January 2019

Parameter Assignment And Schedulability Analysis For Real-Time Multiframe Task Systems

Bo Peng
Wayne State University, bopeng.rt@gmail.com

PARAMETER ASSIGNMENT AND SCHEDULABILITY ANALYSIS FOR REAL-TIME MULTIFRAME TASK SYSTEMS

by

BO PENG

DISSERTATION

Submitted to the Graduate School,

of Wayne State University,

Detroit, Michigan

in partial fulfillment of the requirements

for the degree of

DOCTOR OF PHILOSOPHY

2019

MAJOR: COMPUTER SCIENCE

Approved By:

_____

Advisor Date

_____

_____

_____

_____

ACKNOWLEDGEMENTS

I would like to express my deepest appreciation to my adviser Dr. Nathan Fisher, the chairman of my dissertation committee. Dr. Fisher guided me on both research and teaching during these years. The qualities he brings to work is a model to me. Dr. Fisher is exceptional at providing everything I need to finish this thesis, from research ideas to correcting my writing sentence to sentence. I am grateful and will take the professional demeanor learnt from him with me to the rest of my career. As my teacher and mentor, he has taught me more than I could ever express here.

I would also like to extend my gratitude to Dr. Thidapat Chantem of the Department of Electrical and Computer Engineering at Virginia Tech, USA. She collaborated with our lab and helped me on my publications which directly contribute to this thesis.

I would like to extend my sincere thanks to the other members of my dissertation committee, Dr. Loren Schwiebert and Dr. Daniel Grosu of the Computer Science Department at Wayne State University, along with Dr. Marko Bertogna of the Department of Physics, Informatics and Mathematics at the University of Modena (Italy) for their suggestions and support during my doctoral journey.

I must also thank my family for their love and care which supported me during this arduous and great achievement. Lastly, this dissertation was supported in part by the US National Science Foundation (Grant Nos. CNS-1618185, CNS-1205338, CNS-0953585, IIS-1724227, CSR-1618979) and a grant from Wayne State University's Office of Vice President of Research.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1 INTRODUCTION

1.1 Real-Time Systems

Real-time and embedded systems, which span a broad scope of complexity from microcontrollers to highly complicated and distributed systems, require completion of computation and delivery of service in a timely fashion. The correctness of an operation (an operation is formally redefined later in Chapter 3) depends not only on its logical correctness, but also on the time in which the operation is performed. Examples of real-time systems include digital control, signal processing, telecommunication systems, etc, which provide us numerous important services. The system in a car controls its engine, and brakes the car in time when we hit the brake pad. When we are sick, the system monitors our blood pressure, heartbeats, and many other relating statistics. Unlike nonreal-time systems (e.g., PCs), real-time systems which are often hidden from our view work efficiently and correctly in our daily life.

The most typical real-time systems are digital control systems which consist of sensors, actuators, and digital controllers [47]. Figure 1 shows a brake system of a self-driving car on the roads that require different speed limits. The car needs to brake when runs on a road with a smaller speed limit. The state of the brake system of the self-driving car is monitored by sensors and changed by actuators. The system reads the data from sensors and estimates the current state, and computes a control output based on the current state and the desired state (input). The output in turn actives actuators to let the system be closer to the desired state. In the brake system, the speed limit of the road is the input, and the difference between the current velocity of the car and speed limit is the output. The

