SCHEDULABILITY, RESPONSE TIME ANALYSIS AND NEW MODELS OF P-FRP SYSTEMS

A Dissertation Presented to

the Faculty of the Department of Computer Science

University of Houston

In Partial Fulfillment

of the Requirements for the Degree

Doctor of Philosophy

By

Xingliang Zou

August 2017

SCHEDULABILITY, RESPONSE TIME ANALYSIS AND NEW MODELS OF P-FRP SYSTEMS

Xingliang Zou

APPROVED:

Albert M. K. Cheng, Chairman
Dept. of Computer Science

Guoning Chen
Dept. of Computer Science

Weidong Shi
Dept. of Computer Science

Ji Chen
Dept. of Electrical and Computer Engineering

Dean, College of Natural Sciences and Mathematics

Acknowledgements

Foremost, I am very lucky to have Dr. Albert Mo Kim Cheng as my supervisor. I would like to express my sincere gratitude to Dr. Cheng for the patience, motivation, enthusiasm, and immense knowledge he has shown and given to me. His constant guidance, supports and encouragements as well as the flexibility of choosing research interests leaded all my way on research and writing of this dissertation.

I would like to thank our visiting scholars, Dr. Yu Jiang from Heilongjiang University and Dr. Zhou from Beihang University. The time we spent together on studying and discussing real-time scheduling and broad computer science technologies was among my most cherished memories. Especially Dr. Jiang, I did and will always remember each of our long conversations on research and paper writing all the years even after he finished his visiting and returned China. Thanks my teammates, they have given me encouragements and a wonderful time.

I would like to thank my committee members, Dr. Guoning Chen, Dr. Weidong Shi and Dr. Ji Chen for serving the committee and giving me insightful comments.

Last but most importantly, thank you, my families. My father and mother, it is them that supported me at all aspects in all my life, taught me right from wrong, walked me through my darkest days and encouraged me pursuing my dreams even when sometimes the dreams were somewhat idealistic such as this one was. My sisters and bothers in law, my bothers and sisters in law, they always take so good care of our parents for me, love me and help me whatever and whenever I needed even without my asking. Thanks my nephews and nieces. I am forever grateful for my home of love.

SCHEDULABILITY, RESPONSE TIME ANALYSIS AND NEW MODELS OF P-FRP SYSTEMS

An Abstract of a Dissertation

Presented to

the Faculty of the Department of Computer Science

University of Houston

In Partial Fulfillment

of the Requirements for the Degree

Doctor of Philosophy

By

Xingliang Zou

August 2017

Abstract

Functional Reactive Programming (FRP) is a declarative approach for modeling and building reactive systems. FRP has been shown to be an expressive formalism for building applications of computer graphics, computer vision, robotics, etc. Priority-based FRP (P-FRP) is a formalism that allows preemption of executing programs and guarantees real-time response. Since functional programs cannot maintain state and mutable data, changes made by programs that are preempted have to be rolled back. Hence in P-FRP, a higher priority task can preempt the execution of a lower priority task, but the preempted lower priority task will have to restart after the higher priority task has completed execution. This execution paradigm is called Abort-and-Restart (AR).

Current real-time research is focused on preemptive of non-preemptive models of execution and several state-of-the-art methods have been developed to analyze the real-time guarantees of these models. Unfortunately, due to its transactional nature where preempted tasks are aborted and have to restart, the execution semantics of P-FRP does not fit into the standard definitions of preemptive or non-preemptive execution, and the research on the standard preemptive and non-preemptive may not applicable for the P-FRP AR model. Out of many research areas that P-FRP may demands, we focus on task scheduling which includes task and system modeling, priority assignment, schedulability analysis, response time analysis, improved P-FRP AR models, algorithms and corresponding software.

In this work, we review existing results on P-FRP task scheduling and then present our research contributions: (1) a tighter feasibility test interval regarding the task release offsets as well as a linked list based algorithm and implementation for

scheduling simulation; (2) P-FRP with software transactional memory-lazy conflict detection (STM-LCD); (3) a non-work-conserving scheduling model called Deferred Start; (4) a multi-mode P-FRP task model; (5) SimSo-PFRP, the P-FRP extension of SimSo - a SimPy-based, highly extensible and user friendly task generator and task scheduling simulator.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this chapter, we introduce the definitions of real-time systems, real-time tasks, task scheduling, functional reactive programming(FRP), priority-based functional reactive programming (P-FRP) and its abort-and-restart (AR) execution semantic. We also present the motivations, propositions, contributions and structures of this dissertation.

1.1 Real-Time Systems

Real-time systems are timely response systems in which apart, from giving correct computation output, programs have to complete execution within certain time limits which called deadlines. A running instance of a program that serves certain purposes is called a tasks. Tasks may be virtual computation elements such as threads, processes or data flows.

A task may run once or more times. Each running of a task is called a job. if the recurrence of two consecutive jobs of a task happens at or with a minimum fixed time interval, the task is called periodic or sporadic tasks respectively and the time interval is its period. Additionally, the task is called aperiodic if there is no such a fixed time interval.

Tasks with finishing time requirements are real-time tasks, and the finishing time are called deadlines. In real-time systems, correctness and timeliness of results are both critical. Failing completing a task within its deadline is called a deadline miss. In any of periodic, sporadic and aperiodic circumstances, any failure of real-time tasks to complete execution within the designed time limits results in incorrect or even correct output which can lead to catastrophic or life-threatening situations.

Also, when a set of tasks running on the same platform at the same time, they can have different priorities to others. Task scheduling is thus the research and practice used to arrange the execution order of a set of tasks to satisfy certain criteria.

While a real-time task should complete execution within its deadline, depending on the way of dealing with the consequence of failing the completion timing requirements, there are three categories of real-time systems as following.

I. Hard Real-Time Systems

In a hard real-time system, a deadline miss by any job of a task leads to total system failure which can have serious consequences. Every job of hard real-time task is expected to complete within it deadline in all possible scenarios.

Applications of hard real-time systems can be found in most medical science,

avionic, automotive, telecommunications, space systems, process control and domains where safety and mission criticality are of the utmost importance. Consider an example in the automotive industry, the anti-lock braking systems that are found in the most modern vehicles. When a driver presses the brake pedal forcefully, the vehicle computer should activate the anti-lock braking mechanism quickly. Any delay in the activation of the anti-locking mechanism may cause the vehicle to skid or go out of control leading to a life-threatening situation. The guarantees to complete within a defined time is a critical component in the design of an anti-lock braking system.

II. Soft Real-Time Systems

In a soft real-time system, deadline misses can occur frequently without effecting system stability, but lead to noticeable decrease in output quality or degraded system performance such as slowly responding. All jobs of a soft real-time task can miss their deadline leading to lower quality output but without a serious impact on the overall system.

Video conferencing systems are soft real-time system since, though it is important to render video frames as soon as they arrive, failure to show frames within certain time limits will only affect the video quality and not lead to any life-threatening situations.

III. Firm Real-Time Systems

A system dealing with deadline miss moderately compared to a hard real-time system while stricter than a soft real-time system is called a firm real-time system.

In a firm real-time system, deadline misses which do not occur at frequent intervals do not affect the stability of the system, but the results of a task missing its deadline are considered invalid. Few jobs of a firm task can have deadline misses, however all jobs missing their deadline leads to total system failure.

An aircraft auto-pilot system is an example of a firm real-time system since some jobs can miss their deadlines, but missing deadlines frequently can affect the safety of the aircraft.

1.2 FRP

There are two distinct types of programming paradigms in computer programming: imperative and functional. Functional programming is a programming paradigm that treats computation as the evaluation of mathematical functions and avoids changing-state and mutable data. It is a declarative programming paradigm, which means programming is done with expressions. In functional code, the output value of a function depends only on the arguments that are input to the function, so calling a function F twice with the same value for an argument x will produce the same result *F(x)* each time. Eliminating side effects, i.e., changes in state that do not depend on the function inputs, can make it much easier to understand and predict the behavior of a program, which is one of the key motivations for the development of functional programming.

In contrast, imperative programming changes state with commands in the source language. Imperative programming does have functions in the sense of subroutines,

and not in the mathematical sense. They can have side effects that may change the value of program state. Thus functional programming has a distinct difference from imperative programming in that it is immune to side-effects caused by using states and mutable data.

Since the formal system of λ-calculus (lambda-calculus) was first devised by Church [47] and Kleene [87], many functional programming languages have been invented: LISP, Ocaml, Haskell, Scheme, Erlang, F#, Atom, Scala, and so on. Haskell and Erlang have been studied and commercially developed intensively. Scala is recently adopted by companies such as LinkedIn, Twitter and Walmart [75].

In Computer Science, studying the temporal aspects of programs and analyzing real-time guarantees of a system are collectively studied under "real-time computing". Real-time systems which execute programs based on external events are also called **reactive** systems. An example of a reactive system is the anti-lock braking system we mentioned previously where the vehicle's on-board computer "reacts" to the driver's pressing to the brake pedal by triggering the anti-lock braking system.

FRP, functional reactive programming, integrates time flow and compositional events into functional programming. This provides an elegant way to express computation in domains such as interactive animations, robotics, computer vision, user interfaces, and simulation.

The original formulation of FRP can be found in [64]. In [122], FRP is presented as a framework for constructing reactive applications using the building blocks of functional programming.

The FRP paradigm originates from the Haskell language community and seeks to express easily how programs should react to new input. Conceptually, a data flow graph is built that captures for each value an expression that can be re-evaluated when any dependent value changes. The concept is called *reactive* because it automatically updates graph nodes when dependent nodes change.

In the FRP paradigm, time-varying values are expressed as *behaviors* or *signals*, and the system is notified using *events* when the value of a behavior has changed [101]. A FRP language includes a means of altering or replacing a program based on event occurrences.

- *Behaviors* or *signals*: Functions of time.
- *Events*: Temporal sequences of discrete values.

FRP avoids the *inversion of control* typically associated with systems structured around *callback functions*. A typical pattern associated with inversion of control is backwards structuring of the system behavior and implementation details of individual system functions tend to bubble up to higher layers of the system in order to be able to respond to external stimuli. By avoiding inversion of control, FRP makes it possible to compose system functionality more readily by enabling, e.g., normal nesting of expressions [88] (Chapter 17).

Amsden [2] presents a primary survey of FRP on its literature, implementation, optimization, and uses. Although FRP is often tied to purely functional languages, adaptations of the paradigm to imperative languages exist, notably as a Java library [48] and in C++ as both a language extension adding new grammar [58] and as

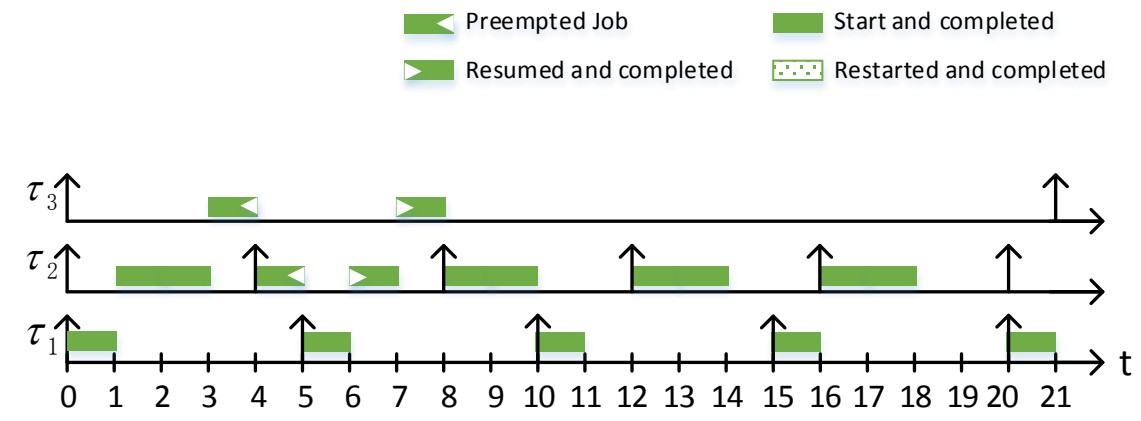
a library based on the standard language [50]. Czaplicki and Chong [49] present Elm, an FRP language focusing on the creation of graphical user interfaces (GUIs). Project [74] presents a simple and practical comparison between FRP libraries.

In a short summary, FRP abstracts real world applications by behaviors (signals) and events, and it can be used to anything reactive.

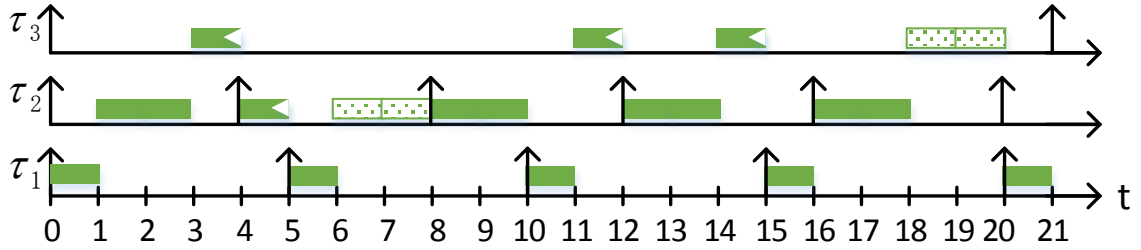
1.3 P-FRP

In real-time systems, the correctness of a program is measured by its logical output as well as its ability to complete within certain time limits. FRP is demonstrated to be effective in modeling and building reactive systems such as graphics, robotic and vision applications. However, another significant feature of real-time systems, priority, is not considered in FRP. To address this problem, P-FRP [84], the priority-based FRP, model has been proposed as a variant of the FRP model.

The P-FRP model supports assigning different priorities to different events, and higher priority events can preempt lower priority ones. However, to maintain guarantees of type safety and stateless execution, the functional programming paradigm requires the execution of a function to be atomic in nature. To comply with this requirement, as well as allow preemption of lower priority events, the P-FRP implements a transactional model of execution. Using only a copy of the state during event execution and atomically committing these changes at the end of execution, the P-FRP model ensures handling an event in a way of "completion or nothing", and thus maintains both the type-safety and the state-less execution paradigm of



(a) Classic Model



(b) P-FRP Model

Figure 1.1: Schedules of fixed priority taskset (τ_i, the i-th task, has execution time C_i and period T_i. $C_1 = 1, C_2 = 2, C_3 = 2, T_1 = 5, T_2 = 4, T_3 = 20$)

the FRP, and does not require the use of synchronization mechanisms between tasks in the system. It has the potential to transform the building of more complicated Cyber-Physical Systems (CPSs). The impact of P-FRP in building safer controllers in automobile anti-lock brake systems has been demonstrated in Christoffersen and Cheng [46].

In the P-FRP model, the transactional execution scheme is also called the Abort-and-Restart (AR), which is a scheme [84] to support the P-FRP, and we also use the P-FRP model or the AR model as the short form of the P-FRP AR model. In

the classic (traditional) preemptive model, the lower priority tasks continue their execution from where they are preempted. It is, however, different in the P-FRP model that when the lower priority tasks are preempted their executions are aborted and restart as new after the higher priority tasks have finished execution. The task scheduling in the classic preemptive model and the P-FRP AR model are shown in Fig.1.1.

Past research in real-time computing for the classic model has focused on systems implemented using imperative programming, which are generally classified under preemptive and non-preemptive systems. Due to its stateless nature of computation, the execution semantics of the P-FRP model is different from classic preemptive and non-preemptive execution and existing state-of-the-art methods developed by real-time researchers for the classic model cannot be applied for ascertaining real-time guarantees in the P-FRP. In this work, we summarize the existing researches on temporal aspects of the P-FRP model, and present our research as well.

1.4 Copy and Restore Operation

In the AR model of P-FRP, when a task starts processing it creates a "scratch" state, which is a *copy* of the current state of the system. Changes made during the processing of this task are maintained inside such a state. When the task has completed, the "scratch" state is *restore*d into the final state in an atomic operation. Therefore, during the restoration and copy operations the task being processed cannot be preempted by higher priority tasks. If the task is preempted after copy

but before the restore operation, the scratch state is simply discarded. The context-switch between tasks only involves a state copy operation for the task that will be commencing processing.

1.5 Contributions

In this work, we have derived several results dealing with real-time analysis of the functional programming model of P-FRP. As previously stated, the transactional nature of execution to P-FRP leads to an execution model where results from prior work cannot be directly applied, hence we had to developed several methods. Based on previous work on P-FRP, we developed more results on P-FRP scheduling. The contributions of our work are summarized below:

- An efficient response time calculation algorithm and its implementation, LList-based RTA algorithm, that can be used on either classic or P-FRP task scheduling.

- Research on the worst case response time and schedulability analysis for the real-time software transactional memory-lazy conflict detection (STM-LCD) model.

- Feasibility interval research. We optimized research on the impact of task's release offsets to task schedulability and the schedulability test interval. Tighter feasibility intervals are found with respect to various task's release offsets.

- An non-work-conserving alternative model, Deferred Start, of the original AR model of P-FRP.

- Multi-mode task model for P-FRP systems. It is the first that multiple modes

instead of single mode for a P-FRP task is proposed in order to reduce the scheduling cost, and thus improve the schedulability of P-FRP task systems.

• SimSo-PFRP. We presented a SimPy based task generator and scheduling simulator with rich algorithms available as an important infrastructure of P-FRP task scheduling research.

1.6 Organization

The rest of this work is organized as following:

Chapter 2 presents the background knowledge including system models, general concepts, terminologies and notations used in this work.

Chapter 3 reviews the priority assignment algorithms used with our observations in both the classic model and the P-FRP AR model.

Chapter 4 summaries previous and existing works as well as our work on P-FRP schedulability analysis, followed by three chapters of detailed introduction of our work on feasibility analysis and improved P-FRP AR models.

Chapter 5 presents details of our work on the P-FRP task schedulability analysis including minimizing the schedulability test interval and some fundamental analysis of uni-processor P-FRP task scheduling.

Chapter 6 presents details of our work on Deferred Start, a non-work-conserving scheduling algorithm for P-FRP task scheduling.

Chapter 7 presents details of our work on the P-FRP multi-mode task scheduling, a novel framework for P-FRP task scheduling.

Chapter 8 presents our work of building SimSo-PFRP, a task generator and simulator with rich algorithms implemented for the P-FRP task scheduling research.

Finally we conclude this work in Chapter 9.

Chapter 2

Background

In this chapter, we introduce the system model, some general concepts, terminologies and notations used in this work.

2.1 Task

τ_i is the i-th of taskset $\Gamma_n = \{\tau_1, \tau_2, ..., \tau_n\}$. The n tasks might be independent if running or not running any task does not depend on status of other tasks. *TasksetSize* or *taskset size* is the number of tasks in a taskset, which is n for Γ_n.

Γ_n is called n-taskset system. An instance of Γ_n is an n-taskset.

J_i^k: An instance or invocation of a task is called a job. J_i^k refers to the k^{th} job of τ_i.

T_i: A constant arrival time period T_i between two successive jobs of the periodic

task τ_i, or the minimal interval between successive jobs of the sporadic task τ_i.

D_i: Relative deadline of the job of τ_i. *Constrained-deadline* means $D_i \leq T_i$, *implicit deadline* means $D_i = T_i$, and *arbitrary deadline* means tasks deadlines may be less than, equal to, or greater than their periods.

Φ_i: Release offset (or phasing) of τ_i. It is the time, relative to time 0, when the first job is released by task τ_i. Let $\Phi_{min}^{\{k\}} = min_{i=1}^{k}\{\Phi_i\}$, and $\Phi_{max}^{\{k\}} = max_{i=1}^{k}\{\Phi_i\}$.

$C_i^m (1 \leq m \leq M_i)$: A job of τ_i has M_i *modes*, C_i^m is the (maximum) computation time of τ_i's job at mode m. Without loss of generality, we assume $0 < C_i^m \leq min\{D_i, T_i\}$ for all i and m. A digraph is used to express the way that modes change. Since this is the first time that the *multi-mode job* is proposed, we also use C_i as the (maximum) computation time of τ_i for the single mode jobs.

Pr_i: Priority of task τ_i, $(1 \leq Pr_i \leq n)$ for Γ_n, and 1 is the highest, n the lowest. In a fixed priority task system, tasks are sorted before analyzing, hence τ_i has priority of i.

U_i^m: Utilization for τ_i at mode m. $U_i^m = C_i^m/T_i$, and $U_i = max_{m=1}^{M_i} U_i^m$. Total utilization of the taskset is $U = \sum_{i=1}^{n} U_i$.

LCM_k: Let $LCM_k = LCM(T_1, T_2, ..., T_k)$ denote the Least Common Multiple (*LCM*, or *hyperperiod*) of the periods of the first k highest priority tasks, $1 \leq k \leq n$, and $LCM_1 = T_1$.

R_i^k: Response time of the J_i^k. It is the time between the release of a job and its completion. In a given priority model, the response time R_i of a task τ_i at a

given release offset pattern is the maximum one of all R_i^k. The Worst-Case Response Time (WCRT) of a task is derived under the worst-case release pattern that leads to the largest interference on the considered task. Such a particular scenario is often referred to as the *critical instant*.

$t_{copy}(i)$: The time taken to make a copy of the state before τ_i starts execution.

$t_{restore}(i)$: The time taken to commit the state after τ_i has completed execution.

P_i: The processing time for τ_i. Processing of a task includes execution as well as copy and restore operations. Hence, $P_i = t_{copy}(i) + C_i + t_{restore}(i)$.

In real-time scheduling, *Integer Time Model* means that temporal parameters are measured by counting the number of clock cycles, hence temporal values have a resolution of one clock cycle. Therefore, Φ_i is assumed to be a non-negative integer value, and other temporal parameters of a task are assumed to be positive integer values.

2.2 Priority

In a taskset, different tasks/jobs can have different priorities compared to other tasks/jobs. We consider three categories.

I. Task-level Priority

In task-level priority, a priority is associated with each task, and all jobs generated by a task have the priority associated with that task. Thus, if task τ_1 has higher

priority than task τ_2 , then whenever both have active jobs, priorities are inherited hence τ_1's jobs will have priority over than τ_2's jobs.

Task-level priority is also called *Fixed Priority* (FP) or *static priority.*

II.Job-level Priority

Unlike task-level priority that priorities are bound to tasks, in job-level priority, for every pair of jobs J_i and J_j, if J_i has higher priority than J_j at some instant in time, then J_i always has higher priority than J_i.

Job-level priority is sometimes called *dynamic priority.* However, there exist a *fully dynamic priority.*

III. Fully Dynamic Priority

In fully dynamic priority, no restrictions are placed on the priorities that may be assigned to jobs, and the relative priority of two jobs may change at any time.

2.3 Task Scheduling

In computer science, scheduling is the method by which a task is assigned to resources that leads to completion of the this task. Resources are computer components such as processor (such as CPU - Central Processing Unit), memory, network bandwidth, etc. We consider processors only in this work. Task scheduling is thus used to assign a processor to a job at a specific moment (time instant). When assigning a job to a processor at time instant t, if a job is running at this processor, there are two ways to

proceed. The first one is called *preemptive scheduling* which compares the two tasks' priorities, and chooses the one with higher priority to execute. If the running one has lower priority, it will be *preempted* and its running information such as memory, registers, etc. will be saved with the cost of memory and processor time, then the new one will be selected to execute. This process is called *context switch*. On the other hand, a *non-preemptive scheduling* is defined if there are no preemptions regardless of the jobs' priorities.

In this work, we consider a hard real-time uni-processor system with hierarchical memory components at first, we then expand our work to multi-processor systems. In a preemptive P-FRP AR system, there are n independent tasks $\Gamma_n = \{\tau_1, \tau_2, ..., \tau_n\}$ to form a *taskset*. We assume that a task can only be preempted at discrete (integer) time instants. We first consider the cost of preemption is zero, then take it into considerations later.

Periodic tasksets may be scheduled in a *synchronous* way if there is some time point at which all tasks arrive at the same time, or in an *asynchronous* way where all tasks never arrive simultaneously. Tasks with common release must be synchronous. However, tasks with different release offsets do not imply they are asynchronous tasks, since they may arrive at the same time in the future. For example, when all task periods are co-prime, the taskset must experience a common release, regardless of any imposed offsets [7][4]. Notice that for an asynchronous scheduling there may still be some tasks (but never all) sharing a common release.

A taskset is referred to be *feasible* with respect to a given system when there exists some scheduling algorithms that can schedule all possible job sequences that

may be generated by the given taskset on that system without missing any deadlines.

A scheduling algorithm is referred to be *optimal* if it can schedule any tasksets when there are any other scheduling algorithms can.

A scheduling algorithm is referred to be *clairvoyant* if it uses information about future events, such as arrival times for future jobs especially for sporadic tasks, actual execution times, which are generally unknown in advance.

A task is referred to be *schedulable* according to a given scheduling policy if its worst-case response time under that scheduling policy is less than or equal to its deadline. Similarly, a taskset is referred to as schedulable according to a given scheduling policy if all of its tasks are schedulable.

A schedulability test is the test used to telling whether or not a taskset is schedulable with respect to certain given conditions such as given system, given release pattern. There are three types of schedulabitlity tests:

I. Sufficient Test

A schedulability test is called sufficient with respect to a scheduling algorithm and a system if all of the tasksets that are deemed schedulable according to the test are in fact schedulable.

II. Necessary Test

A schedulability test is called necessary if all of the tasksets that are deemed unschedulable according to the test are in fact unschedulable.

III. Exact Test

A schedulability test that is both sufficient and necessary is referred to as exact.

A *feasibility interval* is the time interval $[t, t + H)$ such that if all tasks are schedulable in $[t, t + H)$ then the tasks will also be schedulable in the time interval $[0, Z) : Z \to \infty$ [92][7]. Here $[t, t + H)$ is the collection of integers greater than or equal to t and less than $t + H$. A schedulability testing interval $[a, b)$ is the time interval in which the schedulability of a given taskset is checked. In general, the length of $[a, b)$ is not less than that of $[t, t + H)$.

In scheduling theory, a *non-work-conserving* scheduling is one that may let the CPU stay idle even when there are tasks ready for execution, while *work-conserving* does not. Other than classifying scheduling into two types of preemptive and non-preemptive, there is another classification. *Fully preemptive scheduling* allows a running task to be interrupted whenever a higher priority task releases. *Fully non-preemptive scheduling* does not allow a running task to be interrupted by other tasks for any reason. There are scheduling algorithms that lay in between fully preemptive and non-preemptive scheduling algorithms. For example, they set portions of a task preemptive and the rest non-preemptive.

In existing research work for the P-FRP model, it is usually assumed that tasks are independent and there is no precedence constraints between tasks, i.e., tasks cannot be blocked from execution by another task except contending for a processor. Therefore, lock-free and wait-free data sharing have not been considered in the P-FRP model. It is also assumed that once a job starts to execute it will not suspend itself.

A CPU scheduler carries out the processor scheduling activity. CPU Schedulers are often implemented in the way so they keep all processors busy, allow multiple users to share the processors effectively, or to achieve a target quality of service. The concept of scheduling makes it possible to have computer multitasking with a single processor.

A CPU scheduler may aim at one of many goals, for instance, minimizing tasks' response time, maximizing throughput (the total amount of work completed per time unit), or minimizing latency (the time between work becoming enabled and its subsequent completion), maximizing fairness (equal CPU time to each process, or more generally appropriate times according to the priority and workload of each process). In practice, these goals often conflict (e.g., throughput versus latency), thus a scheduler will implement a suitable compromise. In real-time systems, the scheduler needs to schedule tasks to avoid deadline missing.

2.4 Multi-processor Scheduling

Computers are such powerful tools we use in our modern daily life. Our needs and desires upon computers resulted in a rapid increase in both software complexity and the processing demands placed on the underlying hardware. To address demands for increasing processor performance, computer vendors concentrated on increasing processor clock speeds. As this approach has led to problems of both high power consumption and heat dissipation, simply increasing clock speeds in many scenarios is financially even technologically no longer a feasible solution. A decade ago, the

trend was moving toward multi-processor systems. Instead of trying to have one high speed processor to accomplish tasks, a multi-processor system uses multiple relatively lower-speed processors working together to tackle down problems.

Multi-processor systems can be achieved by integrating multiple cores (also called multi-core) in single processor, or integrating multiple processors on single board. With the availability of low-cost multi-core processors and single boards with multiple processors, more real-time and embedded systems are being implemented as multi-core or multi-processor based. While multiple cores / processors increase the throughput of the system, in real-time implementations where tasks have to complete within deterministic bounds, the assignment of tasks to each individual core/processors has to be carefully determined. Presence of multiple processors also gives fail-safe redundancy, a vital requirement in safety critical systems.

Multi-processor systems can generally be divided into two categories from the perspective of scheduling and based on the nature of processors:

I. Heterogeneous Multi-processor System

A heterogeneous multi-processor system can have processors of different types which may be running at different speeds. Furthermore, it is possible that not all tasks are able to execute on all processors.

II. Homogeneous Multi-processor System

In a homogeneous multi-processor system, the processors are identical and thus at the same speed; hence the rate of execution of all tasks is the same on all processors.

Unlike on a uni-processor system, a taskset running on a multi-processor system may be executing on designated or any processor(s). Scheduling algorithms for multi-processor systems can be divided into three categories:

I. Partitioning Scheduling

Partitioning scheduling assigns a processor to every task and all jobs of that task will run on that specific processor.

II. Global Scheduling

Global scheduling on the other hand, allows tasks to migrate among processors during run time, and hence task assignment to processors is dynamic in nature, as compared to fixed processor assignment in partitioned scheduling.

III. Cluster Scheduling

Cluster scheduling is in the "middle" of Partitioning and global scheduling. A task is allowed to run on a specific subset of processors in stead of a single processor or all processors. It is also known as a *semi-partitioning* scheduling.

In global and cluster scheduling, the moving of a task from one processor to another is called migration. According to the degree of migration allowed, there are three disciplines:

(1) No migration. Partitioning scheduling for instance.

(2) Migration allowed, but only at job boundaries (i.e., dynamic partitioning at the job level).

(3) Unrestricted migration (i.e., jobs are also allowed to migrate).

Migration usually comes with *migration cost* such as memory copy, cache loss.

Scheduling algorithms can be run *off-line* before the system is started if all task parameters are known *a priori*, or *on-line* while the system is running.

Chapter 3

Priority Assignment

We introduce related existing work including ours on priority assignments in the classic model and P-FRP AR model.

3.1 On the Classic Model

We first introduce the existing priority assignment research results on uni-processor and multi-processor systems in classic model as following.

3.1.1 Uni-processor Systems

We've mentioned three types of priority assignments in Chapter.2.2.2: task level (static or fixed priority assignment), job level (dynamic priority assignment) and fully dynamic priority assignment.

The most common task level or fixed priority assignment scheme is the rate-monotonic (RM) priority assignment [94]. In RM scheduling, unique priorities are assignment to tasks in which the task with the shortest period is assigned the highest priority, then the task with next shortest period is assigned the second highest priority, and so on so forth. Once a taskset is given, the priorities are assigned and used in entire task scheduling process, and all jobs of a task have the same priority. Liu and Layland [94] have shown that RM is an optimal fixed priority assignment in the classic preemptive execution model for uni-processor task scheduling. The optimality of the RM-priority assignment is derived from the fact that, if a task set is schedulable by any other priority assignment then it is also schedulable by RM priority assignment. However, Leung and Whitehead [91] showed that the optimality for RM priority assignment is valid only for a synchronous (release offset of all tasks is the same) release of tasks.

The most common job level or dynamic priority assignment scheme is the earliest-deadline-first (EDF) [94]. In EDF scheduling, priorities are assigned based on the absolute deadlines of jobs. At any time instant, a job with the nearest deadline has the highest priority. Unlike RM, the priorities are decided on the run time, and jobs of a task may have different priorities. Liu and Layland [94] have shown that EDF is an optimal job level priority assignment in the classic preemptive execution model for uni-processor task scheduling.

The most common fully dynamic priority assignment scheme is the least-laxity-first (LLF) [100]. Laxity is defined as the time that a job can wait until it deemed to miss deadline. It is calculated as a job's absolute deadline minus its remaining

execution time. Thus at each time instant, jobs in the system may have different priorities. Unlike EDF that a job has fixed priority, in LLF, a job has different priorities throughout its execution. LLF is also proven to be an optimal priority assignment in the classic preemptive execution model for uni-processor task scheduling.

We can see that, among these three priority assignment schemes, RM assigns priorities before tasks' execution, and does this assignment only once. EDF assigns priorities at each time when there is a job arriving or finishing execution. LLF calculates and possibly changes priorities at each time instant. So in terms of the time spent on running the priority assignment algorithms, RM has the least complexity, EDF the second, and LLF has the most complexity.

3.1.2 Multi-processor

For priority assignment on multi-processor systems, none of RM, EDF and LLF is optimal.

Horn [70] presented an $O(N^3)$ algorithm (where N is the number of jobs) that is able to determine an optimal multiprocessor schedule for jobs where all of the arrival times and execution times are known *a priori*. This algorithm can be applied to a set of strictly periodic tasks, by considering all of the jobs in the hyperperiod. However, because of its $O(N^3)$ complexity, it is only tractable for tasksets with a relatively short hyperperiod. And this method is not applicable to sporadic tasksets where arrival times are not known in advance.

When scheduling an arbitrary collection of jobs that have more than one distinct

deadline on more than one processor, Hong and Leung [68] [69] showed that such an algorithm would require clairvoyance knowledge of future arrivals and execution times to avoid making decisions that lead to deadline misses. And they proved there is no optimal online scheduling algorithm for such case. Dertouzos and Mok [60] extended this result and showed that knowledge of arrival times is necessary for optimality, even if execution times are known.

Several important algorithms have been proposed for the partitioned allocation of tasks in symmetric homogeneous multiprocessor systems. Dhall and Liu [61] have given the Rate-Monotonic-First-Fit-Scheduling (RMFFS) and Rate-Monotonic-NextFit-Scheduling (RMNFS) algorithms which combine bin-packing and rate-monotonic scheduling to assign tasks to processors. Davari and Dhall [51] [52] have given the First-Fit-Decreasing-Utilization-Factor (FFDUF) and the NEXT-FIT-M algorithms. Oh and Son [106] have given the Rate-Monotonic-First-Fit-Decreasing-Utilization (RMFFDU) which uses a different rate-monotonic schedulability test and is an improvement over the RMFFS and RMNF algorithms.

3.2 On the P-FRP AR Model

Priorities to tasks can be assigned in a static and dynamic way, and the current implementation of P-FRP only allows static priority assignment. Unlike dynamic priority assignment, priorities to tasks in fixed priority scheduling are assignment offline and remain fixed as long as the system is running.

3.2.1 Rate Monotonic, Utilization Monotonic, Deadline Monotonic

In [20],[21], Belwal and Cheng proved that the Rate Monotonic (RM) and the Utilization Monotonic (UM) priority assignments are generally not optimal in the AR model, where RM priority assignment sets a higher priority to a task with a smaller task period (i.e., faster arrival rate) and UM priority assignment orders tasks non-increasingly by task utilization ratios. For 2-task sets, however, the RM and the UM are optimal when one task period is at least two times of the other, and for n-task sets where $n > 2$ the RM priority assignment is optimal only for the case that task periods are harmonic (i.e., integer multiples of each other) and different. Notice that a task with a smaller task period is not necessarily having a larger utilization ratio, for example, a 2-task set with $C_1 = 1$, $T_1 = 3$, and $C_2 = 2$, $T_2 = 4$, where $U_1 = 1/3$ and $U_2 = 1/2$. Notice also that, as stated in [21], under some restrictions RM or UM priority assignment may be optimal in the P-FRP model.

In the classic preemptive model, for periodic task sets with constrained deadlines and scheduled asynchronously, the Deadline Monotonic (DM) priority assignment is generally not optimal, where DM priority assignment orders tasks non-decreasingly by task relative deadlines. In [82], we proved that negative propositions in the classic preemptive model remain negative in the P-FRP model. Therefore, for the same constraints, the DM priority assignment is also generally not optimal in the P-FRP model.

3.2.2 Utilization-and-Rate Monotonic

A priority assignment being U-RM property, i.e., both utilization and rate monotonic, is referred to as the Utilization-and-Rate Monotonic (U-RM) priority assignment, which was mentioned in the paper given by Belwal and Cheng [20], [21]. The idea of the U-RM priority assignment is that a task is assigned with a higher priority if it has a larger utilization ratio and smaller task period. Clearly, this priority assignment is only applicable to task-sets that have the U-RM property. In general this will not be the case.

3.2.3 Execution-time Monotonic

Wong and Burns [123],[124] introduced the Execution-time Monotonic (EM) priority assignment, which assigns a higher priority to a task τ_i which has a larger computation time C_i. Since the EM is not optimal, they derived an improved priority assignment named the Execution-time-toward-Utilization Monotonic (EUM) priority assignment scheme. EUM priority assignment starts with EM ordering and the worst-case is when a task set can only be scheduled by UM ordering (or is not schedulable at all, by only fails at the last task). The lower priority tasks shift up with higher priority level one by one until the task-set is in UM ordering. They showed that the EUM dominated both EM and UM. However, the EUM is also not optimal for the P-FRP model.

In summary, there is still no optimal uni-processor priority assignment discovered for general P-FRP task scheduling at present.

3.2.4 Multi-processor

We introduced priority assignment work on multi-processor classic model above. One common aspect of these algorithms is that they use a sorting order for the first-fit scheme. The sorting is performed on the basis of arrival rates or utilization ratios. Another common aspect of these algorithms is that the schedulability test for tasks assigned to a processor is performed based on criterion defined for the RM scheduling policy.

