithm places the item in the first bin that can accommodate the item. If no bin is found, it opens a new bin and puts the item to the new bin. The Best-Fit (Worst-Fit, respectively) algorithm places each item to the bin with the *lowest* (*largest*, respectively) remaining capacity among all the bins with sufficient capacity to accommodate the item.

In addition, if the items are ordered by a decreasing *ratio* of the value to the weight, of an item, before allocation, it is known as a RAD algorithm, described in [GJ79]. A RAD algorithm is characterized as follows:

- Items are ordered by a decreasing *ratio* of the value to the weight before allocation.
- Items are allocated sequentially by following the order defined above and the allocation algorithm fails only if there is no bin having sufficient capacity to hold the pending item.

It has been shown that the approximation factor of a RAD algorithm is 2 for the bin packing problem [Vaz01, Chapter 9]. More precisely, if a RAD algorithm allocates M bins with $M \geq 2$, the overall weight is strictly larger than $\frac{M}{2}$ times the bin size.[3] Therefore, we can directly use the utilization bound $2 - \sqrt{2}$ as the bin size and reach the following theorem.

Theorem 6.12 (RAD-TUB). *An implicit-deadline multi-mode system τ is feasible under RM scheduling and a RAD algorithm using the total utilization bound $2 - \sqrt{2}$ as the bin size if*

$$U_i \leq 2 - \sqrt{2} \quad \forall \tau_i \in \tau \text{ and } U^{sum} \leq (\frac{2 - \sqrt{2}}{2}) M \cong 0.293 \cdot M \tag{6.30}$$

3 Although the description in the book [Vaz01] is for the first-fit algorithm, the factor 2 holds also for any RAD algorithm. The description in [Vaz01] was for a trivial analysis with overall weight strictly larger than $\frac{M-1}{2}$. The overall weight $\frac{M}{2}$ can be easily achieved as well.

Algorithm 7: First-Fit Decreasing (FFD)

input : A set τ of multi-mode tasks and M identical processors
output: Task allocations Γ_j and the feasibility of system τ
sort the given N tasks in τ s.t. $U_1 \geq U_2 \geq \cdots \geq U_N$;
$\Gamma_j \leftarrow \emptyset, \forall j = 1, 2, \ldots, M$;
for $k = 1, 2, \ldots, N$ **do**
  for $j = 1, 2, \ldots, M$ **do**
    if $U_k \leq 1 - 2\sum_{\tau_i \in \Gamma_j} U_i + \frac{1}{2}(\sum_{\tau_i \in \Gamma_j} U_i)^2 + \frac{1}{2}\sum_{\tau_i \in \Gamma_j}(U_i)^2$ **then**
      $\Gamma_j \leftarrow \Gamma_j \cup \{\tau_k\}$; `// assign` τ_k `to proc.` j
      break (continue the outer loop) ;
  return "infeasible allocation";
return "feasible allocation";

Proof. This comes from the above discussions and the link to the bin packing problem. The condition $U_i \leq 2 - \sqrt{2}$ for every task τ_i is needed since RAD algorithms assume that the size of any item is smaller than the bin size. □

Using the total utilization bound is a simple solution, but we will show in the next subsection that using the QB in Theorem 6.11 can reach much better results.

6.2.3 *Utilization Bound by Using QB*

We consider any reasonable allocation decreasing algorithm using the QB in Theorem 6.11, for example, First-Fit Decreasing (FFD) algorithm using QB (also see Algorithm 7). Note that by using the RAD algorithm, when we consider to assign task τ_k, we know that the utilization of τ_k is no more than the utilization of the tasks that have been assigned onto the processors.

For the rest of the analysis, we assume that the N given tasks are sorted such that $U_1 \geq U_2 \geq \cdots \geq U_N$. Let τ_{Y+1} be the task that fails to be assigned to any of the processors, as Y tasks have been assigned onto the processors successfully. Let Γ_j be the set of the tasks that have been assigned on processor j before task τ_{Y+1} is considered. Thus, by Theorem 6.11, the schedulability condition for each processor that cannot accommodate task τ_{Y+1} can be concluded as follows:

$$\forall j \in [1, M], \quad U_{Y+1} > 1 - 2\sum_{\tau_i \in \Gamma_j} U_i + \frac{1}{2}(\sum_{\tau_i \in \Gamma_j} U_i)^2 + \frac{1}{2}\sum_{\tau_i \in \Gamma_j}(U_i)^2$$

Notice that $\sum_{j=1}^{M} |\Gamma_j| = Y$. By adding these M inequalities together, we therefore have that

$$MU_{Y+1} > M - 2\sum_{i=1}^{Y} U_i + \frac{1}{2}\sum_{i=1}^{Y}(U_i)^2 + \frac{1}{2}\left((\sum_{\tau_i \in \Gamma_1} U_i)^2 + (\sum_{\tau_i \in \Gamma_2} U_i)^2 + ... + (\sum_{\tau_i \in \Gamma_M} U_i)^2 \right) \tag{6.31}$$

Here, we can use Cauchy's Inequality $\sum_i r_i^2 \sum_i s_i^2 \geq (\sum_i r_i s_i)^2$ where all of $r_i, s_i \in \mathbb{R}$. When s_i is replaced by 1, we have $x \sum_i r_i^2 \geq (\sum_i r_i)^2$, where x is the number of terms of i in the summation. By this inequality, we then have that

$$\left((\sum_{\tau_i \in \Gamma_1} U_i)^2 + (\sum_{\tau_i \in \Gamma_2} U_i)^2 + ... + (\sum_{\tau_i \in \Gamma_M} U_i)^2 \right) \geq \frac{1}{M}(\sum_{i=1}^{Y} U_i)^2. \tag{6.32}$$

Consequently, it follows that by Eq. (6.31) and (6.32)

$$MU_{Y+1} > M - 2\sum_{i=1}^{Y} U_i + \frac{1}{2}\sum_{i=1}^{Y}(U_i)^2 + \frac{1}{2M}(\sum_{i=1}^{Y} U_i)^2 \tag{6.33}$$

In the following lemma, we first provide a necessary condition for a RAD algorithm to fail to allocate a task onto the M processors:

Lemma 6.10. *Let Y be the number of multi-mode tasks that have been assigned feasibly on M processors. If a reasonable allocation decreasing (RAD) algorithm using QB fails to allocate task τ_{Y+1}, then the following condition must hold:*

$$U^{sum} > \sum_{i=1}^{Y} U_i > \frac{1 + 2\gamma - \sqrt{1 + 2\gamma + 2\gamma^2}}{(1+\gamma)} \cdot M \tag{6.34}$$

where $\gamma = \frac{Y}{M}$.

Proof. We have shown that the condition in Eq. (6.33) is necessary for an RAD algorithm using QB to fail to allocate a task onto the M processors. Due to the non-increasing utilization ordering of the tasks, we have

$$\sum_{i=1}^{Y} U_i \geq Y U_{Y+1} \tag{6.35}$$

Our objective in this proof is to find the infimum $\sum_{i=1}^{\Upsilon} U_i$ such that Eq. (6.33) and (6.35) always hold. This is equivalent to the following quadratic programming (QP), where U_is are variables:

$$\text{min.} \quad \sum_{i=1}^{\Upsilon} U_i \tag{6.36a}$$

$$\text{s.t.} \quad MU_{\Upsilon+1} \geq M - 2\sum_{i=1}^{\Upsilon} U_i + \frac{1}{2M}(\sum_{i=1}^{\Upsilon} U_i)^2 + \frac{1}{2}\sum_{i=1}^{\Upsilon}(U_i)^2 \tag{6.36b}$$

$$\sum_{i=1}^{\Upsilon} U_i \geq \Upsilon U_{\Upsilon+1} \tag{6.36c}$$

Let λ be the multiplier of the schedulability constraint by Eq. (6.36b) and μ be the multiplier of the constraint by Eq. (6.36c). The Lagrange function is

$$L(U_1, U_2, ..., U_\Upsilon) = \sum_{i=1}^{\Upsilon} U_i - \mu(\sum_{i=1}^{\Upsilon} U_i - \Upsilon U_{\Upsilon+1}) -$$
$$\lambda\left(MU_{\Upsilon+1} - M + 2\sum_{i=1}^{\Upsilon} U_i - \frac{1}{2M}(\sum_{i=1}^{\Upsilon} U_i)^2 - \frac{1}{2}\sum_{i=1}^{\Upsilon}(U_i)^2\right)$$

with derivatives

$$\frac{\partial L}{\partial U_i} = \begin{cases} \Upsilon\mu - M\lambda, & \text{if } i = \Upsilon + 1 \quad (6.37a) \\ 1 - \mu - \lambda(2 - U_i - \frac{1}{M}\sum_{i=1}^{\Upsilon} U_i), & \text{otherwise.} \quad (6.37b) \end{cases}$$

A necessary condition for the minimum is that the two derivatives of (6.37a) and (6.37b) are zero. One can reformulate each derivative in Eq. (6.37b) equal to zero such that U_i is a function of M, μ, λ, and $\sum_{i=1}^{\Upsilon} U_i$. In doing so, we can notice that all the U_i have the same value. It follows that for all $i = 1, 2, \ldots, \Upsilon$:

$$U_1 = U_2 = ... = U_\Upsilon \tag{6.38}$$

To solve these equations, we look at several cases:
Case 1: $\mu = 0$. Since the derivative of (6.37a) must be 0 and $M > 0$, we then have $\lambda = 0$. This in turn results in an infeasible solution in (6.37b).
Case 2: $\lambda = 0$. Similarly, we get $\mu = 0$ by (6.37a). This also leads to an infeasible solution in (6.37b).
Case 3: $\mu \neq 0$ and, $\lambda \neq 0$. In this case, both constraints (6.36b) and (6.36c) must be active. By Eq. (6.38) and the activation of Eq. (6.36c), it follows that $U_{\Upsilon+1} = U_i$. Therefore, we have $U_1 = U_2 = ... = U_\Upsilon = U_{\Upsilon+1}$. After solving quadratic equation in one variable with (active) Eq. (6.36b), we have

$$\forall 1 \leq i \leq Y+1, \quad U_i = \frac{1+2\frac{Y}{M} - \sqrt{1+2\frac{Y}{M}+2\frac{Y^2}{M^2}}}{(1+\frac{Y}{M})} \cdot \frac{M}{Y}$$

It follows that

$$\sum_{i=1}^{Y} U_i = \frac{1+2\frac{Y}{M} - \sqrt{1+2\frac{Y}{M}+2\frac{Y^2}{M^2}}}{(1+\frac{Y}{M})} \cdot M$$

which is identical to Eq. (6.34). Hence, this theorem is proven. □

Thus, we see that Lemma 6.10 is necessary for a RAD algorithm to fail; equivalently, the negation of Lemma 6.10 is a sufficient condition for a system to be feasible for a RAD algorithm using QB:

Theorem 6.13. *An implicit-deadline multi-mode system τ is feasible under RM scheduling and by a RAD algorithm using QB if*

$$U^{sum} \leq \left(\frac{3-\sqrt{5}}{2}\right) M \cong 0.381 \cdot M \tag{6.39}$$

Proof. Notice that the right-hand side of Eq. (6.34) monotonically increases with respect to the value of $\gamma = \frac{Y}{M}$. Thus, the minimization in the right-hand side of Eq. (6.34) happens when γ is minimized. It is evident that at least M tasks can be feasibly assigned to the given processors before the RAD fails; hence, $\gamma \geq 1$. It follows that

$$\frac{1+2\gamma - \sqrt{1+2\gamma+2\gamma^2}}{(1+\gamma)} M \geq (\frac{3-\sqrt{5}}{2}) M \cong 0.381 \cdot M \tag{6.40}$$

Taking the negation of Lemma 6.10, this theorem is proven by contrapositive. □

Figure 6.11 shows the utilization bound under RAD with respect to γ, assumed to be given. Therefore, if a tighter lower bound on γ could be derived, we would be able to achieve better utilization bounds. For example, as shown in Figure 6.11, if $\gamma \geq 2$, we can conclude a utilization bound of $0.464 \cdot M$.