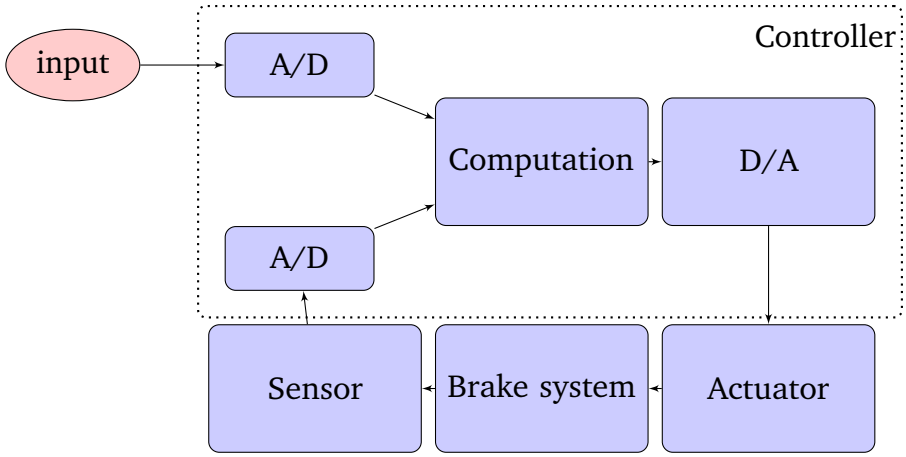


Figure 1: A simple digital control system. The input of the controller is the speed limit of the road and the output is the difference between the current velocity of the car and the speed limit. The output in turn actives actuators to let the speed of the car be closer to the speed limit of the road.

output, in turn, decelerates the car if current speed is larger than the limit.

The resultant controller is a sampled data system which samples sensors readings periodically and carries out its computation time in each period. This sequence of output thus in turn to activate the actuators. In a more complex system, scheduling such operations with time constraints must guarantee that the successful completion of execution does not exceed its deadline. In order to obtain the best performance, the parameters of each operation must be carefully calculated. The determination of the parameters depends on various input data and the different behavior of the processing environment. Before introducing our motivation and objective of this thesis, we introduce basic background information in Chapter 1.1.1 and 1.1.2.

1.1.1 Typical Parameters in Real-Time Systems

Before presenting the motivation and objective of this thesis, we briefly introduce several fundamental parameters [47] in real-time systems. Detailed definitions and explanations of the parameters are presented in Chapter 3.

- Worst-Case Execution Time (WCET):

  Execution time is the amount of time that is required to complete an operation. The execution time of an operation may vary, e.g., when the execution path of the operation (represented by the directed acyclic graph model) contains conditional branches. The execution time of the operation thus depends on the input data which may have different execution paths. Worst-case execution time is the maximum execution time among all paths of an operation.

- Deadline:

  The deadline (absolute) of an operation is the instant time by which its execution is required to be completed. The relative deadline is the amount of time which is relative to release time (the instant that the operation is available for execution). We consider hard deadlines in this thesis. A deadline is hard when the failure to meet the deadline is considered to be a failure.

- Period:

  A period characterizes an operation which repeatedly releases. In this work, we refer the period as the minimum inter-arrival time of two consecutive release time instants. The release time of an operation is the instant of time at which the operation is ready to execute.

A unit of work/operation typically executes recurringly in real-time systems. The parameters describe a recurring operation and the operation is called a *job* in real-time systems. The set of recurring jobs jointly provide a function called a *task*. A task with

parameters forms a real-time model. For example, a task of the sporadic task model consists of an execution time (WCET), a relative deadline, and a period. In this thesis, we analyze complex models of which a job contains subjobs and the subjobs (can form a subtask) have execution order requirements (priorities). The execution order describes the dependable features of subjobs. The subjobs/subtask can also have parameters such as execution time, deadline, etc. We have *task-wise* and *subtask-wise*[1] parameters in this thesis.

1.1.2 Schedulability Analysis in Real-Time Systems

In Chapter 1.1.1, we have introduced some typical parameters in real-time systems. These parameters (i.e., execution time, deadline, period, etc.) of tasks work as input to help decide whether the system is schedulable. A valid schedule is feasible if all tasks in a system complete executing by their deadlines. A system is schedulable using a specific algorithm if the algorithm always produces a feasible schedule for the system. A schedulability analysis (also known as schedulability test) validates whether all tasks in the system can meet all their deadlines when scheduled by a scheduling algorithm.

Schedulability analysis is important, especially in *hard real-time systems*. In hard real-time systems, all deadlines of tasks are hard and must be met. If the schedulability analysis of a system is not correctly launched and further causes a deadline miss, severe results may happen, e.g., the failure of the braking system in a car will cause the loss of money and threat for people's safety.

[1] In this thesis, we refer to an operation as a job which is an instance of a task, and a subjob as a frame of a job. We formally define them in Chapter 3.

1.2 Motivation

We have briefly introduced the certain background, basic parameters and schedulability analysis in real-time systems. Parameters can be considered as input to decide whether the system is schedulable. The effectiveness of the schedulability analysis hinges upon the precision of parameter estimates. In Chapter 1.2.1, we present the traditional ways that calculating/setting parameters in real-time systems. As nowadays applications become more and more complex, we show the difficulties of calculating/setting parameters and the opportunities on working the parameter adaptation problem in Chapter 1.2.2. In Chapter 1.2.3, we introduce the specific area we are working, how we assign the parameters, and why we consider using such methods.