While this criterion is correct in the preemptive execution model, due to a different execution model the RM-priority assignment is not the optimal priority assignment for P-FRP. Hence, none of the first-fit algorithms that have been presented so far are guaranteed to provide correct results in P-FRP.

Belwal and Cheng [22] presented their P-FRP First-Fit-Decreasing Rate (PFFDR), P-FRP First-Fit-Decreasing Utilization Factor (PFFDUF), P-FRP First-Fit-Decreasing Processing Time (PFFDPT). Their results showed that the number of processors required to schedule P-FRP tasks are also higher than the number of processors required to schedule the same tasks in the preemptive execution model, and they proposed that techniques to avoid preemption for some tasks in P-FRP will be useful for enhancing schedulability in this execution model.

Chapter 4

Schedulability Analysis in P-FRP

In this chapter, we discuss fixed priority scheduling for P-FRP tasks. To ensure a system meets all deadlines in the real world, a schedulability test is required for the task-set in the system. In Burns and Wellings' book [38], four characteristics are defined for testing schedulability: *Necessary*, *Sufficient*, *Exact*, and *Sustainable*. Specifically, a *necessary* test means that if a task-set fails the test it will miss at least one deadline; A *sufficient* test means that if a task-set passes the test it will meet all deadlines; An *exact* test means both characteristics of sufficient and necessary; A *Sustainable* test means that a task set maintains schedulable if conditions for scheduling have been improved (e.g., by reducing the utilization of a task).

In the classic preemptive model [94], a sporadic task can be viewed as a periodic one with a minimum arrival time period for schedulability test or response time analysis without changing the schedulability. This is, however, generally not the case in the P-FRP model. Wong and Burns [123],[124] showed that a sporadic task

with a later release may bring a longer response time. This property implies that in the P-FRP model when a sporadic task set is treated as periodic and is schedulable it is not necessarily schedulable when treated as sporadic. Therefore, in this work sporadic tasks will not be considered unless otherwise stated.

4.1 Critical Instant

A critical instant for a task means the time at which a release of that task will lead to the longest response time. The motivation for finding the critical instant is that when scheduling a task-set, we need to determine the worst-case response time to check if the task-set is schedulable.

In fixed priority scheduling with the classic preemptive model, Liu and Layland [94] proved that a task is at its critical instant when the task and all higher priority tasks are released at the same time (synchronous release). Unfortunately, Belwal and Cheng [23],[21] showed that in the P-FRP model a synchronous release of tasks is not necessarily lead to the worst-case response time.

Moreover, in [82], we observed that in the P-FRP model under fixed priority scheduling both synchronous and asynchronous periodic task sets are not necessarily schedulable even if all of the first jobs of tasks can meet their deadlines, and furthermore, the response time of the first job of any task is also not necessarily to be the WCRT, which are different from that in the classic preemptive model.

Considering all possible release-time and abort combinations, Wong and Burns

[123],[124] showed that finding the critical instant for the AR model with periodic and sporadic tasks is intractable. We argue that in the P-FRP model whether there is generally a critical instant or not has not been proved and if the answer is "yes" the problem of how to find the critical instant efficiently remains also open.

4.2 Feasibility Interval

In the P-FRP model, Belwal *et al.* [26] presented the first result on the feasibility interval and showed that, for a schedulable n-task set Γ_n under fixed priority scheduling, the feasibility interval of Γ_n is $[t, t + LCM_n)$, where $t \geq \Phi_{max}^{\{n\}}$. However, it does not hold for some scenarios. Specifically, the interval $[\Phi_{max}^{\{n\}}, \Phi_{max}^{\{n\}} + LCM_n)$ is not a feasibility interval for some scenarios. In [81], we presented an algorithm to calculate the minimal schedulability testing interval for tasks with arbitrary release offsets. They first employed S_i by $S_1 = \Phi_1$, $S_i = max\{\Phi_i, \Phi_i + \lceil (S_{i-1} - \Phi_i)/T_i \rceil \times T_i\}$, $2 \leq i \leq n$ [67], then presented a testing interval as $[0, min\{\Phi_{max}^{\{n\}} + 2LCM_n, S_n + LCM_n\})$ for Γ_n. In [82], we presented their tight feasibility interval as $[\Phi_{min}^{\{k\}}, \Phi_{min}^{\{k\}} + LCM_k)$ for task set Γ_k of each $1 \leq k \leq n$ when tasks' release satisfies both the *basic phasing* and the *initial busy* conditions.

4.3 Necessary Tests

It is known that a necessary test is in fact a property that a model possesses. It is known that for uni-processor n-task systems a simple necessary test is that the total

utilization factor of the task set is at most 1, i.e., $U \leq 1$. This property holds for both the classic preemptive model and the P-FRP model, under fixed or dynamic priority assignment. Since it provides little information on unschedulable task sets which still satisfy $U \leq 1$, the test is of more importance in the theoretical sense than in the practical sense.

Currently, in the P-FRP model few necessary tests are discovered. In [82], we presented that for a periodic task set under a fixed priority assignment, a necessary schedulability test in the P-FRP model is that it is schedulable for the same priority assignment in the classic preemptive model. As stated in [21], there were also some necessary tests for P-FRP tasks satisfying some special restrictions.

4.4 Sufficient Tests

Sufficient tests are more powerful for schedulability test. As shown in [94], in the classic preemptive model for an n-task set, a sufficient schedulability test is $U = n(2^{1/n} - 1)$ under fixed priority scheduling, and a sufficient schedulability test is $U \leq 1$ (being also a necessary test as shown above) under Earliest Deadline First (EDF) scheduling. It is, however, not necessarily the case in the P-FRP model. This is because that in the classic task system the total processing time in a time period is fixed once the tasks are given, while in the P-FRP model the total processing time is changed at run time due to that the restarting discards the incomplete execution, which is equivalent to adding some processing time into the task system. Therefore, this-like utilization-based sufficient test is generally not applicable to P-FRP tasks,

though under some restrictions there can also be some utilization-based sufficient tests, for example, a utilization bound $1/n$ proposed in [16].

As for non-utilization-based sufficient tests, there were some sufficient tests for P-FRP tasks satisfying some special restrictions [21].

Moreover, for periodic P-FRP tasks satisfying basic phasing and initial busy conditions under a given priority assignment, in [82], we proposed somewhat general and efficient simulation-based sufficient tests for both synchronous and asynchronous tasks, and by the sufficient tests the schedulability is only needed to be checked with an optimal search length LCM_{n-1}, the least common multiple of the first $n-1$ higher priority task periods, instead of LCM_n. It is worthy of stating that the sufficient tests are also applicable to sporadic tasks due to that they are simulation-based tests.

4.5 Exact Tests

Actual response time analysis (RTA) [83] is an "exact" schedulability test based on calculating the worst-case response time of a task which includes the time of interference from other higher priority tasks and blocking from lower priority tasks (due to shared resources or non-preemptive scheduling). RTA is not exact unless blocking is exact.

Since it is still unknown whether there is optimal priority assignment for an n-task set in the P-FRP model, existing researches pay first attention to calculating the longest response time under a given priority assignment. If the longest response time

of a task is longer than its deadline, the task will not meet its deadline. The opposite situation is that if the longest response time of the task is less than or equal to its deadline, the task will meet its deadline. The analysis can be applied for arbitrary deadlines. Several methods to compute the exact response time for the P-FRP model are given by researchers. Notice that except for exhausted permutation of priority assignment ($(n-1)!$ for an n-task set) the so-called worst-case response time results derived in existing P-FRP literatures are indeed longest response time under some given priority assignment since currently there is still no optimal priority assignment discovered for general P-FRP tasks.

4.5.1 Iteration-based Response Time Analysis

A natural way to make response time analysis (RTA) in the P-FRP model is to extend the famous fixed point iteration algorithm developed by Audsley *et al.* [5]. Considering extra costs incurred due to the abort of tasks, Ras and Cheng [112] presented the first RTA expression for computing approximate response time bounds in the P-FRP model. However, the expression failed to converge for some scenarios [21] and therefore was an unreliable method to obtain guaranteed results. Belwal *et al.* [27] introduced a $O(N^2)$ polynomial time algorithm that computes an approximate upper bound on the response time. The experimental analysis showed that the quality of the bound given by this algorithm varies significantly.

Wong and Burns [123],[124] presented also a new iterative expression for calculating response time in the P-FRP model. We find, however, that their new expression

is also unable to give correct result for some scenarios, for example, a synchronous 3-task set with implicit deadlines and task parameters are $C_1 = 1$, $C_2 = 2$, $C_3 = 2$, and $T_1 = 4$, $T_2 = 5$, $T_3 = 20$.

To the best of our knowledge, there is at present still no effectively iterative expression for calculating response time in the P-FRP model.

4.5.2 Gap-Enumeration Method

Time-Accurate Simulation (TAS) [16] is a simple way to calculate the response time in the P-FRP model. The approach is to execute a simulation through each time unit within the feasible interval. Aiming to improve the efficiency, the Gap-Enumeration Method (GEM) [16] which enumerated k-gaps was used for reducing the time complexity for a calculation of response time analysis in the P-FRP model. The idea is that there is a time slot which is first allocated with the highest priority task and second with the next higher priority task and so on. When a task is fitted, the response time for the task is found. If the task τ_i cannot find a gap, it will miss its deadline. The gap-search function can be simply explained in that it searches the first k-gap which is fitted with the size of P_k. The authors use a red-black tree (RB-tree) in their experiments. In the P-FRP model, the first job of τ_i meets the deadline but that does not mean other jobs of τ_i will meet deadlines because a synchronous release may not lead to a critical instant. It remains an open question as to whether the Gap-Enumeration Method is sufficient if there is no critical instant.

4.5.3 Idle-period Game Board Algorithm

Belwal and Cheng [18] thought that the GEM was hard to program because an RB-tree was not available as a native function in programming languages. Another technique using a game board was an easier way because the method can be implemented by using a simple array. Moreover, they changed the term Gap-Enumeration to idle-period. This Game Board Algorithm (GBA), however, was essentially the same as the GEM.

4.5.4 Longest Response Time through Time Petri Nets

Belwal *et al.* [28] developed Time Petri Net (TPN) models for schedulability analysis in the P-FRP model. A publicly available TPN tool called ROMEO [65] was used. In the experiment, 50 task sets of sizes varying from 2 to 4 tasks were synthetically generated. The execution times for these task sets were selected from the range [5, 15], while their arrival periods were selected from [30, 100]. On the same platform with an Intel dual-core machine and Microsoft Windows Vista, the exhaustive enumeration took an average of 1, 8 and 20 minutes to run the simulation of 2, 3 and 4-task sets respectively. The queries on TPN models using ROMEO obtained the result with a 3 to 4 seconds delay for a few 4-task sets.

4.5.5 LList-based Exact Test

The above simulation-based methods are in fact exhaustive-search-based methods for response time and schedulability analysis. Moreover, for an n-task set a test interval with at least a length LCM_n, the least common multiple of all n task periods, was required. It is obviously inefficient for larger LCM_n.

Furthermore, in [82], we have obtained and proved an optimal search length LCM_{n-1} instead of LCM_n for n-task sets satisfying the basic phasing and the initial busy conditions for a given priority assignment in the simulation-based exact response time calculation and schedulability test. This is the current best result on the problem.

Based on the previous research, we present a linked list- and simulation-based exact test method for the classic preemptive model under fixed priority scheduling in [97]. The results have shown that the LList-based exact test is a good candidate in exact response-time tests when task periods span no more than three orders of magnitude. Notice that this method is also applicable to the P-FRP model and this is one of their current work.

4.6 A Case Study for the P-FRP AR model

In [112], Ras and Cheng presented also their case study for the P-FRP AR model. Two cases are studied in software and on hardware. The authors use the Generic Avionics Platform [95] task set in their case studies. In the first study, they evaluated

the AR model under RM and EDF scheduling and then compare the results with other models such as non-preemptive, Priority Ceiling Protocol (PCP) [116], and Stack Resource Policy (SRP) [9]. Another study was for hardware: the Analog Devices' ADuC814 micro-controller was used to run the P-FRP compiled code with RM and EDF scheduling. The result of this case study was the number of tasks against the average number of aborts and the average number of aborts against system load.

4.7 Multi-processor Scheduling

Based on their previous work [112], Ras and Cheng [113] derived inequalities for calculating response time upper bound of a task in a symmetric multi-processor (SMP), under RM and EDF scheduling respectively. This is the first attempt extending the AR model to SMP systems.

Belwal and Cheng [22] presented off-line partitioning of tasks in SMP system in order to find the minimum number of processors required to feasibly schedule the tasks in a given P-FRP task set. The authors implemented three first-fit based heuristics: P-FRP First-Fit-Decreasing Rate, P-FRP First-Fit-Decreasing Utilization Factor and P-FRP First-Fit-Decreasing Processing Time. Unlike previous works where theoretical proofs to validate the performance of the first-fit algorithms are derived, the authors only used experimental tasks sets for the heuristics' comparison and validation.

Chapter 5

Schedulability Testing Interval

This chapter studies the schedulability of real-time tasks in the P-FRP model under fixed priority scheduling, one of the influential scheduling policies. Since the abort-and-restart execution paradigm of the P-FRP model is different from that of the classic preemptive model, the schedulability analysis for P-FRP tasks under fixed priority scheduling differs widely. In P-FRP, for a synchronous n-task set under fixed priority scheduling, the least common multiple (LCM) of all n task periods is the typical length of a testing interval for a schedulability test.

In this chapter, we propose and prove a simulation-based tightly sufficient schedulability test in the P-FRP model under fixed priority scheduling for a given priority

*Work of this chapter was published as: Yu Jiang, Albert M. K. Cheng and Xingliang Zou. Schedulability analysis for real-time P-FRP tasks under fixed priority scheduling. 21th IEEE International Conference on Embedded and Real-Time Computing Systems and Application in Hong Kong, 2015, pp. 31-40.

order, covering scenarios from synchronous task release to asynchronous task release with the initial busy condition, and from implicit deadlines to constrained deadlines. The length of a testing interval for the tightly sufficient test is the LCM of the first n-1 task periods. The tightness of the sufficient condition is shown by examples and the optimality of the search length is proved.

5.1 Introduction

Real-time systems are playing a crucial role in our modern society. Systems including robotic controllers, virtual reality systems, multimedia systems, telecommunications, and many safety-critical industrial systems such as automotive electronics, flight control, space missions, digital oil production control, chemical and nuclear plant control, and so on all make use of real-time system technologies.

To meet diversity requirements of these kinds of real-time systems, various task models and scheduling algorithms have been proposed, from *the classic preemptive model* [94] (shortened as *the classic model* in the rest of this work) to *the priority-based functional reactive programming (P-FRP) model* [84], from *the rate monotonic (RM)* to *the deadline monotonic (DM)* and to *the earliest deadline first (EDF)* scheduling algorithms [94],[92],[91],[43].

Using functional reactive programming languages over the traditional imperative programming style present several advantages especially for implementing safety-critical embedded systems. In this programming paradigm, the programmer can intuitively describe safety behaviors of the system, lowering the chance of introducing

bugs in the design phase, while its stateless nature of execution does not require the use of synchronization primitives, thus reducing the complexity of programming. Specifically, the P-FRP model is a variant of the FRP model which has been employed in a wide range of applications such as graphics, robotics and vision [122], [110]. Besides maintaining both type-safety and state-less execution paradigm of FRP, P-FRP also supports assigning different priorities to different tasks and does not require the use of synchronization mechanism between tasks. P-FRP is a relatively new model in embedded and real-time system research, having received attentions from different research groups [84],[17],[39],[123],[124], and it has the potential to transform the implementation of the control components of future complex cyber-physical systems. The impact of P-FRP in building safer controllers in automobile anti-lock brake systems has been demonstrated in [46].

Like the classic model, in the P-FRP model higher-priority tasks can preempt lower-priority ones. Unlike the classic model in which preempted jobs (instances or invocations) can resume execution from the point they were interrupted, in the P-FRP model preempted jobs will, however, roll back and restart execution from the very beginning at some later time, i.e., preempted jobs will resume in a transactional way, in order to meet the natural atomic execution requirement. The P-FRP model is hence characterized by the "abort-and-restart" (AR) execution semantics, and handles a job in a "completing-or-nothing" way. This provides a scheduling and programming model where response times to different tasks can be tweaked by the programmer without affecting the semantic soundness of the program. Moreover, the AR execution semantics would be better than fully non-preemptive scheduling

since it allows an arriving higher-priority task to preempt the currently running lower-priority task even if they have the same critical section while considering work-conserving scheduling. Furthermore, the AR execution semantics is similar to that in the Preemptible Atomic Regions (PAR) model which is a new concurrency control abstraction for real-time systems [98]. In other words, the AR execution semantics has been implemented in a common programming language.

Real-time systems have timing requirements that must be guaranteed, and schedulability analysis and scheduling policy enables these guarantees to be fulfilled. Sufficient tests are usually efficient but are pessimistic. Exact (necessary and sufficient) tests are more powerful than sufficient tests [5]. Thus, discovering faster exact schedulability tests is a persistent research motivation for the real-time community.

5.1.1 Motivations and Contributions

The motivation for the work of this chapter is threefold. (1) Compared with the classic model, the schedulability analysis in the P-FRP model differs widely because of the AR feature and the scheduling uncertainty of lower priority tasks interfered by higher priority tasks, and thus many results for the classic model may not be applied directly to P-FRP tasks. Lacking powerful and efficient schedulability test would prevent the use of the P-FRP model in practical real-time scheduling. In general, finding exact schedulability tests for various kinds of real-time models would be beneficial for both the theory and the engineering of real-time systems. (2) For a P-FRP n-task set, the current schedulability tests require testing within the interval

consisting of the least common multiple (LCM) of all task periods, or are sufficient with a utilization-based bound $1/n$ under restrictions on the task periods, making it less useful as n gets larger [16]. (3) The motivation also comes from efficiency requirement. In the P-FRP model, the schedulability of a task is a function of both the set of higher priority tasks and their specific priorities, hence, the response time of a task is a function of priority ordering [39]. Thus, many more priority order combinations will be checked to seek an optimal or sub-optimal priority assignment, and a more efficient simulation-based schedulability test is needed, which will also be beneficial for the study of response time analysis.

In this chapter, we propose a tightly sufficient schedulability test for periodic P-FRP n-task sets under fixed priority scheduling for a *given priority assignment* (*GPA*). The sufficient condition cannot be improved further since there really exist task sets reaching the condition. The sufficient test covers cases from synchronous task release to asynchronous task release satisfying certain phasing conditions, from implicit deadlines to constrained deadlines. It is a simulation-based schedulability test which reduces the overall search length, from the LCM of all n task periods to that of the first n-1 higher priority task periods, and the search length is optimal for scheduling P-FRP n-task sets. The corresponding proofs are given for validating lemmas and theorems we have provided. Apart from yielding a contribution to the theory and engineering of P-FRP scheduling, we believe that these results will provide inspiration to studies in the P-FRP model.

5.1.2 Organization

Basic concepts and notations used in this chapter can be found in Chapter.2, and the rest of the chapter is organized as follows. Sec. 5.2 provides new results on the tightly sufficient test in the P-FRP model, including their proofs and illustrations, and Sec. 5.3 presents experimental results showing the efficiency and effectiveness of the new test. Sec. 5.4 gives a brief overview of related work, and finally, Sec. 5.5 states conclusions and future research areas.

5.2 Tightly Sufficient Test for P-FRP Tasks

In this section, we provide our new results on simulation-based tightly sufficient schedulability test in the P-FRP model under fixed priority scheduling for any GPA (*given priority assignment*). We first present results that are related to the feasibility interval in Sec. 5.2.1. Then we give a sufficient schedulability test for synchronous start tasks and generalize it for asynchronous start tasks under certain phasing conditions in Sec. 5.2.2, covering cases of both implicit deadlines and constrained deadlines. In Sec. 5.2.3, we prove the optimality of the search length in the schedulability test.

As stated in the chapter of background, for an n-task set of periodic tasks and a GPA, we relabel tasks with τ_1 being the highest priority task, and denote the task set by Γ_n=$\{\tau_1,\tau_2,...,\tau_n\}$. A task set with different GPAs will be treated as different task sets, since in the P-FRP model whether a task set is schedulable is related to the priority order.

For ease of reference, for each k, $1 \leq k \leq n$, let $\Gamma_k = \{\tau_1, \tau_2, ..., \tau_k\}$ $(\subseteq \Gamma_n)$ to denote the first k higher priority tasks. When a given phasing of Γ_n satisfying $0 \leq \Phi_i < T_i$ for all i, $1 \leq i \leq n$, we designate the given phasing with a term of the ***basic phasing condition***. When Γ_n satisfies the basic phasing condition, each Γ_k $(1 \leq k \leq n)$ also satisfies this condition. For simplicity and clarity, we define following two concepts.

Definition 1. *For Γ_n under fixed priority scheduling, an* **i-permissibility interval** *of task τ_i for a GPA, $1 \leq i \leq n$, is a time interval $[t_u, t_v)$ for the GPA, such that: (1) the interval length, i.e., $t_v - t_u$, is no less than C_i time units, (2) no higher priority task is awaiting execution and ready to execute before time t_u, and (3) no higher priority task is released within it.*

This concept differs from that of the gap in [17] which has not taken the requirement of interval length into account.

Definition 2. *For Γ_n under fixed priority scheduling, the* **Initial Busy Condition** for a GPA *refers to the condition: For each i, $2 \leq i \leq n$, $min_{1 \leq j < i}\{\Phi_j\} < \Phi_i + C_i$ and $\Phi_i \leq max_{1 \leq j < i}\{\Phi_j + R_{j,1}^{GPA}\}$, where $R_{j,1}^{GPA}$ is the response time of the first job of task τ_j for the GPA.*

In the *initial busy condition for a GPA*, the former inequality implies that, for the GPA, the first job of task τ_i cannot complete before or at a time at which the first jobs of higher priority tasks begin releasing, and the latter inequality implies that, for the GPA, the first job of task τ_i is released no later than a time at which all the first jobs of higher priority tasks have completed. For checking the initial busy

condition, what we need to know is the response time of the first job instead of the worst-case response time of each task. When Γ_n satisfies the initial busy condition, each Γ_k ($\subseteq \Gamma_n$) also satisfies the initial busy condition, $1 \leq k \leq n$.

5.2.1 Feasibility Interval in the P-FRP Model

In this subsection, for each Γ_k ($\subseteq \Gamma_n$), $1 \leq k \leq n$, we first present a feature of the release pattern by Lemma 1, then further under the *initial busy condition* we present a feature of the effective execution pattern in the P-FRP model by Lemma 2. Further, we present a result on the feasibility interval in the P-FRP model by Theorem 1.

Lemma 1. *Consider a schedulable Γ_n satisfying the basic phasing condition, then, for each k, $1 \leq k \leq n$, and for any $t \geq \Phi_{min}^{\{k\}}$, the release pattern of Γ_k ($\subseteq \Gamma_n$) in the intervals $[t, t + LCM_k)$ and $[t + LCM_k, t + 2LCM_k)$ will be the same.*

Proof. Two intervals $[t, t+LCM_k)$ and $[t+LCM_k, t+2LCM_k)$ have the same length. The basic phasing condition implies that, for each k, $1 \leq k \leq n$, and for each task τ_i, $1 \leq i \leq k$, in Γ_k, there will be exact LCM_k/T_i jobs of τ_i released in each interval $[t, t + LCM_k)$, where $t \geq \Phi_{min}^{\{k\}}$. The claim then follows from both the periodicity of tasks and the LCM_k being the least common multiple of all periods of tasks in Γ_k. ■

This result is a parallel form of that in the classic model for periodic tasks because a release pattern does not depend on the execution semantics.

Lemma 2. *Consider a schedulable Γ_n satisfying both the basic phasing condition and the initial busy condition, under fixed priority scheduling, then, for each k, $1 \leq k \leq n$, and for any $t \geq \Phi_{min}^{\{k\}}$, the effective execution pattern of Γ_k ($\subseteq \Gamma_n$) in the intervals $[t, t + LCM_k)$ and $[t + LCM_k, t + 2LCM_k)$ will be the same.*

Proof. The lemma is proved by induction.

Base: when $k = 1$, there is only task τ_1, and it is easy to see that for any $t \geq \Phi_1$, the effective execution pattern of $\Gamma_1 = \{\tau_1\}$ in the intervals $[t, t + LCM_1)$ and $[t + LCM_1, t + 2LCM_1)$ will be the same.

Inductive hypothesis: Assume that for any $t \geq \Phi_{min}^{\{k-1\}}$, the effective execution pattern of $\Gamma_{k-1} = \{\tau_1, \tau_2, ..., \tau_{k-1}\}$ in the intervals $[t, t+LCM_{k-1})$ and $[t+LCM_{k-1}, t+2LCM_{k-1})$ is the same.

Inductive step: For $\Gamma_k = \Gamma_{k-1} \cup \{\tau_k\}$, by the inductive hypothesis, we know that, starting from $\Phi_{min}^{\{k-1\}}$, in every successive interval with length of LCM_{k-1} time units, the effective execution pattern of Γ_{k-1} will repeat itself one after another. Considering that LCM_k is a multiple of LCM_{k-1}, that Γ_k satisfies both the basic phasing condition and the initial busy condition, that Γ_k is schedulable and tasks are periodic, and that the priority of task τ_k is lower than priorities of tasks in Γ_{k-1} and hence task τ_k produce no preemption during the execution of tasks in Γ_{k-1}, thus, for any $t \geq \Phi_{min}^{\{k\}}$, the effective execution pattern of Γ_{k-1} in the intervals $[t, t + LCM_k)$ and $[t + LCM_k, t + 2LCM_k)$ will be the same.

Next, we prove that, for any $t \geq \Phi_{min}^{\{k\}}$, the effective execution pattern of the task τ_k in the intervals $[t, t + LCM_k)$ and $[t + LCM_k, t + 2LCM_k)$ will be the same.

By the initial busy condition, the first job of task τ_i cannot complete before or at a time at which the first jobs of higher priority tasks begin releasing, and the first job of task τ_i is released no later than a time at which all the first jobs of higher priority tasks have completed, i.e., the first job of task τ_k will have to (re)start executing after that all the first jobs of higher priority tasks have completed. From the above results we have known that, for any $t \geq \Phi_{min}^{\{k\}}$, the effective execution pattern of Γ_{k-1} in the intervals $[t, t + LCM_k)$ and $[t + LCM_k, t + 2LCM_k)$ is the same. We also know that a release of a lower priority task has no impact on the execution of higher priority tasks. Thus, after scheduling all tasks in Γ_{k-1}, for every value of b, $0 \leq b < LCM_k$, if there is a k-permissibility interval of task τ_k starting from time $t+b$, then there will also be a same length k-permissibility interval of task τ_k starting from time $t+LCM_k+b$. Moreover, by Lemma 1, the release pattern of task τ_k in the intervals $[t, t+LCM_k)$ and $[t+LCM_k, t+2LCM_k)$ will be the same. In addition, for any schedulable task, it will (re)start its execution at the starting point of its nearest permissibility interval after its release time and complete its execution in the same permissibility interval. Therefore, for any $t \geq \Phi_{min}^{\{k\}}$, the effective execution pattern of the task τ_k in the intervals $[t, t + LCM_k)$ and $[t + LCM_k, t + 2LCM_k)$ will be the same, and thus the effective execution pattern of $\Gamma_k = \Gamma_{k-1} \cup \{\tau_k\}$ in the intervals $[t, t + LCM_k)$ and $[t + LCM_k, t + 2LCM_k)$ will then be the same. ■

Theorem 1. *Consider a schedulable* Γ_n *satisfying both the basic phasing condition and the initial busy condition, under fixed priority scheduling, then, for each* k, $1 \leq k \leq n$, *a feasibility interval of* Γ_k *is* $[t, t + LCM_k)$, *where* $t \geq \Phi_{min}^{\{k\}}$.

Proof. If a schedulable system runs till time Z, where Z is a sufficiently large positive

integer ($Z \rightarrow +\infty$), then for each k, $1 \leq k \leq n$, we obtain a number $j = \lfloor (Z - t_\Phi)/LCM_k \rfloor$, where $t_\Phi = \Phi_{min}^{\{k\}}$. Therefore, the time interval [0, Z) can be divided into some time intervals and the union of these consecutive time intervals will lead to the time interval [0, Z), i.e.,

$[0, Z) = [0, t_\Phi) \cup [t_\Phi, t_\Phi + j \times LCM_k) \cup [t_\Phi + j \times LCM_k, Z)$, where $[t_\Phi, t_\Phi + j \times LCM_k) = [t_\Phi, t_\Phi + LCM_k) \cup [t_\Phi + LCM_k, t_\Phi + 2LCM_k) \cup ... \cup [t_\Phi + (j-1) \times LCM_k, t_\Phi + j \times LCM_k)$.

By Lemma 1 and Lemma 2, both the release and the effective execution patterns of Γ_k in time intervals $[t_\Phi, t_\Phi + LCM_k)$, $[t_\Phi + LCM_k, t_\Phi + 2LCM_k)$,...,$[t_\Phi + (j - 1) \times LCM_k, t_\Phi + j \times LCM_k)$ will be the same. Thus, schedules of Γ_k in these j time intervals will be the same. Due to the periodicity of tasks, schedules in time intervals $[t_\Phi + j \times LCM_k, Z)$ and $[t_\Phi + (j - 1) \times LCM_k, t_\Phi + (j - 1) \times LCM_k + (Z - (t_\Phi + j \times LCM_k)))$ will also be the same. Therefore, if task set Γ_k is schedulable in time interval $[t_\Phi, t_\Phi + LCM_k)$, it will also be schedulable in intervals $[t_\Phi + LCM_k, t_\Phi + 2LCM_k), ..., [t_\Phi + (j - 1) \times LCM_k, t_\Phi + j \times LCM_k)$, and $[t_\Phi + j \times LCM_k, Z)$. Task set Γ_k is also schedulable in the time interval [0, t_Φ). Hence, the schedulability of Γ_k in the time interval $[t_\Phi, t_\Phi + LCM_k)$ will determine the schedulability of Γ_k in the time interval [0, Z). By its definition, a feasibility interval of the task set Γ_k is $[\Phi_{min}^{\{k\}}, \Phi_{min}^{\{k\}} + LCM_k)$.

For any $t \geq \Phi_{min}^{\{k\}}$, further considering the length of intervals $[t, t + LCM_k)$ is equal to a fixed value LCM_k, the theorem then follows. ■

The basic phasing condition and the initial busy condition guarantee that the

starting point of the first feasibility interval shifts to the left and is from $\Phi_{min}^{\{k\}}$ for Γ_k (instead of from $\Phi_{max}^{\{n\}}$ for Γ_n previously), making less constructing length of schedules in some cases. For example, it needs only to construct a schedule for Γ_n in the interval $[0,\ LCM_n)$ instead of in $[0,\ \Phi_{max}^{\{n\}} + LCM_n)$ due to $\Phi_{min}^{\{n\}} = 0$.

In the following subsection, based on the above results, we present new results on schedulability test in the P-FRP model under fixed priority scheduling.

In both implicit deadlines and constrained deadlines scenarios, in order to be schedulable, each job of a task should complete its execution within its relative deadline D_i time units. Since the scenario with implicit deadlines is a special case of that with constrained deadlines, we present the sufficient schedulability test in a unified form. In the rest of the chapter, without explicit description, we assume that the task set Γ_n is with implicit or constrained deadlines, for a GPA, and under fixed priority scheduling in the P-FRP model.

For Γ_n satisfying both the basic phasing condition and the initial busy condition for a GPA, for each k, $2 \leq k \leq n$, when there are $j_k > 0$ k-permissibility intervals of τ_k in the time interval $[\Phi_{min}^{\{k-1\}},\ \Phi_{min}^{\{k-1\}} + LCM_{k-1})$, denoted by $[t_1, t_2)$, $[t_3, t_4)$,..., $[t_{2j_k-1}, t_{2j_k})$, and the next k-permissibility interval is $[t_{2j_k+1}, t_{2j_k+2})$, we define $l_{max}^{\{k\}}$, a quantity which will be used in following theorems:

$$l_{max}^{\{k\}} = max_{1 \leq i \leq j_k}\{t_1 - \Phi_k + C_k, t_{2i+1} - t_{2i} + 2C_k - 1\}$$

The case of synchronous start can be seen as a special case of asynchronous start where each phasing equals to 0. For clarity, we first present the sufficient test for synchronous start tasks, and then generalize it for asynchronous start tasks.

5.2.2 A Tightly Sufficient Schedulability Test

In this subsection, we first introduce a simulation-based tightly sufficient schedulability test for the scenario of synchronous start and then generalize it for that of asynchronous start. Note that the synchronous start tasks satisfies both the basic phasing condition and the initial busy condition.

Theorem 2. *Consider a synchronous start Γ_n for a GPA, then, for each k, $2 \leq k \leq n$, Γ_k ($\subseteq \Gamma_n$) is schedulable for the GPA if: (1) Γ_{k-1} is schedulable in the time interval [0, LCM_{k-1}), and (2) there is at least one k-permissibility interval of τ_k in the time interval [0, LCM_{k-1}), and $D_k \geq l_{max}^{\{k\}}$, and $T_k \geq l_{max}^{\{k\}}$ or $T_k = LCM_{k-1}$.*

Proof. For synchronous start, $\Phi_k = 0$ $(1 \leq k \leq n)$.

For each k, $2 \leq k \leq n$, by preconditions, Γ_{k-1} will be schedulable since Γ_{k-1} is schedulable in the time interval [0, LCM_{k-1}), which is a feasibility interval of Γ_{k-1} by Theorem 1. In order to prove Γ_k is schedulable, we need to prove that task τ_k is schedulable.

There is at least one k-permissibility interval of τ_k in the time interval [0, LCM_{k-1}), i.e., there are $j_k > 0$ k-permissibility intervals of τ_k in that interval, denoted by $[t_1, t_2)$, $[t_3, t_4)$,..., $[t_{2j_k-1}, t_{2j_k})$, and the next k-permissibility interval is $[t_{2j_k+1}, t_{2j_k+2})$. $D_k \geq l_{max}^{\{k\}} = max_{1 \leq i \leq j_k}\{t_1 + C_k,\ t_{2i+1} - t_{2i} + 2C_k - 1\}$, then there is $D_k \geq t_1 + C_k$ which implies that the first job of task τ_k will not have a deadline miss.

If $T_k = LCM_{k-1}$, there will be only one job of task τ_k released in each interval $[m \times LCM_{k-1}, m \times LCM_{k-1} + LCM_{k-1})$, where $m \geq 0$. Note also that, starting from

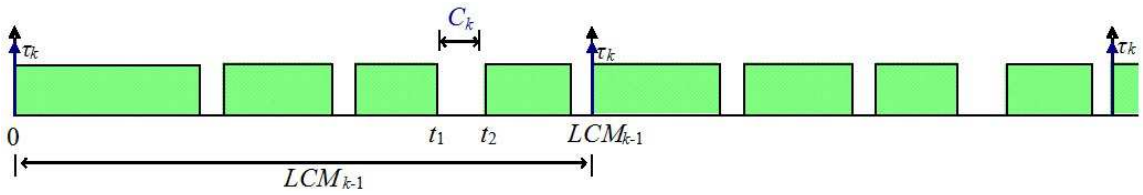


Figure 5.1: Illustration of the case T_k=LCM_{k-1} in the proof of Theorem 2.

time 0 for a synchronous start, in every successive interval with length of LCM_{k-1} time units, the schedule of Γ_{k-1} will repeat itself one after another, including k-permissibility intervals of τ_k in that interval. In this case, task τ_k will keep pace with those tasks in Γ_{k-1} as a whole, as shown in Fig.5.1. Thus, similar to the first job, each of all the other jobs of task τ_k will not have a deadline miss, i.e., τ_k is schedulable. The task set Γ_k is hence schedulable.

We then consider the arrival positions of other jobs of task τ_k in the interval [0, LCM_{k-1}) at the time line. For a k-permissibility interval $[t_{2m-1}, t_{2m})$, $2 \leq m \leq j_k$, (a) if a job arrives at a position between time points t_{2m-2} and t_{2m-1}, then it will complete its execution in the interval $[t_{2m-1}, t_{2m-1} + C_k)$, and its relative deadline D_k should be no less than the relative time $t_{2m-1} + C_k - t_{2m-2}$, i.e., $D_k \geq t_{2m-1} - t_{2m-2} + C_k$; (b) if a job arrives at a position between time points t_{2m-1} and $t_{2m} - C_k$, then it will complete its execution in the k-permissibility interval $[t_{2m-1}, t_{2m})$, and at this time there will be $D_k \geq C_k$; (c) if a job arrives at a position between time points $t_{2m} - (C_k - 1)$ and t_{2m}, then it will not be able to complete its execution in this interval since the remaining maximum interval length $C_k - 1$ is less than C_k. Thus, due to the "completing-or-nothing" way of execution in the P-FRP model, the job will have to be postponed until in the next k-permissibility interval $[t_{2m+1}$,

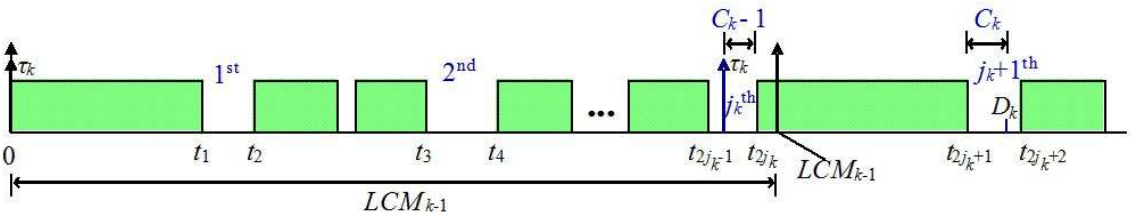


Figure 5.2: Illustration of the case (c) in the proof of Theorem 2.

t_{2m+2}) to complete its execution. At this time, its relative deadline D_k should be no less than the relative time $t_{2m+1} - (t_{2m} - (C_k - 1)) + C_k = t_{2m+1} - t_{2m} + 2C_k - 1$, i.e., $D_k \geq t_{2m+1} - t_{2m} + 2C_k - 1$. Then, considering all the $j_k + 1$ k-permissibility intervals, if $D_k \geq l_{max}^{\{k\}} = max_{1 \leq i \leq j_k}\{t_1 + C_k, t_{2i+1} - t_{2i} + 2C_k - 1\}$, each job of task τ_k will have no deadline miss. An example of the case (c) is illustrated in Fig.5.1.