6.2.4 *Bounds Based on Max Utilization*

In this section, we derive a generalized utilization bound by considering the upper bound on the utilization for ever tasks α. Note that $U_i \leq \alpha$ for every task τ_i. This is motivated by Figure 6.11. If α is small enough, then, we would like to show that γ is also big enough in Lemma 6.10. We here introduce a new function $\phi(\alpha)$ defined as *the*

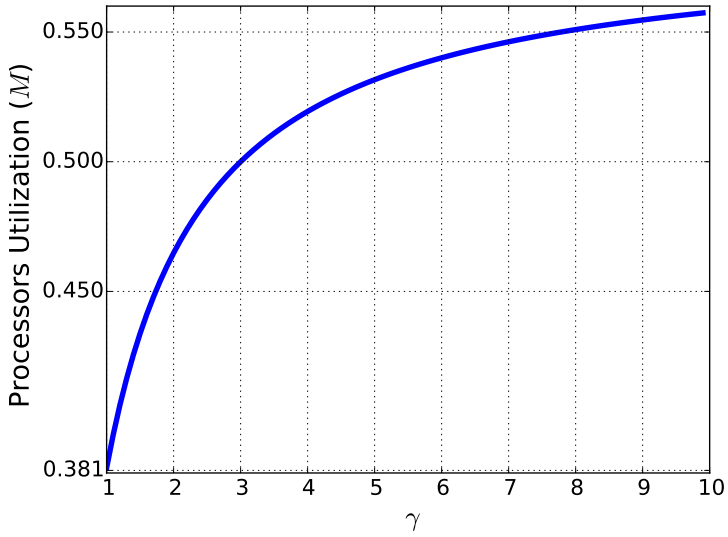


Figure 6.11: Utilization bound of RAD with respect to γ

maximum number of tasks with utilization no more than α that can be always guaranteed to fit into one processor. If we compute $\phi(\alpha)$ by simply combining the total utilization bound of Eq. (6.25) and the fact that a uniprocessor system with one task is schedulable, we can already obtain a bound on $\phi(\alpha) \geq max\left(\left\lfloor \frac{2-\sqrt{2}}{\alpha} \right\rfloor, 1\right)$. However, such a lower bound on $\phi(\alpha)$ is pessimistic, since the bound $2-\sqrt{2}$ by Eq. (6.25) comes from a large amount of tasks in [HC15b]. Hence, instead, we consider the quadratic bound, provided in Theorem 6.11, in hopes of getting a tighter lower bound on $\phi(\alpha)$.

Lemma 6.11. *The maximum number of tasks with utilization no more than α that can be always guaranteed to fit into one processor $\phi(\alpha)$ is at least:*

$$\phi(\alpha) \geq \beta = \left\lfloor \frac{4+\alpha-\sqrt{\alpha^2+8}}{2\alpha} \right\rfloor \tag{6.41}$$

Proof. It is not difficult to see the minimum $\phi(\alpha)$ happens when all the tasks are with utilization α. The value of β can be solved with the following inequality with respect to the upper bound on the utilization for ever task α:

$$\alpha \leq 1 - 2\sum_{i=1}^{(\beta-1)} \alpha + \frac{1}{2}(\sum_{i=1}^{(\beta-1)} \alpha)^2 + \frac{1}{2}\sum_{i=1}^{(\beta-1)} \alpha^2 \tag{6.42}$$

After reformulation, we obtain

$$\alpha^2(\beta-1)^2 + (\beta-1)(\alpha^2-4\alpha) + 2 - 2\alpha \geq 0 \tag{6.43}$$

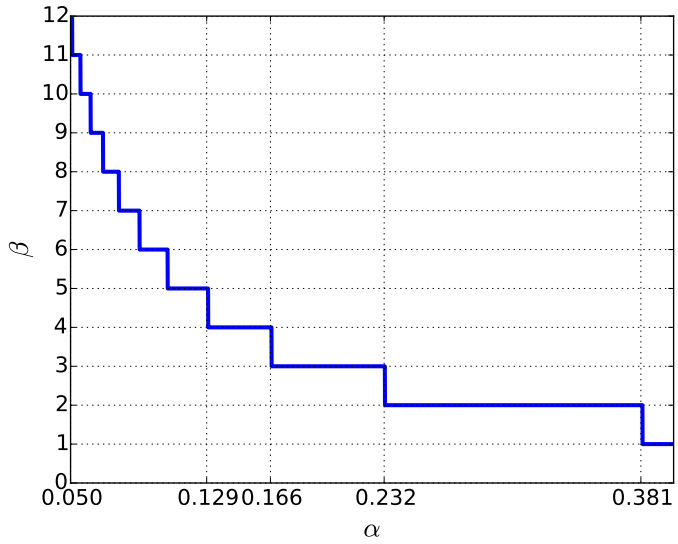


Figure 6.12: The value of β with respect to given α.

Solving it as the standard quadratic inequality w.r.t. β, we obtain

$$\beta \leq \frac{4+\alpha-\sqrt{\alpha^2+8}}{2\alpha} \text{ or } \beta \geq \frac{4+\alpha+\sqrt{\alpha^2+8}}{2\alpha} \tag{6.44}$$

Since we are only interested in providing the lower bound on $\phi(\alpha)$, it is sufficient to take the value with the "$-$" sign as our solution of β. Thus, it indicates a feasible schedule under RM scheduling on one processor if β tasks with utilization no more than α are allocated on the processor. Recall that by definition β must be an integer. Hence, we have to apply the floor function to round it down thereafter. We here conclude this lemma. □

Figure 6.12 depicts β with respect to the utilization upper bound α. This lower bound is tighter than the one by using only Eq.(6.25). For example, if $\alpha = 0.381$, we can conclude that $\beta = 2$ by Lemma 6.11, as can be seen in Figure 6.12; however, $max\left(\left\lfloor\frac{2-\sqrt{2}}{\alpha}\right\rfloor, 1\right) = 1$. We then make use of Lemma 6.11 to derive the following generalized utilization bound taking the value of β:

Theorem 6.14. *For a given α, suppose that β is from Lemma 6.11. A multiprocessor system τ with implicit deadline, multi-mode tasks is feasible under RM scheduling and a reasonable allocation decreasing (RAD) algorithm using QB if*

$$U^{sum} \leq \frac{1+2\beta-\sqrt{1+2\beta+2\beta^2}}{(1+\beta)} \cdot M \tag{6.45}$$

α	$(0.381, 1]$	$(0.231, 0.381]$	$(0.166, 0.231]$	$(0.129, 0.166]$
β	1	2	3	4
utilization	$0.381M$	$0.464M$	$0.500M$	$0.519M$

Table 6.5: Utilization bounds of RAD using QB under α

Proof. By Lemma 6.11, we have $\gamma = \frac{\Upsilon}{M} = \frac{\sum_{j=1}^{M} |\Gamma_j|}{M} \geq \frac{M\beta}{M} = \beta$. Hence, β is a lower bound on γ. Replacing γ in Lemma 6.10 by β, and similarly taking the negation of Lemma 6.10, we can thus conclude this theorem. □

This utilization bound is a generalized result of Theorem 6.13. For example, if $\alpha = 0.23$, then $\beta = 3$. By Theorem 6.14, we then obtain the utilization bound of 50%, shown in Table 6.5. Moreover, if $\alpha \to 0$, then $\beta \to \infty$ and the utilization bound converges to $2 - \sqrt{2}$:

$$\lim_{\beta \to \infty} \frac{2 + \frac{1}{\beta} - \sqrt{2 + \frac{2}{\beta} + \frac{1}{\beta^2}}}{(1 + \frac{1}{\beta})} M = (2 - \sqrt{2})M \approx 0.58M$$

6.2.4.1 *Bounds for Related Models*

In this section, we intend to answer the utilization bound on multiprocessor systems for those related models: the Generalized MultiFrame (GMF) model, the VRB model, and the DRT model. As we only focus on implicit-deadline multi-mode task systems, we also implicitly assume that these GMF, VRB, and DRT models are also with implicit deadlines. We explicitly show that the studied mode change model is a relaxation of these models under the above assumption.

Lemma 6.12. *The studied mode change model is a relaxation of the GMF model , the VRB model, and DRT model if the minimum inter-arrival time to change to the next mode is always equal to the relative deadline of the current task mode.*

Proof. By definition, a generalized multiframe task [BCGM99] must execute each frame sequentially and circularly, starting from any of its frames. The studied multi-mode task model can be equivalently obtained by relaxing the constraint of sequential executions for the generalized multiframe task. Likewise, we can reconstruct the structure of job executions from the directed graph, represented by the DRT model [SEGY11], into a complete graph with a self-loop for each vertex. Regarding the VRB model [DFPS14], the acceleration or deceleration, that determines which modes may be possibly invoked in the next release under the current rotation speed, can be assumed to be an unlimited value. By considering those execution modes invoked on the threshold of rotation speeds, the corresponding mode change model is thereafter obtained. □

We also conclude the utilization bounds for these models:

Corollary 6.1. *The GMF task, the VRB task, and the DRT tasks with implicit deadline on multiprocessor systems achieve the utilization bound provided in Theorems 6.12, 6.13, and 6.14.*

6.2.5 *Evaluations under different RADs*

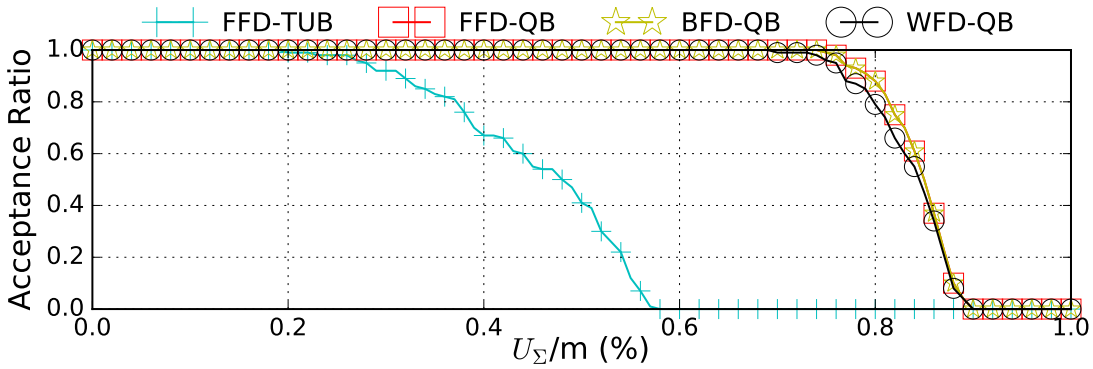
In this subsection, extensive simulations have been carried out in order to quantify the pessimism of the multiprocessor utilization bounds. The metric to compare the results is to measure the *acceptance ratio* of these tests for a given goal of task set utilization. We generate 100 task sets for each utilization level.