1.2.1 Traditional Ways in Getting Parameters

- WCET:

  The analysis of worst-case execution time is an import research branch of real-time systems. WCET, in general, is impossible to exactly calculated; Otherwise, the halting problem can be solved [78]. Even the form of programming in real-time systems is restricted (programs are guaranteed to stop and iterations of loops are bounded), WCET is also hard to get due to worst-case input, in general, is not known beforehand and hard to derive. Due to this reason, researchers estimate WCET by measurement-based and abstraction-based approaches. The measurement-based approach is based on a subset of test cases thus underestimates WCET. The abstraction-based approach is based on the abstract of tasks, which loses detailed information and thus overestimates

WCET.

- Deadline:

  Task deadlines in real-time systems are traditionally calculated from the data that sensors and actuators discover. For example, a self-driving car with a constant speed cannot stop immediately (in an arbitrarily small time) when the car detects an obstacle; instead, the car utilizes a sensor to periodically monitor the distance to the obstacle and an actuator to brake the car. The deadline for this task can be calculated from the speed of the car, the distance between the obstacle and the maximum deceleration of the braking system of the car. This time is also a constraint in turn to the response time of the sensors which discover the obstacle and activate the brake.

- Period: A period can be considered as, e.g., the multiplicative inverse of a sample rate. The period is usually a user-defined parameter according to the requirements of a job. For example, there are sensors and actuators to control the temperature of a room. The system samples and reads the temperature every 10 seconds which is considered as the period of the temperature monitor process.

There is some work [18, 22] that select deadlines and periods under certain constraints, but WCET is usually assumed to be immutable once assigned. Details are introduced in Chapter 2.

1.2.2 Difficulties and Opportunities

We have introduced the basic calculations/settings of some basic parameters. The calculation of parameters in real-time systems can be quite complicated since nowadays applications are becoming more and more complex. The mentioned calculation of car brak-

ing is straightforward; however, the scenario becomes more complex when a task contains image capture, motion detection, stereovision, object recognition, and path planning. These subtasks must work in sequence, e.g., the analysis for the pictures starts after capturing pictures and the analysis must finish before path planning. The system needs to maintain the end-to-end latency of such tasks to assign appropriate subtask periods and relative deadlines. In practice, it has been shown that there are more than one hundred engine control units (ECUs), and more than twenty million lines of code in a typical modern car system [19]. There are fifty to three hundred functions in a task with varying worst-case execution time (WCET), and many shared, global data in a ECU software. There are a set of main challenges [19] of schedulability analysis in such complex applications:

1. There are lots of mode-dependent behaviors.

2. The task periods are randomly generated.

3. Many functions are implemented on top of task self-suspensions.

Due to these challenges, we aim to develop a flexible model to let a subset of parameters to be flexible to be chosen in ranges, and develop efficient algorithms to select the parameters and analyze the schedulability of all tasks.

There is also a large market along these mentioned problems [19]. There are seventy-five million cars produced each year. In Europe, there are around forty-two billion dollars of car sales, and twelve million people are employed for making cars. The number of engine control units (ECUs) also increases rapidly. There are more than one hundred ECUs in a modern car produced in Year 2012, compared to less than five ECUs in a car produced in Year 1980 [19]. The number will continue increasing as future cars will certainly have

more functions, e.g., self-driving, self-docking, and interactive relation with people, etc. In this thesis, we develop flexible models and efficient algorithms to tackle these scheduling challenges on such complex systems.

1.2.3 Flexibly Assigning Parameters

Combining the economic opportunities and corresponding challenging scheduling problems, we aim to develop general models that have the flexibility on choosing the parameters of complex tasks such as self-suspension tasks and end-to-end flows in uniprocessor systems and distributed systems, respectively.

We also develop algorithms that jointly considering the selection of a subset of parameters (i.e., frame relative deadlines and periods) and the schedulability analysis of the system, by linear programming (LP) and mixed-integer linear programming (MILP). The advantage of the combination instead of two separate steps is detailed in Chapter 2.