Therefore, if $T_k \neq LCM_{k-1}$, by $T_k \geq D_k \geq l_{max}^{\{k\}} = max_{1 \leq i \leq j_k}\{t_1 + C_k, t_{2i+1} - t_{2i} + 2C_k - 1\}$, for any job of task τ_k, neither a deadline nor the next job of τ_k will arrive before the current job of τ_k completes, i.e., each job of τ_k will not have a deadline miss (the worst case for D_k is shown in Fig.5.2). Thus, τ_k is schedulable, and the task set Γ_k is hence schedulable. ■

Example: Consider a 3-task set $\Gamma_3 = \{\tau_1, \tau_2, \tau_3\}$ with computation times $C_1 = 3$, $C_2 = 4$, $C_3 = 3$, and $T_1 = D_1 = 9$, $T_2 = D_2 = 12$, $T_3 = D_3 = 32$, released synchronously at time 0, under fixed priorities as the task numbers indicate (1 being the highest). There are $LCM_1 = 9$, $LCM_2 = 36$, and $LCM_3 = 288$. Fig.5.3(a) shows the schedule in the time interval [0, 36), i.e., [0, LCM_2). The upward arrow indicates the release of a new job, the downward arrow indicates the completion of a currently running job, and the cross-mark indicates a lower-priority task being aborted by a higher-priority one. A preempted lower-priority task has to rollback and restart at

some later time.

Task τ_1 is clearly schedulable. For task τ_2, there is only one 2-permissibility interval in the time interval [0, LCM_1). There are $t_1 = 3$, $t_2 = 9$, $t_3 = 12$, and $2C_2 - 1 = 7$. $l_{max}^{\{2\}}$ equals to 10, and $D_2 = T_2 = 12 > l_{max}^{\{2\}}$. Task τ_2 is then schedulable. For task τ_3, there is only one 3-permissibility interval in the time interval [0, LCM_2). At this time, $t_1 = 21$, $t_2 = 24$, $t_3 = t_1 + LCM_2 = 21 + 36 = 57$, and $2C_3 - 1 = 5$. $l_{max}^{\{3\}} = max\{t_1 + C_3, t_3 - t_2 + 2C_3 - 1\} = max\{24, 38\} = 38$, and $D_3 = 32 < l_{max}^{\{3\}}$. By the theorem, task τ_3 will be unschedulable. In fact, all tasks meet their deadlines for the first jobs. It seems that the task set is schedulable and the worst-case response time (WCRT) of the task set is 24. This is, however, not the case. As shown in Fig.5.3(b), the 5th job of τ_3 will arrive at time 128, and the 4th job of τ_3 arriving at time 96 will miss its deadline. In order to make this task set schedulable when released synchronously, the period T_3 should be equal to 36 ($= LCM_2$) or be no less than 38 ($= l_{max}^{\{3\}}$), as required by Theorem 2, and will be an example of synchronous start task sets reaching the sufficient condition. On the other hand, it is enough now to test the schedulability of this task set in a time interval [0, 36) instead of in [0, 288), with an 8 times efficiency improvements.

There are two basic ideas behind the result for testing the schedulability of P-FRP periodic n-task set satisfying the basic phasing condition and the initial busy condition. First, by the concept of k-permissibility interval, for checking whether task τ_k is schedulable, there is no need to consider each gap in a feasibility interval of task set Γ_{k-1}. Second, for task sets satisfying the initial busy condition, the number of jobs of task τ_k in the time interval [0, LCM_{k-1}) will get the largest in

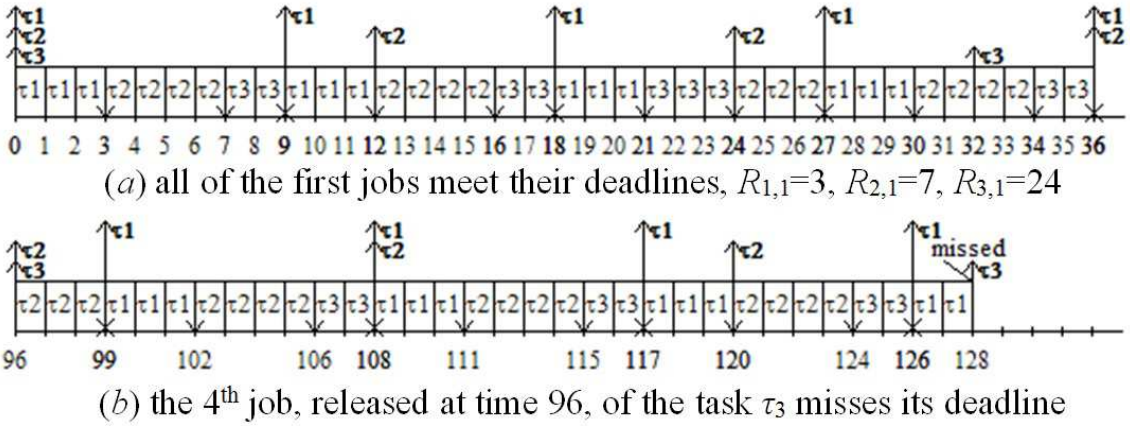


Figure 5.3: A synchronous scenario in the P-FRP model, all the 1st jobs meeting their deadlines (C_1=3, C_2=4, C_3=3; T_1=D_1=9, T_2=D_2=12, T_3=D_3=32). However, the 4th job of task τ_3 will miss its deadline at time 128.

each feasibility interval with a length LCM_{k-1} of Γ_{k-1} for synchronous start and the schedule of Γ_{k-1} comprises of repeated feasibility interval and k-permissibility intervals related to the time interval $[0, LCM_{k-1})$.

We now present a simulation-based sufficient schedulability test for the scenario of asynchronous start, which is a generalized form of that for the scenario of synchronous start.

Theorem 3. *Consider an asynchronous start Γ_n satisfying both the basic phasing condition and the initial busy condition for a GPA, then, for each k, $2 \leq k \leq n$, Γ_k ($\subseteq \Gamma_n$) is schedulable for the GPA if: (1) Γ_{k-1} is schedulable in the time interval $[\Phi_{min}^{\{k-1\}}, \Phi_{min}^{\{k-1\}} + LCM_{k-1})$, and (2) there is at least one k-permissibility interval of τ_k in the time interval $[\Phi_{min}^{\{k-1\}}, \Phi_{min}^{\{k-1\}} + LCM_{k-1})$, and $D_k \geq l_{max}^{\{k\}}$, and when $t_1 - \Phi_k + C_k > LCM_{k-1}$, there is $T_k \geq l_{max}^{\{k\}}$; when $t_1 - \Phi_k + C_k \leq LCM_{k-1}$, there is $T_k \geq l_{max}^{\{k\}}$ or $T_k = LCM_{k-1}$.*

Proof. For each k, $2 \leq k \leq n$, by preconditions, Γ_{k-1} will be schedulable since Γ_{k-1} is schedulable in the time interval $[\Phi_{min}^{\{k-1\}}, \Phi_{min}^{\{k-1\}} + LCM_{k-1})$, which is a feasibility interval of Γ_{k-1} by Theorem 1. In order to prove Γ_k is schedulable, we need to prove that task τ_k is schedulable.

By preconditions, there is at least one k-permissibility interval of τ_k in the time interval $[\Phi_{min}^{\{k-1\}}, \Phi_{min}^{\{k-1\}} + LCM_{k-1})$, i.e., there are $j_k > 0$ k-permissibility intervals of τ_k in that interval, denoted by $[t_1, t_2)$, $[t_3, t_4)$,..., $[t_{2j_k-1}, t_{2j_k})$, and the next k-permissibility interval is $[t_{2j_k+1}, t_{2j_k+2})$. Since $T_k \geq D_k \geq l_{max}^{\{k\}} = max_{1 \leq i \leq j_k}\{t_1 - \Phi_k + C_k, t_{2i+1} - t_{2i} + 2C_k - 1\}$, then there is $T_k \geq D_k \geq t_1 - \Phi_k + C_k$ which implies that the first job of task τ_k will not have a deadline miss.

When $t_1 - \Phi_k + C_k > LCM_{k-1}$: At this time, there is, in fact, only one k-permissibility interval of τ_k in the time interval $[\Phi_{min}^{\{k-1\}}, \Phi_{min}^{\{k-1\}} + LCM_{k-1})$, and then there is only one k-permissibility interval in each of the time interval $[\Phi_{min}^{\{k-1\}} + m \times LCM_{k-1}, \Phi_{min}^{\{k-1\}} + (m+1) \times LCM_{k-1})$ where $m \geq 0$, due to the repetition of the schedule of the schedulable Γ_{k-1}. As observed earlier, due to $T_k (\geq D_k \geq l_{max}^{\{k\}} \geq t_1 - \Phi_k + C_k) > LCM_{k-1}$, the release time points of jobs of task τ_k will drift to right bit by bit relative to certain time points $\Phi_{min}^{\{k-1\}} + m \times LCM_{k-1}$ where $m \geq 1$, and there will be at most one job of task τ_k released before the starting time point of the next k-permissibility intervals of τ_k, and thus task τ_k is schedulable. The task set Γ_k is hence schedulable. This scenario is illustrated in Fig.5.4(b).

Then we consider the arrival positions of other jobs of task τ_k in the interval $[\Phi_{min}^{\{k-1\}}, \Phi_{min}^{\{k-1\}} + LCM_{k-1})$ at the time line. For a k-permissibility interval $[t_{2m-1}, t_{2m})$, $2 \leq m \leq j_k$, (a) if a job of task τ_k arrives at a position between time points t_{2m-2}

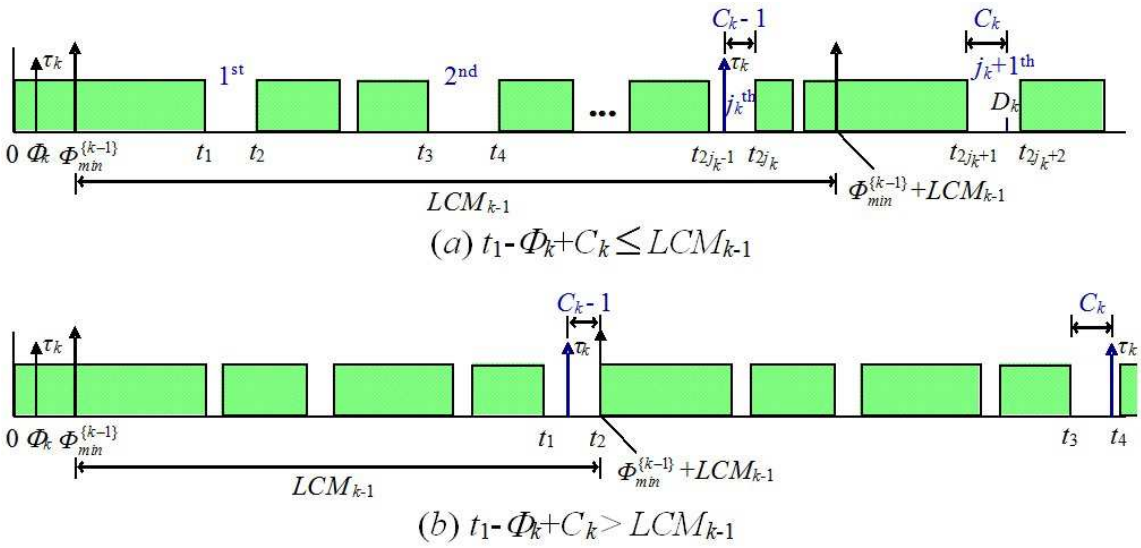


Figure 5.4: Illustration of the case (c) in the proof of Theorem 3.

and t_{2m-1}, then it will complete its execution in the interval $[t_{2m-1},\ t_{2m-1}+C_k)$, and its relative deadline D_k should be no less than the relative time $t_{2m-1}+C_k-t_{2m-2}$, i.e., $D_k \geq t_{2m-1}-t_{2m-2}+C_k$; (b) if a job of task τ_k arrives at a position between time points t_{2m-1} and $t_{2m}-C_k$, then it will complete its execution in the k-permissibility interval $[t_{2m-1},\ t_{2m})$, and at this time there will be $D_k \geq C_k$; (c) if a job of task τ_k arrives at a position between time points $t_{2m}-(C_k-1)$ and t_{2m}, then it will not be able to complete its execution in this interval since the remaining maximum interval length C_k-1 is less than C_k. Thus, due to the "completing-or-nothing" way of execution in the P-FRP model, the job will have to be postponed until in the next k-permissibility interval $[t_{2m+1},\ t_{2m+2})$ to complete its execution. At this time, its relative deadline D_k should be no less than the relative time $t_{2m+1}-(t_{2m}-(C_k-1))+C_k = t_{2m+1}-t_{2m}+2C_k-1$, i.e., $D_k \geq t_{2m+1}-t_{2m}+2C_k-1$. Then, considering all the j_k+1 k-permissibility intervals, if $D_k \geq l_{max}^{\{k\}} = max_{1\leq i\leq j_k}\{t_1-\Phi_k+C_k, t_{2i+1}-t_{2i}+2C_k-1\}$,

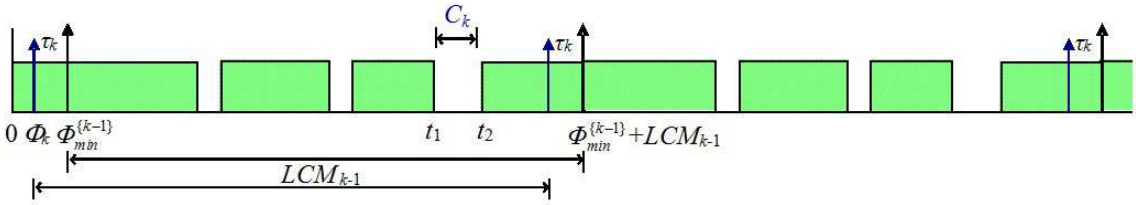


Figure 5.5: Illustration of the case T_k=LCM_{k-1} when $t_1 - \Phi_k$+$C_k \leq LCM_{k-1}$ in the proof of Theorem 3.

each job of task τ_k will have no deadline miss. An example of the case (c) is illustrated in Fig.5.4(a).

When $t_1 - \Phi_k + C_k \leq LCM_{k-1}$: If $T_k = LCM_{k-1}$, then task τ_k will keep pace with those tasks in Γ_{k-1} as a whole, as shown in Fig.5.5. There is at least one k-permissibility interval of task τ_k in the time interval $[\Phi_{min}^{\{k-1\}}, \Phi_{min}^{\{k-1\}} + LCM_{k-1})$ and, starting from time $\Phi_{min}^{\{k-1\}}$, in every successive interval with length of LCM_{k-1} time units, the schedule of Γ_{k-1} will repeat itself one after another, including k-permissibility intervals of τ_k in that interval. Thus, similar to the first job, each of all the other jobs of task τ_k will not have a deadline miss, i.e., τ_k is schedulable. The task set Γ_k is hence schedulable.

If $T_k \neq LCM_{k-1}$, by $T_k \geq D_k \geq l_{max}^{\{k\}} = max_{1 \leq i \leq j_k}\{t_1 - \Phi_k + C_k, t_{2i+1} - t_{2i} + 2C_k - 1\}$, for any job of task τ_k, neither a deadline nor the next job of τ_k will arrive before the current job of τ_k completes, i.e., each job of τ_k will not have a deadline miss (the worst case for D_k is shown in Fig.5.4(a)). Thus, τ_k is schedulable, and Γ_k is hence schedulable. ■

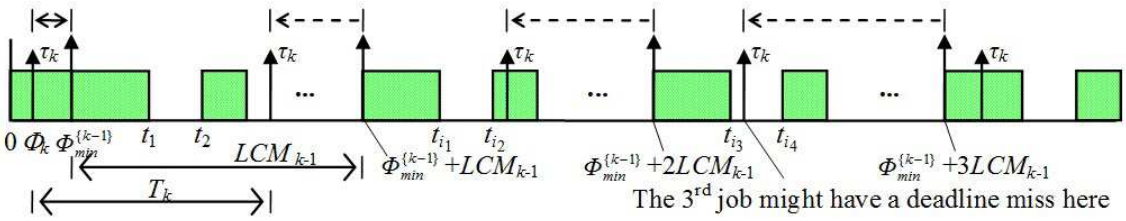
The proof of Theorem 3 follows the same reasoning of the proof for Theorem 2 with explicit consideration on phasing. Besides making the starting point of the first

feasibility interval of Γ_k to be $\Phi_{min}^{\{k\}}$, the purpose of the phasing conditions in the theorem is also aiming to meet a possible need of recursively using this test in other ways of simulation.

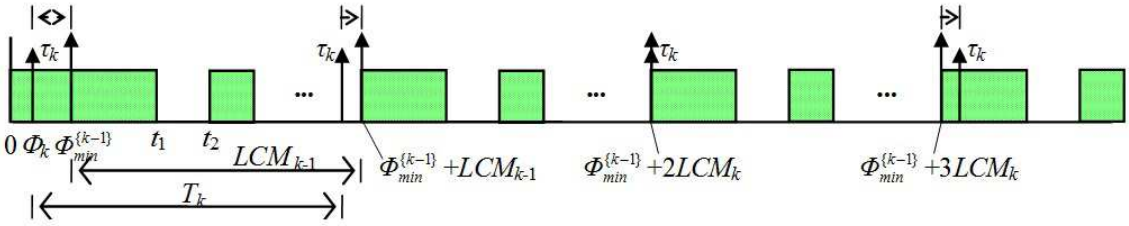
An example for an asynchronous start is a 3-task set where $C_1 = 3$, $C_2 = 4$, $C_3 = 3$; $T_1 = D_1 = 9$, $T_2 = D_2 = 12$, $T_3 = D_3 = 35$; and $\Phi_1 = 2$, $\Phi_2 = 1$, $\Phi_3 = 0$. In this scenario, it is also unschedulable though all the first jobs have meet their deadlines, since the 2nd job of the τ_3 arriving at time 35 will miss its deadline in the time interval [68, 73). In order to make the task set schedulable when scheduled asynchronously with the above phasing, the period T_3 should be no less than 36, as required by Theorem 3, where $\Phi_{min}^{\{2\}} + LCM_2 = 1 + 36 = 37$, $t_1 = 32$, $t_2 = 37$, $2C_3 - 1 = 5$, and $l_{max}^{\{3\}} = 36$. At this time, it is also an asynchronous start example of task sets reaching the sufficient condition. In fact, we observed that in the P-FRP model under fixed priority scheduling a periodic n-task set ($n \geq 2$), for both synchronous start and asynchronous start, is not necessarily schedulable even if all of the first jobs of tasks can meet their deadlines. Furthermore, in the P-FRP model under fixed priority scheduling the response time of the first job of any task is also not necessarily to be the WCRT, which is different from that in the classic model.

5.2.3 Discussion

We note that, when $T_k < LCM_{k-1}$, the release time points of jobs of task τ_k will drift to left bit by bit relative to certain time points $\Phi_{min}^{\{k-1\}} + m \times LCM_{k-1}$, and when $T_k > LCM_{k-1}$, the release time points of jobs of task τ_k will drift to right



(a) $T_k < LCM_{k-1}$ ($< l_{max}^{\{k\}}$), the release time points of jobs of task τ_k drifting to the left bit by bit from time points $\Phi_{min}^{\{k-1\}} + m \times LCM_{k-1}$, where $m \geq 1$.



(b) $LCM_{k-1} < T_k$ ($< l_{max}^{\{k\}}$), the release time points of jobs of task τ_k drifting to the right bit by bit from time points $\Phi_{min}^{\{k-1\}} + m \times LCM_{k-1}$, where $m \geq 1$.

Figure 5.6: Illustration of the drifting of release time points of task τ_k when $T_k \neq LCM_{k-1}$ and $T_k < l_{max}^{\{k\}}$.

bit by bit relative to certain time points $\Phi_{min}^{\{k-1\}} + m \times LCM_{k-1}$, where $m \geq 1$. If $T_k < LCM_{k-1} \leq l_{max}^{\{k\}}$ or $LCM_{k-1} < T_k < l_{max}^{\{k\}}$, then certain job of τ_k may eventually miss its deadline at some time point, as shown in Fig.5.6.

In addition, it is easy to derive a sufficient schedulability test for a 2-task set Γ_2 from Theorem 3, as Lemma 3 states.

Lemma 3. *Consider* $\Gamma_2 = \{\tau_1, \tau_2\}$ *satisfying the basic phasing condition and the initial busy condition for a GPA, then,* Γ_2 *is schedulable for the GPA if:* $T_1 \geq C_1 + C_2$, *and* $D_2 \geq l_{max}^{\{2\}}$, *and (a)* $T_2 \geq l_{max}^{\{2\}}$ *when* $\Phi_1 + C_1 - \Phi_2 + C_2 > T_1$, *or (b)* $T_2 \geq l_{max}^{\{2\}}$ *or* $T_2 = T_1$ *when* $\Phi_1 + C_1 - \Phi_2 + C_2 \leq T_1$, *where* $l_{max}^{\{2\}} = max\{\Phi_1 + C_1 - \Phi_2 + C_2, C_1 + 2C_2 - 1\}$.

Proof. For a 2-task set Γ_2, Γ_1=$\{\tau_1\}$ is schedulable due to $C_1 \leq D_1 \leq T_1$. Inequality $T_1 \geq C_1 + C_2$ implies that there is one (and only one) 2-permissibility interval of τ_2, i.e., $[t_1, t_2)$, in the time interval $[\Phi_1,\ \Phi_1{+}T_1)$, where t_1=$\Phi_1{+}C_1$ and t_2=$\Phi_1{+}T_1$, and the starting point of the next 2-permissibility interval is t_3=$\Phi_1{+}T_1{+}C_1$, and thus $l_{max}^{\{2\}}$=$max\{t_1 - \Phi_2 + C_2,\ t_3 - t_2 + 2C_2 - 1\}$=$max\{\Phi_1 + C_1 - \Phi_2 + C_2,\ C_1 + 2C_2 - 1\}$. The rest parts of the proof follow the same reasoning of the proof for Theorem 3. The lemma is then proved. ■

Since $l_{max}^{\{2\}} \geq C_1{+}C_2$, the necessary schedulability test of $min\{T_1,\ T_2\}\geq C_1{+}C_2$ in [16] is just a corollary of Lemma 3.

We note that due to the repetition of the schedule of Γ_{k-1}, two time intervals $[\Phi_{min}^{\{k-1\}} + LCM_{k-1}, t_{2j_k+1})$ and $[\Phi_{min}^{\{k-1\}}, t_1)$ have the same length. Therefore, $t_{2j_k+1} - t_{2j_k} = (t_{2j_k+1} - (\Phi_{min}^{\{k-1\}} + LCM_{k-1})) + ((\Phi_{min}^{\{k-1\}} + LCM_{k-1}) - t_{2j_k}) = (t_1 - \Phi_{min}^{\{k-1\}}) + ((\Phi_{min}^{\{k-1\}} + LCM_{k-1}) - t_{2j_k}) = t_1 + LCM_{k-1} - t_{2j_k}$, and thus $l_{max}^{\{k\}}$ is also equal to $max_{1\leq i\leq j_k-1}\{t_1 - \Phi_k + C_k, t_{2i+1} - t_{2i} + 2C_k - 1,\ t_1 + LCM_{k-1} - t_{2j_k} + 2C_k - 1\}$, which has relationship only with the k-permissibility intervals of τ_k in the interval $[\Phi_{min}^{\{k-1\}}, \Phi_{min}^{\{k-1\}} + LCM_{k-1})$.

Moreover, after simulating all tasks in Γ_{k-1} in the time interval $[\Phi_{min}^{\{k-1\}}, \Phi_{min}^{\{k-1\}} + LCM_{k-1})$ we will have $2j_k$ boundary values of j_k k-permissibility intervals and it needs only to test all LCM_k/T_k jobs of τ_k in this interval. At this time for testing the j^{th} job of τ_k it needs only to make a comparison between Loc_k^j and relative boundary values, where

$$Loc_k^j = \begin{cases} mod_k^j & \text{if } j = 1 \text{ or } mod_k^j \geq \Phi_{min}^{\{k-1\}} \ , \\ mod_k^j + LCM_{k-1} & \text{if } mod_k^j < \Phi_{min}^{\{k-1\}} \ . \end{cases}$$

where $mod_k^j = \Phi_k + (j-1)T_k \ mod \ LCM_{k-1}$, and there is no need to test each job of τ_k in a time-tick way (i.e., simulating the schedule from time 0 one time tick after another). Notice also that $LCM_k/T_k \leq LCM_{k-1}$. Hence, it needs only to search in this interval, with a length of LCM_{k-1}, to check whether the task τ_k is schedulable, as Lemma 4 states.

Lemma 4. *Consider Γ_n satisfying the basic phasing condition and the initial busy condition for a GPA, then, for each k, $2 \leq k \leq n$, it is sufficient to search in the interval $[\Phi_{min}^{\{k-1\}}, \Phi_{min}^{\{k-1\}} + LCM_{k-1})$ for checking whether task τ_k is schedulable.*

Proof. It follows from the above discussion. ■

This lemma tells us that there is no need to simulate the corresponding schedule at the time line beyond the time point $\Phi_{min}^{\{k-1\}} + LCM_{k-1}$. In fact, based on Lemma 4, a pseudo-polynomial time algorithm for computing the LRT^{GPA} (*the largest response time of a task set for a GPA*) is derived. Moreover, compared with the previous results in [17],[18], for each k, $2 \leq k \leq n$, when testing the schedulability of task τ_k, we reduce the search length from LCM_k to LCM_{k-1} when $T_k \neq LCM_{k-1}$. Actually, we have Theorem 4.

Theorem 4. *Consider Γ_n satisfying the basic phasing condition and the initial busy condition for a GPA, then, for each k, $2 \leq k \leq n$, the search length, being LCM_{k-1},*

in the simulation-based schedulability test for checking the schedulability of Γ_k *is optimal.*

Proof. For each k, $2 \leq k \leq n$, as shown in Lemma 4, it needs to search with no more than a length of LCM_{k-1} for checking the schedulability of task τ_k. On the other hand, it indeed requires to search with at least a length of LCM_{k-1} for testing the schedulability of task τ_k in a simulation-based way, since, in general, all possible execution states of k-1 higher priority tasks are embodied in a feasibility interval of Γ_{k-1} with a length of LCM_{k-1}. In other words, if a testing length is no more than LCM_{k-2} when intending to test the schedulability of Γ_k, certain execution states of Γ_{k-1} will be missing and thus it is unable to test the schedulability of task τ_k, let alone the optimality. ■

It should be pointed out that Theorem Theorem 2 and Theorem 3 provide a recursive method which can be applied step-by-step from the highest priority task to each of the lower-priority ones in the n-task set. It does not, however, imply the two theorems can only be used recursively. In fact, when using traditional time-tick-based simulation method, for the sufficient test, we can use no more than $5(n\text{-}1)$ variables to record the related boundary values of k-permissibility intervals of task τ_k in the time interval $[\Phi_{min}^{\{k-1\}}, \Phi_{min}^{\{k-1\}}+LCM_{k-1})$, i.e., t_1, t'_{2i+1}, t'_{2i}, $max\{t'_{2i+1} - t'_{2i}\}$, and the values of $l_{max}^{\{k\}}$, calculate the $l_{max}^{\{k\}}$ when the simulated schedule reaches at the time $\Phi_{min}^{\{k-1\}}+LCM_{k-1}$, and then check the schedulability of Γ_k according to the theorems. Thus, for example, for a system with n synchronously released tasks having the same period T, the length of the interval used by the sufficient test is still T, no need to test

n-1 times a time interval of length T. At this time, there is only one k-permissibility interval of τ_k in the interval [0, T), and we need no more than 2(n-1)+1 variables for recording the related values of k-permissibility intervals and the values of $l_{max}^{\{k\}}$, $2 \leq k \leq n$.

In addition, based on previous results in [21], we derive a new necessary test for P-FRP tasks by Lemma 5.

Lemma 5. *For a periodic task set under a fixed priority assignment, a necessary schedulability test in the P-FRP model is that it is schedulable for the same priority assignment in the classic model.*

Proof. As stated in [21], (1) if a task set is schedulable for some fixed priority assignment in the classic model, then it is not necessarily schedulable for the same priority assignment in the P-FRP model; and (2) if a task set is schedulable for some fixed priority assignment in the P-FRP model, then it will also be schedulable for the same priority assignment in the classic model, the lemma then follows. ■

By Lemma 5, negative propositions in the classic model, such as that, for periodic task sets with constrained deadlines and scheduled in an asynchronous way, the DM scheduling is not optimal, will remain negative in the P-FRP model.

It is seen that the maximum search length has been reduced from LCM_n to LCM_{n-1} when $T_n \neq LCM_{n-1}$ for testing the schedulability of Γ_n which satisfies the basic phasing condition and the initial busy condition (including the scenario of synchronous start) for a GPA, and it is optimal in terms of the search length.

However, it is still not a polynomial-time schedulability test. In fact, we conjecture that, in the P-FRP model under fixed priority scheduling, the problem of deciding whether a given periodic n-task set is schedulable for a GPA on m processors is NP-hard for each $m \geq 1$, i.e., it is unlikely that there will be a polynomial-time exact schedulability test unless P=NP; otherwise, for all those periodic n-task sets which are deemed schedulable in both the P-FRP and classic models, denoted by Γ_{ds}, we can use the polynomial-time exact schedulability test in the P-FRP model under fixed priority scheduling to check the schedulability of a given task set in Γ_{ds}, then, by Lemma 5, it implies that, in the classic model under fixed priority scheduling, we are able to check the schedulability of any task set in Γ_{ds} in polynomial-time, and this will contradict previous results in the general case (as stated in Sec.5.4). Note that the conjecture does not imply that there will be no polynomial time exact schedulability test in the classic model for task sets with particular parameters. Till now there is no formal proof for the hardness of the above decision problem in the P-FRP model.

Finally, it should be emphasized that the above sufficient schedulability test is applicable for periodic P-FRP n-task sets satisfying both the basic phasing condition and the initial busy condition under fixed priority scheduling and a GPA. For the scenarios of dealing with arbitrary phasing, finding the phasing that leads to the worst-case behavior, and optimizing priority assignment are left for future work.

5.3 Experimental Results

Besides providing theoretical proofs in Sec.5.2, we present experimental results in this section for showing the efficiency improvement and the effectiveness of the new suffcient schedulability test. The efficiency improvement refers to improvements on the search length and the testing time, while the effectiveness means that, by the new sufficient test, a given task set satisfying the sufficient condition is deemed schedulable and a given unschedulable task set is also deemed unschedulable. In fact, simply comparing the LCM_{n-1} with the LCM_n of the same n-task set for the same given priority order provides an evidence of the efficiency improvement on the search length of the new test.

Experiment setup: We generate six groups of task sets, from 3-task to 8-task sets, by varying tasks' parameters and task set size. An n-task set is generated as follows. Randomly generate n positive integer values within a given range, sort them in an increasing order, and set them in sequence to the period T_i, $1 \leq i \leq n$; Use the UUniFast algorithm [30] to generate n utilization factors U_i in a decreasing order, $1 \leq i \leq n$, such that the total utilization $U = \sum_{i=1}^{n} U_i$ equal to a given value, e.g., 0.5 or 0.4 for a 3-task set and a 4-task set respectively; Then let the computation time C_i to be $T_i \times U_i$. Each phasing Φ_i is set to 0 or 1 for satisfying the initial busy condition. RM scheduling is employed. Two different period ranges are used, aiming to validate the sufficient test with different scenarios, the range of periods from 61 to 301 for the groups of 3-, 4- and 5-task sets and the range of periods from 51 to 79 for the groups of 6-, 7- and 8-task sets. Each of the task sets in each group is likely

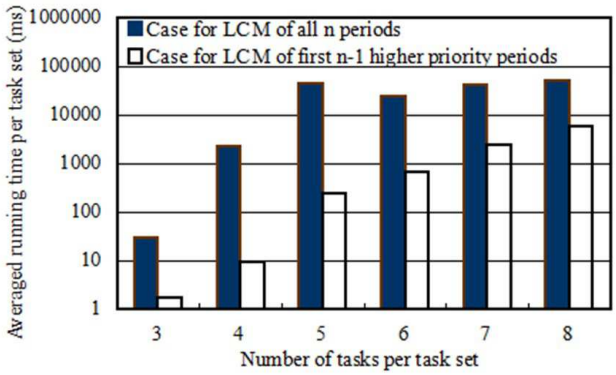


Figure 5.7: Comparison between cases for different testing interval lengths.

to be unique since they are generated independently. The experiment is conducted on a PC with CPU i7-3770 3.4GHz and 16GB memory.

Fig.5.7 shows ours improvement by a comparison of averaged testing time per task set between cases for different search lengths. The testing time is averaged over 300 *schedulable* task sets satisfying the sufficient condition for each group. Groups with different period ranges are put in the same figure. The Y-axis is shown with a logarithmic scale due to high values of testing time of the case for the search length of LCM_n.

We can see that for different groups of task sets all have efficiency improvement. In average the testing time per set in groups of 3- through 7-task sets has been improved for at least one order of magnitude, especially for groups of 4- and 5-task sets having more than two order of magnitudes. For the group of 8-task sets, the testing time per set by the test with a search length of LCM_n is as more than 9 times long as that using the new test, which implies that in average for this group of 8-task sets the new test with an LCM_{n-1} search length needs only one-ninth time of

that of an LCM_n. In addition, by comparing groups of 3-, 4-, and 5-task sets with that of 6-, 7-, and 8-task sets, we can see that more improvements are obtained for task sets with larger range of periods.

On the other hand, in the experiments we generate 60000 6-, 7-, or 8-task sets (20000 per group) and pick out totally 10168 task sets that are all in fact unschedulable (confirmed by the test with a search length of LCM_n) and they are all deemed unschedulable by the new test. At this time, in average the testing time per task set for each group by the new test is between 14.8ms and 113.4ms, having been improved for at least one order of magnitude because of less search length.

To sum up, the new sufficient test is of not only effectiveness but also having efficiency improvement in average both on the search length and on the testing time, compared with the test with a search length of LCM_n.

5.4 Related Research Work

To date, there is an immense body of research on the classic model, and excellent surveys can be found in [43],[57],[6], [117],[13],[55]. In contrast, research on the P-FRP model is just beginning. In this section, we give a brief review on some results on the classic and P-FRP models under fixed priority scheduling on uni-processor systems that are pertinent to the theme of this chapter.

In the classic model under fixed priority scheduling, as to the exact schedulability tests for n-task sets, there are two kinds of generally equivalent computation-based

methods which are based on either processor demands (or workload) [89],[33], or on the response time analysis (RTA) [5],[7]. Using the *workload* $W_i(t)=\sum_{j=1}^{i}\lceil t/T_j\rceil \cdot C_j$ given in [89], which is the cumulative demand on the processor incurred by jobs of equal or higher priority than the task τ_i over [0, t] when time 0 is a *critical instant*, a periodic task set with implicit deadlines is schedulable for any phasing under fixed priority scheduling if and only if $max_{1\leq j\leq n} min_{t\in s_i}\{W_i(t)/t\}\leq 1$, where $s_i=\{k \cdot T_j\colon\ j=1,...,i;\ k=1,...,\lfloor T_i/T_j\rfloor\}$.This expression has been extended for coping with arbitrary deadlines [90], and for accommodating asynchronous systems [29]. A more efficient *Hyperplanes* test was also proposed [33], which can be tuned using a parameter to balance acceptance ratio versus complexity. An important concept, termed the *busy period*, for schedulability analysis has been introduced in [90].

Using the *RTA-based method* proposed in [5],[7], a periodic task set with constrained deadlines is schedulable under DM scheduling if and only if $R_i \leq D_i$ for each task, where the R_i can be computed using the iterative formula, i.e.,$R_i^{(k)}=C_i+\sum_{1\leq j<i}\lceil R_i^{(k-1)}/T_j\rceil \cdot C_j$, until $R_i^{(k)} = R_i^{(k-1)}$ and $R_i^{(k)} \leq D_i$, where the initial value of $R_i^{(0)}$ is set to C_i, and only a subset of points in the interval [0, D_i] need to be examined for feasibility. Other initial values such as $\sum_{1\leq j\leq i} C_j$ or $R_{i-1}+C_i$ can also be used [6], and further effective initial values are explored for faster convergence and efficiency improvement [96],[30]. Other exact tests such as that in [108] deal with cases under a condition of certain period ratios. In addition, response time upper bounds can also be used as sufficient tests [31],[56]. Moreover, the standard RTA algorithm has been extended to cater for situations of arbitrary deadlines and release jitters without restrictions on the task periods [121]. There are also other extensions

such as accounting for blocking time.