Task utilization values were generated from a uniform distribution but with the constraint that they summed to a constant desired total utilization $U_{\sum}$. We adopted the UUnifast-Discard to generate task sets. The cardinality of a task set was decided according to the ratio of the number of tasks to the number processors, in one of three values $[2, 5, 10]$, provided that the number of available processors is given. We evaluated three RAD algorithms mentioned earlier, namely First-Fit Decreasing, Best-Fit Decreasing, and Worst-Fit Decreasing, using the total utilization bound in Theorem 6.12, denoted by FFD-TUB, BFD-TUB, and WFD-TUB, receptively, and those using QB in Theorem 6.13, denoted by FFD-QB, BFD-QB, and WFD-QB, receptively.

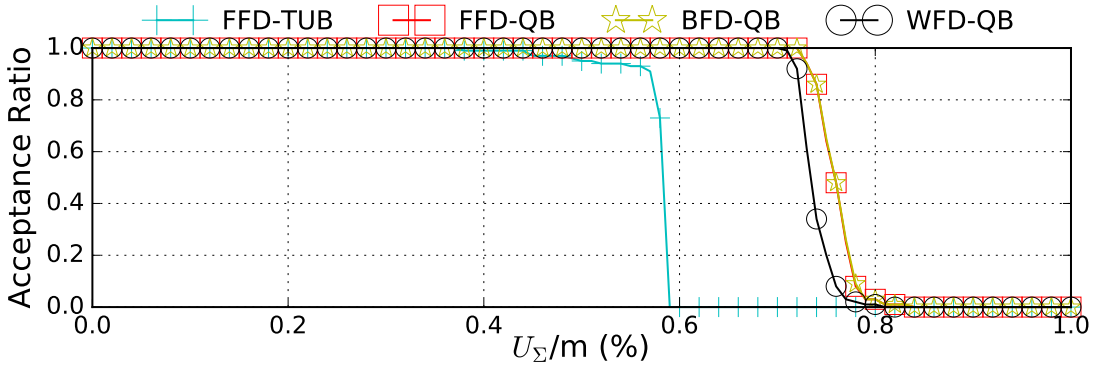
The *remaining capacity* δ_j on processor j for an RAD using TUB and QB is defined as follows: for TUB we use $\delta_j = 2 - \sqrt{2} - \sum_{\tau_i \in \Gamma_j} U_i - U_k$, and for QB we use $\delta_j = 1 - 2\sum_{\tau_i \in \Gamma_j} U_i + \frac{1}{2}(\sum_{\tau_i \in \Gamma_j} U_i)^2 + \frac{1}{2}\sum_{\tau_i \in \Gamma_j}(U_i)^2 - U_k$, where task τ_k is the task being allocated.

Results. Figure 6.13 and Figure 6.14 show the acceptance ratio under difference tasks on a platform with 4 processors and 16 processors, respectively, e.g., $M = 16$, $\frac{N}{M} = 5$ implies the number of tasks is $N = 16 \times 5 = 80$. Note that for better readability, we only show the result of FFD-TUB. The performance of FFD-TUB is identical to that of BFD-TUB and slightly better than that of WFD-TUB, similar to the cases where QB is used. The first and most obvious observation is that the RAD algorithm using QB is far more effective than that using TUB. We then notice that FFD-QB and BFD-QB perform alike and are superior to WFD, across all the settings. We also notice that the maximum effective utilization of RAD decreases noticeably from 90% down to 70% when the ratio $\frac{N}{M}$ increases from 2 to 10, with a fixed number of processors (e.g. Figure 6.13 (a), (b), and (c)). On the other hand, the variance of the number of processors would slightly affect the acceptance ratio, as the ratio of the number of tasks to the number of processors is fixed (e.g. Figure 6.13 (a) and Figure 6.14 (a)).

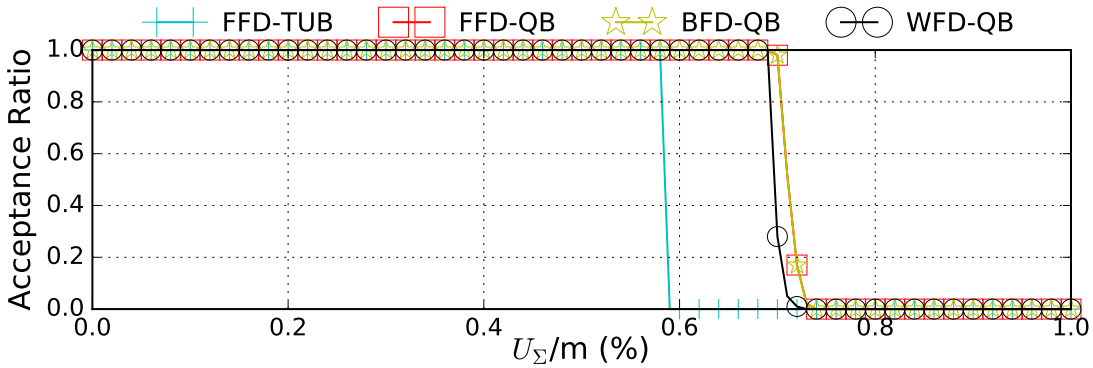
Overall, in our simulations, the simple RAD heuristics combined together with the quadratic bound can admit task sets even with noticeable high utilization, from 70% and up to 90%.



(a) $M = 4$ and $\frac{N}{M} = 2$



(b) $M = 4$ and $\frac{N}{M} = 5$

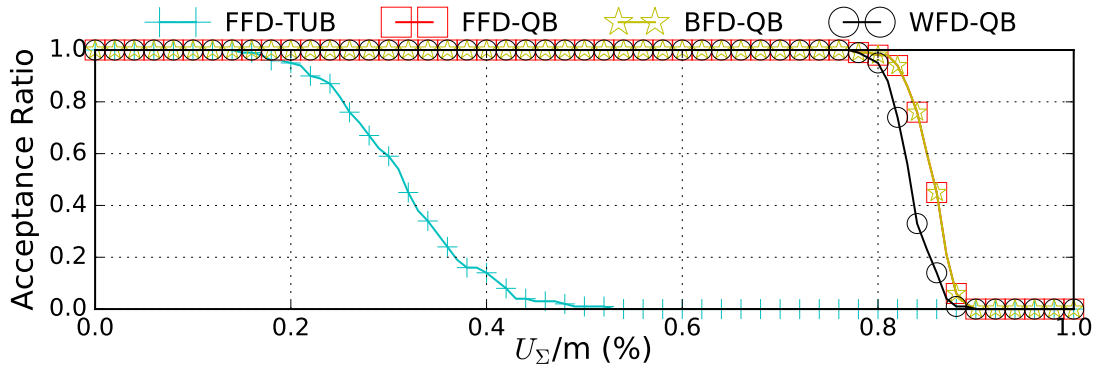


(c) $M = 4$ and $\frac{N}{M} = 10$

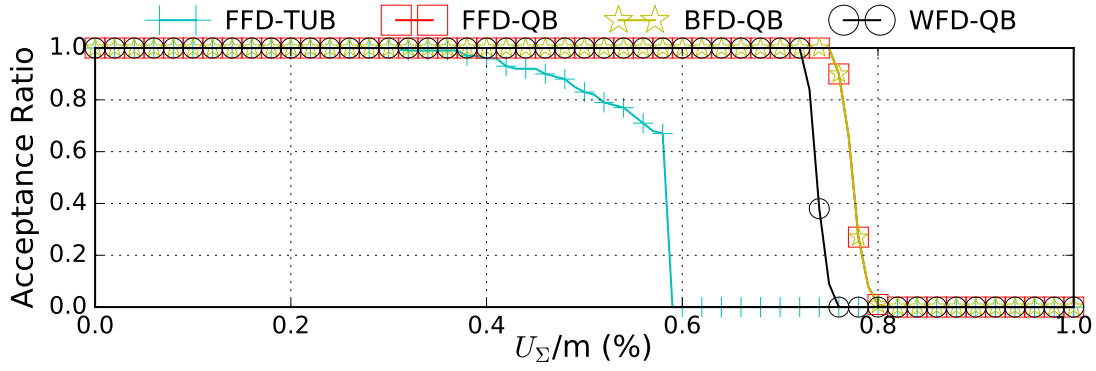
Figure 6.13: The acceptance ratio by FFD, BFD, and WFD using the quadratic bound under under different task-to-processor ratios on a 4-processor platform.

6.3 SUMMARY

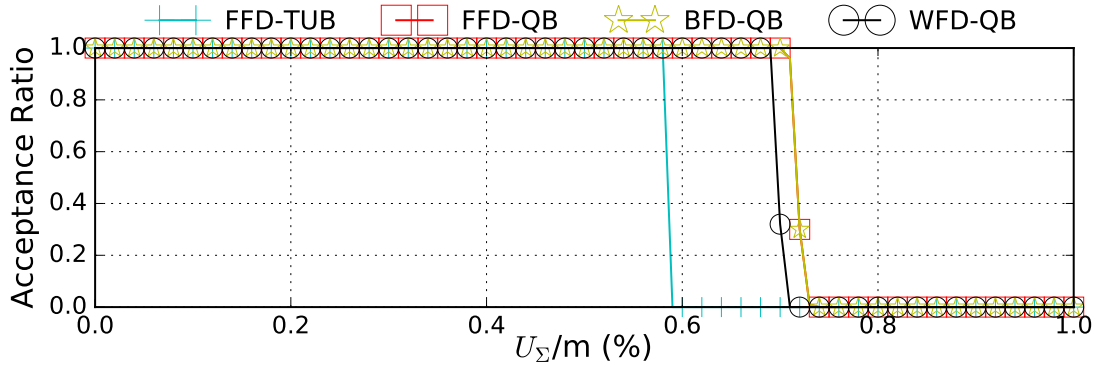
A multi-mode task model is a natural generalization model that represents higher expressiveness over the sporadic task model. This dissertation addresses the scheduling problem of mode change real-time tasks under fixed-priority scheduling.