1.3 Objective

In this thesis, we aim to create general flexible models that are capable of optimally selecting the parameters under EDF (earliest deadline first) scheduling in both uniprocessor and distributed systems. We focus on the tasks which consist of frames that execute in order. Such tasks represent many important applications like transactions (end-to-end flows) and self-suspending tasks. Under such flexible frameworks, we aim to develop efficient algorithms that select parameters (e.g., deadlines and periods) and generate schedulability analysis jointly.

1.3.1 Thesis

The thesis of this document is:

The schedulability ratio of task sets can be improved in parameter-adaptable models than traditional parameter-fixed models. That is, flexible parameters can be tuned to improve system schedulability. Efficient algorithms for selecting these parameters can utilize the linear and mixed-integer linear programming methods. Such algorithms can efficiently select parameters and schedule a task set at the same time. The enhanced schedulability benefits can be evaluated and verified using randomly generated datasets. Such a universal model can be applied in many areas such as scheduling self-suspending tasks in uniprocessor systems and end-to-end flows in distributed systems, respectively.

1.4 Summary of Contributions

The main contributions of this thesis are listed as follows:

1. We propose new models GMF-PA (the generalized multiframe model with parameter adaptation) and dGMF-PA (distributed GMF-PA) which permit flexible selections of frame relative deadlines and periods under EDF scheduling.

2. In uniprocessor systems, we develop a parameter selection algorithm based on MILP that selects frame deadlines and periods in the GMF-PA model. We prove that the algorithm is a necessary schedulability test. When parameters are assumed to be integers, the algorithm is a sufficient and necessary schedulability test, i.e., an exact test. We develop an MILP-based approximation algorithm and prove that this method is a sufficient schedulability test.

3. In distributed systems, we extend the algorithms used in uniprocessor systems to be

capable of selecting frame deadlines and periods for the dGMF-PA tasks.

4. We apply our parameter selection algorithm and its approximation algorithm on multiple-segment self-suspending tasks in uniprocessor systems. The system is also extended to an arbitrary-deadline system. We also apply our algorithms on end-to-end flows in distributed systems.

5. We prove that the speed-up factor of the MILP-based approximation algorithm is $1 + \epsilon$ with respect to the exact schedulability test of GMF-PA (dGMF-PA) tasks under EDF scheduling.

6. We give a case study that applies our MILP-based algorithms to a tracking robot car. A tracking task consists of several subtasks (e.g., motion detection, stereovision, object recognition, etc.) and we relax the assumption that some subtasks of a tracking task are independent. We extensively evaluate our algorithms by varying four parameters such as network bandwidth, database size, etc.

7. We give a concave approximation algorithm based on the MILP algorithm and prove the speed-up factor of the algorithm is $(1 + \delta)^2$ with respect to the exact schedulability test of GMF-PA tasks under EDF scheduling on uniprocessors. The positive constant δ is a user-defined constant which can be made arbitrarily close to zero.

8. Since there is no known tractable way to solve a concave programming problem, we develop a LP-based heuristic algorithm based on the concave approximation algorithm for GMF-PA tasks. The LP-based algorithm is an efficient schedulability test and can select frame parameters at the same time.

9. We apply the LP-based algorithm to schedule multiple-suspending tasks. To exploit the unique property of one-suspending tasks, as opposed to multi-suspending tasks, we present an improved heuristic algorithm for GMF-PA tasks.

10. We conduct extensive experiments and show that the LP-based algorithms with fixed numbers of iterations outperform previous work in terms of schedulability and average running time. The fixed numbers of iterations make the LP-based algorithms pseudo-polynomial (the input size depends on the maximum interval length [8]), which is more efficient than the MILP-based approach.

1.5 Organizations

The following table shows the next chapters of this thesis:

Table 1: Chapter Contribution Summary

Chapter #	Contribution
Chapter 2	Related work
Chapter 3	Models and definitions used in the rest of the thesis
Chapter 4	Parameter selection and schedulability analysis in uniprocessor systems
Chapter 5	A case study on a robot tracking system
Chapter 6	A linear programming based approximation algorithm
Chapter 7	Parameter selection and schedulability analysis in distributed systems
Chapter 8	Conclusion of this thesis
Chapter 9	List of publications

CHAPTER 2 RELATED WORK

In this chapter, we introduce the related work of our GMF-PA and dGMF-PA models. GMF-PA and dGMF-PA models extend the GMF model and we introduce the related models in Chapter 2.1. The traditional models, e.g., the GMF model, are considered to be fixed parameter models. That is, the parameters are fixed during the task specification time. We introduce the related work of flexible parameter models in Chapter 2.2. We introduce the applications such as self-suspension tasks and end-to-end flows in Chapter 2.3 and Chapter 2.4, respectively.