In general, however, in the classic model, the problem of deciding whether any given periodic task set is schedulable under fixed priority scheduling, for both synchronous and asynchronous systems, is NP-hard, unless P=NP [92],[91],[12], [35], without resource (or speed) augmentations. Approximate schedulability tests with tunable complexity have also been proposed [32],[35]. The non-polynomial behavior is due to the feasibility test itself and not the priority assignment [8], since only a polynomial number of priority assignments, at most being $O(n^2)$, needs to be checked for both synchronous and asynchronous n-task sets [7],[8]. In addition, it has been proved that response time computation for periodic tasks with implicit deadlines under RM scheduling is NP-hard [62], and thus approximate techniques are employed [105]. Moreover, for some special tasks such as those with harmonic period length, meaning that the periods of tasks pairwise divide each other, there are exact polynomial-time algorithms for both testing the schedulability and computing the response time [36].

In the P-FRP model, the authors of [17],[18] present the Gap-enumeration [17] and idle-period game board [18] algorithms for computing the actual response time, and then the schedulability of the task set is determined. However, the algorithms need to test within the interval consisting of the LCM of all task periods; On the other hand, both algorithms have some slack in performance due to higher search costs or Red-Black balance-tree maintaining costs [18].

As to the sufficient test, the authors of [16] proposed one for P-FRP n-task sets, with a utilization bound being $1/n$ under the restriction of $T_n < T_i$ and $T_i \leq n \cdot T_n$ for

$1 \leq i < n$, making it less useful as n gets larger. In addition, the authors of [112] tried to extend the standard RTA method for coping with the problem of RTA in the P-FRP model. Furthermore, the authors of [123],[124] provided an equivalent equation to that in [112], and considered it as a sufficient test in their studies. However, a special case indicates that the resulting expression does not converge for all situations [17]. This research practice indicates that results derived in the classic model usually do not apply directly to the P-FRP model because of the AR feature and hence the uncertain response time of the P-FRP tasks [39]. Thus, schedulability analysis in the P-FRP model deserves more research effort. To the best of our knowledge, no simulation-based tightly sufficient test for P-FRP tasks for a given priority order is available till work of this chapter.

5.5 Conclusions

In this chapter, we have presented a new simulation-based sufficient schedulability test for periodic P-FRP task sets which satisfy both the basic phasing condition and the initial busy condition for a given priority order, under fixed priority scheduling, covering both implicit and constrained deadlines. The sufficient test, including the optimality of the search length for testing, is validated by both theoretical proofs and experimental results. To the best of our knowledge, this is the first time such simulation-based tightly sufficient test is presented in the P-FRP model, showing the first test that has the maximum search length being the LCM of periods of the first n-1 higher priority tasks. One significance of this optimal search length is that,

when LCM_n is (much) larger than LCM_{n-1}, previous tests constructing a schedule of length at least LCM_n may be infeasible while the above test may be feasible. Though the new test is not efficient enough, it is the current best result of simulation-based schedulability tests and its search length cannot be improved further.

The schedulability and response time analysis problems under non-fixed priority scheduling policies as well as those dealing with arbitrary deadlines and finding optimal priority assignments in the P-FRP model are future work. Considering the blocking time, sporadic tasks, robustness during overloads, resource sharing, and task interactions also deserve further study. The scheduling theory for the P-FRP model is far from complete and must be fully addressed for it to be practically used in the implementation of complex real-time control systems.

Chapter 6

Deferred Start

In real-time systems, Functional Reactive Programming (FRP) is already playing an important role and will potentially be more so in the future. Priority-based (preemptive) FRP (P-FRP), a variant of FRP with more real-time characteristics, demands more research in its scheduling and timing analysis. Its Abort-and-Restart (AR) nature indicates that reducing preemptions can be critical for improving system performance. In this chapter, we present a new non-work-conserving scheduling model, P-FRP Deferred Start (DS) model, to reduce certain preemptions for task sets with fixed priorities. We analyze some properties and feasibility problem of the P-FRP DS model. We provide a prototype algorithm to verify our analysis. The algorithm has polynomial-time complexity in terms of the total number of jobs

*Work of this chapter was published as: Xingliang Zou, Albert M. K. Cheng and Yu Jiang. A Non-Work-Conserving Model for P-FRP Fixed Priority Scheduling. 13th IEEE International Conference on Embedded Software and Systems in Chengdu, 2016.

within a given time interval. Experiments show the domination on schedulability rate and task response time to those of the P-FRP AR model. The schedulability rate increases up to 243.1% and the task response time saving is up to 24.11% for 10-task sets in our experiments.

6.1 Introduction

In real-time systems, the correctness of a program is measured by its logical output as well as its ability to complete within certain time limits. FRP (Functional Reactive Programming) [122] is demonstrated to be effective in modeling and building real-time reactive systems such as graphics, robotic and vision applications. However, a significant feature of real-time systems, priority, is not considered in FRP. To address this problem, the P-FRP (Priority-based FPR) [84] model has been proposed as a variant of the FRP model. P-FRP maintains both the type-safety and the state-less execution paradigm of FRP, and supports priorities assigned to different tasks while not requiring the use of synchronization mechanisms between tasks in the system. It has the potential to transform the building of more and more complicated Cyber Physical Systems (CPSs). Christoffersen and Cheng [46] presented an impact of P-FRP in building controllers in automobile anti-lock brake systems.

In classic preemptive scheduling [94], a preempted task pauses execution, and resumes from where it paused (Fig.6.1(a)) when it is given CPU time again. In order to maintain the state-less paradigm of the FRP model, unlike the classic preemptive model, P-FRP uses an Abort-and-Restart (AR, Fig.6.1(b)) semantics where if a

lower priority task is interrupted by a higher priority task, it has to restart from the beginning when it is resumed. Thus the P-FRP model is also called the *P-FRP AR* model, or simply shortened as the *AR* model. This provides a scheduling and programming model where response times of different tasks can be tweaked by the programmer without affecting the semantic soundness of the program.

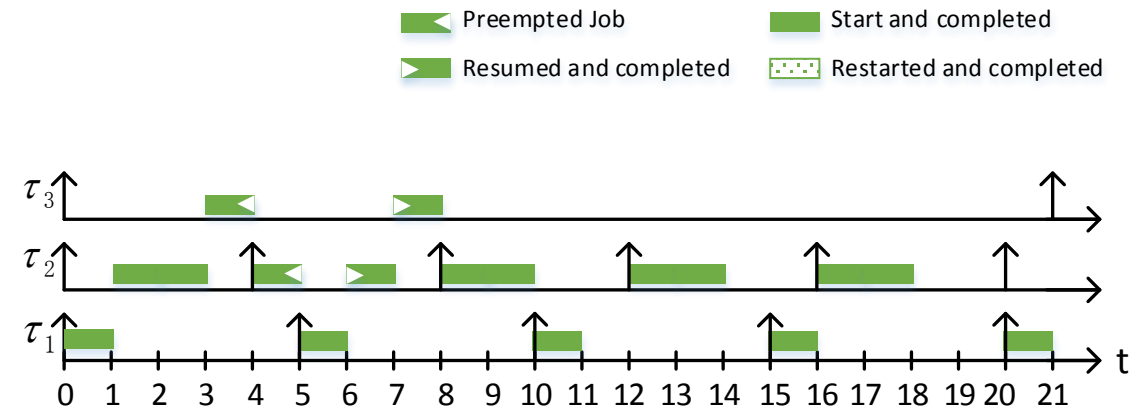
6.1.1 Motivations and Contributions

It is obvious that the restarting in the AR model wastes the executed time (we call it the *abort cost*) and therefore the schedulability would benefit from reducing the number of preemptions (aborts). Our contributions in this chapter include:

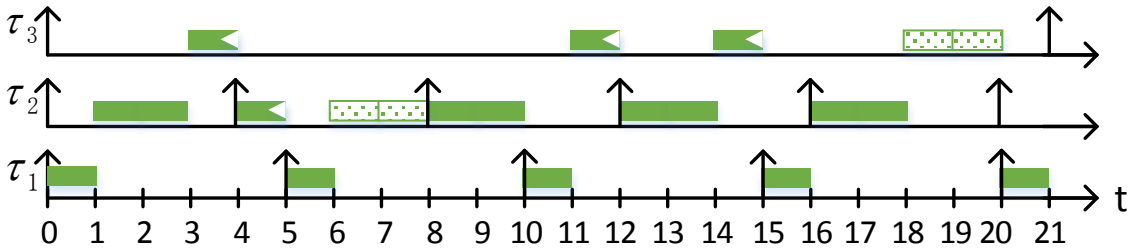
1) We propose a new model, the *P-FRP Deferred Start* (*DS*) model, or shortened as the *DS* model, for P-FRP systems. The DS model (see Fig.6.1(c)) reduces the number of preemptions by postponing the starting of a job. It is a non-work-conserving model and applies to task sets with any given priority assignments. The DS model has equivalent or better schedulability and tasks' response time performance compared to that of the AR model, which is not true in any other known non-work-conserving models.

2) We present the schedulability and response time analysis for the DS model. Experiments and analysis indicate the superior schedulability rate and shorter response time in the DS model compared to the AR model.

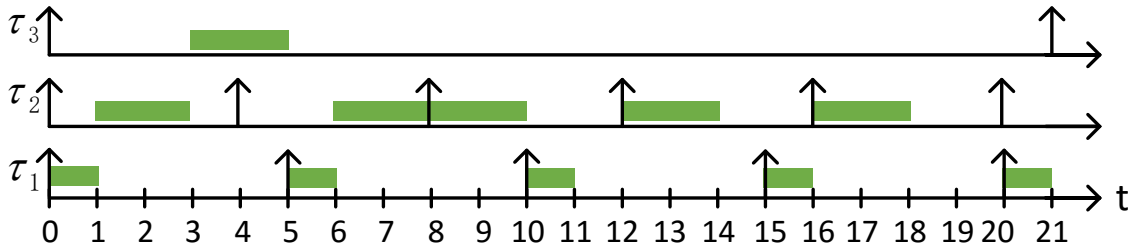
3) We present and implement a prototype algorithm for the exact schedulability test in the DS model. The algorithm has polynomial-time complexity in terms of the



(a) The classic preemptive model



(b) The P-FRP AR model



(c) The P-FRP DS model

Figure 6.1: Schedules of synchronously released fixed priority task set ($C_1 = 1$, $C_2 = 2$, $C_3 = 2$, $T_1 = 5$, $T_2 = 4$, $T_3 = 21$, implicit deadlines).

total number of jobs within a given time interval. The algorithm and implementation aim to improve the existing algorithms in P-FRP task scheduling.

6.1.2 Organization

The rest of the chapter is organized as follows. Basic concepts and notations used in this chapter can be found in Chapter.2. Section 6.2 gives a brief overview on related works. Section 6.3 presents the DS model and its some properties. Section 6.4 describes our algorithm for the exact schedulability test in the DS model. Section 6.5 provides the experiments and results analysis, and Section 6.6 concludes this chapter.

6.2 Related Work

In the classic preemptive model using RM (Rate Monotonic) scheduling for periodic tasks with deadlines equal to periods [94], an n-task set is schedulable if the first instance of each task can finish within its deadline when these tasks are released at the same time (*synchronous release*, or *critical instant*). RM is optimal among other existing fixed priority scheduling algorithms for synchronously released periodic task sets [94]. When deadline is less than period, Deadline Monotonic (DM) uses deadlines instead of periods and is similar to RM [38]. For arbitrary deadline task-sets with potentially some tasks having deadlines greater than periods, neither the RM nor the DM schemes can be used [90]. There are extensive analysis of schedulability for

the tasks with given priority assignments under the classic preemptive model [38], [57] in decades.

However, these analyses do not apply directly to the P-FRP model due to the abort-and-restart semantics of P-FRP systems. Belwal and Cheng [20] have shown that RM is not optimal in the P-FRP AR model with synchronous release and it is even unknown if there exists an optimal one other than exhaustive tests over all possible priority assignments. Belwal and Cheng [16] presented a utilization-based analysis showing that the current schedulability condition only holds true with the utilization bound of $1/n$ under certain restrictions on periods and release scenarios. Jiang *et al.* [82], [81] presented their feasibility intervals and schedulability analysis for the AR model of the P-FRP systems. Wong *et al.* [123] conducted research on other priority assignment algorithms: Utilisation Monotonic (UM), Execution-time Monotonic (EM), and a combination of UM and EM, Execution-time-toward-Utilisation Monotonic (EUM). By comparing with the Exhaustive Search (ES) schema, they confirmed that none of RM, DM, EM, or EUM is optimal in the AR model. The reality that there is so far no optimal priority assignment algorithm for the P-FRP AR model is one of the motivations of the work in this chapter to study alternative models of P-FRP systems.

Alternatives of the AR model are researched too. Real-time Java is designed to allow programmers to develop a real-time application using the Java language. In [98], Manson *et al.* introduced the example of Preemptible Atomic Regions (PAR) for real-time Java. It is a new concurrency control abstraction for real-time systems. The basic notions of the AR model and the PAR model are similar but PAR makes

a log of the shared resource and then the state of the resource will be rolled back if the task is preempted. Wong *et al.* [125] proposed the Deferred Abort (DA) model where a task is divided into two regions: the first region is AR and the second region is non-preemptive and non-abortable. The motivation of the DA model is to reduce the number of preemptions by eliminating some preemptions. The difference between AR and DA is that AR has no non-preemptive region and a higher priority task preempts a running lower priority task instantly when it is released. Notice that the deferred preemption has been studied in the classic preemptive model, such as [37], [53], [42], and the deferred abort can be viewed as an application of the deferred preemption in the P-FRP model. The concepts of priority level-i active period, and Δ-critical instant were also introduced. Preliminary research of the DS model was presented in [127].

The most effective way to reduce preemption cost is to disable preemptions completely, which means a non-preemptive scheduling (NPS). In most NPS, however, a task can experience blocking from other tasks. A careful arrangement of task execution can avoid the blocking, while it most likely works in a non-work-conserving way. NPS has received less attention compared to preemptive scheduling. The schedulability conditions in NPS are NP-hard in the strong sense as Jeffay *et al.* [80] shown, and in the same paper, they also presented an exact (necessary and sufficient) test for schedulability of non-preemptive EDF (npEDF) for periodic tasks with arbitrary release offsets. A comprehensive schedulability analysis of non-preemptive systems was performed by George *et al.* [66]. Sufficient conditions for the schedulability of periodic (and sporadic) tasks scheduled by non-preemptive rate-monotonic (npRM)

have been studied [66], [3], [109]. Clairvoyant EDF (cEDF) [63] and group-based EDF (gEDF) [93] are two heuristic algorithms for this NP-hard problem.

Since the problem is NP-hard, special cases were studied, such as harmonic task sets [59], [44], [103]. It should be stated that in [103] the Precautious-RM algorithm was proposed and it was an O(1) fixed priority based algorithm which can be used in many more cases [102], [104], for example, that Nasri and Fohler [102] presented Efficient Precautious-RM (EP-RM) as non-preemptive scheduling algorithms under RM (rate-monotonic) priority assignment for less restricted harmonic task sets where the periods of tasks are an integer multiple of the shortest task period. Baruah and Chakraborty [11] analyzed the schedulability of the non-preemptive recurring task model. Buttazzo and Cervin [40] used the non-preemptive task model to reduce jitter. Preemption normally works well to meet timing requirement in real-time system design; however, in many cases, a fully preemptive scheduler produces many unnecessary preemptions. To reduce the runtime overhead due to preemptions and still preserve the schedulability of the task set, limited preemption has been investigated [14], [42].

6.3 Deferred Start of P-FRP

We introduce the DS model for the P-FRP systems in this section. We first give the definition and several properties of the DS model, then present feasibility interval analysis for this new model.

6.3.1 The Model

We analyze further the classic preemptive model, the classic non-preemptive model, the P-FRP AR model, and the P-FRP DA model, for comparison purpose; then we give the definition of the DS model for P-FRP tasks.

1) In the classic preemptive model, when a task is running and a higher priority task releases and takes the control of processor, the lower priority task pauses execution. And when the interrupted task gets again the control of processor, it resumes from the point when it was paused; the already-executed part of the task remains effective.

2) In the classic non-preemptive model, once a task, whatever its priority, starts executing, no task can interrupt it. Each successfully scheduled task thus occupies a continuous block of time at the length of the task's execution time. Non-preemptive scheduling brings a question that is called priority inversion. When priority inversion happens, the tasks with higher priorities have to wait for the completion of the tasks with lower priorities.

3) In the P-FRP AR model, a newly released higher priority task interrupts the running task instantly in the same way it does in the classic preemptive model. However, for the interrupted task, it must start from the beginning whenever it is given control of the processor again. No matter how much processor time it has spent in the uncompleted executions and how many times a task is interrupted, there has to be a continuous block of time no shorter than the length of the task's execution time in order to let it finishes execution. This AR model can also be called the

fully-AR model.

4) In the P-FRP DA model, a task contains two regions: preemptive and non-preemptive regions. It works in the preemptive region in the same way the AR model does, and works in the non-preemptive region in the same way the classic non-preemptive model does.

From the above analysis, we can see that there are executed but incomplete execution regions for an individual task in both the classic preemptive model and the P-FRP AR model (including the DA model). The difference is that the executed but incomplete execution regions of the same task in the classic preemptive model are valid towards the complete execution of the task, while those in the P-FRP AR model are not. In fact, an executed but incomplete execution region has no contribution to and effect on the completion of a P-FRP task in the P-FRP model. So there are two steps leading to the DS model from the AR model. In the first step, we use "idle" periods to replace the executed but incomplete execution regions in the AR scheduling. In the second step, without affecting the execution of tasks with higher priorities, we move backward the complete execution of some lower-priority task if it fits in a "idle" period. By repeating these steps for tasks from the second highest-priority task (no need for the highest-priority task) to the lowest-priority task, we get the DS model scheduling of the P-FRP task sets.

In practical algorithms, scheduling in the DS model (DS scheduling) is simplified as follows: a task is scheduled as if it was in the AR model with the condition that it is scheduled to start executing only when there is enough time to finish its execution. As a result, the number of preemptions in DS scheduling decreases to zero, which

makes it seem to be a non-work-conserving non-preemptive scheduling. It differs, however, from existing classic non-preemptive scheduling algorithms in that:

1) the completion time, hence the response time, of a higher-priority task in DS scheduling is not affected by tasks with lower priorities. A task yields the processor only when it will be preempted by higher-priority tasks and that would make the incomplete execution useless according to the P-FRP semantics, and not because of the arrival of lower-priority tasks. It is still a priority-based model but there is no priority inversion as found in the classic non-preemptive model.

2) in the P-FRP AR model, preemptions happen strictly according to the priority settings, thus the response time of a task is optimal in the sense that its response time is not affected by any lower priority task. The DS model follows the same rule regarding to priority settings, so the DS model is not comparable to those classic non-preemptive models that allow a task delay execution while it should start instantly.

3) there is no non-preemptive region in DS scheduling as found in the DA model and some classic non-preemptive models.

By shifting from work-conserving to non-work-conserving, compared to the AR model, in the DS model, (1) there is no waste of execution time; (2) because higher-priority tasks yield the processor if it cannot finish execution in some time interval, a lower-priority task has higher chance to be scheduled successfully and to be scheduled earlier for execution, thus the schedulability rate of task sets increases, and the response times of individual tasks decrease.

Table 6.1 shows where the DS model is in the context of preemptive P-FRP.

Table 6.1: Preemptive P-FRP

	Fully-AR	Non-fully-AR
Work-Conserving	**AR** [84]	**DA**[125]
Non-Work-Conserving	***DS***	

Both the existing AR and DA models are work-conserving, and to the best of our knowledge, the DS model is the first non-work-conserving model proposed for P-FRP systems.

The DS model that is presented in this chapter is based on fully AR, that leaves the variations of the DS model open topics which are neither fully preemptive nor fully non-preemptive.

6.3.2 Properties

We present several properties of the DS model in this subsection.

Theorem 5. *In P-FRP systems, if a task set is schedulable in the AR model, it is also schedulable in the DS model.*

Proof. The proof is conducted using an inductive method for a Γ_n task system,

Base: when $n = 1$, where there is only task τ_1, the proof is obvious that both schedulings are the same, thus if it is schedulable in the AR model then it is schedulable in the DS model.

Inductive hypothesis: For $n = k$, suppose that if Γ_k is schedulable in the AR model, then it is also schedulable in the DS model.

Inductive step: When $n = k + 1$, i.e., $\Gamma_{k+1} = \Gamma_k \cup \{\tau_{k+1}\}$, the aim of this step is to prove that if Γ_{k+1} is schedulable in the AR model, then it is also schedulable in the DS model. Without loss of generality, we can assume the newly added task in Γ_{k+1} is τ_{k+1}; otherwise, we can still assume that the Γ_k consists of the first k highest priority tasks and by the inductive hypothesis, this Γ_k is schedulable in both the AR and the DS models. Notice also that adding a task with the lowest priority does not change the schedulability of the existing task set according to the definition of the AR and the DS models.

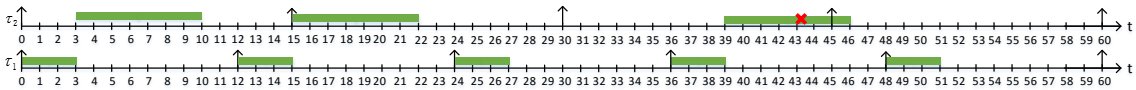
For $\Gamma_{k+1} = \Gamma_k \cup \{\tau_{k+1}\}$ which is schedulable in the AR model, by inductive hypothesis, Γ_k is schedulable in the DS model, and now we need to prove τ_{k+1} is schedulable in the DS model.

It is noted that, for the same task set Γ_{k+1}, for each job of each task, its release time point is the same in both the AR model and the DS model, and a lower priority task has no effect on the schedule of higher priority tasks. For any job $J_{k+1}^j (j \geq 1)$ of the task τ_{k+1}, since τ_{k+1} is schedulable in the AR model, suppose that the release time point of job J_{k+1}^j is t_r and the completion time point is t_c. Given the schedule S_{DS} of Γ_k in the DS model and there are m ($m \geq 0$) unoccupied time intervals in the schedule S_{DS} between the time point t_r and the time point $t_c - C_{k+1}$ that have length no shorter than C_{k+1}. If $m = 0$, the finish time of job J_{k+1}^j in the DS model is the same as that in the AR model. If $m > 0$, job J_{k+1}^j is scheduled successfully at a time point in the DS model ahead of that in the AR model. The theorem is then proved. ■

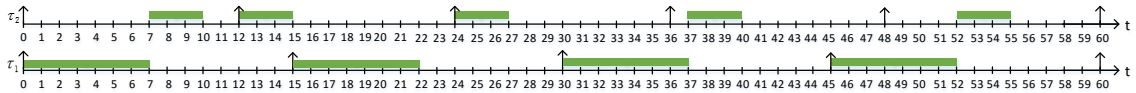
Lemma 6. *In P-FRP systems, if a task set is schedulable in the DS model under*

Table 6.2: An example task set

	Priority	Execution Time C_i	Period T_i
τ_1	1	1	5
τ_2	2	2	4
τ_3	3	2	10



(a) Using RM priority assignment: $\tau_1(3, 12)$, $\tau_2(7, 15)$



(b) Using a non-RM priority assignment: $\tau_1(7, 15)$, $\tau_2(3, 12)$

Figure 6.2: Schedules of synchronously released fixed priority task set ($C_1 = 7, C_2 = 3, T_1 = 15, T_2 = 12$, implicit deadlines) in the DS model.

a given priority assignment, there is no guarantee that it is schedulable in the AR model under the same priority assignment.

Proof. We use an example to show this property.

Tasks in Table 6.2 all have implicit deadlines and release synchronously. Simply referring to the Figs.6.1(b) and (c), it is clear that Γ_3 is schedulable in the DS model, but the first job of τ_3 will miss its deadline in the AR model. ■

Theorem 5 together with Lemma 6 imply that an AR model's schedulabity test must be sufficient but may not be necessary for the DS model.

From the definition and analysis of the DS model, we also have Lemma 7.

Lemma 7. *Compared to the AR model, the DS model reduces the response time of*

a lower priority task without delaying the completion of higher priority tasks, and hence it can decrease the response time of all tasks except the two highest priority ones.

Proof. For a task system Γ_n of $1 \leq n \leq 2$, tasks' response time in the DS model are the same as those in the AR model (see the analysis on τ_1 and τ_2 below).

When $n \geq 3$, according to the scheduling rules of the DS model, we have that:

1) τ_1, the task with the highest priority, is scheduled right at the time when it is released, and its response time is hence exactly its execution time, C_1.

2) After τ_1 is scheduled, τ_2 is selected. For a certain job of τ_2, in the AR model, before it finally finishes execution at time t, it might have been aborted k ($k > 0$) times, hence wasting k time intervals, $[t_1, t_2), [t_3, t_4), ..., [t_{2k+1}, t_{2k+2})$. On the other hand, when it is scheduled in the DS model, its completion time is still t since none of the k time intervals is long enough and the DS scheduler simply skips them. So the response time of τ_2 in both models is the same, except that k time intervals are idle in the DS model. If there is no abort, we apply the same reasoning to τ_3.

3) After τ_2 is scheduled, τ_3 is selected. In the DS model, jobs of τ_3 may be able to use the time intervals which are wasted in the AR model and saved in the DS model after scheduling jobs of τ_2. Even though τ_3 may not be able to benefit in shortening its response time from the saving intervals resulting from scheduling τ_2, it does not suffer lengthening of its response time. Besides, the time intervals that are not used by τ_3 may also be used in scheduling other lower priority tasks.

4) For τ_4 and others, similar reasoning as in 3) applies.

Hence, Lemma 7 is proved. ■

Theorem 5, Lemma 6, and Lemma 7 show that the task set schedulable in the AR model is also schedulable in the DS model, and some of the task sets that are not schedulable in the AR model can be schedulable in the DS model. Also, for a task set schedulable in both the AR and the DS models, its response time in the DS model is not longer than that in the AR model. Thus the DS model dominates the AR model in terms of task schedulability and response time.

6.3.3 Feasibility Interval Analysis

We give a concept first. In the DS model, when there is an available time interval but a task is not able to use it because of its short length, we call the task is *interfere*d by higher-priority tasks.

Lemma 8. *The schedule of* Γ_n *in the DS model has at least* $(n-1)!$ *interfering combinations.*

Proof. Consider first a periodic task set Γ_n whose tasks release only once. Task τ_1 has $n-1$ choices of lower priority tasks to interfere; τ_2 has $n-2$ choices of lower priority tasks to interfere. This continues until τ_{n-1} which has only one choice to interfere. Finally, τ_n has zero choices because there is no lower priority task. In this case, there are $(n-1)!$ interfering combinations. When tasks release more than

once, the number of choices increases, and the number of interfering combinations is therefore at least $(n-1)!$. ■

Lemma 9. *In the DS model for P-FRP systems, the rate-monotonic priority assignment is not an optimal priority assignment with synchronous release of tasks.*

Proof. If we can give a P-FRP task set in the DS model which is not schedulable using the RM priority assignment, but is schedulable by another priority assignment, it is sufficient to prove this lemma.

Consider a 2-task system Γ_2: $C_1 = 7, T_1 = 15, C_2 = 3, T_2 = 12$ that released synchronously at time 0, as shown in Fig.6.2. Using the same method shown in Fig.6.1, in the time interval of [0, 60), the third job of τ_1 finishes at time point 46 which is 1 time unit later than its deadline when using the RM priority assignment (Fig.6.2(a)). After exchange their priority, both tasks are scheduled successfully (Fig.6.2(b)). The lemma is hence proved. ■

In fact, there is still no known priority assignment or release pattern to form the critical instant of the DS model in P-FRP systems as Theorem 6 indicates.

Theorem 6. *To find the critical instant for the DS model in a periodic n-task system, there are exponential number of state should be checked.*

Proof. Lemma 8 shows that there are at least $(n-1)!$ interfering combinations for an n periodic task system, and all of which must be checked for the worst-case to be found. These two properties in isolation and together show that it is an

exponentially increasing number of release patterns to check in order to define the critical instant. ■

Thus, efficient feasibility interval analysis and schedulability test algorithms and implementations for a given release pattern are demanded before finding the WCRT of an n-task system.

One of differences between the DS model and the AR model is that in the DS model a time interval that is unable to accommodate the complete execution of a task τ_i may be able to accommodate the complete execution of a lower priority task τ_k. We then provide a new definition of the *k-permissibility interval in DS* of a task τ_k in the DS model as follows, which is of difference in condition (2) from that in [82].

Definition 3. *For Γ_n under fixed priority scheduling, a* k-permissibility interval in DS *of task τ_k is a time interval $[t_u, t_v)$ such that: (1) the interval length, i.e., $t_v - t_u$, is no less than C_k time units, (2) the interval length $t_v - t_u$ is less than C_i time units if a higher priority task τ_i is awaiting execution and ready to execute before time t_u, and (3) no higher priority task is released within it.*

When a given phasing of Γ_n satisfying $0 \leq \Phi_k < T_k$ for all $k, 1 \leq k \leq n$, we designate the given phasing with a term of the *basic phasing condition* [82]. When Γ_n satisfies the basic phasing condition, each Γ_k $(1 \leq k \leq n)$ also satisfies this condition. The term of the *Initial Busy Condition* remains the same as that in the AR model [82].

Definition 4. *For Γ_n under fixed priority scheduling, the Initial Busy Condition for a given priority assignment refers to the condition: For each $i, 2 \leq i \leq n, min_{1 \leq j < i}\{\Phi_j\} < \Phi_i + C_i$ and $\Phi_i \leq max_{1 \leq j < i}\{\Phi_j + R_{j,1}\}$ where $R_{j,1}$ is the response time of the first job of task τ_j for the same given priority assignment [82].*

In [82], the authors have presented their research on the feasibility interval in P-FRP systems specifically with the AR model. The results are also hold in the DS model since both the DS and AR model are the P-FRP models. Also, from Theorem 5 we know that a sufficient test in the AR model is also a sufficient test in the DS model.

Theorem 7. *[Theorem 3.3 in [82]] Consider a schedulable Γ_n satisfying both the basic phasing condition and the initial busy condition, under fixed priority scheduling, then for each $k, 1 \leq k \leq n$, a feasibility interval of Γ_k is $[t, t + LCM_k)$, where $t \geq \Phi_{min}^{\{k\}}$.*

The above results show the periodicity of P-FRP task scheduling in the DS model. This periodicity makes it possible to check the schedulability of a task set in a finite time interval, and apply it to the infinite releases of the tasks, thus makes $[\Phi_{min}^{\{k\}}, \Phi_{min}^{\{k\}} + LCM_k)$ the first feasibility interval.

In the next section, we present an exact schedulability test for the DS model.

6.4 LList-based Exact Schedulability Test

There is neither mathematical- nor simulation-based either on computation time or memory space for P-FRP DS model. Response Time Analysis- (RTA-) based schedulability test is a practical method in such scenario. Belwal and Cheng presented the gap-enumeration [17] and idle-period game board [18] algorithms for computing the actual response time of P-FRP tasks in the AR model. However, on the one hand, the algorithms need to test within an interval with length of LCM of all task periods; On the other hand, both algorithms have slack in performance due to high search costs or high Red-Black balance-tree maintenance costs [18].

Lv *et al.* [97] presented a linked list- (LList-) and simulation-based exact test method for the classic preemptive model under fixed priority scheduling. The preliminary results have shown that the LList-based exact test is a good candidate in exact response-time tests when task periods span no more than three orders of magnitude. Notice that this method is also applicable to the P-FRP tasks in both the AR and the DS models. In the P-FRP model, though the optimal search length LCM_{n-1} in the exact test is still exponential-time complexity in terms of the number of tasks, it is polynomial-time complexity in terms of the total number of jobs within the test interval [82]. We use the same method in this work.

Also, from the analysis in above section, we have the following lemma:

Lemma 10. *In the DS model, consider* Γ_n *satisfying both the basic phasing condition and the initial busy condition, under fixed priority scheduling, then for each* $k, 1 < k \leq n$, Γ_k *is schedulable in* $[\Phi_{min}^{\{k\}}, \Phi_{min}^{\{k\}} + LCM_k)$ *if* Γ_{k-1} *is schedulable in*

$[\Phi_{min}^{\{k-1\}}, \Phi_{min}^{\{k-1\}} + LCM_{k-1})$ *and* τ_k *is schedulable in* $[\Phi_{min}^{\{k\}}, \Phi_{min}^{\{k\}} + LCM_k)$.

Proof. It naturally follows the proof of Theorem 7. ■

Lemma 10 indicates that Γ_n can be scheduled in a recursive and incremental manner, and it is crucial in improving the performance of the implementations.

6.5 Experiments and Results Analysis

6.5.1 Experiment Setup

The experiments are conducted on a desktop computer with a CPU of i3-4130 3.4GHz, 8 GB memory and *Ubuntu 14.04.3 LTS 64-bit Desktop* operating system. The objective is to deliver comparisons between the AR model and the DS model on the schedulable rate increasing for task sets and the response times decreasing for individual tasks that achieved in the DS model.

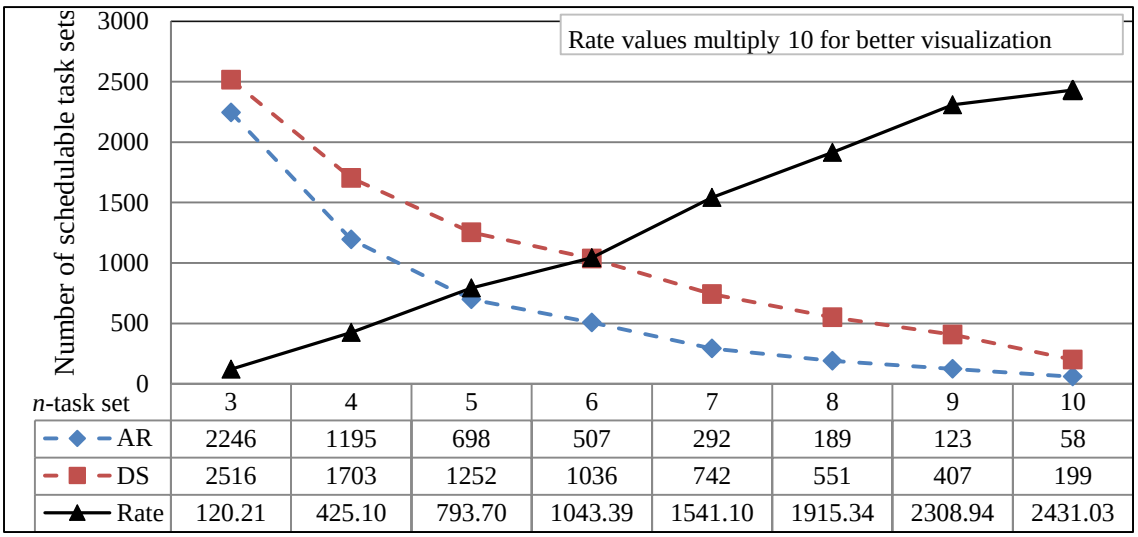
From the discussion above, the DS model scheduling can be viewed as a kind of non-preemptive scheduling, it thus shares the same limitation with non-preemptive scheduling that the utilization factor of an infeasible task set can be close to 0 [104]. Hence the utilization factor itself of task sets in the experiments is not our concern. We choose a utilization factor that is neither very high nor very low. And with this chosen utilization factor, we should have enough number of schedulable task sets as well as that of non-schedulable task sets in the AR model for the purpose of comparison. In the task set generation, we only consider a necessary condition of

$U \leq 1$. Adopting more necessary conditions can increase the quality of task sets in terms of overall schedulable rate. While it is not important to this chapter as the overall schedulable rate does not affect the comparisons.

As we have stated in above sections, the difference between the AR model and the DS model with n-task system Γ_n where $n = 1, 2$ is trivial. So our experiments start from 3-task sets. The task sets we generate have 3, 4, ..., 10 tasks respectively. For each group, we generate 5000 task sets. The periods T_k are randomly generated in range [15, 70] and deadlines are set to the corresponding periods. Note that the period range is not of significance to our comparison objective. Release offsets are randomly set to 0 or 1 satisfying the initial busy condition. The total utilization factor of a task set is set to 0.6. We use the *UUniFast* algorithm [30] to generate n $(n = 3, 4, .., 10)$ utilization factors $U_k (1 \leq k \leq n)$ in a descending order such that the total utilization factor $U = \sum_{k=1}^{n} U_k$ equals to the given value, 0.6, in this experiment. We then shuffle those U_k to a random order for the consideration of generalization. The computation time of each task is computed by $C_k = U_k \times T_k$, and set $C_k = 1$ if it is 0 after this computation.