(a) $M = 16$ and $\frac{N}{M} = 2$



(b) $M = 16$ and $\frac{N}{M} = 5$



(c) $M = 16$ and $\frac{N}{M} = 10$

Figure 6.14: The acceptance ratio by FFD, BFD, and WFD using the quadratic bound under under different task-to-processor ratios on a 16-processor platform.

For uniprocessors, we prove that a utilization bound of $2 - \sqrt{2} \approx 0.5857$ can be guaranteed in implicit-deadline multi-mode systems under RM scheduling. In addition,

we prove that a 38% utilization bound can be guaranteed for implicit-deadline multi-mode tasks on multiprocessor systems by using any RAD algorithm, e.g. FFD, if each mode is prioritized according to RM scheduling policy.

7 CONCLUSIONS AND FUTURE WORK

7.1 SUMMARY

Timing analysis must be carried out in association with scheduling algorithms. An exact schedulability test will be in vain if associated with poor scheduling algorithms; similarly, a scheduling algorithm is meaningless when the corresponding schedulability test is unavailable or intractable. As far as schedulability is concerned, these two dimensions should be considered equally at the same time, and one should seek as far as possible to find solutions that are either optimal or closely optimal. In this regard, we look at problems from the perspective of speedup factor — a metric that quantifies both the pessimism of the test and the non-optimality of the scheduling algorithm.

On the other hand, several assumptions on the conventional sporadic task model must be assured: (*i*) all tasks are assumed to be independent; (*ii*) tasks cannot suspend themselves; and (*iii*) tasks' behaviors are entirely periodic/sporadic. In this dissertation, we discuss the problems and solutions related to the above assumptions not valid.

In Chapter 3, we propose the response time analysis for sporadic arbitrary-deadline tasks on multiprocessor systems under fixed-priority scheduling. Further, we derive a linear-time response time bound that provides an amenable solution for the the admission control and the interactive system and prototyping. We further conclude that the linear-time response time bound can yield high performance compared to the state-of-the-art TDA tests, especially in case of large task periods, where time-demand analysis tests suffer from high computational complexity.

In Chapter 4, we observe that response-time analysis for multicore platforms with shared resources can be symmetrically approached from two perspectives: a *core-centric* and a *shared-resource-centric* perspective. The common "core-centric" perspective is that a task executes on a core until it suspends the execution due to shared resource accesses. The potentially less intuitive "shared-resource-centric" perspective is that a task performs requests on shared resources until suspending itself back to perform computation on its respective core. Based on the above observation, we provide a pseudo-polynomial-time schedulability test and response-time analysis for constrained-deadline sporadic task systems. In addition, we propose a task partitioning algorithm that achieves a speedup factor of 7, compared to the optimal schedule. This constitutes the first result in this research line with a speedup factor guarantee. The experimental evaluation demonstrates that our approach can yield high acceptance ratios if the tasks have only a few resource access segments.

In Chapter 5, we study the problem of real-time tasks with synchronizations on multiprocessor systems. We show that the proposed resource-oriented partitioned scheduling using PCP combined with a reasonable allocation algorithm can achieve a non-trivial speedup factor guarantee. Specifically, we prove that our task mapping and resource allocation algorithm has a speedup factor $11 - 6/(m+1)$ on a platform comprising m processors, where a task may request at most one shared resource and the number of requests on any resource by any single job is at most one. Earlier results by Andersson and Easwaran [AE10] achieved a speedup factor $12(1 + 3r/4m)$ by using G-EDF and virtualization under the same resource access model, where r is the number of shared resources. The effectiveness of our proposed algorithm is highly supported by the evaluation results. Empirical results show that our algorithm along with either PCP or NPP for task sets with utilization below 50% is nearly optimal in the sense that task sets not deemed schedulable by our algorithm are also not schedulable using any scheduling policy. Further, we observe that while the additional overhead is incurred by the implementation of PCP in each single processor, the improvement over the simple non-preemptive scheduling by using PCP is very little: the long blocking incurred by the non-preemptive scheduling can be painlessly removed by finding a good task partitioning. Therefore, this enables us to use the simple non-preemptive scheduling for resource sharing on multiprocessor systems while keeping high schedulability.

In Chapter 6, we study a model that is a generalization of the periodic task model, called multi-mode task model: a task has several modes specified with different execution times and periods to switch during runtime, independent of other tasks. The proposed test for uniprocessor systems can be efficiently run in polynomial time and is shown to be comparable to the existing state-of-the-art pseudo-polynomial tests for FPT scheduling. On one hand, we show that the FPT scheduling can perform rather poorly in the worst case. On the other hand, we prove that a utilization bound of $2 - \sqrt{2} \approx 0.5857$ can be guaranteed in implicit-deadline multi-mode systems under RM scheduling, one example of FPM scheduling. Moreover, we study the problem of allocating a set of multi-mode tasks on a homogeneous multiprocessor system. We present a scheduling algorithm using any RAD algorithm for task allocations for scheduling multi-mode tasks on multiprocessor systems. We prove that this algorithm achieves 38% utilization for implicit-deadline RM scheduled multi-mode tasks on multiprocessor systems. We then further improve this bound by considering the upper bound on the utilization for every task. We explicitly also conclude that the studied mode change model is a relaxation of the GMF model [BCGM99], the VRB model [DFPS14] (also known as the adaptive variable-rate (AVR) model [BBB14; BNB15]), and the DRT model [SEGY11]. As a consequence, we also derived utilization bounds for these models on multiprocessor systems.

7.2 FUTURE WORK

Release enforcement. In the presence of suspension, our analyses done in Chapter 4 and 5 must take into account the *carry-in* effect. Such an effect adversely affects schedulability, causing additional interferences to be counted in the timing analysis. Recently, there have been quite a few researches [CL14; HC16a; BHCL16; PF16] in favor of *release enforcement*—a mechanism by deterministically releasing internal jobs to potentially enhance predictability. It is worth seeing whether the release enforcement can offset the unfavorable carry-in effect, thereafter bringing down the speedup factor.

Multiple resource accesses and multiple requests on each resource with constant speedup factor. In Chapter 5, although we do not implicitly show the speedup factor of our proposed algorithm for multiple resource accesses and multiple requests on each resource, i.e., $N > 1$ and $Q > 1$, informally speaking, it would be however the value related to N and Q. This implies that our proposed algorithm does not scale very well in the presence of multiple resource accesses and multiple requests on one resource in the sense of speedup factor. Empirical results also indicate the downside of our proposed algorithm under multiple resource accesses and multiple requests on each resource. In future work, we aim at bridging this gap, by developing an algorithm with a constant speedup factor. Furthermore, it is also interesting in providing quality-guaranteed algorithms for resource sharing with nested critical sections.

Constrained-deadline tasks with synchronization. In Chapter 5, we restrict ourselves on implicit deadlines. However, extending implicit deadlines into constrained deadlines might not be as easy as it seems. In constrained-deadline partitioned systems, *density* and *load* are two key factors to algorithms with non-optimality [BF07]. Unlike for utilizations, as used under implicit-deadline systems, there is no *additivity* for *density* and *load*. This may lead to some property not being available under constrained-deadline systems while packing shared resources. It would be definitely interesting to see how to pack those resources such that some property is preserved, e.g., Lemma 5.5.

BIBLIOGRAPHY

[Abe+13] Andreas Abel et al. "Impact of Resource Sharing on Performance and Performance Prediction: A Survey." In: *CONCUR 2013 - Concurrency Theory - 24th International Conference, CONCUR 2013, Buenos Aires, Argentina, August 27-30, 2013. Proceedings*. 2013, pp. 25–43.

[AB09] Sebastian Altmeyer and Claire Burguière. "A New Notion of Useful Cache Block to Improve the Bounds of Cache-Related Preemption Delay." In: *21st Euromicro Conference on Real-Time Systems, ECRTS 2009, Dublin, Ireland, July 1-3, 2009*. 2009, pp. 109–118. DOI: 10.1109/ECRTS.2009.21. URL: http://dx.doi.org/10.1109/ECRTS.2009.21.

[AB11] Sebastian Altmeyer and Claire Maiza Burguière. "Cache-related preemption delay via useful cache blocks: Survey and redefinition." In: *Journal of Systems Architecture* 57.7 (2011), pp. 707–719.

[Alt+15] Sebastian Altmeyer et al. "A generic and compositional framework for multicore response time analysis." In: *Proceedings of the 23rd International Conference on Real Time Networks and Systems, RTNS 2015, Lille, France, November 4-6, 2015*. 2015, pp. 129–138. DOI: 10.1145/2834848.2834862. URL: http://doi.acm.org/10.1145/2834848.2834862.

[ABJ01] Björn Andersson, Sanjoy K. Baruah, and Jan Jonsson. "Static-Priority Scheduling on Multiprocessors." In: *Proceedings of the 22nd IEEE Real-Time Systems Symposium (RTSS 2001), London, UK, 2-6 December 2001*. 2001, pp. 193–202. DOI: 10.1109/REAL.2001.990610. URL: http://dx.doi.org/10.1109/REAL.2001.990610.

[AE10] Björn Andersson and Arvind Easwaran. "Provably good multiprocessor scheduling with resource sharing." In: *Real-Time Systems* 46.2 (2010), pp. 153–159. DOI: 10.1007/s11241-010-9105-6. URL: http://dx.doi.org/10.1007/s11241-010-9105-6.

[AJ00] Björn Andersson and Jan Jonsson. "Fixed-priority preemptive multiprocessor scheduling: to partition or not to partition." In: *7th International Workshop on Real-Time Computing and Applications Symposium (RTCSA 2000), 12-14 December 2000, Cheju Island, South Korea*. 2000, pp. 337–346. DOI: 10.1109/RTCSA.2000.896409. URL: http://dx.doi.org/10.1109/RTCSA.2000.896409.

[Aud91] Neil C Audsley. *Optimal priority assignment and feasibility of static priority tasks with arbitrary start times*. Citeseer, 1991.

[Aud01] Neil C. Audsley. "On priority assignment in fixed priority scheduling." In: *Inf. Process. Lett.* 79.1 (2001), pp. 39–44. DOI: 10.1016/S0020-0190(00)00165-4. URL: http://dx.doi.org/10.1016/S0020-0190(00)00165-4.

[ABRTW93] Neil C. Audsley et al. "Applying new scheduling theory to static priority pre-emptive scheduling." In: *Software Engineering Journal* 8.5 (1993), pp. 284–292. URL: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=238595.