2.1 The Generalized Multiframe Model

The generalized multiframe model (GMF) was first introduced by Baruah et al. [8] to extend the sporadic task model and multiframe task model (MF) [52]. A sporadic task has an execution time, a deadline, and a period, which can be considered as a one-frame task model. An MF task has multiple frames each of which has its own execution time. All frames of a job have identical separation time (frame period) and relative deadline. The GMF model which extends these two models is more flexible. Each frame has its own execution time, frame period, and frame relative deadline. Frames in the MF and GMF models must execute in sequence. In the non-cyclic GMF task model [54], frames can execute out of order and thus reduce the pessimism of the modeling of software-defined radio [53]. The recurring real-time task (RRT) model [10] is a generalization of the GMF model to handle conditional codes. The non-cyclic recurring real-time task (non-cyclic RRT) model [7] can generalize all the models referred above. The digraph model [71] further generalizes the non-cyclic RRT model to allow arbitrary directed graph (allows

loops), and the feasibility problem on preemptive uni-processor systems remains tractable (pseudo-polynomial complexity with bounded system utilization). A complete review of the above models in uniprocessor systems is surveyed by Stigge and Yi [72].

There are many applications based on the MF and GMF models. Ding et al. [31] scheduled a set of tasks with the I/O blocking property under the MF model. Andersson [5] presented the schedulability analysis of flows in multi-hop networks comprising software-implemented Ethernet switches, according to the GMF model. Liu et al. [48] applied the MF task model and presented a sufficient schedulability analysis on messages over a CAN-based system with mixed message queues. Ekberg et al. [32] developed an optimal resource sharing protocol for the GMF model. The authors combined the protocol with EDF scheduling to optimal schedule GMF task sets with shared resources. Lipari and Bini [44] gave a component-based approach to consider the task allocation and parameter assignment of pipelines of tasks in multi-processor systems. Their frame deadline assignment algorithm is based on the minimization of the bandwidth required by the transactions, while our algorithm shown later is based on the system schedulability.

The GMF model has great advantages and been applied to multiple areas, as described earlier. However, current related models typically assume that parameters are fixed after task specification time. In the GMF-PA model (detailed in Chapter 3.2) which extends the GMF model, frame parameters are flexible under system constraints and can be chosen by the MILP-based approach in uniprocessor systems. The dGMF-PA model (detailed in Chapter 3.3) extends the GMF-PA model to be a flexible model in distributed systems. The dGMF-PA model can be applied to end-to-end flows in distributed systems. Similar flexible models, such as the parameter-adaptation model [22] and elastic model [18], are also used

in many applications.

2.2 Flexible Parameter Models

The GMF model has great advantages and is applied to multiple areas. However, its current related models assume that parameters are fixed after the task specification time. Instead, a subset of parameters is flexible to be chosen under frame constraints and task constraints (detailed in Chapter 3) in our GMF-PA and dGMF-PA models. We utilized the advantage of flexible models to make frame periods and deadlines be flexible to be chosen. The relationship between fixed parameter models and flexible parameter models are shown in Figure 2.