6.5.2 Acceptance Ratio

Acceptance ratio means the proportion of the number of task sets deemed to be schedulable by a test method to the total number of task sets tested. Numbers of schedulable task sets in our experiments are listed in Fig.6.3. The rows of *AR* and *DS* are the numbers of schedulable task sets in the AR and the DS model



n-task set	3	4	5	6	7	8	9	10
AR	2246	1195	698	507	292	189	123	58
DS	2516	1703	1252	1036	742	551	407	199
Rate	120.21	425.10	793.70	1043.39	1541.10	1915.34	2308.94	2431.03

Figure 6.3: Schedulable Task Set in AR and DS.

respectively. In the third row, *Rate* is calculated by $(DS - AR)/AR \times 100$ (%). For better visualization, we amply the percentage numbers by 10 in the figure.

We can see that the numbers of schedulable task sets in the DS model dominate those in the AR model, and the DS model has 12.0%, 42.5%, 79.3%, 104.34%, 154.1%, 191.5%, 230.9%, 243.1% more schedulable tasks. In this experiment, the improvement of the acceptance ratio increases along with the increasing of the tasks number in a task set. The reason is that with more tasks in a task set, the potentially wasted time increases in the AR model, and hence the acceptance ratio of a task system benefits more from applying the DS model.

Table 6.3: CPU Time Saved by DS

Γ_n	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
CPU_Saving(%)	7.39	6.11	5.98	5.68	5.41	5.19	4.98	4.48

6.5.3 CPU Time Cost

We analyze the CPU time saving in the DS model compared to that in the AR model. For each schedulable task set in both models, we sum up the CPU time assigned to a task in the feasibility interval, $Tick_{AR}$ and $Tick_{DS}$, then let $Per\_Task\_Set\_CPU\_Saving = (Tick_{AR} - Tick_{DS})/Tick_{DS}$. We calculate the average number CPU_Saving of all such tasks and show in Table 6.3.

Table 6.3 shows that the DS model saves from 4.48% up to 7.39% CPU time compared to the AR model. The ratio is not significant, and it is descending with the ascending of the number of tasks in a task set. When comparing this small number and the descending trend to the big number and the ascending trend in the acceptance ratio change in Fig.6.3, it reveals the significance the abort-and-restart wastes time to the P-FRP tasks. It also reveals that wasting of small pieces of time could result in huge schedulability lost. The facts make the DS model a promising alternative for the P-FRP systems.

6.5.4 Response Time

Experimental results confirm that scheduling tasks in the DS model has no task response time increase compared to that in the AR model. We list statistics of the response time changes of task sets in our experiment. From Table 6.4, Γ_3 has 2246

Table 6.4: Task Sets with Response Time Decreased

Γ_n	Number of Schedulable Task Sets	Number of Task Sets Having Response Time Decreased	Ratio (%)
3	2246	348	15.49
4	1195	594	49.71
5	698	542	77.65
6	507	433	85.40
7	291	273	93.81
8	188	183	97.34
9	123	117	95.12
10	58	57	98.28

schedulable task sets in both the AR and the DS model, and 348 (15.49%) of them have response time decreased. As of Γ_{10}, 57 out of 58 (98.28%) have response time decreased. The portion of task sets that have response time decreased ranges from 15.49% to 98.28%, and the number increases as the number of tasks in a task set increases. The reason is that more tasks in a task set leads to more interfering cost for lower-priority tasks in the AR model.

We calculate the average response time decreasing rate for τ_k, shown in Fig.6.4. As expected, there is no response time change to the first two highest-priority tasks. We notice that the lower the task priority is, the higher the response time decreasing rate is. The reason is that with lower priority, a task has a higher chance of being interfered by other tasks. For example, τ_3 of Γ_{10} has 4.11% response time decreasing, and the number monotonically increases to 24.11% for τ_{10}.

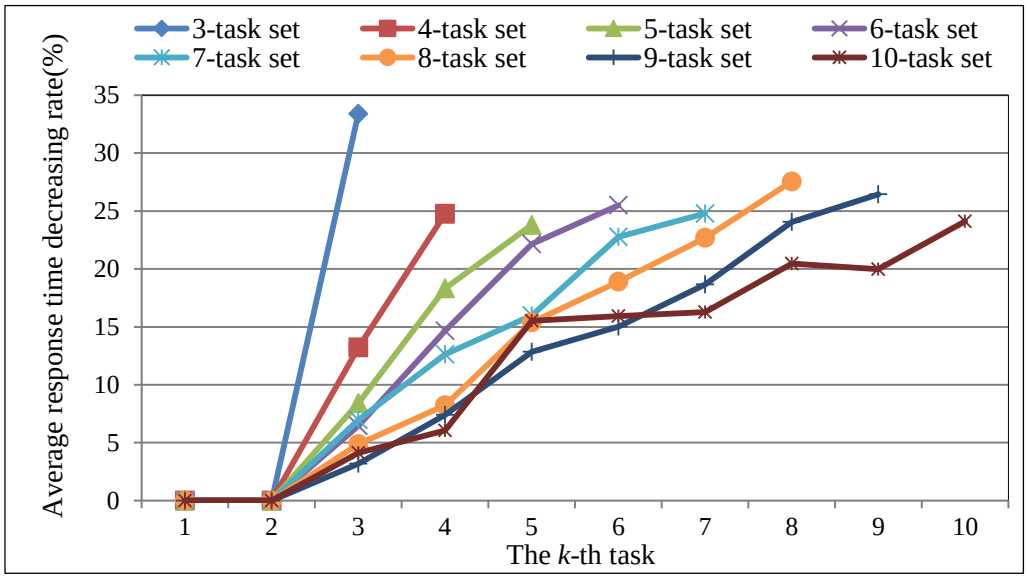


Figure 6.4: Average Response Time Decreasing Rate.

6.6 Conclusion and Future Work

We have proposed our research on Deferred Start (DS), a new alternative model for preemptive P-FRP systems. We stated that there is no known critical instant for P-FRP task sets in this new model. We have presented several properties, and also analyzed the feasibility interval problem of the DS model. Analysis and experiments show that the DS model increases the schedulability rate and decreases the response times of P-FRP tasks compared to the AR model.

Moreover, from the generalized analysis we have presented in this chapter, considering the fact that the DS model scheduling is a type of non-preemptive scheduling, less generalized cases such as task sets with harmonic periods, with wider-distributed periods are worth of further study.

Chapter 7

Multi-Mode Task Model

Functional Reactive Programming (FRP) provides an elegant way to express computation in domains such as interactive animations, robotics, computer vision, user interfaces, and simulation. Priority-based (preemptive) FRP (P-FRP), a variant of FRP with more real-time characteristics, demands research in its scheduling and timing analysis. Different from the classic preemptive model, in a P-FRP system, when a task is preempted, all changes made by the task are discarded and after higher priority tasks complete their execution the preempted task will restart from the beginning (abort-and-restart). P-FRP is thus able to capture changes of the task in time and provides an option other than the classic preemptive model in certain scenarios.

*Work of this chapter was published as: Xingliang Zou, Albert M. K. Cheng, Carlos Rincon and Yu Jiang. Multi-Mode P-FRP Task Scheduling. *Outstanding Paper Award*, 20th IEEE International Symposium on Real-Time Computing, Toronto, Canada, 2017.

In the P-FRP model, previous studies use the largest execution time of a task for all its restarted jobs. In practice, however, when considering the changing/unchanging inputs/outputs of the task or the memory effects such as cache-hit in loading code and data, the restarted jobs likely consume less time than its largest execution time. In this chapter, for the first time we present a multi-mode P-FRP task framework and two particular scenarios for the framework that are able to reflect such effects and then improve the performance of a developing commercial software platform. We show that the multi-mode task P-FRP system has significant schedulability improvements over the original P-FRP model.

7.1 Introduction

Functional Reactive Programming (FRP) is a framework for constructing reactive applications. It exploits many of the characteristics of high-level languages such as abstraction, maintainability, and comprehensibility. Nowadays, FRP is used in many applications such as robotics, animation, and computer vision [79]. Research results have also been presented on the semantics of FRP [50], [77], [119]. Applicability of functional programming for real-time systems is laso explored by the research community [119] [20] [123].

In Schneider Electric [73], a project called *modelergy* is under development and the outcome is an OTS/Simulation-as-a-Service product. It is briefly a cloud platform to host various applications. The platform itself consists of three major modules: the *portal* is responsible for user management; the *guest-agent* is a bridge that connects

all modules, and the *platform* is the core module that hosts the client applications and manages the entire environment. The hosted applications also have intensive activities internally and throughout the underlying host platform. There are many idempotent functions in *modelergy* that are naturally suitable for FRP. FRP also facilitates scaling out of the *modelergy*. We adapted P-FRP [84], a priority-based FRP variant, into *modelergy* to address the priority issue that different activities require different priorities. Though in *modelergy*, there are more than one task sharing the same priority, and there are various types of tasks, in this work we simplify the range of tasks to the subset of periodic and sporadic tasks, and there is only one task at each priority level.

Since P-FRP uses an *abort-and-restart* (AR) semantics where once a lower priority task is interrupted by a higher priority one, it has to restart from the beginning after released higher priority tasks complete, the cost of P-FRP must be addressed in order to improve the resource efficiency such as efficient CPU and memory utilization. Kazemi and Cheng [85] studied the P-FRP task execution time on a scratchpad memory-based platform, and showed that the task execution time changes because of the memory hierarchy. Their work is also one of the motivations for this work.

The contribution of this chapter includes: adopting P-FRP into a real-world product; proposing a multi-mode task framework for P-FRP systems; intensively analyzing and examining the proposed framework with two typical scenarios from different perspectives. The results show that our achievements push P-FRP forward into more practical use.

The rest of the chapter is organized as follows. In Section 7.2, we review the

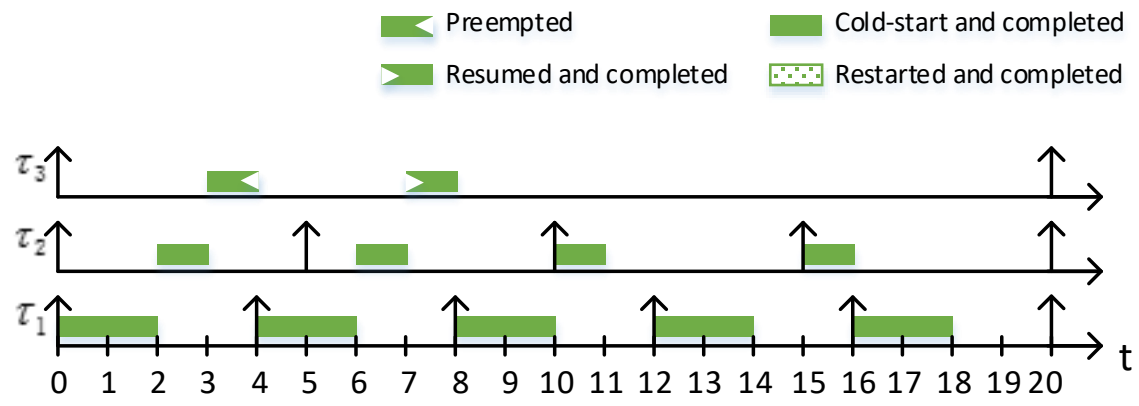
related work of FRP and P-FRP. In Section 7.3, we propose the multi-mode tasks scheduling for P-FRP systems, followed by the experiments and corresponding analysis of the proposed models in Section 7.4. Finally, we conclude the chapter in Section 7.5.

7.2 Related Works

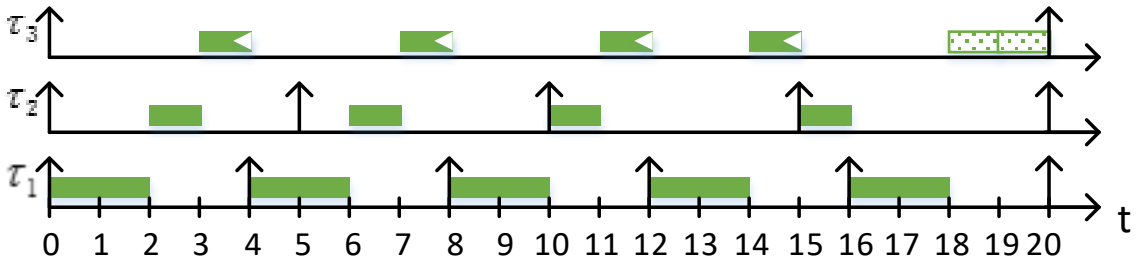
Hu *et al.* [77] reviewed the impact of functional programming, how it has changed the way we construct, verify and even think about programs. Medeiros [99] prensented the demands of the reactivity paradigm. Although FRP is often tied to pure functional languages, adaptations of the paradigm to imperative languages exist. Library [48] in Java and C++ works as a language extension by adding new grammar [58] and as a library based on the standard languages [50]. Czaplicki and Chong [49] presented Elm, an FRP language focusing on the creation of graphical user interfaces (GUIs). Project [74] presented a simple and practical comparison among various FRP libraries.

One of the significant features of FRP is that its stateless execution eliminates the use of synchronization primitives, and thus reduces the complexity of programming. Programmers can intuitively describe safety behaviors of a system and lower the chance of introducing bugs.

In real-time systems, the correctness of a program is measured by its logical output as well as its ability to complete within certain time and with proper prioritization. P-FRP has been proposed in [84] to address priority issue of FRP. P-FRP



(a) The classic model



(b) The P-FRP AR model

Figure 7.1: Classic and P-FRP AR models RM scheduling
($C_1 = 2, C_2 = 1, C_3 = 2, T_1 = D_1 = 4, T_2 = D_2 = 5, T_3 = D_3 = 20$)

maintains both the type-safety and the state-less execution paradigm of FRP, and supports priorities assigned to different tasks while not requiring the use of synchronization mechanisms between tasks in the system. Zou *et al.* [130] presented a detailed survey of current P-FRP scheduling research. Fig.7.1 shows different schedules of an example taskset under classic and P-FRP semantics.

P-FRP is neither classic preemptive nor classic non-preemptive systems. Unlike the classic preemptive systems, there are two job morphs in P-FRP systems because of abort-and-restart: *cold-started job* and *restarted job*. Cold-started job is a job that

is released at the beginning of each period, while a restarted job is the re-invocation of a preempted job. In the original P-FRP systems, cold-started and restarted jobs have the same execution time.

Due to the abort-and-restart execution semantics of P-FRP systems, many research results in the classic preemptive/non-preemptive/limited preemptive models do not apply to the P-FRP model. For the P-FRP AR model, Belwal and Cheng [20], Wong *et al.* [123] presented their researches on priority assignment. Jiang *et al.* [82], [81] presented their feasibility intervals and schedulability analysis for the P-FRP AR model in order to find tighter feasibility intervals.

Because of the high inefficiency of the original P-FRP paradigm, alternative models are also studied. Zhou *et al.* [126] presented their research on WCRT and schedulability analysis for real-time software transactional memory-lazy conflict detection for P-FRP tasks. Wong *et al.* [125] proposed the Deferred Abort model to reduce the number of preemptions in scheduling P-FRP tasks. Zou *et al.* [127] proposed a non-work-conserving Deferred Start model to eliminate preemptions in scheduling P-FRP tasks.

However, none of these researches considers the different execution time of cold-started jobs and restarted jobs. In this chapter, we proposed a new task model that has different modes for *cold start* and *restarted* jobs that uniquely exist in P-FRP systems.

Altmeyer and Navet [1] proposed a declarative modeling and execution framework

for real-time and Cyber Physical Systems(CPS). In CPS, the characteristics of a real-time task may be able to change over time, for example, the computational demand or the resource allocation. Such behavior is referred to as *mode changes* [115], [120], [78] and the importance of mode changes for real-time systems has been presented in many perspectives, such as aircraft control systems, automotive Electronic Control Units (ECU) [54], [114], etc.

Typically, there are two categories of mode changes in hard real-time systems. One is associated to all tasks and the other is associated to individual tasks. In the former, all tasks have to switch to their new parameters and before the new mode of the task system is fully established, no further mode changes are permitted [115], [120]. In the latter, there is no synchronous mode change imposed on all tasks and different tasks may progress through their different execution modes independent of each other. The latter category of mode changes can be found in the generalized multiframe (GMF) model [10], digraph real-time model (DRT) [118], variable rate-dependent behavior (VRB) task model [34], [41], [54], [86], and a generalization form [78] of the sporadic model [100]. These mode changes are usually associated to tasks. To the best of our knowledge, there is no research on the P-FRP model combining both the abort-and-restart semantics and the mode change mechanism.

7.3 Multi-Mode P-FRP Tasks

In this section, we first introduce the system model, some basic concepts and notations used in the chapter, and then propose the multi-mode task model.

7.3.1 Notations

We consider a real-time system with single processor, hierarchical memory components, and rich input/output interfaces. In a P-FRP system, there are n independent, preemptive *multi-mode* tasks $\Gamma_n = \{\tau_1, \tau_2, ..., \tau_n\}$ to form a *taskset* where each task τ_i has M_i ($M_i \geq 1$) execution *modes* for each of its jobs to select. $\tau_{i,j}^m$ is the j-th job of τ_i at mode m. $\tau_{i,j}^m$ has the maximum execution time C_i^m, the minimum inter-arrival time of a sporadic task or period of a periodic task T_i, and the *relative deadline* D_i. The smallest of C_i^ms is denoted by C_i^S, and the largest is C_i^L. We restrict ourselves to either $D_i = T_i$ for the *implicit deadline* or $D_i \leq T_i$ for the *constrained-deadline*. The difference of execution time between two consecutive modes is called *mode gap*. In a fixed priority scheduling, τ_1 has the highest priority.

Φ_i: The *initial release offset* (or *phasing*) of a task τ_i. It is the time, relative to time 0, when the first job is released by τ_i. Let $\Phi_{min}^{\{k\}} = min_{i=1}^{k}\{\Phi_i\}$, and $\Phi_{max}^{\{k\}} = max_{i=1}^{k}\{\Phi_i\}$.

U_i^m: The utilization factor of task τ_i for its job at mode m where $U_i^m = C_i^m/T_i$, and $U_i = max_{m=1}^{M_i} U_i^m = C_i^L/T_i$. The total utilization factor of the taskset is $U = \sum_{i=1}^{n} U_i$, and $U \leq 1$.

LCM_i: The Least Common Multiple (*LCM*, or *hyperperiod*) of the periods of the first i highest priority tasks. $LCM_i = LCM(LCM_{i-1}, T_i)$, $2 \leq i \leq n$, and $LCM_1 = T_1$.

7.3.2 Multi-Mode P-FRP

In the P-FRP AR model, a restarted job has the same execution time as the cold-started job does, hence wastes the executed time of the aborted job (abort cost). It is critical to reduce the abort cost of P-FRP. For example, if the input of a task doesn't change, there is no need to discard all the work that has already been done. We propose a multi-mode task P-FRP model. In this model, a task has multiple modes in terms of multiple execution time options. A light *expert system* is used to select a mode of execution for any cold-started or restarted jobs. The expert system has a knowledge database based on the characteristics of the tasks and the running platforms, and uses deductions to select the right mode for a particular job. In this chapter, the output of the expert system is the chosen execution time of a particular job (cold-started or restarted one). If an expert system always returns the remaining execution time of the preempted job for any restarted jobs, this multi-mode P-FRP task scheduling is the same as the classic full preemptive scheduling. On the other hand, if it always returns the longest execution time, i.e., C_i^L, to any jobs of τ_i, it is reduced to the P-FRP AR model.

In the rest of the section, we present two specific scenarios, Interface-aware and Memory-aware multi-mode P-FRP task models in subsections 7.3.3 and 7.3.4 respectively. The Interface-aware model focuses on the features of a task itself, while the Memory-aware one relates to the entire taskset and the whole system.

7.3.3 Interface-aware Multi-Mode P-FRP

In *modelergy*, there are some tasks responsible for functions such as rendering GUI or retrieving/storing data to the Cassandra database. In such cases, the task execution time is highly dependent on the input in that the execution time is proportional to the amount of changes in the input, and partial change of the input will result in only partial re-execution. For example, a 0-1 matrix A as the input represents a black-white image, and matrix B as the output is the reverse of A. So the task conducts the calculation such that $B[i][j] = 0\ if\ A[i][j] = 1,\ or\ B[i][j] = 1\ if\ A[i][j] = 0$. In this example, the restarted job needs only to continue computing if there is no change in A, or to redo part of the preempted work if it detects some related changes. The execution time of the restarted job hence can be significantly shorter than the full execution time. Since in this type of systems, the execution mode is highly related to the task interface (input and output) to the outside world, we call it *Interface-aware Multi-Mode P-FRP* model. Note that in fixed priority scheduling for this model, the task with the highest priority has no restarted jobs, and thus its jobs always run in the same mode.

We give an exemplary InterA of Interface-aware Multi-Mode P-FRP model below. Here InterA stands for Inter(face)A(ware).

InterA. *InterA refers to an Interface-aware multi-mode P-FRP model such that:*

(1) A task τ_i has M_i modes for each of its jobs to ***switch in sequential*** *from mode* m*=1 to mode $m = M_i$ when a job is started or restarted;*

(2) All C_i^ms are in a non-increasing order, i.e., $C_i^1 \geq C_i^2 \geq ... \geq C_i^{M_i}$; and

$C_i^L = C_i^1$, $C_i^S = C_i^{M_i}$ *in this scenario;*

(3) Both the period T_i *and the relative deadline* D_i *remain the same for each job at different modes;*

(4) The rules for mode changing are as follows:

(i) if task τ_i *has a job running at mode* m $(m < M_i)$ *when it is preempted, and the time unit the job has executed is not less than the mode gap between mode* m *and its successor* $m+1$*, i.e.,* $C_i^m - C_i^{m+1}$*, it proceeds to mode* $m+1$ *when it is restarted; otherwise, it remains at current mode* m*, and we call such scenario* ***mode transit frozen****;*

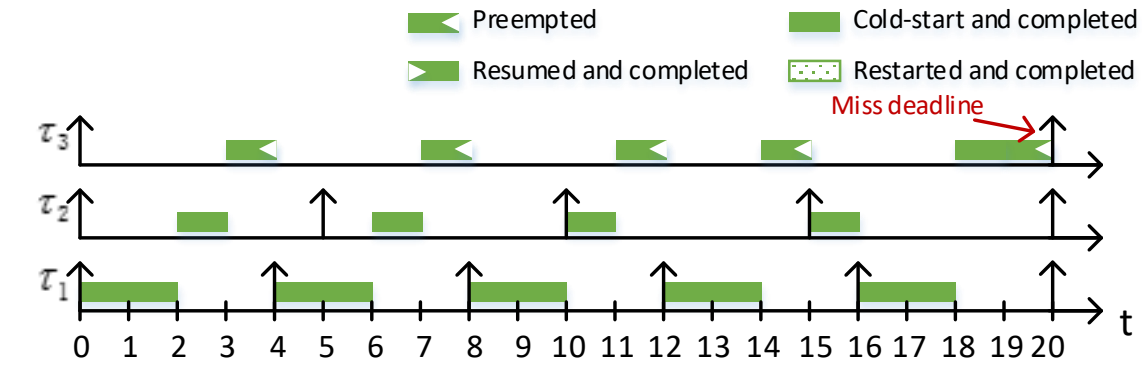
(ii) if current mode of $\tau_{i,j}$ *is the last one (i.e.* $m = M_i$*), it remains at the same mode in all following restarted of* $\tau_{i,j}$*.*

InterA denotes a type of tasks of which the executions are conducted gradually, i.e., with finite amount of checkpoints, and when a task is preempted, it rolls back with certain steps instead of to the beginning as P-FRP AR does. Note that InterA is not all but one type of Interface-aware multi-mode P-FRP systems.

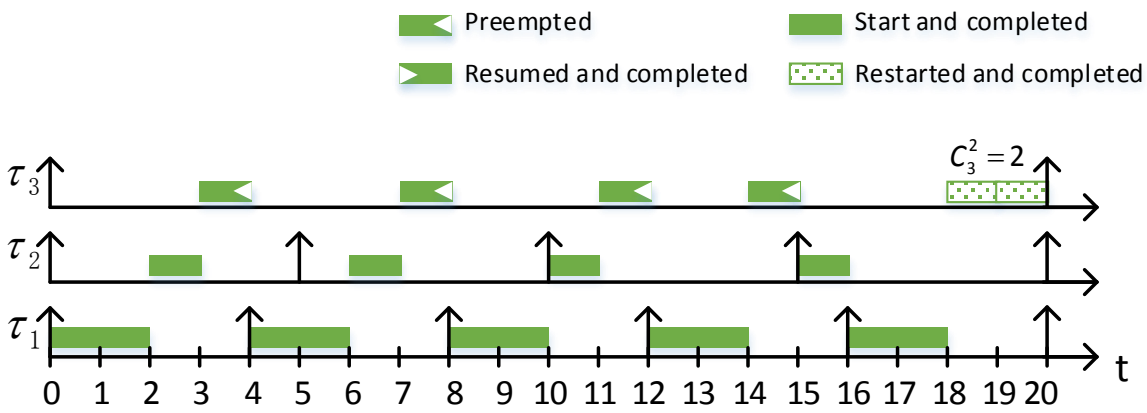
Example. We give scheduling of an InterA taskset in Fig.7.2. We compare the taskset's Rate Monotonic (RM) scheduling between P-FRP AR and InterA models.

In this example, $M_i = 2$ for all tasks, but it is clear that τ_1 runs only in mode *m=1*, while both modes are used for jobs of other tasks: *m=1* for the *cold-started* jobs, and *m=2* is for the *restarted* jobs, and $C_i^1 \geq C_i^2$ for $i = 2, 3$.

The P-FRP AR scheduling uses the mode of *m=1* only. The results are shown in Fig.7.2(a). The first job of task τ_3 is interrupted five times before its deadline



(a) The P-FRP AR model



(b) The P-FRP InterA model

Figure 7.2: P-FRP AR and InterA models RM scheduling
($C_1 = 2; C_2^1 = C_2^2 = 1; C_3^1 = 3, C_3^2 = 2. T_1 = D_1 = 4, T_2 = D_2 = 5, T_3 = D_3 = 20$)

at time point 20, and finally misses its deadline because all restarted jobs require 3 time units. So the taskset is not schedulable by P-FRP AR scheduling.

On the other hand, both modes are used in the multi-mode P-FRP scheduling shown in Fig.7.2(b). Since the restarted job of τ_3 asks for 2 time units in mode *m=2*, and it is scheduled at the time [18, 20) with reduced execution time. Hence the taskset is schedulable by P-FRP InterA scheduling.

From the above description and example, we can see that the Interface-aware multi-mode P-FRP task scheduling reduces the number of preemptions and tasks' response time, and is thus potentially able to schedule more tasks.

7.3.4 Memory-aware Multi-Mode P-FRP

Unlike the Interface-aware P-FRP model which uses information of changing input/output to select the execution mode, there is another type of system in which the internal memory effect dominates the decision making. We denote this P-FRP model as the *Memory-aware P-FRP model*.

We consider a specific scenario. A typical job life cycle includes: (1) Code is loaded from the hard drive and data is loaded from the hard drive or the hierarchical memory (or memory cache); (2) Computation is done by processor(s); and (3) Results are committed to the hierarchical memory.

In the P-FRP AR model, when a periodic/sporadic task is aborted, the work done in phases (2) and (3) is discarded. Due to the existence of hierarchical memory, however, the time spent in phase (1) can be less when a job is restarted; for example,

all or part of the task code is still in the caches and needs not to be read again from the hard drives. Hence the job's execution time varies. The difference can also be caused by the eviction of the code from fast memory (cache, scratchpad, etc.). We consider the Least Recently Used (LRU) replacement algorithm for the fast memory, thus the amount of code remaining in the fast memory is related to the amount of other tasks whose jobs have executed between the execution of two consecutive jobs of a task. We refer to this amount of tasks as *JID*:

Definition 1. *JID (Job Instantiating Distance) of τ_i refers to the amount of tasks, τ_k's of $k \neq i$, that have jobs executed since the last time τ_i has been executed.*

JID denotes the number of tasks executed between two consecutive jobs of task τ_i. So JID is in range of [0, n-1] for Γ_n. The execution time of a job is typically shorter with a smaller JID. Note that either of the two consecutive jobs can be cold-started or restarted, and the tasks corresponding to JID can be either the higher priority ones or the lower priority ones relative to τ_i. On the other hand, in this case, the highest priority tasks can also run in different mode. Fig.7.3 shows some examples. We can see $JID = 1$ for the restarted of job $\tau_{2,1}$ since only τ_1 is executed in time interval [1, 4), and $JID = 2$ for cold-started $\tau_{2,2}$ because τ_3 and τ_1 are executed in time interval [5, 9).

From the definition and Fig.7.3, we can see that there are two compositions of JIDs: JID_1 denotes those JIDs calculated for the restarted jobs, and JID_2 are those calculated for the cold-started jobs. So JID_1 is within a single period of the task, while JID_2 crosses periods because it is calculated using the interval between releasing of a cold-started job and the finishing of its previous job. JID_2 can be

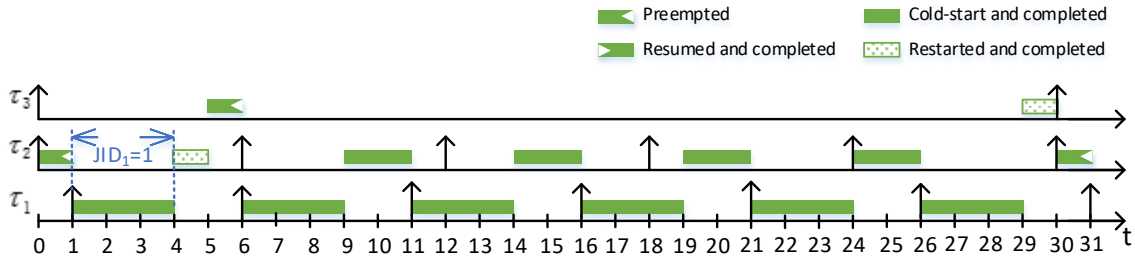
Table 7.1: JIDA Multi-mode Tasks

	T_i	Φ_i	C_i^m		
			C_i^1	C_i^2	C_i^3
τ_1	5	1	2	2	3
τ_2	6	0	1	1	2
τ_3	30	0	1	1	2

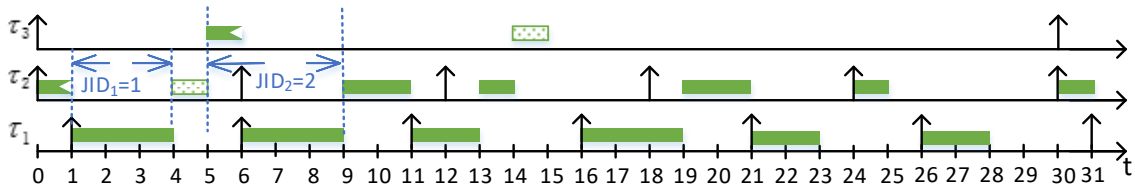
ignored in some scenarios. In *modelergy*, we have some memory-intensive tasks of which each cold-started job flushes the fast memory (sometimes the fast memory means the data having a copy in the local machine), for example, a task that loads data from networking repositories, thus hit or miss of the fast memory only has effects on the restarted job.

For convenience, we denote the scenario that only JID_1 is used as the *local*-JID scheduling (Fig.7.3.(a)), and the other scenario as the *global*-JID scheduling (Fig.7.3(b)) where both JID_1 and JID_2 are used. Compared to the aforementioned memory-intensive tasks, there are also other tasks such as CPU-intensive ones where the fast memory-hit/miss crosses any cold-started or restarted jobs. Global-JID captures these scenarios more accurately. Note that in Fig.7.3, in the scenario of local-JID the highest priority task τ_1 has only one mode since it is never interrupted by other tasks, while in the scenario of global-JID, τ_1 does have multiple modes because the lower priority tasks, τ_2 and τ_3, are executed between execution of two consecutive jobs of τ_1.

This memory effect is not considered in previous studies of P-FRP systems. Therefore, we combine mode changing with the P-FRP model to take into account the memory effect in its schedulability analysis. In this specific JID-based model of



(a) The P-FRP local-JID model



(b) The P-FRP global-JID model

Figure 7.3: P-FRP local- and global- JID models RM scheduling of tasks in Table 7.1

Memory-aware P-FRP system, a job usually needs to reload part instead of all of its code (and data) when there are jobs from the same task having been executed earlier.

We also give an exemplary scenario, JIDA, of the Memory-aware multi-mode task P-FRP system. Here JIDA stands for JID A(ware).

JIDA. *JIDA refers to a JID-based memory-aware multi-mode task P-FRP system such that:*

(1) For any job of τ_i in Γ_n, its running modes are decided using a task and system dependent function, i.e. the mode function *of $m = func\_mode(JID)$; M_i is the number of* ms*, and thus also task and system dependent.*

(2) The execution times C_i^m, $1 \leq m \leq M_i$, are determined by a domain and system dependent function $C_i^m = func\_comp(m)$. C_i^ms are typically in a non-decreasing order, i.e., $C_i^1 \leq C_i^2 \leq ... \leq C_i^{M_i}$; and $C_i^L = C_i^{M_i}$, $C_i^S = C_i^1$;

(3) Both the period T_i and the relative deadline D_i remain the same for each job at different modes.

For an exemplary linear system, we define the *mode functions* such as Equation (7.1) for the global-JID scheduling, and Equation (7.2) for the local-JID scheduling:

$$m = JID + 1 \quad \text{for all jobs} \tag{7.1}$$

$$m = \begin{cases} n & \text{for cold-started jobs} \\ JID + 1 & \text{for restarted jobs} \end{cases} \tag{7.2}$$

For this system, m is in range of [1, n] and $M_i = n$, and we give an example taskset and its scheduling as follows.

Example. We give an example taskset Γ_3 shown in Table 7.1. There are three modes, i.e. M_i=3 for each job, and its local- and global-JID schedulings are shown in Fig.7.3.

In this example, $C_i^1 \leq C_i^2 \leq C_i^3$ for each $1 \leq i \leq 3$, and $C_i^S = C_i^1$, $C_i^L = C_i^3$. Note that $\tau_{1,1}$ starts at time 1, $\tau_{2,1}$ and $\tau_{3,1}$ start at time 0, and all of these three first jobs are in the mode with C_i^L.

The local-JID scheduling is shown in Fig.7.3(a). All cold-start jobs are executed in mode *m=3*. The first job of τ_2, $\tau_{2,1}$, is interrupted by τ_1 at time point 1. When

it is restarted at time point 4, only τ_1 is executed in the time interval [1, 4), hence $JID = 1$ and $m = JID + 1 = 2$ for $\tau_{2,1}$, so the job is executed in the mode with $C_2^2 = 1$ and is completed at time point 5. Note that the taskset is unschedulable in the P-FRP AR model since there is not enough time for $\tau_3, 1$ to complete before its deadline.

The global-JID scheduling is shown in Fig.7.3.(b). Unlike those in the local-JID scenario, only some of cold-started jobs of τ_1 and τ_2 are in mode m=*3*, and the rest of them are in mode m=*2*. Thus τ_3 is able to finish executing at time point 15 which is much earlier than time point 30 as it completes in the local-JID scheduling. In fact, τ_3 is not schedulable for any $C_3^m > 1$ in local-JID, while it is schedulable with $C_3^L \leq 2$ in the global-JID scenario.

We can see that for all cold-started $\tau_{k,j}, j > 1$, the local-JID and global-JID scenarios have different scheduling. The local-JID scheduling always chooses the mode that has the longest execution time C_i^L for the cold-started jobs, while in global-JID scheduling, the mode is always determined by JID and is likely not the mode with C_i^L. Thus global-JID has at least the same performance as local-JID does in this example.

We also can see that under the same priority assignment and fixed priority scheduling, both local- and global-JID schedulings reduce the response time of some tasks in the taskset. The JID-based memory-aware P-FRP scheduling is thus potentially able to schedule more tasks than the P-FRP AR scheduling does.

In the next section, we provide experiments and analysis on the P-FRP Interface-aware and Memory-aware models.

7.4 Experiments and Analysis

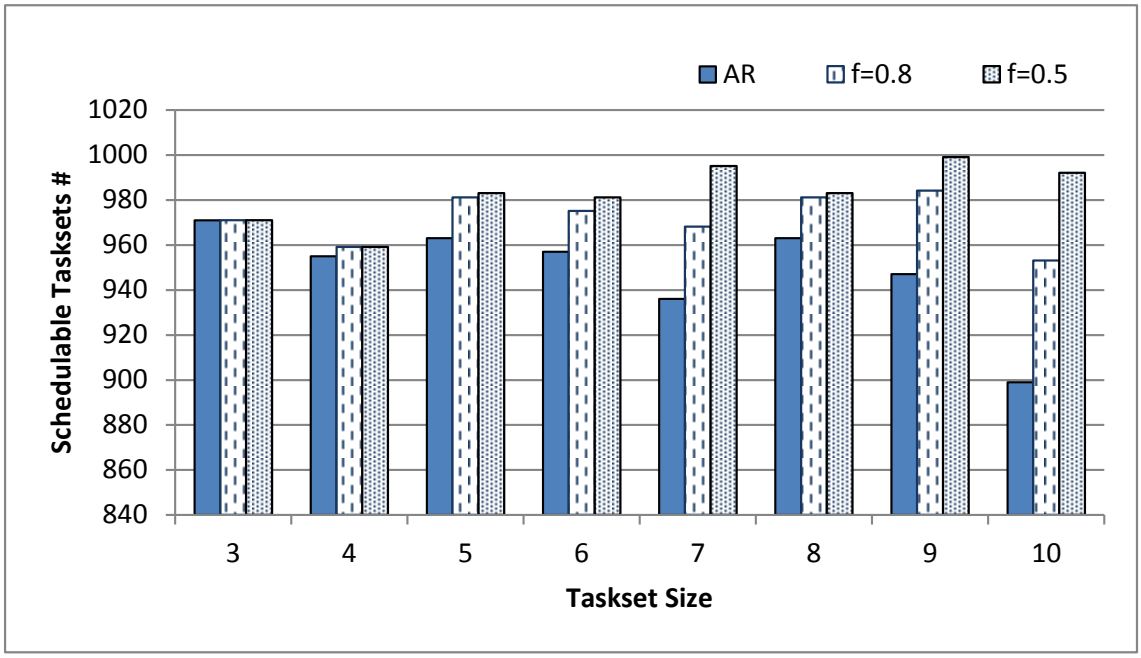
We design experiments for the multi-mode P-FRP model to observe how schedulability changes:

(1) when applying the new multi-mode P-FRP task scheduling and the original P-FRP AR scheduling; and

(2) when task execution time changes in various ways in the multi-mode scheduling.