[Aut] *AUTOSAR release 4.2*. 2014. URL: http://www.autosar.org.

[Bak03] Theodore P. Baker. "Multiprocessor EDF and Deadline Monotonic Schedulability Analysis." In: *Proceedings of the 24th IEEE Real-Time Systems Symposium (RTSS 2003), 3-5 December 2003, Cancun, Mexico*. 2003, pp. 120–129. DOI: 10.1109/REAL.2003.1253260. URL: http://dx.doi.org/10.1109/REAL.2003.1253260.

[Bak06] Theodore P. Baker. "An Analysis of Fixed-Priority Schedulability on a Multiprocessor." In: *Real-Time Systems* 32.1-2 (2006), pp. 49–71. DOI: 10.1007/S11241-005-4686-1. URL: http://dx.doi.org/10.1007/S11241-005-4686-1.

[BC07a] Theodore P. Baker and Michele Cirinei. "Brute-Force Determination of Multiprocessor Schedulability for Sets of Sporadic Hard-Deadline Tasks." In: *Principles of Distributed Systems, 11th International Conference, OPODIS 2007, Guadeloupe, French West Indies, December 17-20, 2007. Proceedings*. 2007, pp. 62–75.

[Bar05] Sanjoy K. Baruah. "The Limited-Preemption Uniprocessor Scheduling of Sporadic Task Systems." In: *17th Euromicro Conference on Real-Time Systems (ECRTS 2005), 6-8 July 2005, Palma de Mallorca, Spain, Proceedings*. 2005, pp. 137–144. DOI: 10.1109/ECRTS.2005.32. URL: http://dx.doi.org/10.1109/ECRTS.2005.32.

[Bar07] Sanjoy K. Baruah. "Techniques for Multiprocessor Global Schedulability Analysis." In: *Proceedings of the 28th IEEE Real-Time Systems Symposium (RTSS 2007), 3-6 December 2007, Tucson, Arizona, USA*. 2007, pp. 119–128. DOI: 10.1109/RTSS.2007.35. URL: http://dx.doi.org/10.1109/RTSS.2007.35.

[BB08] Sanjoy K. Baruah and Theodore P. Baker. "Schedulability analysis of global edf." In: *Real-Time Systems* 38.3 (2008), pp. 223–235. DOI: 10.1007/s11241-007-9047-9. URL: http://dx.doi.org/10.1007/s11241-007-9047-9.

[BBMS10] Sanjoy K. Baruah et al. "Improved multiprocessor global schedulability analysis." In: *Real-Time Systems* 46.1 (2010), pp. 3–24. DOI: 10.1007/s11241-010-9096-3. URL: http://dx.doi.org/10.1007/s11241-010-9096-3.

[BCGM99] Sanjoy K. Baruah et al. "Generalized Multiframe Tasks." In: *Real-Time Systems* 17.1 (1999), pp. 5–22. DOI: 10.1023/A:1008030427220. URL: http://dx.doi.org/10.1023/A:1008030427220.

[BF07] Sanjoy K. Baruah and Nathan Fisher. "Global Deadline-Monotonic Scheduling of Arbitrary-Deadline Sporadic Task Systems." In: *Principles of Distributed Systems, 11th International Conference, OPODIS 2007, Guadeloupe, French West Indies, December 17-20, 2007. Proceedings*. 2007, pp. 204–216.

[BF08] Sanjoy K. Baruah and Nathan Fisher. "Global Fixed-Priority Scheduling of Arbitrary-Deadline Sporadic Task Systems." In: *Distributed Computing and Networking, 9th International Conference, ICDCN 2008, Kolkata, India, January 5-8, 2008.* 2008, pp. 215–226.

[BMR90] Sanjoy K. Baruah, Aloysius K. Mok, and Louis E. Rosier. "Preemptively Scheduling Hard-Real-Time Sporadic Tasks on One Processor." In: *IEEE Real-Time Systems Symposium*. 1990, pp. 182–190.

[BBA11] Andrea Bastoni, Björn B. Brandenburg, and James H. Anderson. "Is Semi-Partitioned Scheduling Practical?" In: *Euromicro Conference on Real-Time Systems (ECRTS)*. 2011, pp. 125–135. DOI: 10.1109/ECRTS.2011.20. URL: http://dx.doi.org/10.1109/ECRTS.2011.20.

[BC07b] Marko Bertogna and Michele Cirinei. "Response-Time Analysis for Globally Scheduled Symmetric Multiprocessor Platforms." In: *Proceedings of the 28th IEEE Real-Time Systems Symposium (RTSS 2007), 3-6 December 2007, Tucson, Arizona, USA*. 2007, pp. 149–160. DOI: 10.1109/RTSS.2007.31. URL: http://dx.doi.org/10.1109/RTSS.2007.31.

[BCL05a] Marko Bertogna, Michele Cirinei, and Giuseppe Lipari. "Improved Schedulability Analysis of EDF on Multiprocessor Platforms." In: *17th Euromicro Conference on Real-Time Systems (ECRTS 2005), 6-8 July 2005, Palma de Mallorca, Spain, Proceedings*. 2005, pp. 209–218.

[BCL05b] Marko Bertogna, Michele Cirinei, and Giuseppe Lipari. "New Schedulability Tests for Real-Time Task Sets Scheduled by Deadline Monotonic on Multiprocessors." In: *Principles of Distributed Systems, 9th International Conference, OPODIS 2005, Pisa, Italy, December 12-14, 2005, Revised Selected Papers*. 2005, pp. 306–321.

[Bin15] Enrico Bini. "The Quadratic Utilization Upper Bound for Arbitrary Deadline Real-Time Tasks." In: *IEEE Trans. Computers* 64.2 (2015), pp. 593–599. DOI: 10.1109/TC.2013.209. URL: http://dx.doi.org/10.1109/TC.2013.209.

[BB04] Enrico Bini and Giorgio C. Buttazzo. "Schedulability Analysis of Periodic Fixed Priority Systems." In: *IEEE Trans. Computers* 53.11 (2004), pp. 1462–1473. DOI: 10.1109/TC.2004.103. URL: http://dx.doi.org/10.1109/TC.2004.103.

[BB05] Enrico Bini and Giorgio C. Buttazzo. "Measuring the Performance of Schedulability Tests." In: *Real-Time Systems* 30.1-2 (2005), pp. 129–154. DOI: 10.1007/s11241-005-0507-9. URL: http://dx.doi.org/10.1007/s11241-005-0507-9.

[BNRB09] Enrico Bini et al. "A Response-Time Bound in Fixed-Priority Scheduling with Arbitrary Deadlines." In: *IEEE Trans. Computers* 58.2 (2009), pp. 279–286. DOI: 10.1109/TC.2008.167. URL: http://dx.doi.org/10.1109/TC.2008.167.

[BMMNB14] Alessandro Biondi et al. "Exact Interference of Adaptive Variable-Rate Tasks under Fixed-Priority Scheduling." In: *26th Euromicro Conference on Real-Time Systems, ECRTS 2014, Madrid, Spain, July 8-11, 2014*. 2014, pp. 165–174. DOI: 10.1109/ECRTS.2014.38. URL: http://dx.doi.org/10.1109/ECRTS.2014.38.

[BNB15] Alessandro Biondi, Marco Di Natale, and Giorgio C. Buttazzo. "Response-time analysis for real-time tasks in engine control applications." In: *International Conference on Cyber-Physical Systems (ICCPS)*. 2015, pp. 120–129. DOI: 10.1145/2735960.2735963. URL: http://doi.acm.org/10.1145/2735960.2735963.

[BAHCN15] Konstantinos Bletsas et al. *Errata for three papers (2004-05) on fixed-priority scheduling with self-suspensions*. Tech. rep. CISTER-TR-150713. CISTER, 2015.

[BLBA07] Aaron Block et al. "A Flexible Real-Time Locking Protocol for Multiprocessors." In: *RTCSA*. 2007, pp. 47–56. DOI: 10.1109/RTCSA.2007.8. URL: http://dx.doi.org/10.1109/RTCSA.2007.8.

[BFPRT73] Manuel Blum et al. "Time Bounds for Selection." In: *J. Comput. Syst. Sci.* 7.4 (1973), pp. 448–461. DOI: 10.1016/S0022-0000(73)80033-9. URL: http://dx.doi.org/10.1016/S0022-0000(73)80033-9.

[Bra11] B.B. Brandenburg. "Scheduling and Locking in Multiprocessor Real-Time Operating Systems." PhD thesis. The University of North Carolina at Chapel Hill, 2011.

[Bra13] Björn B. Brandenburg. "Improved analysis and evaluation of real-time semaphore protocols for P-FP scheduling." In: *Real-Time and Embedded Technology and Applications Symposium, RTAS*. 2013, pp. 141–152. DOI: 10.1109/RTAS.2013.6531087. URL: http://dx.doi.org/10.1109/RTAS.2013.6531087.

[Bra14a] Björn B. Brandenburg. "The FMLP+: An Asymptotically Optimal Real-Time Locking Protocol for Suspension-Aware Analysis." In: *Euromicro Conference on Real-Time Systems (ECRTS)*. 2014, pp. 61–71. DOI: 10.1109/ECRTS.2014.26. URL: http://dx.doi.org/10.1109/ECRTS.2014.26.

[BA10] Björn B. Brandenburg and James H. Anderson. "Optimality Results for Multiprocessor Real-Time Locking." In: *Real-Time Systems Symposium (RTSS)*. 2010, pp. 49–60. DOI: 10.1109/RTSS.2010.17. URL: http://dx.doi.org/10.1109/RTSS.2010.17.

[Bra14b] Björn Bernhard Brandenburg. "Blocking Optimality in Distributed Real-Time Locking Protocols." In: *LITES* 1.2 (2014), 01:1–01:22. DOI: 10.4230/LITES-v001-i002-a001. URL: http://dx.doi.org/10.4230/LITES-v001-i002-a001.

[BCDH16] Georg von der Brüggen et al. "Exact Speedup Factors for Linear-Time Schedulability Tests for Fixed-Priority Preemptive and Non-preemptive Scheduling." In: *Information Processing Letters (IPL)* (2016). URL: doi:10.1016/j.ipl.2016.08.001.

[BCH15] Georg von der Bruggen, Jian-Jia Chen, and Wen-Hung Huang. "Schedulability and Optimization Analysis for Non-preemptive Static Priority Scheduling Based on Task Utilization and Blocking Factors." In: *27th Euromicro Conference on Real-Time Systems, ECRTS 2015, Lund, Sweden, July 8-10, 2015*. 2015, pp. 90–101. DOI: 10.1109/ECRTS.2015.16. URL: http://dx.doi.org/10.1109/ECRTS.2015.16.

[BCHC16] Georg von der Brüggen et al. "Systems with Dynamic Real-Time Guarantees in Uncertain and Faulty Execution Environments." In: *Real-Time Systems Symposium (RTSS)*. Porto, Portugal, 2016.