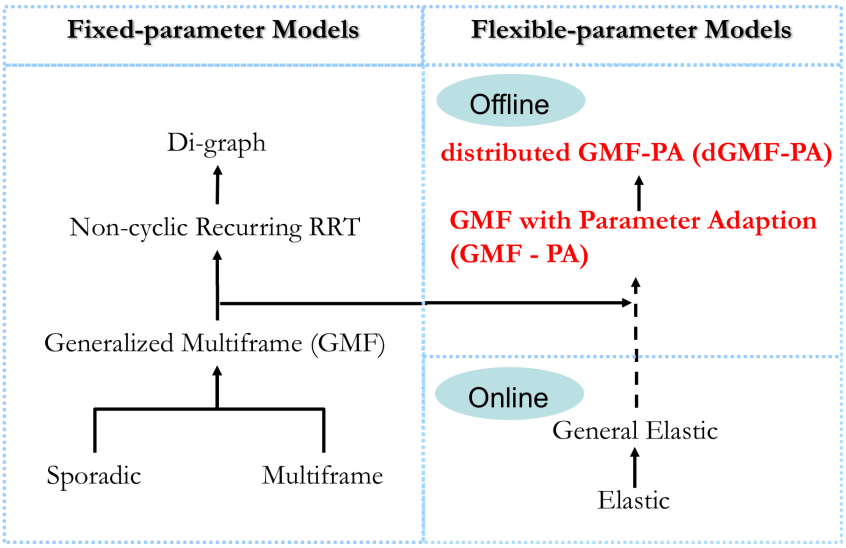


Figure 2: A solid arrow line between two models means generalization. For instance, the dGMF-PA model generalizes the GMF-PA model. The dashed arrow line between two models means partial generalization. For example, task periods are flexible in the elastic model but are fixed in the GMF-PA model. If task periods are fixed, the GMF-PA model generalizes the elastic model.

.

A new research direction of task models is motivated by control applications for combustion engines. Tasks can be triggered by specific crankshaft rotation angles and the angular

velocity of the engine. Task periods which depend on the angular velocity thus are variable. Buttazzo et al. [17] gave a sufficient utilization-based feasibility test on fixed-priority and EDF scheduling. Biondi et al. [13] gave an exact schedulability analysis under EDF scheduling, and showed the average performance gain of EDF over FP scheduling over a set of simulation results. It is interesting to consider mapping our flexible model to such task models in the future.

2.3 The Self-Suspending Task Model

A typical self-suspension task model [47] contains two computational frames separated by a self-suspending frame. After the first computational frame finishes, the job suspends executing the other computational frame until an external operation completes. The order of the frames is required and a task suspends itself to communicate with external devices, I/O operations, computation offloading, etc. We call such tasks *one-suspension* self-suspending tasks.

We apply the GMF-PA model in uniprocessor systems to self-suspending tasks in this thesis. Self-suspending tasks were first defined in real-time systems by Rajkumar [65] and the author gave a sufficient schedulability test for fixed-priority tasks[2]. Scheduling self-suspending tasks is NP-hard [66] in strong sense even a task system has only one self-suspension task. Nelissen et al. [55] also showed that the timing analysis of sporadic self-suspending tasks is also not easy, and they computed exact worst-case response times using mixed-integer linear programming (MILP) algorithm. Ekberg and Yi [34] have also shown that the schedulability test of implicit-deadline tasks under fixed-priority scheduling

[2]Chen and Brandenburg [26] further explained that the algorithm [65] may cause ever-increasing suspension lengths which incur deadline misses for self-suspending tasks.

in uniprocessor system is NP-hard, which indicates the scheduling of self-suspension tasks is also hard in such environment.

The traditional scheduling algorithms [47] which consider self-suspending delays as parts of computation times (i.e., the tasks are transformed to one-frame/sporadic tasks) are quit efficient, but sacrifice significant system capacity, especially when the delay of a self-suspension is large. A much accurate way to model the self-suspensions is to consider the self-suspensions as carry-in jobs [79] or jitters [55]. In this case, the scheduling algorithms which explicitly consider the delays of suspensions receive much attention on the tasks with at most one self-suspending frame and tasks with multiple self-suspending frames.

Due to the hardness of such scheduling problems, Chen and Liu [23] gave the fixed-relative-deadline (FRD) scheduling algorithm to improve the schedulability on sporadic self-suspending tasks on uniprocessor systems, and quantified the quality of their approach by analyzing the speed-up factors with respect to the optimal FRD scheduling and any arbitrary feasible scheduling algorithms. Chen et al. [25] gave a framework to analyze response time in fixed-priority systems for self-suspension tasks. Brüggen et al. [77] gave a greedy algorithm that assigns frame deadlines, and proved the speed-up factor of their algorithm. Later, they [76] proposed a hybrid self-suspension model which limits the upper bounds for parameters e.g., task execution time, task suspension time, the number of suspensions, etc.