There is still no known mathematical calculations to conduct schedulability test for P-FRP systems as of present. A simulation-based RTA (Response Time Analysis) schedulability test is applicable in such condition. The simulation runs in the feasibility interval. Lv et al. [97] presented an efficient simulation-based algorithm, LList-based algorithm LList-RTA, to calculate the response time for either classic task scheduling or P-FRP task scheduling. A task is schedulable only if all of its jobs complete within deadlines, and a taskset is schedulable only if all of its tasks are schedulable.

Synthetic tasksets are used in the experiments. The priority assignment algorithm is RM. All experiments are run on a desktop computer with a CPU of i3-4130 3.4GHz, 8 GB memory and *Ubuntu 14.04.3 LTS 64-bit Desktop* operating system. We describe



(a) U=0.2

Figure 7.4: Interface-aware P-FRP: Schedulability Comparisons

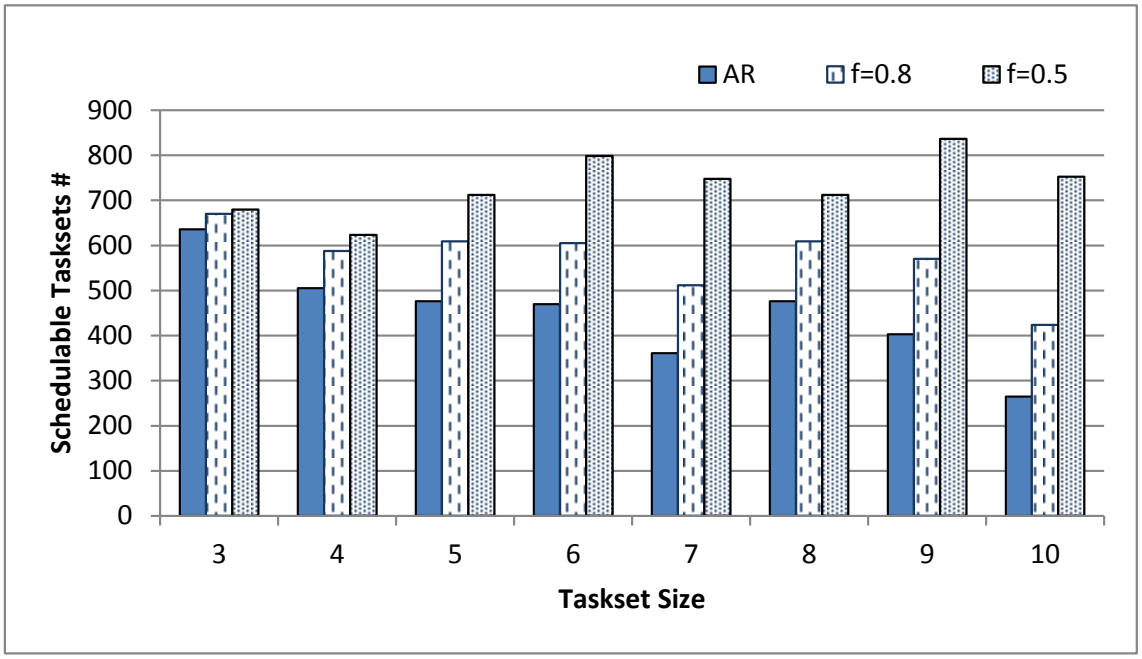
how we generate the synthetic tasks for the experiments in 7.4.1, followed by the experiments and analysis of Interface-aware and Memory-aware multi-mode P-FRP task scheduling in 7.4.3 and 7.4.3.

7.4.1 Task Generation

Following are the rules we use in task generation:

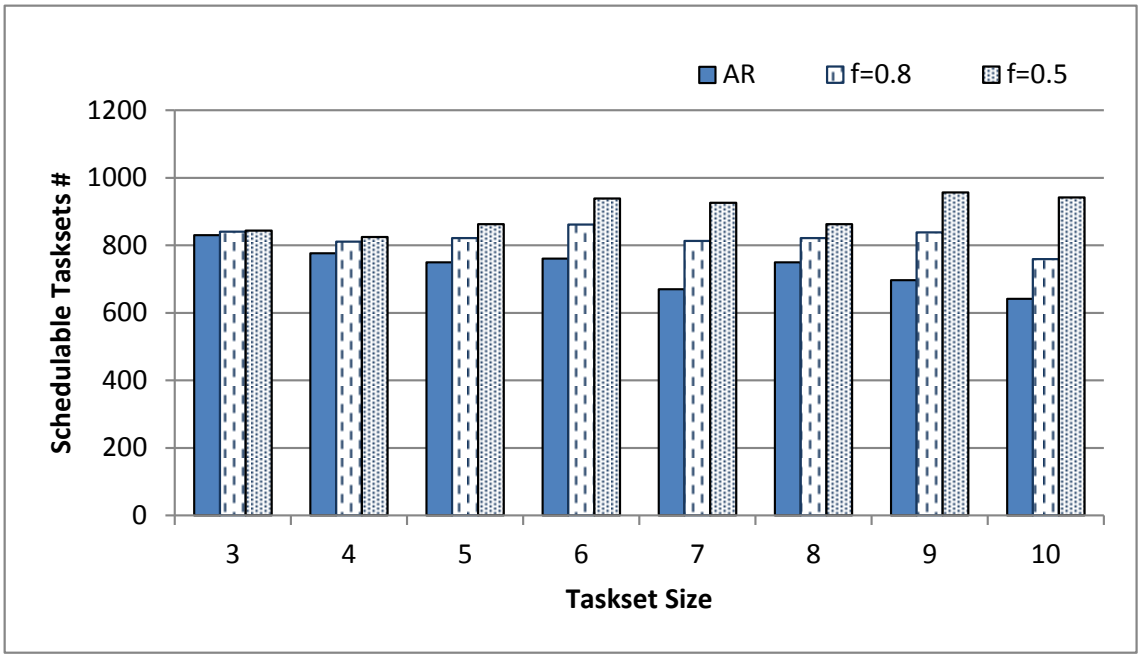
(1) All tasks are periodic (or sporadic tasks but arriving at their minimum intervals) with implicit-deadline;

(2) The number of tasks in a taskset, n, is in range [3, 10];



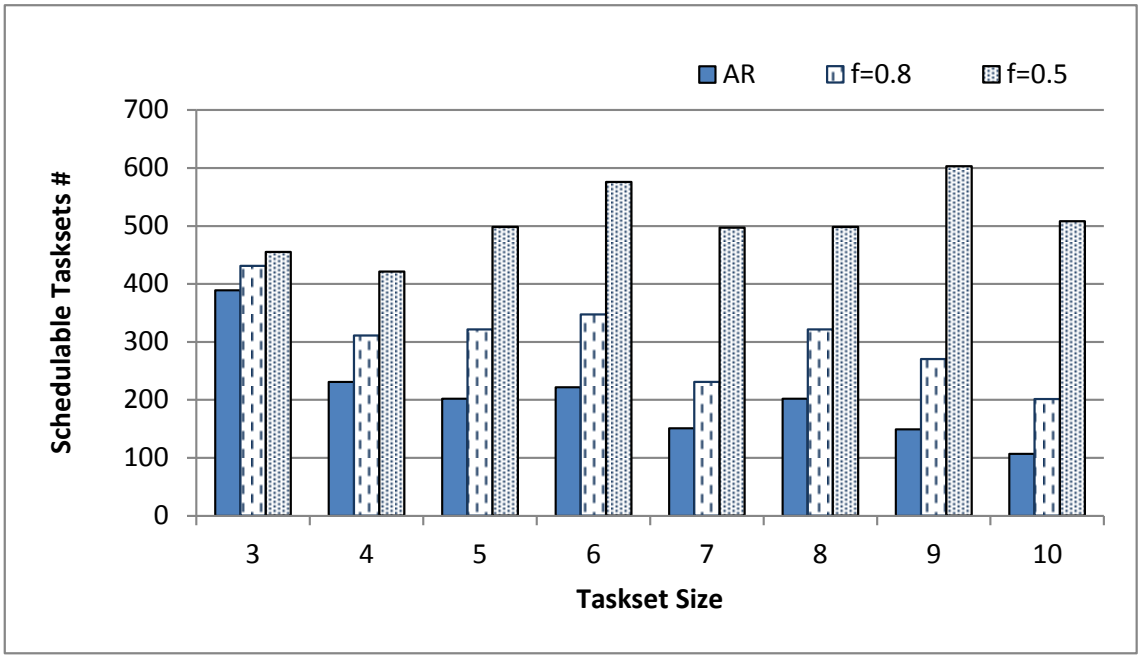
(b) U=0.3

Figure 7.4: Interface-aware P-FRP: Schedulability Comparisons (con't)



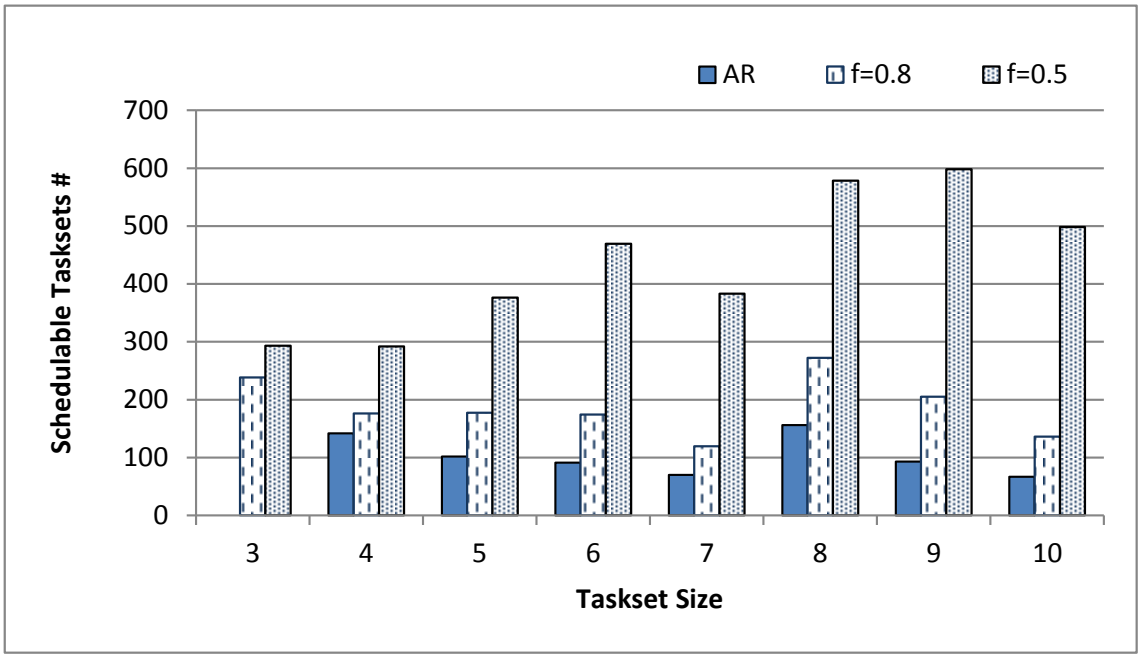
(c) U=0.4

Figure 7.4: Interface-aware P-FRP: Schedulability Comparisons (con't)



(d) U=0.5

Figure 7.4: Interface-aware P-FRP: Schedulability Comparisons (con't)



(e) U=0.6

Figure 7.4: Interface-aware P-FRP: Schedulability Comparisons (con't)

(3) Task periods are generated according to a *log-uniform distribution*. The log-uniform distribution of a variable x is such that $ln(x)$ has a uniform distribution. T_i is assigned to τ_i, and so forth. We choose periods in range of [20, 200] based on our application;

(4) Utilization U for a taskset is one of the values in $\{0.2, 0.3, 0.4, 0.5, 0.6\}$. At each U, the *UUniFast* Algorithm [30] is used to generate U_is for τ_is;

(5) Mode functions and methods of calculating execution times are different for InterA and JIDA as follows.

For InterA:

(i) Three modes are set for each job, i.e. $M_i = 3$ for $1 \leq i \leq n$ in the experiments;

(ii) The execution time in adjacent modes changes with a factor of f_a. f_a is set to 0.8 or 0.5 to simulate the slightly or moderately changing of execution times;

(iii) Task execution times are calculated by $C_i^L = C_i^1 = U_i \times T_i$. For each $2 \leq m \leq M_i$, $C_i^m = \lceil C_i^{m-1} * f_a \rceil$;

For JIDA:

(i) C_i^L and C_i^S of task τ_i are generated as follows: $C_i^L = U_i \times T_i$, $C_i^S = f_c \times C_i^L$. f_c indicates the range of the task execution time. f_c is set to one of the values in $\{0.2, 0.5, 0.8\}$ in different experiments.

(ii) Unlike InterA, M_is in JIDA are calculated instead of being given explicitly. In our experiments, the system capability regarding to performance of the fast

memory is expressed by a factor f_m, and f_m is used to obtain M_i and m and finally C_i^m. With larger f_m, execution time in different mode changes slower. M_i, m and C_m^m are calculated by Equation (7.3), (7.4) and (7.5) respectively.

$$M_i = \lceil 1/f_m \rceil \tag{7.3}$$

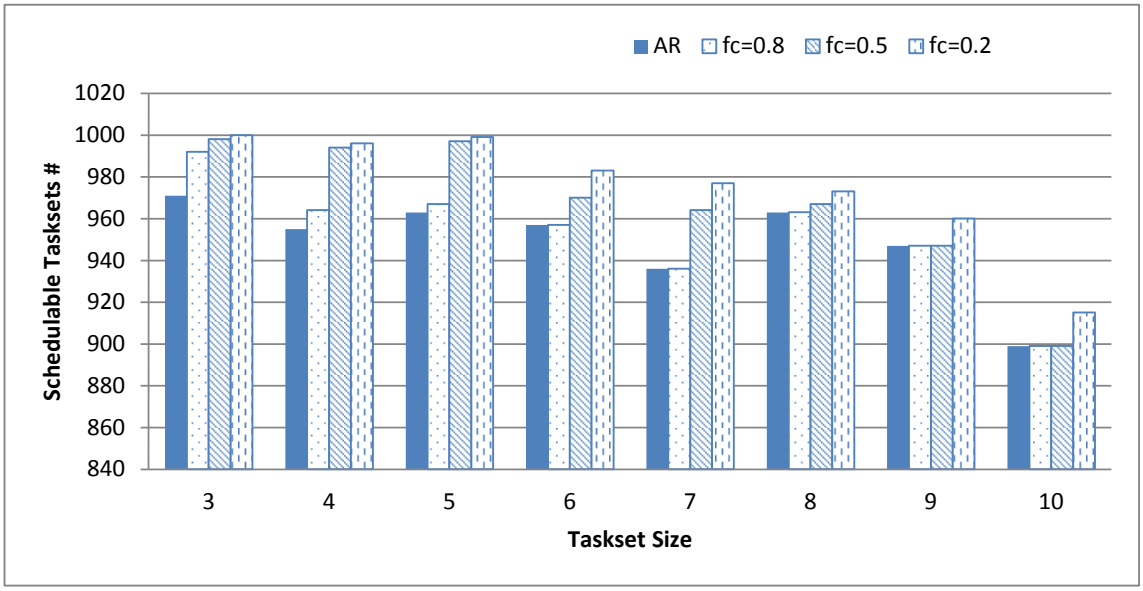
$$m = \begin{cases} JID + 1 & \text{if } JID < M_i \\ M_i & \text{if } JID \geq M_i \end{cases} \tag{7.4}$$

$$C_i^m = min(C_i^S + (C_i^L - C_i^S) * f_m * (m-1), C_i^L) \tag{7.5}$$

Therefore, f_m is the slope of the curve representing changes in the execution time with JID values. f_m is selected from the set of $\{0.1,\ 0.2,\ 0.3\}$ to simulate the three systems with different performance when running the tasksets with their hierarchical memory subsystems. Specifically, $f_m = 0.1$ means a system has better performance than the one with $f_m = 0.2$, and a system with $f_m = 0.2$ has better memory performance the one with $f_m = 0.3$.

(6) We generate 1000 different tasksets for each configuration pair (n, U). For example, Γ_3 has 1000 tasksets at utilization 0.2, and another 1000 tasksets at utilization 0.3, and so on. So there are totally 40,000 basic tasksets, and the actual experimental tasksets are generated using the basic tasksets and various configurations of f_a, f_m and f_c as described above;

Utilization = 0.2:



(a) $f_m = 0.1$

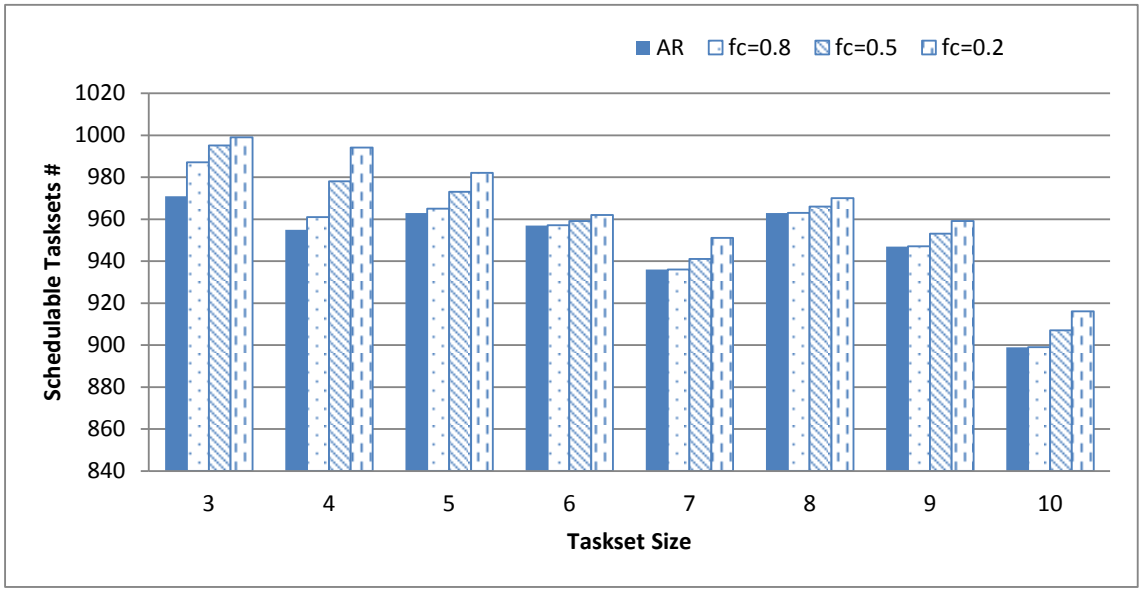
Figure 7.5: Memory-aware P-FRP: Schedulability Comparisons

(7) Tasks are synchronously released at time 0, and the schedulability test time interval is $[0, LCM_n)$ for Γ_n.

7.4.2 Interface-aware P-FRP

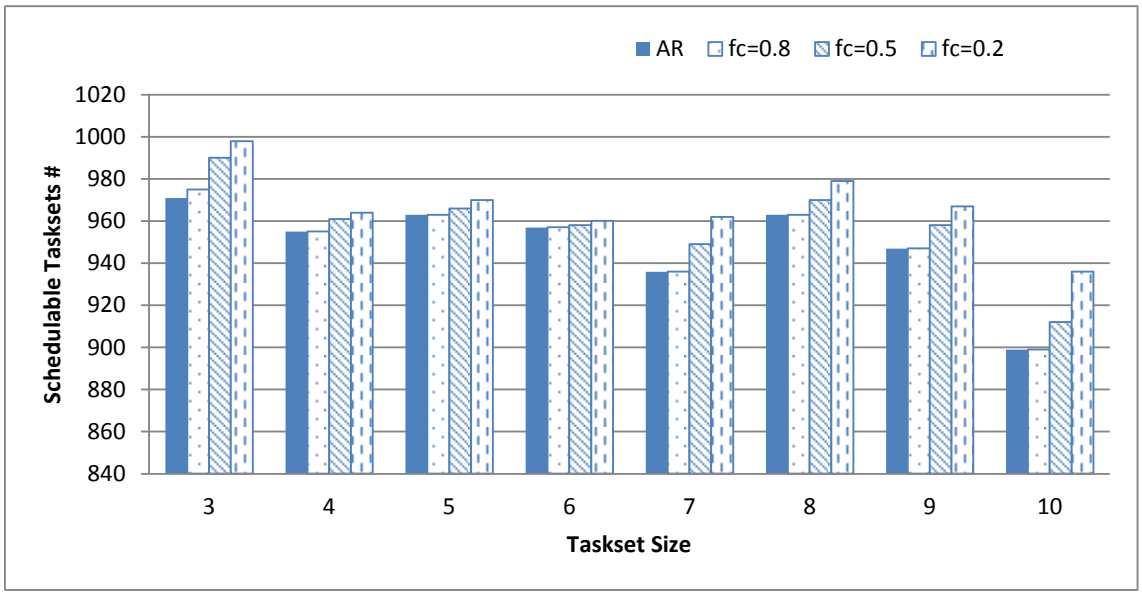
We focus on the schedulability improvement of the proposed multi-mode tasks over the P-FRP AR model, and how the changing f_a affects the schedulability in the multi-mode tasks scheduling.

We use the notation $InterA_{f_a=x}$ to denote that f_a is set to x in a certain experiment. We run LList-RTA schedulability test to all experimental tasksets, and obtain the number of schedulable tasksets respectively in the P-FRP AR, $InterA_{f_a=0.5}$ and



(b) $f_m = 0.2$

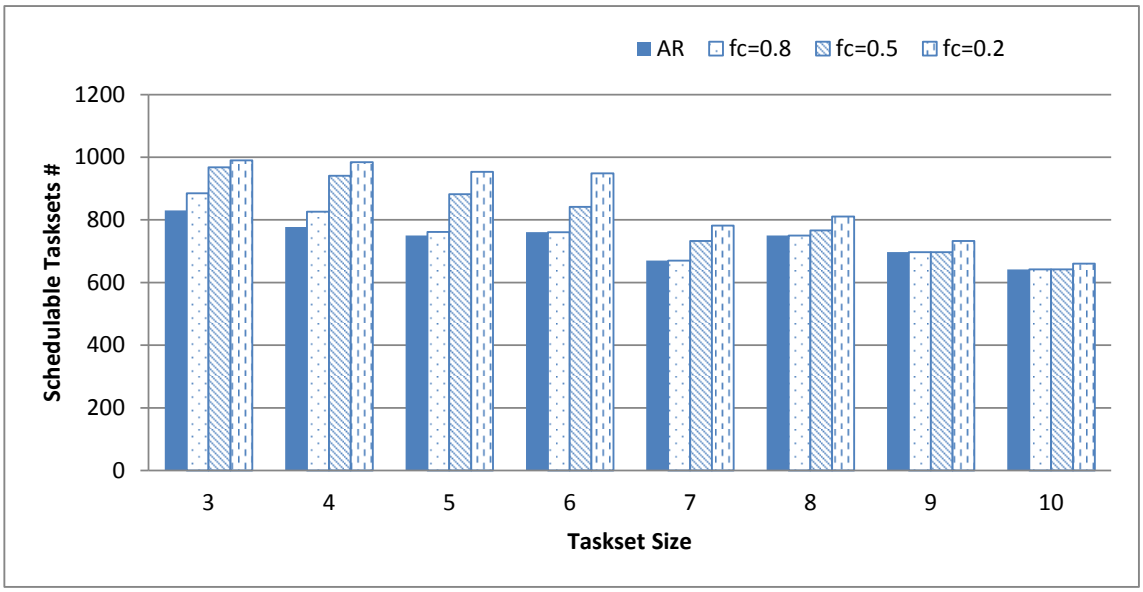
Figure 7.5: Memory-aware P-FRP: Schedulability Comparisons (con't)



(c) $f_m = 0.3$

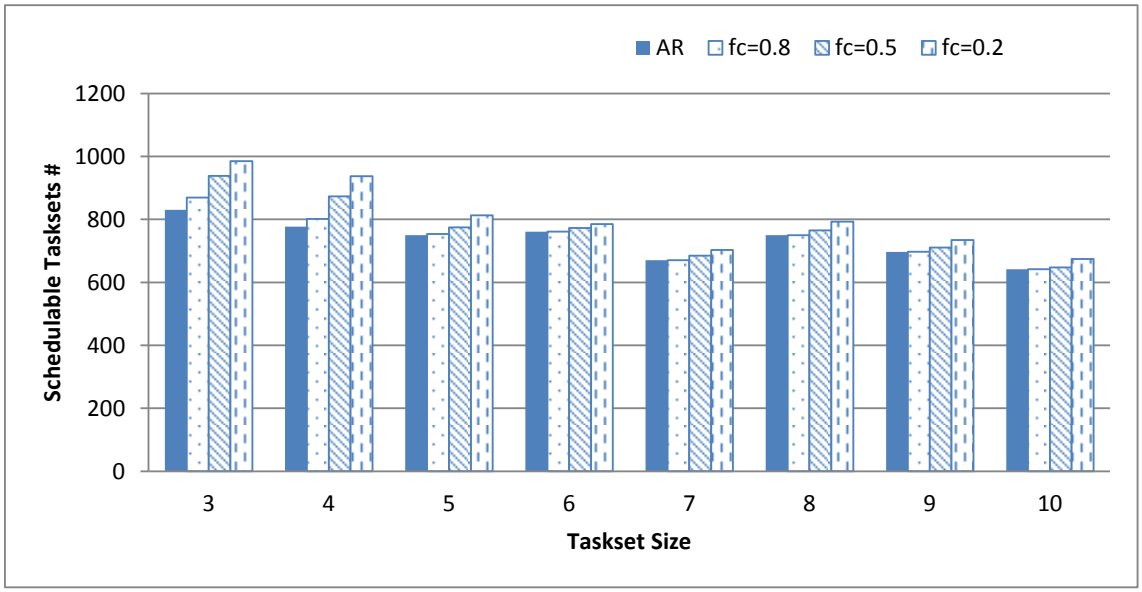
Figure 7.5: Memory-aware P-FRP: Schedulability Comparisons (con't)

Utilization = 0.3:



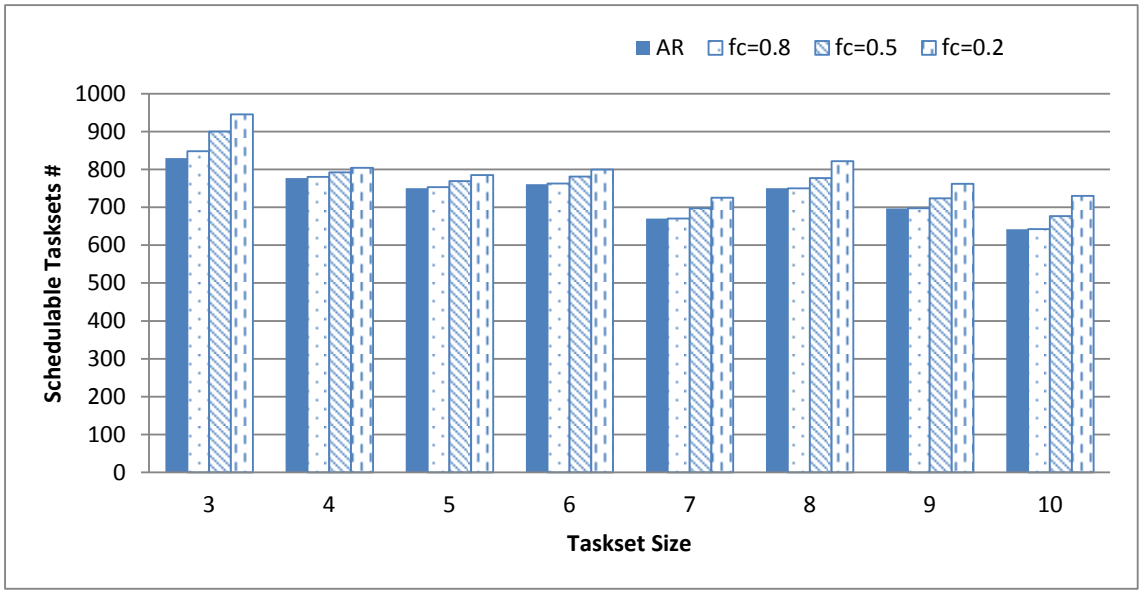
(d) $f_m = 0.1$

Figure 7.5: Memory-aware P-FRP: Schedulability Comparisons (con't)



(e) $f_m = 0.2$

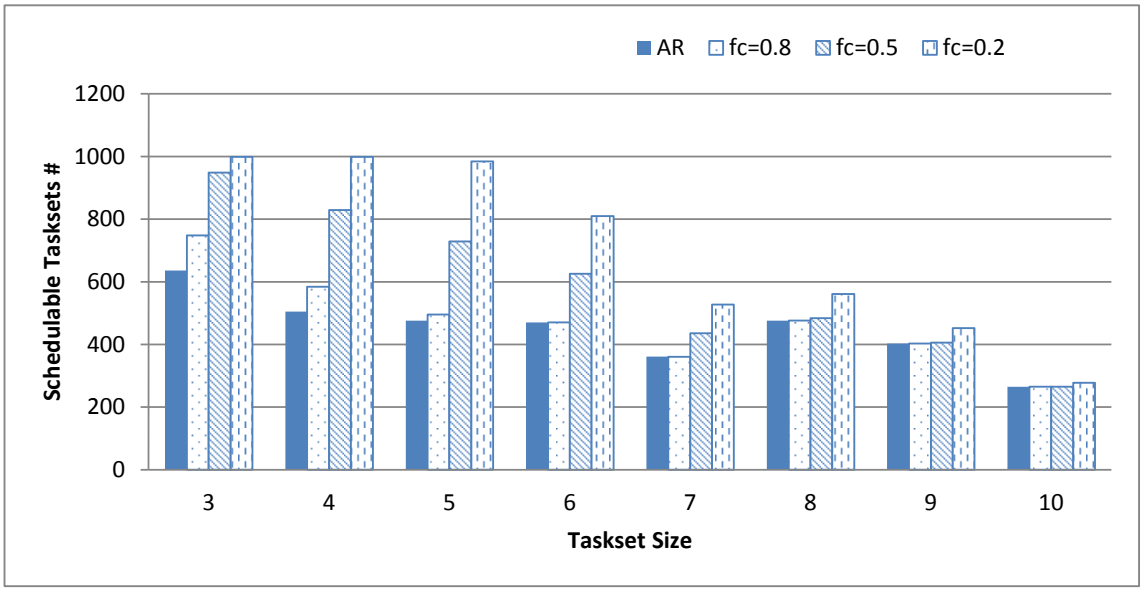
Figure 7.5: Memory-aware P-FRP: Schedulability Comparisons (con't)



(f) $f_m = 0.3$

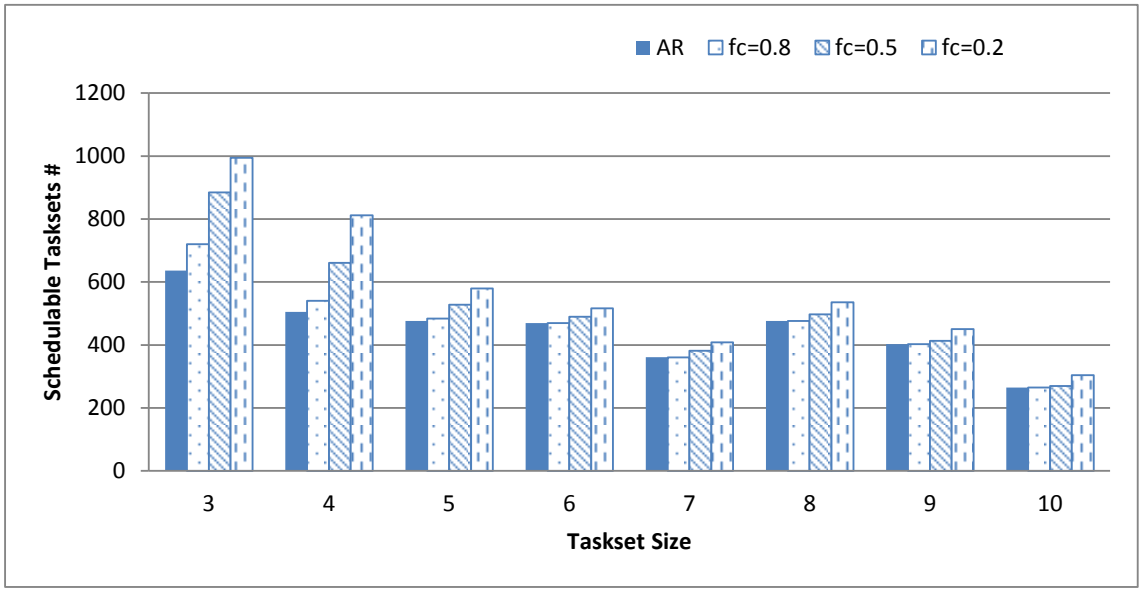
Figure 7.5: Memory-aware P-FRP: Schedulability Comparisons (con't)

Utilization = 0.4:



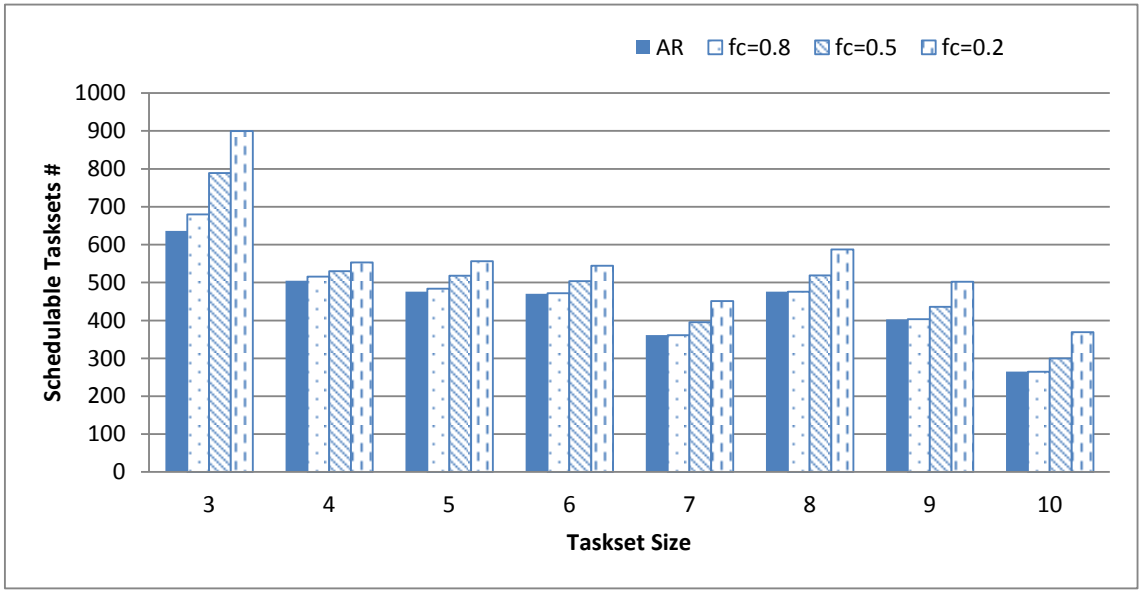
(g) $f_m = 0.1$

Figure 7.5: Memory-aware P-FRP: Schedulability Comparisons (con't)