[BHCL16] Georg von der Brüggen et al. "Uniprocessor Scheduling Strategies for Self-Suspending Task Systems." In: *Proceedings of the 24th International Conference on Real-Time Networks and Systems, RTNS 2016, Brest, France, October 19-21, 2016*. 2016, pp. 119–128. DOI: 10.1145/2997465.2997497. URL: http://doi.acm.org/10.1145/2997465.2997497.

[Bur93] Alan Burns. *Preemptive priority based scheduling: An appropriate engineering approach*. University of York, Department of Computer Science, 1993.

[BW13] Alan Burns and Andy J. Wellings. "A Schedulability Compatible Multiprocessor Resource Sharing Protocol - MrsP." In: *Euromicro Conference on Real-Time Systems (ECRTS)*. 2013, pp. 282–291. DOI: 10.1109/ECRTS.2013.37. URL: http://dx.doi.org/10.1109/ECRTS.2013.37.

[BF93] Ricky W. Butler and George B. Finelli. "The Infeasibility of Quantifying the Reliability of Life-Critical Real-Time Software." In: *IEEE Trans. Software Eng.* 19.1 (1993), pp. 3–12. DOI: 10.1109/32.210303. URL: http://dx.doi.org/10.1109/32.210303.

[But11] Giorgio C. Buttazzo. *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications, Third Edition*. Vol. 24. Real-Time Systems Series. Springer, 2011. ISBN: 978-1-4614-0675-4. DOI: 10.1007/978-1-4614-0676-1. URL: http://dx.doi.org/10.1007/978-1-4614-0676-1.

[BBY13] Giorgio C. Buttazzo, Marko Bertogna, and Gang Yao. "Limited Preemptive Scheduling for Real-Time Systems. A Survey." In: *IEEE Trans. Industrial Informatics* 9.1 (2013), pp. 3–15. DOI: 10.1109/TII.2012.2188805. URL: http://dx.doi.org/10.1109/TII.2012.2188805.

[BBB14] Giorgio C. Buttazzo, Enrico Bini, and Darren Buttle. "Rate-adaptive tasks: Model, analysis, and design issues." In: *Design, Automation & Test in Europe Conference & Exhibition, DATE 2014, Dresden, Germany, March 24-28, 2014*. 2014, pp. 1–6. DOI: 10.7873/DATE.2014.266. URL: http://dx.doi.org/10.7873/DATE.2014.266.

[Car+04] John Carpenter et al. "A Categorization of Real-Time Multiprocessor Scheduling Problems and Algorithms." In: *Handbook of Scheduling - Algorithms, Models, and Performance Analysis*. 2004. URL: http://www.crcnetbase.com/doi/abs/10.1201/9780203489802.ch30.

[CC11] Jian-Jia Chen and Samarjit Chakraborty. "Resource Augmentation Bounds for Approximate Demand Bound Functions." In: *Proceedings of the 32nd IEEE Real-Time Systems Symposium, RTSS 2011, Vienna, Austria, November 29 - December 2, 2011*. 2011, pp. 272–281. DOI: 10.1109/RTSS.2011.32. URL: http://dx.doi.org/10.1109/RTSS.2011.32.

[CHL15] Jian-Jia Chen, Wen-Hung Huang, and Cong Liu. "k2U: A General Framework from k-Point Effective Schedulability Analysis to Utilization-Based Tests." In: *Real-Time Systems Symposium (RTSS)*. 2015.

[CHL16] Jian-Jia Chen, Wen-Hung Huang, and Cong Liu. "k2Q: A Quadratic-Form Response Time and Schedulability Analysis Framework for Utilization-Based Analysis." In: *Real-Time Systems Symposium (RTSS)*. Porto, Portugal, 2016.

[CL14] Jian-Jia Chen and Cong Liu. "Fixed-Relative-Deadline Scheduling of Hard Real-Time Tasks with Self-Suspensions." In: *Proceedings of the IEEE 35th IEEE Real-Time Systems Symposium, RTSS 2014, Rome, Italy, December 2-5, 2014*. 2014, pp. 149–160. DOI: 10.1109/RTSS.2014.31. URL: http://dx.doi.org/10.1109/RTSS.2014.31.

[CNH16] Jian-Jia Chen, Geoffrey Nelissen, and Wen-Hung Kevin Huang. "A Unifying Response Time Analysis Framework for Dynamic Self-Suspending Tasks." In: *Euromicro Conference on Real-Time Systems (ECRTS)*. Toulouse, France, 2016.

[Che+16] Jian-Jia Chen et al. *Many Suspensions, Many Problems: A Review of Self-Suspending Tasks in Real-Time Systems*. Tech. rep. 854. (Status: Preprint). Department of Computer Science, TU Dortmund, 2016. URL: http://ls12-www.cs.tu-dortmund.de/daes/media/documents/publications/downloads/2016-chen-techreport-854.pdf.

[CB02] Antoine Colin and Guillem Bernat. "Scope-Tree: A Program Representation for Symbolic Worst-Case Execution Time Analysis." In: *14th Euromicro Conference on Real-Time Systems (ECRTS 2002), 19-21 June 2002, Vienna, Austria, Proceedings*. 2002, p. 50. DOI: 10.1109/EMRTS.2002.1019185. URL: http://dx.doi.org/10.1109/EMRTS.2002.1019185.

[CP00] Antoine Colin and Isabelle Puaut. "Worst Case Execution Time Analysis for a Processor with Branch Prediction." In: *Real-Time Systems* 18.2/3 (2000), pp. 249–274. DOI: 10.1023/A:1008149332687. URL: http://dx.doi.org/10.1023/A:1008149332687.

[Con+05] FlexRay Consortium et al. "FlexRay communications system-protocol specification." In: *Version* 2.1 (2005), pp. 198–207.

[DB08] Robert I. Davis and Alan Burns. "Response Time Upper Bounds for Fixed Priority Real-Time Systems." In: *Proceedings of the 29th IEEE Real-Time Systems Symposium, RTSS 2008, Barcelona, Spain, 30 November - 3 December 2008*. 2008, pp. 407–418. DOI: 10.1109/RTSS.2008.18. URL: http://dx.doi.org/10.1109/RTSS.2008.18.

[DB11a] Robert I. Davis and Alan Burns. "A survey of hard real-time scheduling for multiprocessor systems." In: *ACM Comput. Surv.* 43.4 (2011), p. 35. DOI: 10.1145/1978802.1978814. URL: http://doi.acm.org/10.1145/1978802.1978814.

[DB11b] Robert I. Davis and Alan Burns. "Improved priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems." In: *Real-Time Systems* 47.1 (2011), pp. 1–40. DOI: 10.1007/s11241-010-9106-5. URL: http://dx.doi.org/10.1007/s11241-010-9106-5.

[DFPS14] Robert I. Davis et al. "Schedulability tests for tasks with Variable Rate-dependent Behaviour under fixed priority scheduling." In: *20th IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS 2014, Berlin, Germany, April 15-17, 2014*. 2014, pp. 51–62. DOI: 10.1109/RTAS.2014.6925990. URL: http://dx.doi.org/10.1109/RTAS.2014.6925990.

[DTGDP15] Robert I. Davis et al. "Quantifying the Exact Sub-optimality of Non-preemptive Scheduling." In: *2015 IEEE Real-Time Systems Symposium, RTSS 2015, San Antonio, Texas, USA, December 1-4, 2015*. 2015, pp. 96–106. DOI: 10.1109/RTSS.2015.17. URL: http://dx.doi.org/10.1109/RTSS.2015.17.

[Dur98] Tom Durkin. "What the Media Couldn't Tell You About Mars Pathfinder." In: *Robot Science &* (1998).

[EA09] Arvind Easwaran and Björn Andersson. "Resource Sharing in Global Fixed-Priority Preemptive Multiprocessor Scheduling." In: *Real-Time Systems Symposium (RTSS)*. 2009, pp. 377–386. DOI: 10.1109/RTSS.2009.37. URL: http://dx.doi.org/10.1109/RTSS.2009.37.

[ESD10] Paul Emberson, Roger Stafford, and Robert I Davis. "Techniques for the synthesis of multiprocessor tasksets." In: *International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS 2010)*. 2010, pp. 6–11.

[ETA] ETAS Group. *ASCET Software Products*. URL: http://www.etas.com/en/products.

[GLN01] Paolo Gai, Giuseppe Lipari, and Marco Di Natale. "Minimizing Memory Utilization of Real-Time Task Sets in Single and Multi-Processor Systems-on-a-Chip." In: *Real-Time Systems Symposium (RTSS)*. 2001, pp. 73–83. DOI: 10.1109/REAL.2001.990598. URL: http://dx.doi.org/10.1109/REAL.2001.990598.

[GJ79] M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979. ISBN: 0-7167-1044-7.

[GGL13] Gilles Geeraerts, Joël Goossens, and Markus Lindström. "Multiprocessor schedulability of arbitrary-deadline sporadic tasks: complexity and antichain algorithm." In: *Real-Time Systems* 49.2 (2013), pp. 171–218. DOI: 10.1007/s11241-012-9172-y. URL: http://dx.doi.org/10.1007/s11241-012-9172-y.

[GSYY09] Nan Guan et al. "New Response Time Bounds for Fixed Priority Multiprocessor Scheduling." In: *Proceedings of the 30th IEEE Real-Time Systems Symposium, RTSS 2009, Washington, DC, USA, 1-4 December 2009*. 2009, pp. 387–397. DOI: 10.1109/RTSS.2009.11. URL: http://dx.doi.org/10.1109/RTSS.2009.11.

[HRW15] Sebastian Hahn, Jan Reineke, and Reinhard Wilhelm. "Towards compositionality in execution time analysis: definition and challenges." In: *SIGBED Review* 12.1 (2015), pp. 28–36. DOI: 10.1145/2752801.2752805. URL: http://doi.acm.org/10.1145/2752801.2752805.

[HZNLD14] Gang Han et al. "Experimental Evaluation and Selection of Data Consistency Mechanisms for Hard Real-Time Applications on Multicore Platforms." In: *IEEE Trans. Industrial Informatics* 10.2 (2014), pp. 903–918. DOI: 10.1109/TII.2013.2290585. URL: http://dx.doi.org/10.1109/TII.2013.2290585.

[HAMWH99] Christopher A. Healy et al. "Bounding Pipeline and Instruction Cache Performance." In: *IEEE Trans. Computers* 48.1 (1999), pp. 53–70. DOI: 10.1109/12.743411. URL: http://dx.doi.org/10.1109/12.743411.

[HLK11] Pi-Cheng Hsiu, Der-Nien Lee, and Tei-Wei Kuo. "Task synchronization and allocation for many-core real-time systems." In: *International Conference on Embedded Software, (EMSOFT)*. 2011, pp. 79–88. DOI: 10.1145/2038642.2038656. URL: http://doi.acm.org/10.1145/2038642.2038656.