Multiple self-suspending task models [39] can be divided into two main categories: the dynamic self-suspending task model and the multiple-segment self-suspending task model. Dynamic self-suspending tasks consider the maximum aggregate frame execution time and suspension delay among tasks, i.e., suspensions can take place at any time. Liu and Anderson

scheduled self-suspending tasks in soft [45] and hard [46] real-time multiprocessor systems. Chen et al. [27] gave a response time analysis framework for dynamic self-suspending tasks with fixed priorities. In contrast to the dynamic self-suspending task model, the multiple-segment self-suspending task model explicitly ensures that all frames must work in order. In a system where each task has multiple-segment self-suspending frames, Kim et al. [42] used a MILP-based algorithm to assign frame priorities and deadlines based on the response time analysis in fixed-priority scheduling systems. Later, Kim et al. [41] pointed out there is an error in the paper [42], and they correct it mainly in Sections 3.2 and 4.1 of the paper. We also create MILP-based algorithms to select frame parameters (See more details in Chapter 4.3). The difference is that our analysis is based on the time interval analysis in EDF-scheduled systems and their analysis is based on the response time analysis in fixed-priority scheduling systems. Thus, the scheduling algorithms using the MILP technique are different. Huang and Chen [39] proposed FRD scheduling on self-suspending tasks with fixed priorities. The multiple-segment self-suspending task model explicitly ensures that all frames work in order. In this thesis, we propose the GMF-PA model and parameter-adaptation algorithms that extend the FRD scheduling in arbitrary-deadline systems. We also apply the GMF-PA model to multiple-segment self-suspending tasks.

Chen et al. [27] showed that several papers of self-suspending task scheduling were flawed, and some results were unfortunately adopted in the literature of self-suspending task scheduling. We utilize the classic model GMF to develop schedulability analysis to ensure the correctness of our work at most. Some recent work. [14, 56] also addressed the flaws of self-suspension scheduling. Recent reviews [24, 28, 29] are given on scheduling self-suspension tasks under earliest-deadline-first (EDF) and fixed-priority (FP) scheduling.

Related Work of Self-Suspension Tasks				
Scheduling algorithm	Suspending frames per job			
	One segment		Multiple segments	
	Implicit	Explicit	Implicit	Explicit
EDF scheduled systems	Ridouard et al. [66] Chen et al. [27]	Chen and Liu [23] Brüggen et al. [76, 77]	Liu et al. [79] Chen et al. [27]	**This research: Chapters 4, 5, and 6**
Fixed priority scheduled systems	Rajkumar [65] Chen and Brandenburg [26] Ridouard et al. [66]	Huang and Chen [39] Chen et al. [25, 27] Ekberg and Yi [34]	Nelissen et al. [55] Liu and Anderson [45, 46]	Huang and Chen [39] Chen et al. [27]

Table 2: Related Work of Self-Suspension Tasks

Many applications have used the self-suspending task model. In mobile cloud computing, uploading a portion of work can both benefit from reducing the execution time and energy of tasks in mobile devices. The time taken by the cloud can be considered as a self-suspension delay of a task on the mobile side. Ahmed et al. [4] analyzed the effect of tuning network-centric parameters on runtime application migration. Based on optical character recognition (OCR) applications on android devices, they analyzed that the length of suspension delay is effected by the size of data, network bandwidth, etc. Nimmagadda et al. [57] analyzed the decision-making that whether a part of workload should be offloaded to a server in a tracking robot car. In Chapter 5, we relax an assumption made in the paper [57] and give extensive evaluations based on the case study.

Table 2 shows the classification of the related work of self-suspension task scheduling algorithms.

2.4 The Scheduling of End-to-End Flows

The dGMF-PA (distributed GMF-PA) model extends the GMF-PA model (see details in Chapter 3.2). The dGMF-PA model is used in distributed systems and can transform to the GMF-PA model when the number of processors is one. We apply the dGMF-PA model to end-to-end flows and compare the schedulability ratio and running time with the latest work.

The schedulability analysis of distributed real-time systems has received much attention. Most applications in distributed real-time systems can be modeled by end-to-end flows/end-to-end tasks/transactions in which subtasks/frames of a flow execute in a chainlike manner. The end-to-end tasks can be traced back to the job-shop model [49] in which a task consists of a chain of jobs. A job in this model is ready to execute when its preceding job completes executing. Based on the job-shop model, Bettati [11] first focused on the scheduling of end-to-end tasks to meet their end-to-end timing constraints in distributed systems. Schedulability analysis of such models has been proposed both for the fixed priority (FP) scheduling and earliest deadline first scheduling (EDF).