(h) $f_m = 0.2$

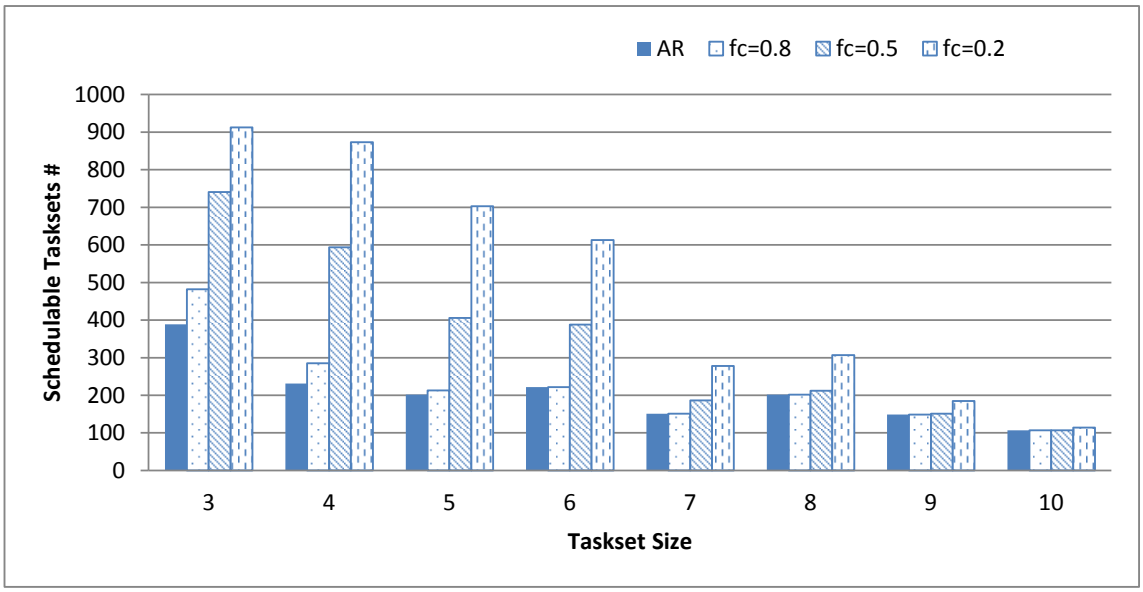
Figure 7.5: Memory-aware P-FRP: Schedulability Comparisons (con't)



(i) $f_m = 0.3$

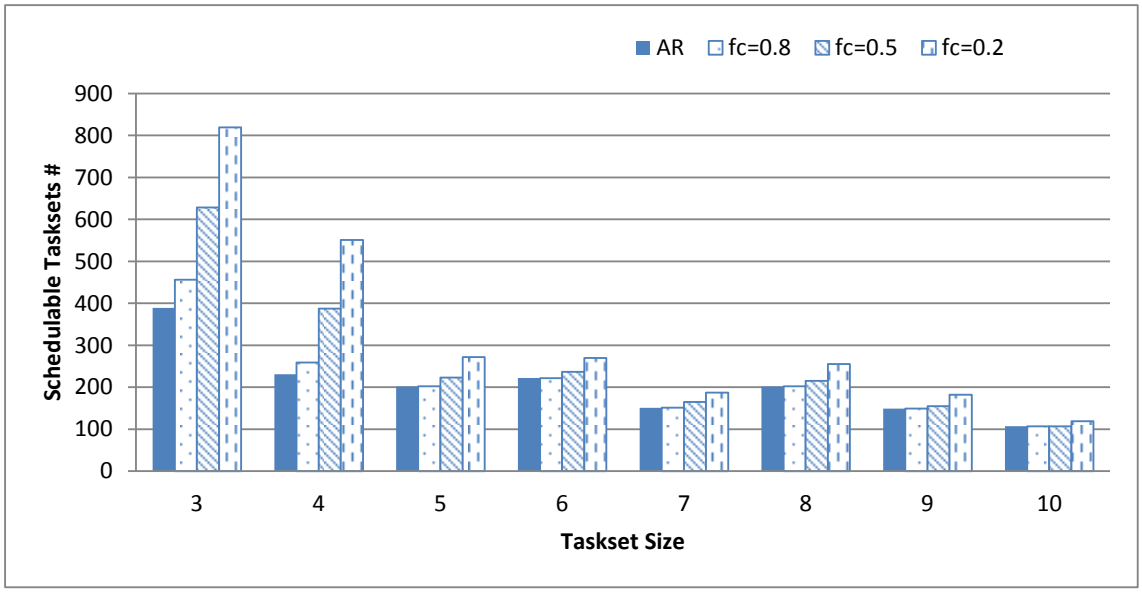
Figure 7.5: Memory-aware P-FRP: Schedulability Comparisons (con't)

Utilization = 0.5:



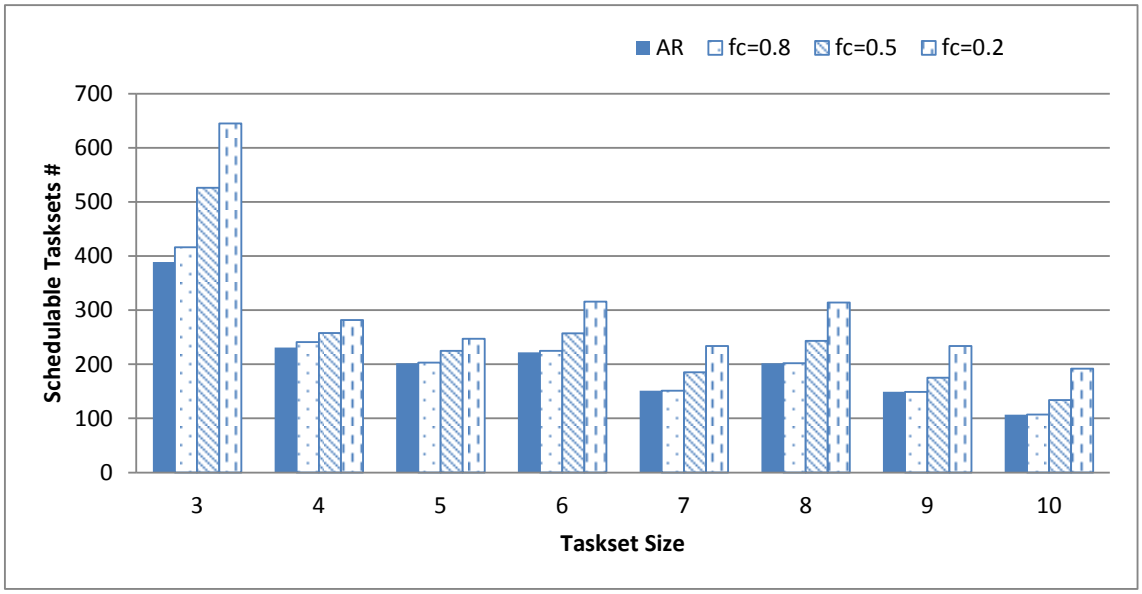
(j) $f_m = 0.1$

Figure 7.5: Memory-aware P-FRP: Schedulability Comparisons (con't)



(k) $f_m = 0.2$

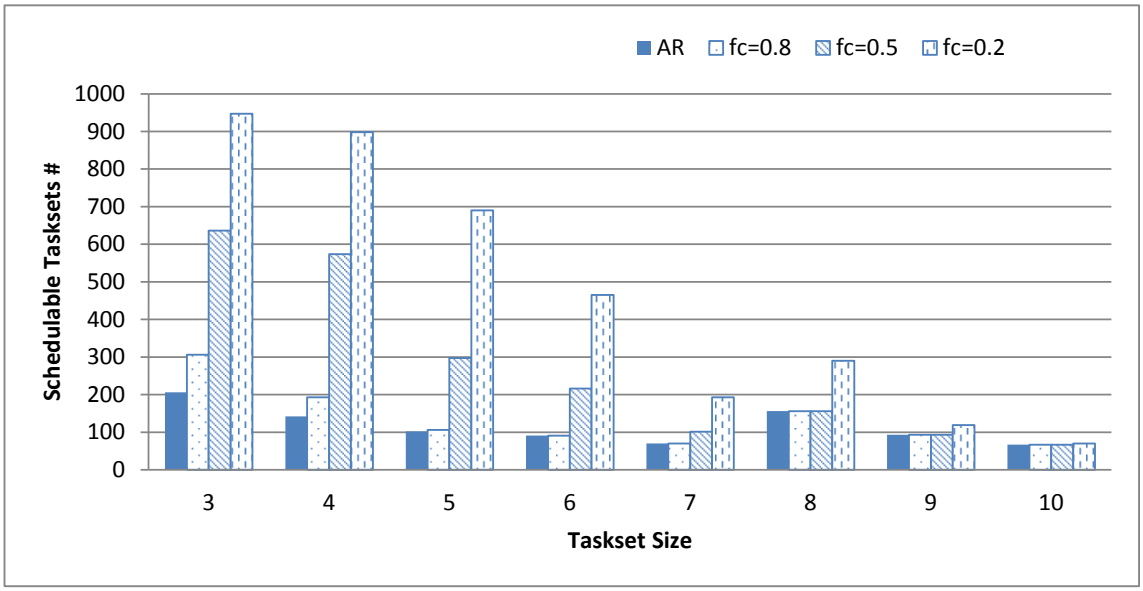
Figure 7.5: Memory-aware P-FRP: Schedulability Comparisons (con't)



(l) $f_m = 0.3$

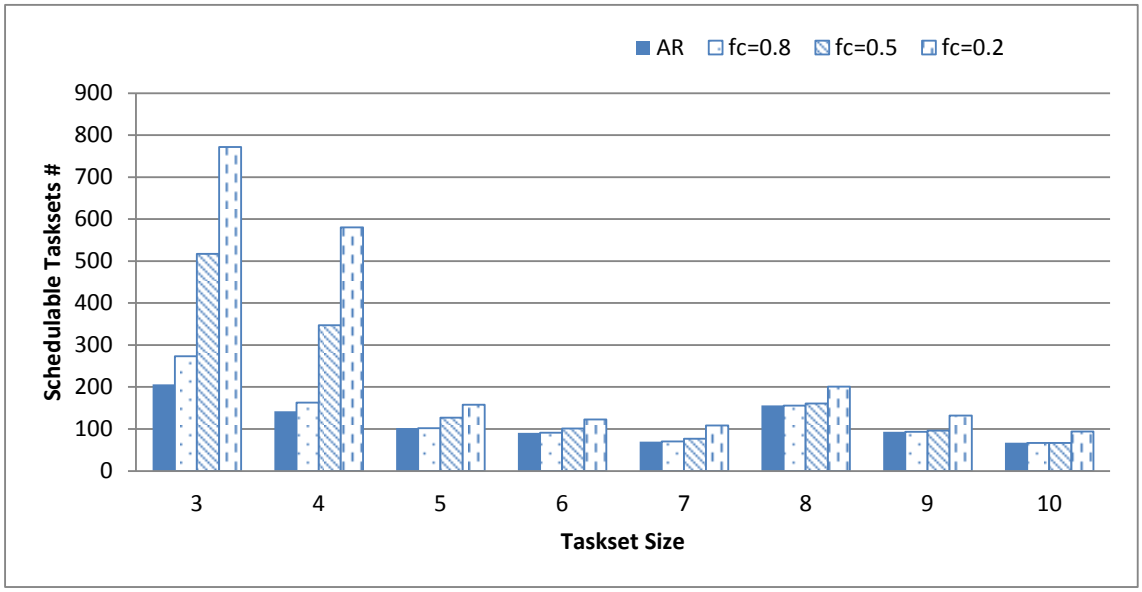
Figure 7.5: Memory-aware P-FRP: Schedulability Comparisons (con't)

Utilization = 0.6:



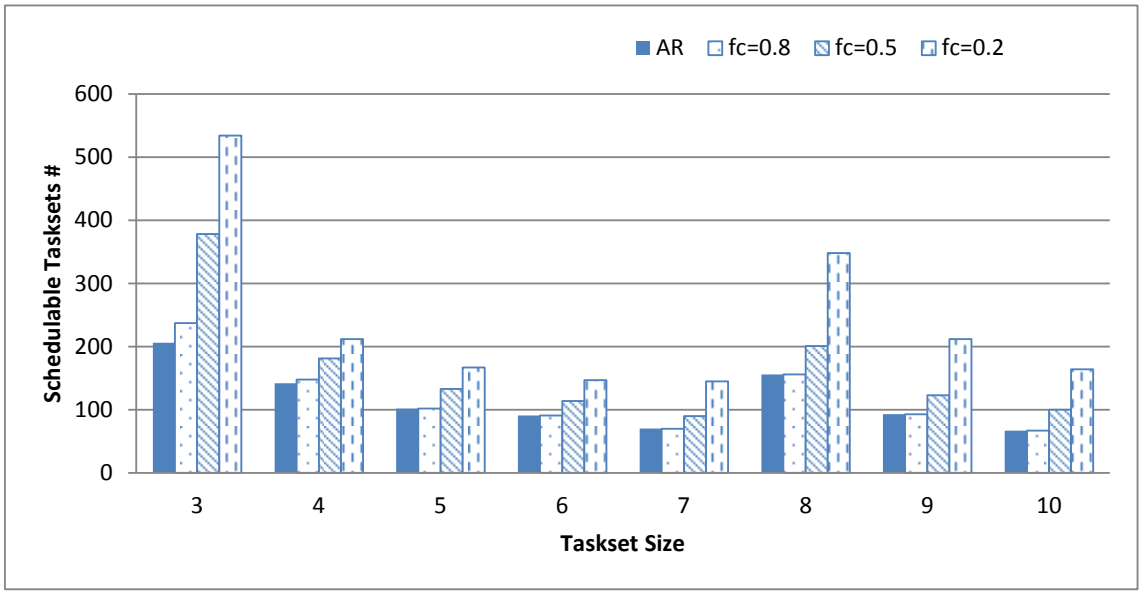
(m) $f_m = 0.1$

Figure 7.5: Memory-aware P-FRP: Schedulability Comparisons (con't)



(n) $f_m = 0.2$

Figure 7.5: Memory-aware P-FRP: Schedulability Comparisons (con't)



(o) $f_m = 0.3$

Figure 7.5: Memory-aware P-FRP: Schedulability Comparisons (con't)

$InterA_{f_a=0.8}$ at each utilization setting. The numbers are shown in Fig.7.4.

First, the figure shows that under all of these configurations, InterA outperforms AR in terms of having more schedulable tasksets. Second, it also shows that the improvements increase generally when the utilization increases, and at each utilization level, the improvements increase when the taskset size increases.

We also find in Fig.7.4 that $InterA_{f_a=0.5}$ has larger improvement than $InterA_{f_a=0.8}$ under most configurations, and has the same improvements over the P-FRP AR in rest of configurations, and it is as our expectations stated in the task generation subsection.

In extreme circumstances, schedulability rate drops to very low levels in P-FRP AR model when the utilization is high (such as U=0.6 in Fig.7.4.(e)), and by using multi-mode task model, it restores to decent levels.

7.4.3 Memory-aware P-FRP

We use the notation $JIDA_{f_c=x}$ to denote that f_c is set to x. We obtain the number of schedulable tasksets respectively in the P-FRP AR and P-FRP JIDA with various configuration tuples (n, U, f_c, f_m). Results are shown in Fig.7.5. Bastoni *et al.* [15] introduced a method to show multiple numbers of schedulable tasksets in a single graph by weighting those numbers base on utilizations. While in P-FRP scheduling, not only the relative value, i.e. utilization, but also the absolute values of execution time and period affect schedulability. Thus we show the original numbers in separate graphs.

In each sub-figure, we can see that $JIDA_{f_c=0.2}$ can schedule more than $JIDA_{f_c=0.5}$ does, and $JIDA_{fc=0.5}$ can schedule more tasksets than $JIDA_{f_c=0.8}$ does. Observing horizontally at each utilization level, such improvements caused by f_c increase when f_m increases. The reason is that with larger f_m, the execution time differences increase, and thus results in higher schedulability. Observing Fig.7.5 vertically, we can see that the multi-mode memory-aware scheduling outperforms the P-FRP AR model under all configurations, and improvements increase non-linearly as utilization increases, and as much as 700%.

We also see that the schedulability rate of the P-FRP AR scheduling decreases sharply with increasing utilization and taskset size as shown in Fig.7.5.(j)-(o), while the multi-mode memory-aware scheduling maintains decent performance in such scenarios.

To sum up, our experiments result and analysis show that the proposed models are able to schedule one to seven times more tasksets compared to the P-FRP AR model.

7.5 Conclusions

Confronting the challenges of a real-world project, we proposed to use P-FRP to resolve the difficulties. In order to overcome drawbacks of P-FRP that cause over-consumption of the system (computation) resources, we proposed a multi-mode P-FRP task framework. We also presented two specific models for the multi-mode task

framework according to two practical situations, the Interface-aware and Memory-aware models. The Interface-aware multi-mode task model focuses on a task itself, while the Memory-aware one addresses the whole taskset and the runtime system.

In previous studies of the original P-FRP model, the largest execution time of a task is used for all its restarted tasks. In some practical scenarios, however, the restarted tasks likely consume less time than the largest one when considering the changing/unchanging input/output or the memory effect such as cache-hit in loading code and data. Our proposed Interface-aware and Memory-aware P-FRP models are able to reflect such effects. Simulation results show that the schedulability is improved significantly when more accurate execution time of a task in *cold start* and *restarted* cases is addressed.

Chapter 8

SimSo-PFRP

When research tends to solve real world problems, it seems to be a better approach to evaluate the research on real systems. However, it is hard even impossible to find such real systems sometimes. P-FRP is such a case because as we've shown in previous chapters, P-FRP is a new model, and it involves many research areas which do not have valid research results at present. As a consequence, simulation is a good compromise to efficiently evaluate our research. In previous chapters, we have presented some algorithms and experiments. In most of our previous work, experiments are executed by our C++ based programs in consideration of the run time efficiency. However, we believe it is a critical step to build a more generic and extension-friendly platform to facilitate the research of P-FRP task scheduling and timing analysis. In this chapter, we present our work on this endeavor.

8.1 Introduction

We've shown that many task scheduling research results are no longer applicable for P-FRP, it is necessary to check then modify existing research results and conduct new research in P-FRP. On the other hand, P-FRP targets on solving complicated problems and scalability is of its strong merit. It is highly important to conduct the research of P-FRP task scheduling on multi-processor platform. In their excellent survey paper [55], Davis and Burns referenced more than thirty real-time multi-processor scheduling algorithms. And more than a dozen of new algorithms have emerged since then. Thus a generic and extension-friendly simulation platform is in highly demanding.

Our work is an extension to an existing project named SimSo [45] [71]. We call our work **SimSo-PFRP**, meaning that it is a P-FRP extension of SimSo. In considerations of self-complete and easier understanding of our work, we include some details and descriptions of SimSo in this chapter.

In [45], Chéramy *et al* wrote "SimSo is a simulator designed for the comparison and the understanding of real-time scheduling policies. It is designed to facilitate the implementation of schedulers in a realistic way. Currently, more than twenty-five scheduling algorithms are available in SimSo. A particular attention is paid to the control of the computation time of the jobs therefore introducing more flexibility, for instance by taking into account cache-related preemption delays. In addition, SimSo offers an easy way to generate the tasksets, to perform simulations and to collect data from the experiments".

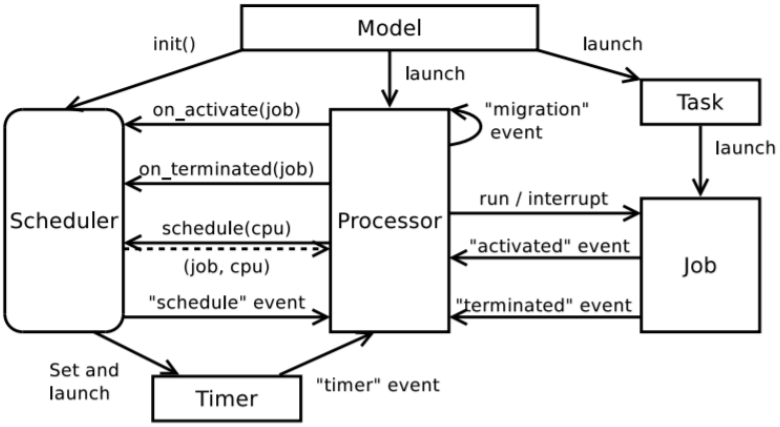


Fig. 1. Interactions between main class instances. *Processor*, *Task*, *Job* and *Timer* are *Process* objects and can have multiple instances.

Figure 8.1: Simso Architecture

We choose SimSo because it is the best match among others to our target, including the option of building one from scratch by ourselves. SimSo is a real-time scheduling simulator designed to be easy to use as well as to extend, and it is freely available under an open source license.

8.2 Architecture

The core of SimSo relies on SimPy, a process-based discrete-event simulation framework. The use of discrete-event simulation allows it to deal with short and long duration at the same cost. Its process-based nature offers a convenient way to express the behavior of the simulated components. SimSo uses a *Configuration* object what contains all the information about the system, such as tasksets, processors, duration, scheduler, etc.

To give a whole picture of SimSo, our Fig.8.1 refers to a figure in [45]. We also quote [45] on the description of SimSo:

"The figure shows the main classes of SimSo and their mutual interactions. SimSo uses objects to simulates real system modules: processors, tasks, jobs, timers, etc. Each of these objects simulates the behavior of the corresponding part on the system: *Tasks* release the jobs; *Jobs* emulate the execution of the task's code; *Timers* can launch a method on a processor at a given time; etc. The instances of *Processors* are actually the central part of the simulation because they simulate both a processor and the operating system executing on it. Each processor can execute a job or be interrupted to execute a method of the scheduler. Finally, the *Scheduler* object is not an active process. It could be considered as a part of the operating system and as a consequence, its methods are only called by the Processors."

"The *Model* object is the conductor of the simulation. It takes as a parameter the *Configuration* object. When the *run_model* method is called, the objects described above are created and launched."

"The design of SimSo allows it to take into consideration various time overheads that occur during the life of the system. This includes direct overheads such as context-switches and scheduler calls (with fixed time penalties) but also indirect overheads with a simplified system of locks to forbid the parallel execution of a scheduler if needed. Such overheads are applied on the processor they are supposed to occur (e.g., the time spent in the scheduler is taken into account on the processor that called the scheduler)."

SimSo has already implemented more than 25 schedulers including uni- and multi-processor algorithms. The main uni-processor schedulers such as RM, DM, FP, EDF and M-LLF [107] are implemented. It even includes DVFS (Dynamic Voltage and Frequency Scaling) schedulers such as Static-EDF and CC-EDF [111]. For multi-processor systems, both partitioning and global scheduling algorithms are available. For *partitioning* scheduling, P-EDF and P-RM using decreasing First-Fit assignment algorithm are implemented. It also provides a class to use Next-Fit, Best-Fit, Worst-Fit algorithms. For *global* scheduling, SimSo implemented following schedulers: G-RM, G-EDF, G-FL [13], EDF-US[29], PriD [18], EDZL, M-LLF [24] and more recently U-EDF [22]. SimSo also implemented some PFair schedulers: LLREF [9], LRETL [15], DP-WRAP [21], BF [31] and NVNLF [14]. For *semi-partitioning*, SimSo implemented EDHS [19], EKG [2] and RUN [26].

8.3 Execution Time Model

Task scheduling is essentially to manage the execution order and duration of jobs. In ideal classic preemptive scheduling, the total job duration is fixed as its WCET. The remaining execution time of a job decreases monotonically. It keeps decreasing when the job is executing, and remains unchanged when the jobs is preempted till it resumes execution then decreases. This ideal model is oversimplified of the reality. SimSo uses the term *Execution Time Model* (ETM) to denote this procedure. The authors claim their purpose of using ETM is to customize duration of jobs.

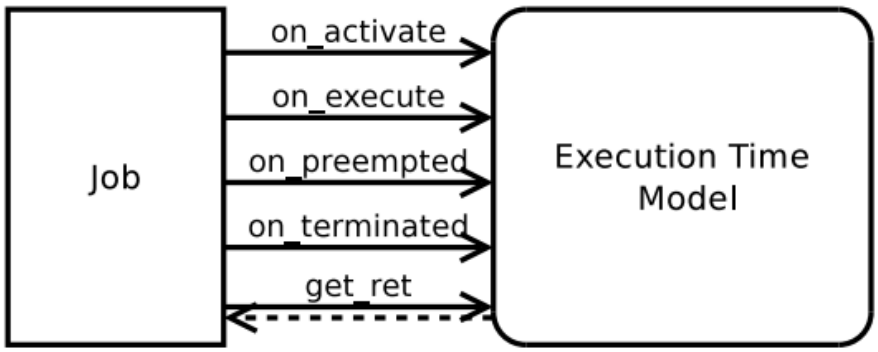


Fig. 3. Interface of any execution time model.

Figure 8.2: Simso Architecture

There are a few reasons that make us to say that the ideal model is oversimplified. In task scheduling, the use of WCET is common. However, it is in fact very pessimistic: First, WCET is the worst-case this an upper bound of the execution of a job which is hardly reached. In most of time, a job likely finishes in less or much less time than WCET. Second, the preemption and migration cost are ignored which in is not true in real world. In fact, in many cases, preemption cost and migration cost can dominate the time consumption in the system. Now it is time to recall our P-FRP AR model. Unlike the classic model, P-FRP has an additional abort cost. When a preempted P-FRP job restarts, determining its execution time is crucial and much more complicated than in class model. Because of this, ETM is a fundamental part of our extension to SimSo.

Fig.3 in [45] (our Fig.8.2) depicts the ETM class in SimSo. The ETM object is informed by the jobs of any scheduling event (activate, execute, preempted, terminated, etc.). The job will use the *get_ret* method to get a lower bound of its remaining

execution time and, when that time is up, the job calls that method again until it returns 0.

Beside the basic WCET model, SimSo also implemented an ACET, Average Execution Time, execution model. The ACET model uses a normal distribution defined by its mean, its standard deviation and is bounded by the WCET. Another model detects the preemptions and migrations and extends the WCET of the job using fixed time penalties. It also has a more complex model that tries to simulate the state of the caches. We implemented three P-FRP ETMs in SimSo-PFRP.

8.4 SimSo-PFRP

We implemented three ETMs in SimSo-PFRP:

OFRP - *Original P-FRP AR;*

JIDA - *Memory-aware multi-mode P-FRP;*

InterA - *Interface-aware multi-mode P-FRP.*

Fig. 8.3 shows the example code of OFRP ETM. All ETM classes are derived from a base class, AbstractedExecutionTimeModel. A specific ETM needs to implement the functions accordingly. We use OFRP class as example. Function __init__ is called automatically when a OFRP object is created, and does some initialization such as:

self.sim = sim # save simulation object.

self.executed = # create an empty list to save executed time of a job.

```
# original FRP: discard executed work at preempted if is not finished
from simso.core.etm.AbstractExecutionTimeModel \
    import AbstractExecutionTimeModel

class OFRP(AbstractExecutionTimeModel):
    def __init__(self, sim, _):
        self.sim = sim
        self.executed = {}
        self.on_execute_date = {}
        self.num_preempted = {}
        self.on_preempted_date = {}
    def init(self):
        pass
    def update_executed(self, job):
        if job in self.on_execute_date:
            self.executed[job] += (self.sim.now() - self.on_execute_date[job]
                                   ) * job.cpu.speed

            del self.on_execute_date[job]
    def on_activate(self, job):
        self.executed[job] = 0
        self.num_preempted[job] = 0
        self.on_preempted_date[job] = 0
    def on_execute(self, job):
        if self.on_preempted_date[job] < self.sim.now():  # last job is not me
            self.num_preempted[job] += 1
            self.executed[job] = 0 # P-FRP AR: restart === discard executed
        self.on_execute_date[job] = self.sim.now()
        print(self.sim.now(), job.task.name, "etm.on_execute.executed", self.executed[job])
    def on_preempted(self, job): # will be triggered even it will be picked as the next job
        self.update_executed(job)
        self.on_preempted_date[job] = self.sim.now()
        print(self.sim.now(), job.task.name, "etm.on_preempted.executed", self.executed[job])
    def on_terminated(self, job):
        self.update_executed(job)
        del self.on_preempted_date[job]
    def on_abort(self, job):
        self.update_executed(job)
        del self.on_preempted_date[job]
    def get_executed(self, job):
        if job in self.on_execute_date:
            c = (self.sim.now() - self.on_execute_date[job]) * job.cpu.speed
        else:
            c = 0
        print(self.sim.now(), job.task.name, "etm.get_executed", self.executed[job], c)
        return self.executed[job] + c
    def get_ret(self, job):
        wcet_cycles = int(job.wcet * self.sim.cycles_per_ms)
        return int(wcet_cycles - self.get_executed(job))
    def update(self):
        for job in list(self.on_execute_date.keys()):
            self.update_executed(job)
```

Figure 8.3: OFRP of SimSo-PFRP

self.on_execute_date = # create an empty list to execution start time of a job.

self.num_preempted = # create an empty list to save the number of preemptions times of a job.

self.on_preempted_date = # create an empty list to save when preemptions happens to a job.

Function ***init*** *is called explicitly when it is necessary.*

Function ***on_activate*** *is called when a job is released (cold-start).*

Function ***on_execute*** *is called when a job is scheduled to start or restart to execute. Since the variable executed[job-i] saves the time executed for the job-i, we set this variable to 0, and that implements the semantic of the original P-FRP AR model: discarding the incomplete execution.*

Function ***on_preempted*** *is called when a job is preempted.*

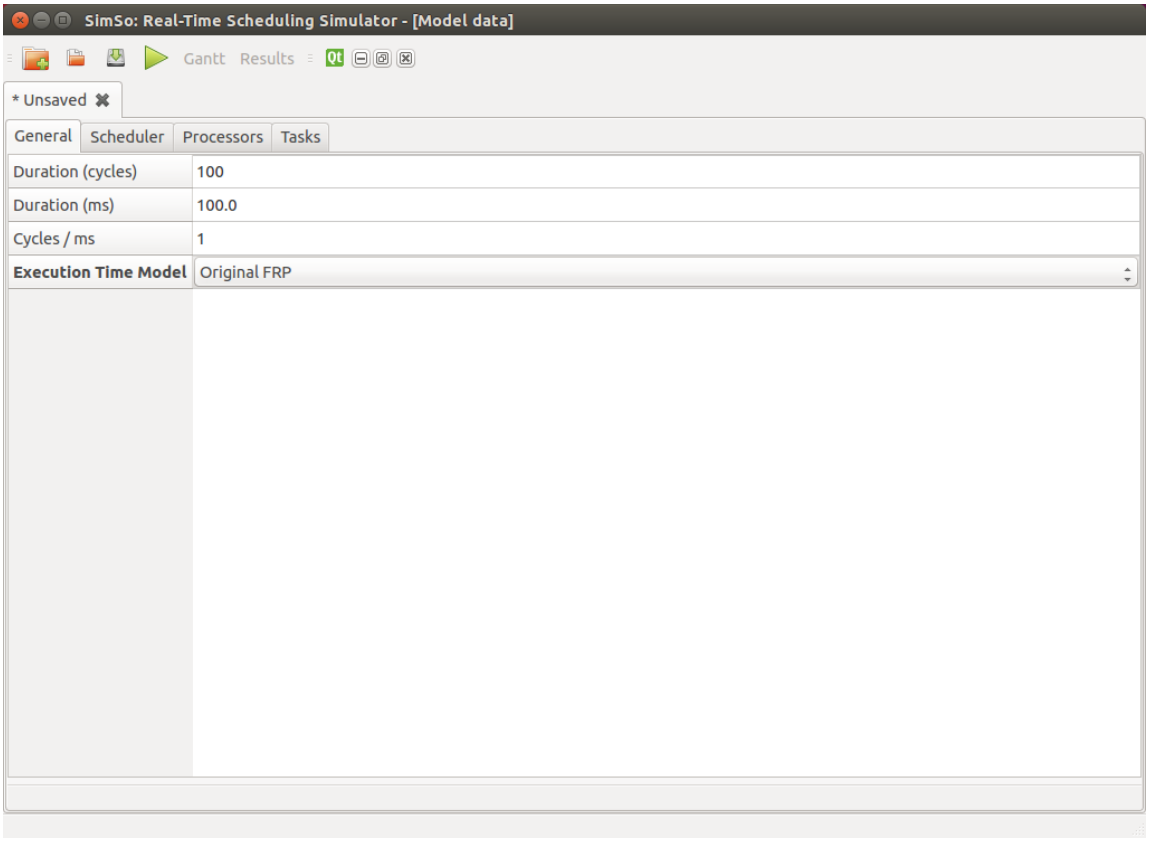
Function ***on_terminated*** *is called when a job finishes correctly.*

Function ***on_abort*** *is called when a job is aborted before finishing.*

Function ***get_executed*** *calculates the time a job has executed.*

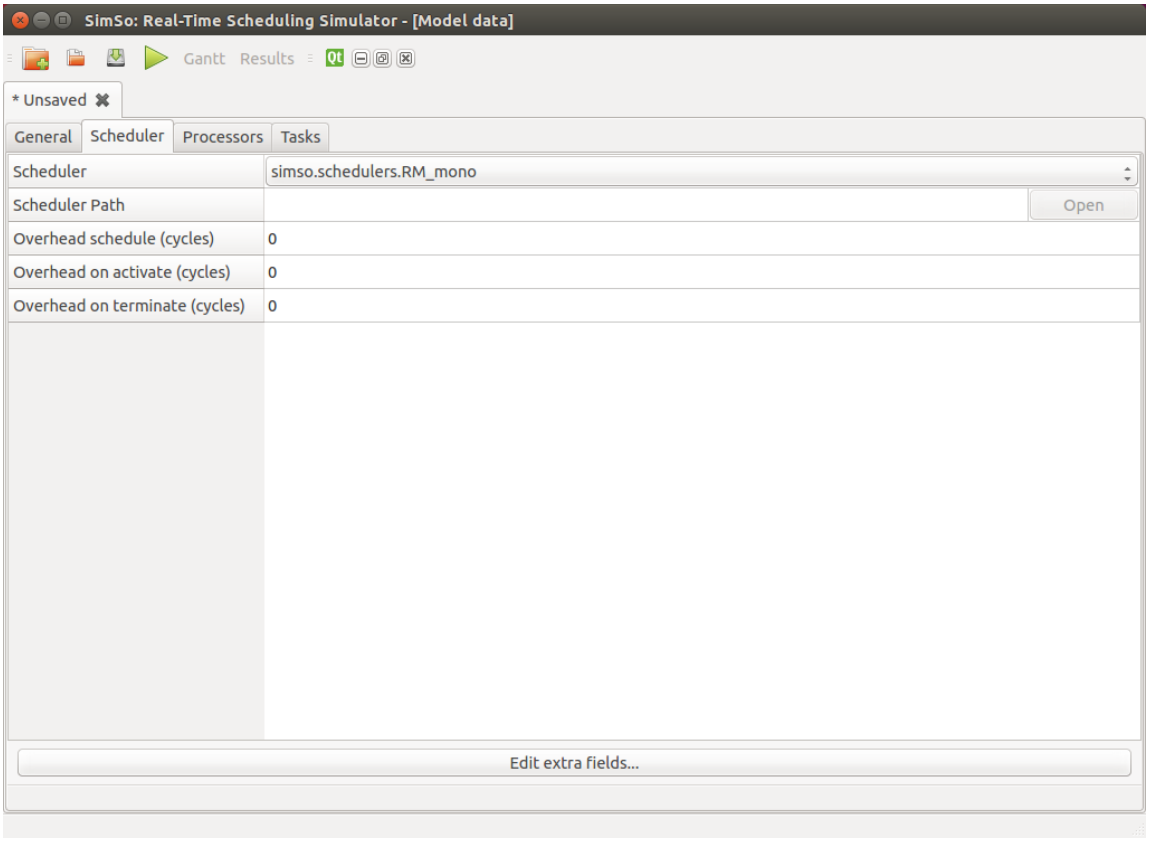
Function ***get_ret*** *returns the remaining time of a job.*

SimSo has a fully support of any functions using Python script. However, it also includes a QT [76] based GUI for users to have a better visualization how SimSo works. Fig.8.4 shows an example. In Fig.8.4.(a) we choose OFRP as the ETM; in Fig.8.4.(b) we choose uni-processor RM priority assignment algorithm; we choose uni-processors with a unit speed in Fig.8.4.(c) and we create a taskset with three periodic tasks in Fig.8.4.(d). After we run the this configuration, the experiment results are