[HC15a] Wen-Hung Huang and Jian-Jia Chen. "Response Time Bounds for Sporadic Arbitrary-Deadline Tasks under Global Fixed-Priority Scheduling on Multiprocessors." In: *International Conference on Real-Time Networks and Systems (RTNS)*. Lille, France, 2015.

[HC15b] Wen-Hung Huang and Jian-Jia Chen. "Techniques for Schedulability Analysis in Mode Change Systems under Fixed-Priority Scheduling." In: *IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*. (Best Paper Award). Hong Kong, 2015.

[HC16a] Wen-Hung Huang and Jian-Jia Chen. "Self-Suspension Real-Time Tasks under Fixed-Relative-Deadline Fixed-Priority Scheduling." In: *Design, Automation and Test in Europe (DATE)*. Dresden, Germany, 2016.

[HC16b] Wen-Hung Huang and Jian-Jia Chen. "Utilization Bounds on Allocating Rate-Monotonic Scheduled Multi-Mode Tasks on Multiprocessor Systems." In: *Design Automation Conference (DAC)*. Austin, TX, USA, 2016.

[HCR16] Wen-Hung Huang, Jian-Jia Chen, and Jan Reineke. "MIRROR: Symmetric Timing Analysis for Real-Time Tasks on Multicore Platforms with Shared Resources." In: *Design Automation Conference (DAC)*. Austin, TX, USA, 2016.

[HCZL15] Wen-Hung Huang et al. "PASS: Priority Assignment of Real-Time Tasks with Dynamic Suspending Behavior under Fixed-Priority Scheduling." In: *Design Automation Conference (DAC), San Francisco, CA, USA*. 2015.

[HYC16] Wen-Hung Huang, Maolin Yang, and Jian-Jia Chen. "Resource-Oriented Partitioned Scheduling in Multiprocessor Systems: How to Partition and How to Share?" In: *Real-Time Systems Symposium (RTSS)*. (Outstanding Paper Award). Porto, Portugal, 2016.

[Joh73] David S Johnson. "Near-optimal bin packing algorithms." PhD thesis. Massachusetts Institute of Technology, 1973.

[KP95] Bala Kalyanasundaram and Kirk Pruhs. "Speed is as Powerful as Clairvoyance." In: *36th Annual Symposium on Foundations of Computer Science, Milwaukee, Wisconsin, 23-25 October 1995*. 1995, pp. 214–221.

[KFMCR14] Timon Kelter et al. "Static analysis of multi-core TDMA resource arbitration delays." In: *Real-Time Systems* 50.2 (2014), pp. 185–229. DOI: 10.1007/s11241-013-9189-x. URL: http://dx.doi.org/10.1007/s11241-013-9189-x.

[KLR12] Junsung Kim, Karthik Lakshmanan, and Ragunathan Rajkumar. "Rhythmic Tasks: A New Task Model with Continually Varying Periods for Cyber-Physical Systems." In: *2012 IEEE/ACM Third International Conference on Cyber-Physical Systems, ICCPS 2012, Beijing, China, April 17-19, 2012*. 2012, pp. 55–64. DOI: 10.1109/ICCPS.2012.14. URL: http://dx.doi.org/10.1109/ICCPS.2012.14.

[Koo14] Phil Koopman. "A case study of Toyota unintended acceleration and software safety." In: *Presentation. Sept* (2014).

[KG93] Hermann Kopetz and Günter Grünsteidl. "TTP - A Time-Triggered Protocol for Fault-Tolerant Real-Time Systems." In: *Digest of Papers: FTCS-23, The Twenty-Third Annual International Symposium on Fault-Tolerant Computing, Toulouse, France, June 22-24, 1993*. 1993, pp. 524–533. DOI: 10.1109/FTCS.1993.627355. URL: http://dx.doi.org/10.1109/FTCS.1993.627355.

[LNR09] Karthik Lakshmanan, Dionisio de Niz, and Ragunathan Rajkumar. "Coordinated Task Scheduling, Allocation and Synchronization on Multiprocessors." In: *Real-Time Systems Symposium, (RTSS)*. 2009, pp. 469–478. DOI: 10.1109/RTSS.2009.51. URL: http://dx.doi.org/10.1109/RTSS.2009.51.

[LGPWS14] Kai Lampka et al. "A formal approach to the WCRT analysis of multicore systems with memory contention under phase-structured task sets." In: *Real-Time Systems* 50.5-6 (2014), pp. 736–773. DOI: 10.1007/s11241-014-9211-y. URL: http://dx.doi.org/10.1007/s11241-014-9211-y.

[Leh90] John P. Lehoczky. "Fixed Priority Scheduling of Periodic Task Sets with Arbitrary Deadlines." In: *Proceedings of the Real-Time Systems Symposium - 1990, Lake Buena Vista, Florida, USA, December 1990*. 1990, pp. 201–209. DOI: 10.1109/REAL.1990.128748. URL: http://dx.doi.org/10.1109/REAL.1990.128748.

[LSD89] John P. Lehoczky, Lui Sha, and Y. Ding. "The Rate Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behavior." In: *Proceedings of the Real-Time Systems Symposium - 1989, Santa Monica, California, USA, December 1989*. 1989, pp. 166–171. DOI: 10.1109/REAL.1989.63567. URL: http://dx.doi.org/10.1109/REAL.1989.63567.

[LDS07] Chuanpeng Li, Chen Ding, and Kai Shen. "Quantifying the cost of context switch." In: *Proceedings of the Workshop on Experimental Computer Science, Part of ACM FCRC, San Diego, 13-14 June 2007*. 2007, p. 2. DOI: 10.1145/1281700.1281702. URL: http://doi.acm.org/10.1145/1281700.1281702.

[LM95] Yau-Tsun Steven Li and Sharad Malik. "Performance Analysis of Embedded Software Using Implicit Path Enumeration." In: *DAC*. 1995, pp. 456–461. DOI: 10.1145/217474.217570. URL: http://doi.acm.org/10.1145/217474.217570.

[LM97] Yau-Tsun Steven Li and Sharad Malik. "Performance analysis of embedded software using implicit path enumeration." In: *IEEE Trans. on CAD of Integrated Circuits and Systems* 16.12 (1997), pp. 1477–1487. DOI: 10.1109/43.664229. URL: http://dx.doi.org/10.1109/43.664229.

[LL73] C. L. Liu and James W. Layland. "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment." In: *J. ACM* 20.1 (1973), pp. 46–61. DOI: 10.1145/321738.321743. URL: http://doi.acm.org/10.1145/321738.321743.

[LC14] Cong Liu and Jian-Jia Chen. "Bursty-Interference Analysis Techniques for Analyzing Complex Real-Time Task Models." In: *Proceedings of the IEEE 35th IEEE Real-Time Systems Symposium, RTSS 2014, Rome, Italy, December 2-5, 2014*. 2014, pp. 173–183. DOI: 10.1109/RTSS.2014.10. URL: http://dx.doi.org/10.1109/RTSS.2014.10.

[LM11] Paul Lokuciejewski and Peter Marwedel. *Worst-Case Execution Time Aware Compilation Techniques for Real-Time Systems*. Springer, 2011. ISBN: 978-90-481-9928-0. DOI: 10.1007/978-90-481-9929-7. URL: http://dx.doi.org/10.1007/978-90-481-9929-7.

[LYGY10] Mingsong Lv et al. "Combining Abstract Interpretation with Model Checking for Timing Analysis of Multicore Software." In: *RTSS*. 2010, pp. 339–349.

[MBBMB15] Alessandra Melani et al. "Memory-processor co-scheduling in fixed priority systems." In: *Proceedings of the 23rd International Conference on Real Time Networks and Systems, RTNS 2015, Lille, France, November 4-6, 2015*. 2015, pp. 87–96. DOI: 10.1145/2834848.2834854. URL: http://doi.acm.org/10.1145/2834848.2834854.

[Mok83] Aloysius K Mok. "Fundamental design problems of distributed systems for the hard-real-time environment." In: (1983).

[NNB10] Farhang Nemati, Thomas Nolte, and Moris Behnam. "Partitioning Real-Time Systems on Multiprocessors with Shared Resources." In: *Principles of Distributed Systems - International Conference, OPODIS*. 2010, pp. 253–269.

[Nan] *NHTSA-NASA Study of Unintended Acceleration in Toyota Vehicles*. Tech. rep. National Aeronautics and Space Administration (NASA), Jan. 2011.

[PSCCT10] Rodolfo Pellizzoni et al. "Worst case delay analysis for memory interference in multicore systems." In: *Design, Automation and Test in Europe, DATE 2010, Dresden, Germany, March 8-12, 2010*. 2010, pp. 741–746. DOI: 10.1109/DATE.2010.5456952. URL: http://dx.doi.org/10.1109/DATE.2010.5456952.

[PF16] Bo Peng and Nathan Fisher. "Parameter adaption for generalized multiframe tasks and applications to self-suspending tasks." In: *Embedded and Real-Time Computing Systems and Applications (RTCSA), 2016 IEEE 22nd International Conference on*. IEEE. 2016, pp. 49–58.

[PK89] Peter P. Puschner and Christian Koza. "Calculating the Maximum Execution Time of Real-Time Programs." In: *Real-Time Systems* 1.2 (1989), pp. 159–176. DOI: 10.1007/BF00571421. URL: http://dx.doi.org/10.1007/BF00571421.

[Raj90] R. Rajkumar. "Real-Time Synchronization Protocols for Shared Memory Multiprocessors." In: *10th International Conference on Distributed Computing Systems (ICDCS)*. 1990, pp. 116–123. DOI: 10.1109/ICDCS.1990.89257. URL: http://dx.doi.org/10.1109/ICDCS.1990.89257.

[RSL88] Ragunathan Rajkumar, Lui Sha, and John P. Lehoczky. "Real-Time Synchronization Protocols for Multiprocessors." In: *Real-Time Systems Symposium (RTSS)*. 1988, pp. 259–269. DOI: 10.1109/REAL.1988.51121. URL: http://dx.doi.org/10.1109/REAL.1988.51121.

[RC04] Jorge Real and Alfons Crespo. "Mode Change Protocols for Real-Time Systems: A Survey and a New Proposal." In: *Real-Time Systems* 26.2 (2004), pp. 161–197. DOI: 10.1023/B:TIME.0000016129.97430.c6. URL: http://dx.doi.org/10.1023/B:TIME.0000016129.97430.c6.

[Rei09] Jan Reineke. "Caches in WCET Analysis: Predictability - Competitiveness - Sensitivity." PhD thesis. Saarland University, 2009. ISBN: 978-3-941071-69-8. URL: http://www.epubli.de/shop/buch/Caches-in-WCET-Analysis-Jan-Reineke-9783941071698/12835.