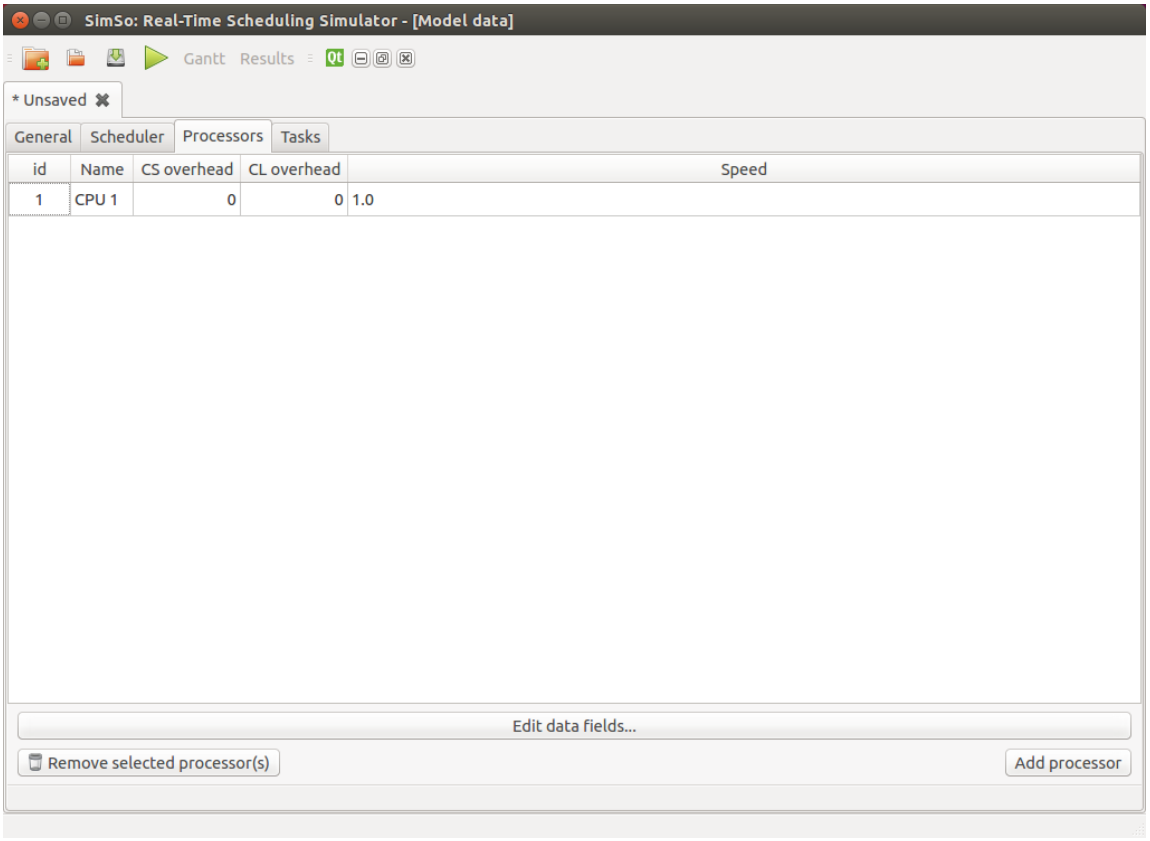
(a) Select ETM - OFRP

Figure 8.4: SimSo-PFRP: Example of OFRP Scheduling



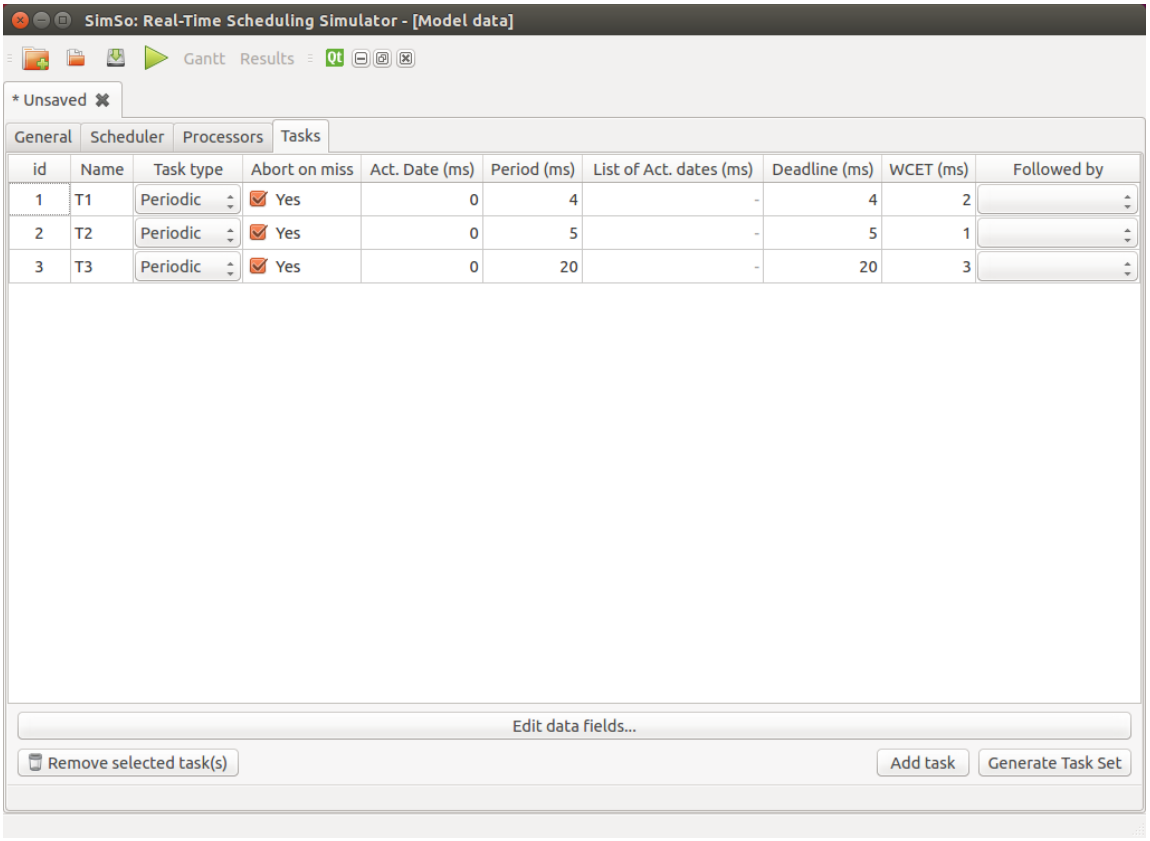
(b) Priority Assignment - RM

Figure 8.4: SimSo-PFRP: Example of OFRP Scheduling (con't)



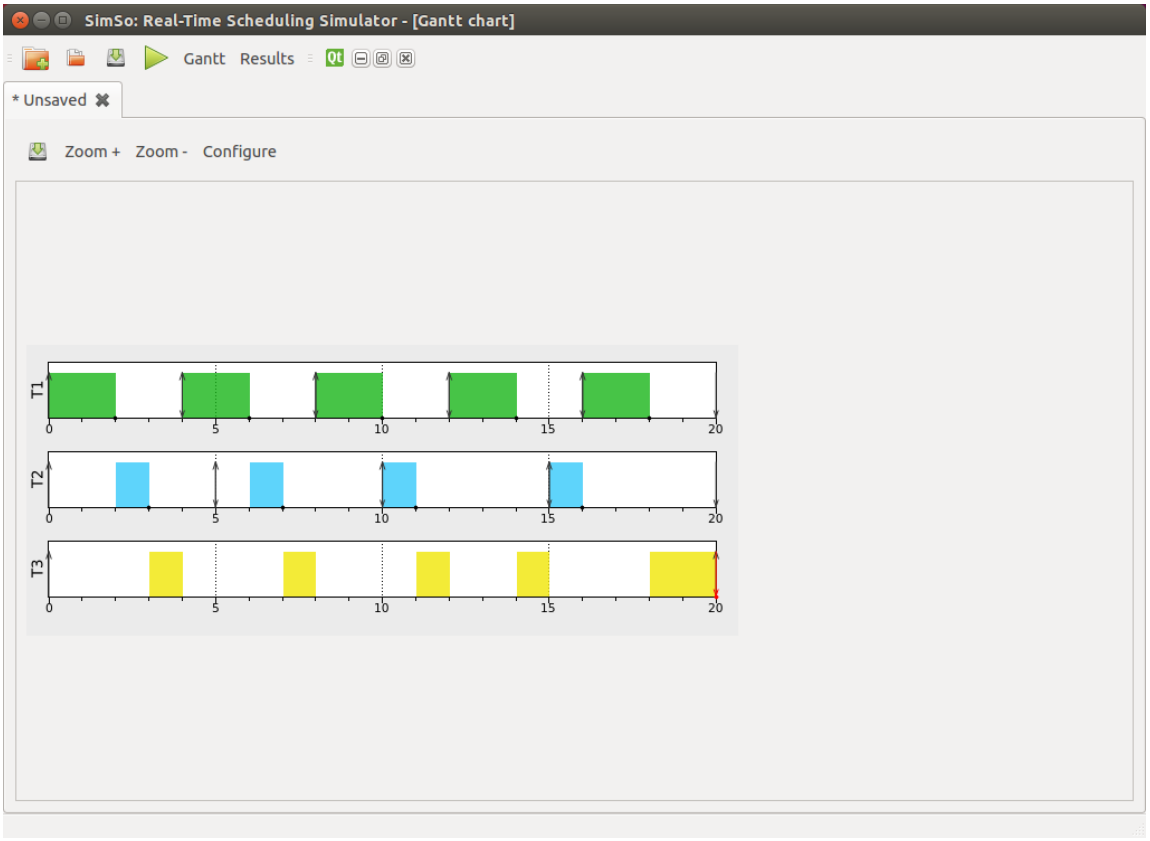
(c) Number of Processors - Uniprocessor

Figure 8.4: SimSo-PFRP: Example of OFRP Scheduling (con't)



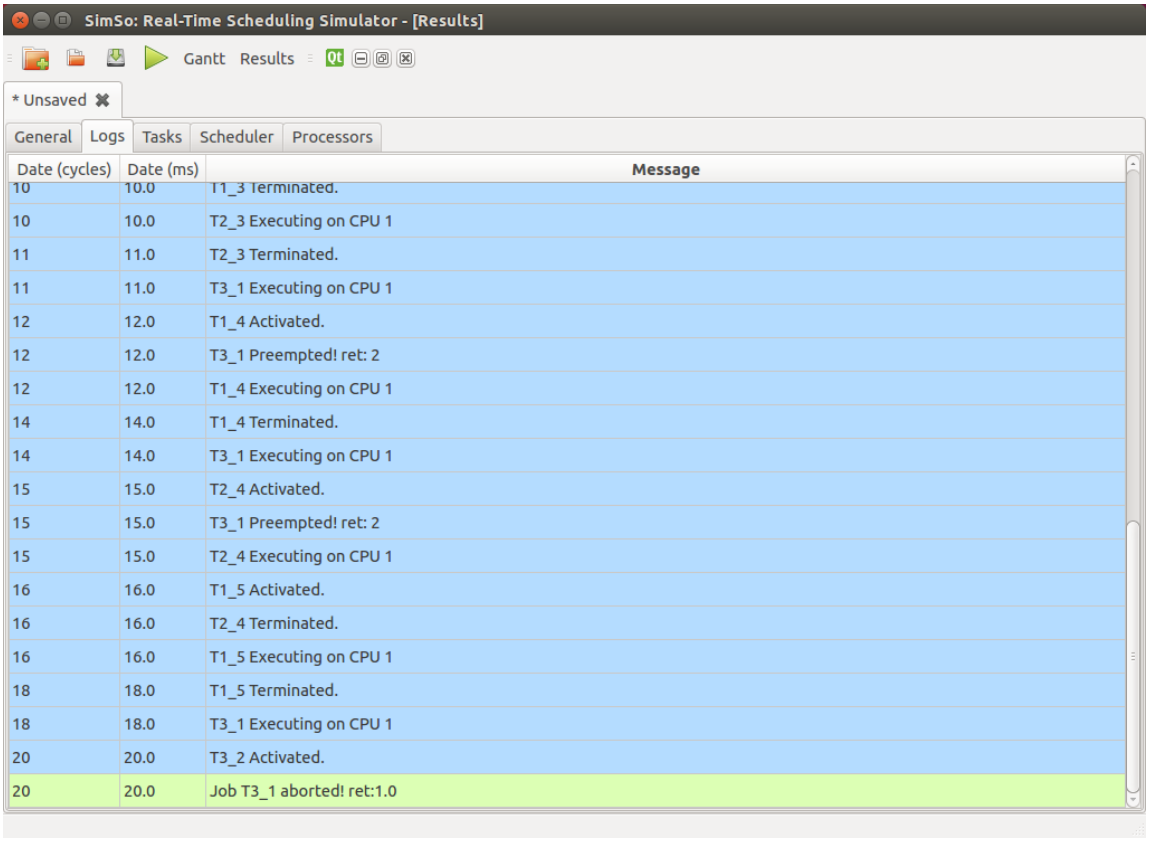
(d) Taskset

Figure 8.4: SimSo-PFRP: Example of OFRP Scheduling (con't)



(e) Scheduling Results - Gantt

Figure 8.4: SimSo-PFRP: Example of OFRP Scheduling (con't)



(f) Scheduling Results - Log

Figure 8.4: SimSo-PFRP: Example of OFRP Scheduling (con't)

shown in Fig.8.4.(e) and (f). Fig.8.4.(e) is a graphic display of the scheduling, and Fig.8.4.(f) shows the log of events. We can see that τ_3 misses its deadline at time instant 20 which is the end of the hyperperiod of experimental tasksets, hence the experimental taskset is correctly deemed to be unschedualable.

Compared to OFRP, InterA and JIDA are more complicated in implementation since they have multiple candidate modes to select one at in functions such as *on_activate* and *on_execute*.

Fig.8.5 and Fig.8.6 show code fragments accordingly.

The source code of SimSo-FRP can be found at https://github.com/jeffreyzou/simso.

```
    def on_execute(self, job):
    if self.on_preempted_date[job] < self.sim.now():
        self.num_preempted[job] += 1

        # calculate jid
        unique_tasks = set(job.task.other_tasks)
        jid = len(unique_tasks)
        # new computation time based on jid (and self.executed[job])
        # c = func(jid)
        # self.current_wcet[job] = c

        # clear its own other_tasks list
        #job.task.other_tasks.clear()   #python 3.3+
        #del job.task.other_tasks[:] # a little slower
        job.task.other_tasks[:] = []

        # update itself to others
        for t in self.sim.task_list:
            if t != job.task:
                t.other_tasks.append(job.task)

        self.executed[job] = 0

    self.on_execute_date[job] = self.sim.now()
```

Figure 8.5: JIDA of SimSo-PFRP

```
    def on_execute(self, job):
    if self.on_preempted_date[job] < self.sim.now():
        self.num_preempted[job] += 1

        # new computation time based on number of preemptions (and self.executed[job])
        # c = func(job.get_num_premptions)
        # self.current_wcet[job] = c

        self.executed[job] = 0

        self.on_execute_date[job] = self.sim.now()

    def on_preempted(self, job):
        self.update_executed(job)
    self.on_preempted_date[job] = self.sim.now()
```

Figure 8.6: InterA of SimSo-PFRP

Chapter 9

Conclusions

P-FRP is a promising exploration regarding to the needs of more complicated CPS systems. In this work, we reviewed existing researches on P-FRP task scheduling. We also reviewed and presented our work on this topic which listed as following:

• An efficient response time calculation algorithm and its implementation, LList-based RTA algorithm, that can be used on either classic or P-FRP task scheduling.

• Research on the worst case response time and schedulability analysis for the real-time software transactional memory-lazy conflict detection (STM-LCD) model.

• Feasibility interval research. We optimized research on the impact of task's release offsets to task schedulability and the schedulability test interval. Tighter feasibility intervals are found with respect to various task's release offsets.

• An non-work-conserving alternative model, Deferred Start, of the original AR model of P-FRP.

• Multi-mode task model for P-FRP systems. It is the first that multiple modes instead of single mode for a P-FRP task is proposed in order to reduce the scheduling cost, and thus improve the schedulability of P-FRP task systems.

• SimSo-PFRP. We presented a SimPy based task generator and scheduling simulator with rich algorithms available as an important infrastructure of P-FRP task scheduling research.

Based on our work, further multi-mode P-FRP task model will leads the way of more practical application of P-FRP model in real world. P-FRP-capable programming language or compiler/interpreter and computer architecture shall be great help and research area to P-FRP ecosystem. There are also more implementations and tests can be done on SimSo-PFRP.

Bibliography

[1] S. Altmeyer and N. Navet. Towards a declarative modeling and execution framework for real-time systems. ACM SIGBED Review 2016, 13(2):30-33.

[2] E. Amsden. A survey of functional reactive programming. http://www.cs.rit.edu/~eca7215/frp-independentstudy/Survey.pdf

[3] B. Andersson and E. Tovar. The utilization bound of non-preemptive rate-monotonic scheduling in Controller Area Networks is 25%. IEEE International Symp. on Industrial Embedded Systems (SIES), 2009, pp. 11-18.

[4] N. C. Audsley and A. Burns. On fixed priority scheduling, offsets and co-prime task periods. Information Processing Letters, 1998, 67(2): 65-69.

[5] N. C. Audsley, A. Burns, M. Richardson, and K. Tindell, A. Wellings. Applying new scheduling theory to static priority preemptive scheduling. Software Engineering Journal, 1993, 8(5): 284-292.

[6] N. C. Audsley, A. Burns, R. I. Davis, K. W. Tindell, and A. J. Wellings. Fixed priority scheduling: A historical perspective. *Real-Time Syst.*, 1995, 8(2-3):173–198.

[7] N. C. Audsley. Optimal priority assignment and feasibility of static priority tasks with arbitrary start times. TR-YCS-164, U. of York, 1991.

[8] N. C. Audsley. On priority-assignment in fixed priority scheduling. *Inform. Process. Lett.*, 2001, 79(1):39–44.

[9] T. P. Baker. Stack-based scheduling of real-time processes. Real-Time Systems, 1991, 3(1): 67-100.

[10] S. Baruah, D. Chen, S. Gorinsky, and A. Mok. Generalized multiframe tasks. Real-Time Systems, 1999, 17(1):5-22.

[11] S. K. Baruah and S. Chakraborty. Schedulability analysis of non-preemptive recurring real-time tasks. Parallel and Distributed Processing Symposium, International, 0:149, 2006

[12] S. Baruah, R. Howell, and L. Rosier. Feasibility problems for recurring tasks on one processor. *Theoretical Computer Sci.*, 1993, 118(1):3–20.

[13] S. K. Baruah and A. Burns. Fixed-priority scheduling of dual-criticality systems. In *Real-Time Networks and Systems (RTNS) 2013*, pp. 173–181.

[14] S. Baruah. The limited-preemption uniprocessor scheduling of sporadic task systems, ECRTS, 2005, pp. 137-144.

[15] A. Bastoni, B. B. Brandenburg, J. H. Anderson, Cache-related preemption and migration delays: Empirical approximation and impact on schedulability. In Proceedings of the 6th International Workshop on Operating Systems Platforms for Embedded Real-Time Applications, 2010, pp. 33-44.

[16] C. Belwal and A. M. K. Cheng. A Utilization based Sufficient Condition for P-FRP. 9th IEEE/IFIP Int'l Conf. on EUC, 2011, pp. 237-242.

[17] C. Belwal and A. M. K. Cheng. Determining actual response time in P-FRP. ACM PADL 2011, pp 250-264.

[18] C. Belwal and A. M. K. Cheng. Determining actual response time in P-FRP using idle-period game board. ISOR, 2011, pp. 136-43.

[19] C. Belwal and A. M. K. Cheng. Lazy versus eager conflict detection in software transactional memory: A real-time schedulability perspective. IEEE Embedded Systems Letters, 2011, 3(1): 37-41

[20] C. Belwal and A. M. K. Cheng. On priority assignment in P-FRP. RTAS 2010 WiP Session.

[21] C. Belwal and A. M. K. Cheng. Optimal priority assignments in P-FRP. TR-UH-CS-11-03, University of Houston, 2011.

[22] C. Belwal and A. M. K. Cheng. Partitioned scheduling of P-FRP in symmetric homogenous multiprocessors. IEEE/IFIP 9th International Conf. on Embedded and Ubiquitous Computing (EUC), 2011, pp. 47-54.

[23] C. Belwal and A. M. K Cheng. Reducing the number of preemptions in P-FRP. IEEE RTSS 2010 WiP Session.

[24] C. Belwal and A. M. K. Cheng. Schedulability analysis of transactions in software transactional memory using timed automata. IEEE 10th International Conf. on Trust, Security and Privacy in Computing and Communications (TrustCom), 2011, pp.1091-1098.

[25] C. Belwal and A. M. K. Cheng. Scheduling conditions for real-time software transactional memory. IEEE Embedded Systems Letters, 2011, 3(3): 93-96.

[26] C. Belwal, A. M. K. Cheng, and B. Liu. Feasibility interval for the transactional event handlers of P-FRP. Journal of Computer and System Sciences, 2013, 79(5): 530-541.

[27] C. Belwal, A. M. K. Cheng, and Y. Wen. Response time bounds for event handlers in the priority based functional reactive programming (P-FRP) paradigm. 2012 ACM Research in Applied Computation Symp., 2012, pp. 282-287.

[28] C. Belwal, A. M. K. Cheng, and Y. Wen. Time petri nets for schedulability analysis of the transactional event handlers of P-FRP. 2012 ACM Research in Applied Computation Symp., 2012, pp. 257-262.

[29] G. Bernat. Response time analysis of asynchronous real-time systems. *Real-Time Syst.*, 2003, 25(2-3):131–156.

[30] E. Bini and G. C. Buttazzo. Measuring the performance of schedulability tests. Real-Time System, 2005, 30(1-2):129-154.

[31] E. Bini and S. K. Baruah. Efficient computation of response time bounds under fixed-priority scheduling. In *RTNS 2007*, 2007, pp. 95–104.

[32] E. Bini and G. C. Buttazzo. The space of rate monotonic schedulability. In *IEEE RTSS 2002*, pp. 169–178.

[33] E. Bini and G. C. Buttazzo. Schedulability analysis of periodic fixed priority systems. *IEEE Trans. on Computers*, 2004, 53(11):1462–1473.

[34] A. Biondi, A. Melani, M. Marinoni, M. D. Natale, and G. Buttazzo. Exact interference of adaptive variable-rate tasks under fixed-priority scheduling. In Euromicro Conference on Real-Time Systems (ECRTS), 2014, pp. 165-174.

[35] V. Bonifaci, H. L. Chan, A. M. Spaccamela, and N. Megow. Algorithms and complexity for periodic real-time scheduling. *ACM Tran. on Algor.*, 2012, 9(1):A6:1–19.

[36] V. Bonifaci, A. Marchetti-Spaccamela, N. Megow, and A. Wiese. Polynomial-time exact schedulability tests for harmonic real-time tasks. In *IEEE RTSS 2013*, 2013, pp. 236–245.

[37] R. J. Bril, J. J. Lukkien, and W. F. J. Verhaegh. Worst-case response time analysis of real-time tasks under Fixed-priority scheduling with deferred preemption. Real-Time Systems, August 2009, 42(1-3):63-119.

[38] A. Burns and A.J. Wellings. Real-time systems and programming languages. Pearson Education, 4th edition, 2009.

[39] A. Burns. Is audsley's scheme the most expressive optimal priority assignment algorithm? In *the 8th Real-Time Scheduling Open Problems Seminar (RTSOPS)*, July 2013, pp. 8–11.

[40] G. Buttazzo and A. Cervin. Comparative assessment and evaluation of jitter control methods. International Conference on Real-Time and Network Systems, 2007.

[41] G. C. Buttazzo, E. Bini, and D. Buttle. Rate-adaptive tasks: Model, analysis, and design issues. In Design, Automation and Test in Europe Conference and Exhibition (DATE), 2014, pp. 1-6.

[42] G. C. Buttazzo, M. Bertogna, and Gang Yao. Limited preemptive scheduling for real-time systems. IEEE Transactions on Industrial Informatics, 2013, 9(1):3-15.

[43] G. C. Buttazzo. Rate monotonic vs. EDF: Judgment day. *Real-Time Syst.*, 2005, 29(1):5–26.

[44] Y. Cai and M. C. Kong. Nonpreemptive scheduling of periodic tasks in uni- and multiprocessor systems. Algorithmica, 1996, vol. 15, no. 6, pp. 572-599.

[45] M. Chéramy, P. Hladik and A. Déplanche. SimSo: A Simulation Tool to Evaluate Real-Time Multiprocessor Scheduling Algorithms. 5th International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS), Jul 2014, Madrid, Spain. pp. 6-p, 2014.

[46] K. R. Christoffersen and A. M. K. Cheng. Model-based design: Antilock brake system with priority-based functional reactive programming. RTSS 2013 WiP Session.

[47] A. Church. An unsolvable problem of elementary number theory. American Journal of Mathematics 58, pp.345-363, 1936.

[48] Courtney, A. Frappe: Functional reactive programming in Java. In Practical Aspects of Declarative Languages Springer. 2001, pp. 29-44.

[49] E. Czaplicki, S. Chong. Asynchronous functional reactive programming for GUIs. In ACM SIGPLAN Notices (Vol. 48, pp. 411-422). ACM, 2013.

[50] X. Dai, G. Hager, J. Peterson. Specifying behavior in C++. ICRA 2002, pp. 153-160.

[51] S. Davari and S. K. Dhall. An on-line algorithm for real-time tasks allocation. Proceedings of IEEE RTSS 1986, pp.194-200.

[52] J. I. David, K. G. Larsen and A. Skou. Model-based framework for schedulability analysis using UPPAAL 4.1. In Model-Based Design for Embedded Systems, ed. Gabriela Nicolescu and Pieter J. Mosterman, CRC Press, 2010, pp. 93-119.

[53] R. I. Davis and M. Bertogna. Optimal Fixed priority scheduling with deferred preemption. RTSS 2012, pp. 39-50.

[54] R. I. Davis, T. Feld, V. Pollex, and F. Slomka. Schedulability tests for tasks with variable rate-dependent behaviour under fixed priority scheduling. In Real-Time and Embedded Technology and Applications Symposium (RTAS), 2014, pp. 51-62.

[55] R. I. Davis and A. Burns. A survey of hard real-time scheduling for multiprocessor systems. ACM Comp. Surv. vol. 43, 2011, pp. 35:1-44.

[56] R. I. Davis and A. Burns. Response time upper bounds for fixed priority real-time systems. In *IEEE RTSS 2008*, 2008, pp. 407–418.

[57] R. I. Davis. A review of fixed priority and EDF scheduling for hard realtime uniprocessor systems. ACM SIGBED Review, 2014, 11(1):8-19.

[58] C. Demetrescu, I. Finocchi, A. Ribichini. Reactive Imperative Programming with Dataflow Constraints. In ACM SIGPLAN Notices, 2011, Vol. 46, pp. 407-426.

[59] J. S. Deogun and M. C. Kong. On periodic scheduling of time-critical tasks. IFIP World Computer Congress, 1986, pp. 791-796.

[60] M. L. Dertouzos and A. K. Mok. Multiprocessor scheduling in a hard real-time environment. IEEE Trans. Softw. Eng. 15(12), 1989, pp. 1497-1506.

[61] S. K. Dhall and C. L. Liu. On a real-time scheduling problem. Operation Research, 1978, 26(1), pp.127-140.

[62] F. Eisenbrand and T. Rothvoss. Static-priority real-time scheduling: Response time computation is NP-hard. In *IEEE RTSS 2008*, 2008, pp. 397–406.

[63] C. Ekelin. Clairvoyant non-preemptive EDF scheduling. ECRTS, 2006, pp. 23-32.

[64] C. Elliott, P. Hudak. Functional Reactive Animation. ICFP, 1997, 32(8): 263-273.

[65] G. Gardey, D. Lime, M. Magnin, and O.H. Roux. Romeo: A tool for analyzing time petri nets. Lecture Notes in Computer Science 3576 (17th International Conf. on CAV 2005), Springer, 2005, pp. 418-423.

[66] L. George, N. Rivierre, M. Spuri. Preemptive and Non-Preemptive Real-Time UniProcessor Scheduling. Research Report RR-2966, INRIA, 1996.

[67] J. Goossens and R. Devillers. The non-optimality of the monotonic priority assignments for hard real-time offset. Real-Time Systems, 1997, 13(2): 107-126.

[68] K. S. Hong and J. Y. T. Leung. On-line scheduling of real-time tasks. In Proceedings of the Real-Time Systems Symposium, 1988, pp. 244-250.

[69] K. S. Hong and J. Y. T. Leung. On-line scheduling of real-time tasks. IEEE Trans. Comp. 41, 1992, pp. 1326-1331.

[70] W. A. Horn. Some simple scheduling algorithms. Naval Res. Logist. Quart. 21, 1974, pp. 177-185

[71] http://projects.laas.fr/simso/

[72] http://simpy.readthedocs.org/

[73] http://www.schneider-electric.com/

[74] https://github.com/gelisam/frp-zoo

[75] https://typesafe.com/company/casestudies, 09/24/2014.

[76] https://www.qt.io/

[77] Z. Hu, J. Hughes, M. Wang. How functional programming mattered. Natl Sci Rev (2015)2(3):349-370.

[78] W. Huang and J. Chen. Techniques for schedulability analysis in mode change systems under fixed-priority scheduling. IEEE RTCSA 2015, pp. 176-186

[79] P. Hudak, J. Hughes, S. P. Jones, P. Wadler. A history of Haskell: being lazy with class. Proceedings of ACM HOPL III, 12-1-12-55, 2007.

[80] K. Jeffay, D. F. Stanat, and C. U. Martel. On non-preemptive scheduling of period and sporadic tasks. IEEE Symposium on Real-Time Systems, 1991, pp. 129-139.

[81] Y. Jiang, Q. Zhou, X. Zou, and A. M. K. Cheng. Minimal Schedulability Testing Interval for Real-Time Periodic Tasks with Arbitrary Release Offsets. IEEE ICESS 2014, pp. 611-614.

[82] Y. Jiang, A. M. K. Cheng, and X. Zou. Schedulability Analysis for Real-Time P-FRP Tasks Under Fixed Priority Scheduling. RTCSA 2015, pp. 31-40.

[83] M. Joseph and P. Pandya. Finding response times in a real-time system. Computer Journal, 1986, 29(5): 390-395.

[84] R. Kaiabachev, W. Taha, and A. Zhu. E-FRP with priorities. ACM EMSOFT 2007, pp.221-230.

[85] Z. Kazemi, A. M. K. Cheng. A Scratchpad Memory-Based Execution Platform for Functional Reactive System and its Static Timing Analysis. RTAS 2015 WiP session, pp. 176-181.

[86] J. Kim, K. Lakshmanan, and R. R. Rajkumar. Rhythmic tasks: A new task model with continually varying periods for cyber-physical systems. In International Conference on Cyber-Physical Systems (ICCPS), pages 55-64, 2012.

[87] S. Kleene. A theory of positive integers in formal logic. American Journal of Mathematics 57, 1935, pp.153-173 and 219-244.

[88] S. Krishnamurthi(2012). Programming Languages: Application and Interpretation (Second Edition.). Providence, 2012.

[89] J. Lehoczky, L. Sha, and Y. Ding. The rate monotonic scheduling algorithm: Exact characterization and average case behavior. In *IEEE RTSS 1989*, 1989, pp. 166–171.

[90] J. P. Lehoczky. Fixed Priority Scheduling of Periodic Task Sets with Arbitrary Deadlines. RTSS, 1990, pp. 201-209.

[91] J. Y. T. Leung and J. Whitehead. On the complexity of fixed-priority scheduling of periodic, real-time tasks. Performance Evaluation (Netherlands), 1982, 2(4), pp. 237-250.

[92] J. Y. T. Leung and M.L. Merrill. A note on preemptive scheduling of periodic, real-time tasks, Information Processing Letters, 1980, 11(3): 115-118.

[93] W. Li, K. Kavi, and R. Akl. A non-preemptive scheduling algorithm for soft real-time systems. Computers and Electrical Engineering, 2007, vol. 33, no. 1, pp. 12-29.

[94] C. L. Liu and L. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. Journal of ACM, 1973, 20(1): 46-61.

[95] C. D. Locke, D. R. Vogel, and T. J. Mesher. Building a predictable avionics platform in Ada: A case study. IEEE RTSS 1991, pp. 181-189.

[96] W. C. Lu, K. J. Lin, H. W. Wei, and W. K. Shih. Period-dependent initial values for exact schedulability test of rate monotonic systems. In *Int'l Parallel and Distributed Process. Symp.(IPDPS)2007*, pp. 1–8.

[97] J. Lv, X. Zou, A. M. K. Cheng, and Yu Jiang. Using Linked List in Exact Schedulability Tests for Fixed Priority Scheduling. RTAS 2016 WiP Session, pp. 1-1.

[98] J. Manson, J. Baker, A. Cunei, S. Jagannathan, M. Prochazka, B. Xin, and J. Vitek. Preemptible atomic regions for real-time Java. RTSS 2005, pp. 62-71.

[99] A. Medeiros. Dynamics of Change: Why Reactivity Matters. Tame the dynamics of change by centralizing each concern in its own module. Comm. of the ACM, October 2016, Pages 42-46.

[100] A. K. Mok. Fundamental design problems of distributed systems for the hard-real-time environment. 1983.

[101] C. Monsanto. Liftless Functional Reactive Programming. The 21st International Symposium on Implementation and Application of Functional Languages (IFL), 2009

[102] M. Nasri and G. Fohler. Non-Work-Conserving Scheduling of Non-Preemptive Hard Real-Time Tasks Based on Fixed Priorities. Real-Time Network and Systems (RTNS'15), 2015, pp. 309-318.

[103] M. Nasri and M. Kargahi. Precautious-RM: a predictable non-preemptive scheduling algorithm for harmonic tasks. Real-Time Systems, vol. 50, no. 4, 2014, pp. 548-584.

[104] M. Nasri, S. Baruah, G. Fohler, and M. Kargahi. On the optimality of EDF and RM for non-preemptive real-time harmonic tasks. Real-Time Network and Systems (RTNS'14), 2014, pp. 331-340.

[105] T. Nguyen, P. Richard, and E. Bini. Approximation techniques for response-time analysis of static-priority tasks. *Real-Time Syst.*, 2009, 43(2):147–176.

[106] Y. Oh and S. Son. Fixed-priority scheduling of periodic tasks on multiprocessor systems. Technical Report. UMI Order Number: CS-95-16., University of Virginia, 1995.

[107] S. H. Oh and S. M. Yang. A modified least-laxity-first scheduling algorithm for real-time tasks. In Proc. of RTCSA, 1998, pp. 31-36.

[108] M. Park and H. Park. An efficient test method for rate monotonic schedulability. *IEEE Trans. on Computers*, 2014, 63(5):1309–1315.

[109] M. Park. Non-preemptive fixed priority scheduling of hard real-time periodic tasks. International Conference on Computational Science, 2007, pp. 881-888.

[110] J. Peterson, P. Hudak, A. Reid, and G. D. Hager. FVision: A declarative language for visual tracking. In *Practical Aspects of Declarative Languages (PADL) 2001*, pp. 304–321.

[111] P. Pillai and K. G. Shin. Real-time dynamic voltage scaling for low power embedded operating systems. In Proc. of SOSP 01, 2001, 35(5): 89-102.

[112] J. Ras and A. M. K. Cheng. Response time analysis for the abort-and-restart task handlers of the priority-based functional reactive programming (P-FRP) paradigm. IEEE RTCSA 2009, pp. 305-314.

[113] J. Ras and A. M. K. Cheng. Response time analysis of the abort-and-restart model under symmetric multiprocessing. IEEE 10th Int'l Conf. on Computer and Information Technology (CIT), 2010, pp. 1954-1961.

[114] J. Real and A. Crespo. Mode change protocols for real-time systems: A survey and a new proposal. Real-time systems, 2004, 26(2):161-197.

[115] L. Sha, R. Rajkumar, J. Lehoczky, and K. Ramamritham. Mode change protocols for priority-driven preemptive scheduling. Real-Time Systems,1(3):243-264, 1989.

[116] L. Sha, R. Rajkumar, and J. P. Lehoczky. Priority inheritance protocols: An approach to real-time synchronization. IEEE Transactions on Computers, 1990, 39(9): 1175-1185.

[117] L. Sha, T. Abdelzaher, K. E. Årzén, A. Cervin, T. P. Baker, A. Burns, G. Buttazzo, M. Caccamo, J. P. Lehoczky, and A. K. Mok. Real time scheduling theory: A historical perspective. *Real-Time Syst.*, 2004, 28(2-3):101–155.

[118] M. Stigge, P. Ekberg, N. Guan, and W. Yi. The digraph real-time task model. In Real-Time and Embedded Technology and Applications Symposium (RTAS), 2011, pp. 71-80.

[119] W. Taha, P. Hudak, and Z. Wan. Directions in functional programming for real-time applications. In Embedded Software. Springer, 2001, pp. 185-203.

[120] K. W. Tindell, A. Burns, and A. J. Wellings. Mode changes in priority preemptively scheduled systems. In Real-Time Systems Symposium, 1992, pp. 100-109.

[121] K. W. Tindell, A. Burns, and A. J. Wellings. An extendible approach for analysing fixed priority hard real-time tasks. *Real-Time Syst.*, 1994, 6(2):133–151.

[122] Z. Wan and P. Hudak. Functional reactive programming from first principles. ACM SIGPLAN PLDI 2000, pp. 242-252.

[123] H. C. Wong and A. Burns. Improved Priority Assignment for the Abort and-Restart (AR) Model. Technical Report YCS-2013-481, University of York, Department of Computer Science, 2013.

[124] H. C. Wong and A. Burns. Schedulability analysis for the abort-and-restart (AR) model. ACM 22nd International Conf. on Real-Time Networks and Systems (RTNS), 2014, pp. 119-127.

[125] H. C. Wong and A. Burns. Priority-based Functional Reactive Programming (P-FRP) using Deferred Abort. RTCSA 2015, pp. 227-236.

[126] Q. Zhou, Y. Li, X. Zou, A. M. K. Cheng, Y. Jiang. Worst Case Response Time and Schedulability Analysis for Real-Time Software Transactional Memory-Lazy Conflict Detection (STM-LCD). ACM SIGBED Review 2016, 13(2):14-19.

[127] X. Zou, A. M. K. Cheng, and Y. Jiang. Deferred Start: A Non-Work-Conserving Model for P-FRP Fixed Priority Task Scheduling. RTSS 2015 WiP Session, pp. 373-373.

[128] X. Zou, A. M. K. Cheng and Y. Jiang. A Non-Work-Conserving Model for P-FRP Fixed Priority Scheduling. 13th IEEE International Conference on Embedded Software and Systems in Chengdu, 2016

[129] X. Zou, and A. M. K. Cheng. Memory-aware Response Time Analysis for P-FRP Tasks. RTAS 2016 WiP Session, pp. 1-1.

[130] X. Zou, A. M. K. Cheng, and Y. Jiang. P-FRP Task Scheduling: A Survey. The first CPSWeek Workshop on Declarative Cyber-Physical Systems (DCPS 2016), IEEE, 2016, pp. 1-8.