[RLPKL11] Jan Reineke et al. "PRET DRAM controller: bank privatization for predictability and temporal isolation." In: *Proceedings of the 9th International Conference on Hardware/Software Codesign and System Synthesis, CODES+ISSS 2011, part of ESWeek '11 Seventh Embedded Systems Week, Taipei, Taiwan, 9-14 October, 2011*. 2011, pp. 99–108. DOI: 10.1145/2039370.2039388. URL: http://doi.acm.org/10.1145/2039370.2039388.

[SBB11] Luca Santinelli, Giorgio C. Buttazzo, and Enrico Bini. "Multi-moded Resource Reservations." In: *17th IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS 2011, Chicago, Illinois, USA, 11-14 April 2011*. 2011, pp. 37–46. DOI: 10.1109/RTAS.2011.12. URL: http://dx.doi.org/10.1109/RTAS.2011.12.

[SCT10a] Andreas Schranzhofer, Jian-Jia Chen, and Lothar Thiele. "Dynamic Power-Aware Mapping of Applications onto Heterogeneous MPSoC Platforms." In: *IEEE Trans. Industrial Informatics* 6.4 (2010), pp. 692–707.

[SCT10b] Andreas Schranzhofer, Jian-Jia Chen, and Lothar Thiele. "Timing Analysis for TDMA Arbitration in Resource Sharing Systems." In: *16th IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS 2010, Stockholm, Sweden, April 12-15, 2010*. 2010, pp. 215–224. DOI: 10.1109/RTAS.2010.24. URL: http://dx.doi.org/10.1109/RTAS.2010.24.

[SPCTC10] Andreas Schranzhofer et al. "Worst-case response time analysis of resource access models in multi-core systems." In: *Proceedings of the 47th Design Automation Conference, DAC 2010, Anaheim, California, USA, July 13-18, 2010*. 2010, pp. 332–337. DOI: 10.1145/1837274.1837359. URL: http://doi.acm.org/10.1145/1837274.1837359.

[SRL90] Lui Sha, Ragunathan Rajkumar, and John P. Lehoczky. "Priority Inheritance Protocols: An Approach to Real-Time Synchronization." In: *IEEE Trans. Computers* 39.9 (1990), pp. 1175–1185. DOI: 10.1109/12.57058. URL: http://dx.doi.org/10.1109/12.57058.

[SRLR89] Lui Sha et al. "Mode Change Protocols for Priority-Driven Preemptive Scheduling." In: *Real-Time Systems* 1.3 (1989), pp. 243–264. DOI: 10.1007/BF00365439. URL: http://dx.doi.org/10.1007/BF00365439.

[Sho10] Michael Short. "The case for non-preemptive, deadline-driven scheduling in real-time embedded systems." In: *Lecture notes in engineering and computer science: Proceedings of the World Congress on Engineering*. 2010.

[SH98] Mikael Sjödin and Hans Hansson. "Improved Response-Time Analysis Calculations." In: *Proceedings of the 19th IEEE Real-Time Systems Symposium, Madrid, Spain, December 2-4, 1998*. 1998, pp. 399–408. DOI: 10.1109/REAL.1998.739773. URL: http://dx.doi.org/10.1109/REAL.1998.739773.

[SR90] John A. Stankovic and Krithi Ramamritham. "Editorial: What is Predictability for Real-Time Systems?" In: *Real-Time Systems* 2.4 (1990), pp. 247–254. DOI: 10.1007/BF01995673. URL: http://dx.doi.org/10.1007/BF01995673.

[SEGY11] Martin Stigge et al. "The Digraph Real-Time Task Model." In: *17th IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS 2011, Chicago, Illinois, USA, 11-14 April 2011*. 2011, pp. 71–80. DOI: 10.1109/RTAS.2011.15. URL: http://dx.doi.org/10.1109/RTAS.2011.15.

[SY13] Martin Stigge and Wang Yi. "Combinatorial Abstraction Refinement for Feasibility Analysis." In: *Proceedings of the IEEE 34th Real-Time Systems Symposium, RTSS 2013, Vancouver, BC, Canada, December 3-6, 2013*. 2013, pp. 340–349. DOI: 10.1109/RTSS.2013.41. URL: http://dx.doi.org/10.1109/RTSS.2013.41.

[SL14] Youcheng Sun and Giuseppe Lipari. "A Weak Simulation Relation for Real-Time Schedulability Analysis of Global Fixed Priority Scheduling Using Linear Hybrid Automata." In: *22nd International Conference on Real-Time Networks and Systems, RTNS '14, Versaille, France, October 8-10, 2014*. 2014, p. 35. DOI: 10.1145/2659787.2659814. URL: http://doi.acm.org/10.1145/2659787.2659814.

[SLGY14] Youcheng Sun et al. "Improving the response time analysis of global fixed-priority multiprocessor scheduling." In: *2014 IEEE 20th International Conference on Embedded and Real-Time Computing Systems and Applications, Chongqing, China, August 20-22, 2014*. 2014, pp. 1–9. DOI:

10.1109/RTCSA.2014.6910543. URL: http://dx.doi.org/10.1109/RTCSA.2014.6910543.

[TBW94] Ken Tindell, Alan Burns, and Andy J. Wellings. "An Extendible Approach for Analyzing Fixed Priority Hard Real-Time Tasks." In: *Real-Time Systems* 6.2 (1994), pp. 133–151. DOI: 10.1007/BF01088593. URL: http://dx.doi.org/10.1007/BF01088593.

[TBW92] Ken W Tindell, Alan Burns, and Andy J Wellings. "Mode changes in priority preemptively scheduled systems." In: *Real-Time Systems Symposium*. 1992, pp. 100–109.

[Van93] Pamela H. Vance. "Knapsack Problems: Algorithms and Computer Implementations (S. Martello and P. Toth)." In: *SIAM Review* 35.4 (1993), pp. 684–685. DOI: 10.1137/1035174. URL: http://dx.doi.org/10.1137/1035174.

[Vaz01] Vijay V Vazirani. *Approximation algorithms*. Springer, 2001.

[WS99] Yun Wang and Manas Saksena. "Scheduling Fixed-Priority Tasks with Preemption Threshold." In: *6th International Workshop on Real-Time Computing and Applications Symposium (RTCSA '99), 13-16 December 1999, Hong Kong, China*. 1999, p. 328. DOI: 10.1109/RTCSA.1999.811269. URL: http://dx.doi.org/10.1109/RTCSA.1999.811269.

[WA12] Bryan C. Ward and James H. Anderson. "Supporting Nested Locking in Multiprocessor Real-Time Systems." In: *Euromicro Conference on Real-Time Systems ECRTS*. 2012, pp. 223–232. DOI: 10.1109/ECRTS.2012.17. URL: http://dx.doi.org/10.1109/ECRTS.2012.17.

[WA13] Bryan C. Ward and James H. Anderson. "Fine-grained multiprocessor real-time locking with improved blocking." In: *International Conference on Real-Time Networks and Systems, RTNS*. 2013, pp. 67–76. DOI: 10.1145/2516821.2516843. URL: http://doi.acm.org/10.1145/2516821.2516843.

[WB13a] Alexander Wieder and Björn B. Brandenburg. "Efficient partitioning of sporadic real-time tasks with shared resources and spin locks." In: *International Symposium on Industrial Embedded Systems, (SIES)*. 2013, pp. 49–58. DOI: 10.1109/SIES.2013.6601470. URL: http://dx.doi.org/10.1109/SIES.2013.6601470.

[WB13b] Alexander Wieder and Björn B. Brandenburg. "On Spin Locks in AUTOSAR: Blocking Analysis of FIFO, Unordered, and Priority-Ordered Spin Locks." In: *Real-Time Systems Symposium*. 2013, pp. 45–56. DOI: 10.1109/RTSS.2013.13. URL: http://dx.doi.org/10.1109/RTSS.2013.13.

[Wil+08] Reinhard Wilhelm et al. "The worst-case execution-time problem - overview of methods and survey of tools." In: *ACM Trans. Embedded Comput. Syst.* 7.3 (2008), 36:1–36:53. DOI: 10.1145/1347375.1347389. URL: http://doi.acm.org/10.1145/1347375.1347389.

[Wil+09] Reinhard Wilhelm et al. "Memory Hierarchies, Pipelines, and Buses for Future Architectures in Time-Critical Embedded Systems." In: *IEEE Trans. on CAD of Integrated Circuits and Systems* 28.7 (2009), pp. 966–978. DOI: 10.1109/TCAD.2009.2013287. URL: http://dx.doi.org/10.1109/TCAD.2009.2013287.

[YWB15] Maolin Yang, Alexander Wieder, and Björn B. Brandenburg. "Global Real-Time Semaphore Protocols: A Survey, Unified Analysis, and Comparison." In: *Real-Time Systems Symposium (RTSS)*. 2015, pp. 1–12. DOI: 10.1109/RTSS.2015.8. URL: http://dx.doi.org/10.1109/RTSS.2015.8.

ACRONYMS

ETCS	Electronic Throttle Control System
NASA	National Aeronautics and Space Administration
FPM	Fixed-Priority Mode-level
FPT	Fixed-Priority Task-level
DM	Deadline-Monotonic
RM	Rate-Monotonic
EDF	Earliest-Deadline-First
BCET	Best-Case Execution Time
WCET	Worst-Case Execution Time
WCRT	Worst-Case Response Time
FF	First-Fit
BF	Best-Fit
WF	Worst-Fit
NF	Next-Fit
FFD	First-Fit Decreasing
BFD	Best-Fit Decreasing
WFD	Worst-Fit Decreasing
FFRM	First-Fit Rate-Monotonic
DBF	Demand Bound Function
RAD	Reasonable Allocation Decreasing
QB	Quadratic Bound
OPA	Optimal Priority Assignment
LKAS	Lane Keeping Assist System

ASCET	Advanced Simulation and Control Engineering Tool
RTOS	Real-Time Operating System priority ceiling protocol (PCP)
PCP	Priority Ceiling Protocol
PIP	Priority Inheritance Protocol
NPP	Non-Preemptive Protocol
TDA	Time-Demand Analysis
DRT	Digraph Real-Time
CPS	Cyber-Physical System
VRB	Variable Rate-dependent Behavior
GMF	Generalized MultiFrame
DRT	Digraph Real-Time
ABS	Anti-lock Brake System
CFG	Control Flow Graph
ILP	Integer Linear Programming
TDMA	Time Division Multiple Access
LLF	Least-Laxity-First
TDA	Time-Demand Analysis
RTA	Response-Time Analysis
TTP	Time-Triggered Protocol
UCB	Useful–Cache-Block
CRPD	Cache-Related Preemption Delay

COLOPHON

This document was typeset in LaTeX using the typographical look-and-feel `classicthesis`. The bibliography is typeset using `biblatex` with `natbib`. No electrons were harmed in the creation of this thesis.