In the category of FP scheduling algorithms, Tindell and Clark [74] first proposed a holistic analysis, which was later improved by the offset-based analysis [58]. Such analysis calculates the worst-case response time of each subtask to set the offset and jitter of the succeeding subtask. The calculation is iterative. Li et al. [43] applied the holistic schedulability tests to an industrial scheduling tool of a TDMA software radio protocol, and shows the advantage and limits of the holistic analysis. The FP scheduling of end-to-end flows was further improved by the offset-based slanted technique [50] which exploits the

interdependencies among subtasks using offsets. Schlatow and Ernstthis [69] presented a busy-window analysis for end-to-end tasks in static-priority preemptive systems. Casini et al. [20] gave response time and scheduling analysis of parallel tasks with suspensions.

In the category of EDF scheduling algorithms, one-time deadline assignments such as PD [47] (proportional deadline assignment) and NPD [47] (normalized PD) are efficient but pessimistic on schedulability test. The deadline assignment is fixed after the task specification time. Buttazzo et al. [15] divided end-to-end flows to fixed segments, and the segment deadlines were assigned by the execution time distribution. Such one-time assignments are efficient; however, the assignments do not consider the resource competition of sub-jobs, which may fail to schedule a task system. Iterative assignment algorithms such as the offset-based analysis [59] were presented based on the similar analysis in FP scheduling. Pellizzoni and Lipari [60] provided new response time analysis and an iterative algorithm to improve the schedulability analysis. In the iterative-based algorithms, the deadline assignment of subtasks affects the offsets and jitters of themselves which in turn will affect the deadline assignment. In the category of online scheduling algorithms, Hong et al. [37] optimally assigned local frame deadlines of end-to-end flows given known release time of tasks. Hong et al. [38] extended their algorithms to assign the deadlines of DAG tasks. Rahni et al. [64] gave a survey of scheduling real-time transactions. Table 3 shows the classification of the related work of end-to-end flows scheduling algorithms.

There are also some other interesting algorithms in industry, e.g., the reservation algorithms [16] in a reservation server which assigns task priorities based on the system supply. Casini et al. [21] analyzed the response time and scheduling of chainlike tasks in Robot Operating Systems (specifically ROS 2). Another programming framework is

Related Work of End-to-End Flows			
Scheduling algorithm	Deadline assignment algorithm		
	Iterative	Non-iterative	
		Fixed assignment	Combined optimization
EDF scheduled systems	Palencia and Harbour [59],Pellizzoni and Lipari [60], Hong et al. [37, 38]	Bettati [11],Liu [47], Chen and Liu [23]	**This research: Chapter 7**
Fixed priority scheduled systems	Tindell and Clark [74],Palencia and Harbour [50, 58], Schlatow and Ernstthis [69], Li et al. [43], Casini et al. [20]	Future Work	Future Work

Table 3: Related Work of End-to-End Flows

OpenMP [3] and the tasks are scheduled with order constraints and synchronization. Sun et al. [73] proposed the response time analysis and scheduling of OpenMP tasks for parallel real-time systems on multi-cores.

In this thesis, we utilize the interface of demand bound functions [6] and mathematical programming to assign frame deadlines and analyze system schedulability. The iterative-based algorithms cannot easily compute a demand bound function during an interval length because the response time of a subtask depends on the end-to-end flows in all processors.

2.5 Summary

In this chapter, we present the related work of our flexible models GMF-PA and dGMF-PA. Such models in related work have numerous applications. In general, we combine the advantage of the parameter-flexible models and the state-of-the-art models to create our flexible models. For each model, we create a scheduling algorithm and its approximation

algorithm based on the MILP and LP algorithms. The evaluations in this thesis shows that our scheduling algorithms based on such flexible models can schedule more systems than fixed-parameter models (in which the parameters are fixed after task specification time).

CHAPTER 3 MODELS AND NOTATIONS

In this chapter, we review the GMF [8] model in Chapter 3.1. Based on the GMF model, we define our GMF-PA model in Chapter 3.2 in uniprocessor systems. The multiple-segment self-suspending task model and how to apply our GMF-PA model to the multiple-segment self-suspending tasks are presented in Chapter 3.2.1. Based on the GMF-PA model, we define our dGMF-PA model in Chapter 3.3 in distributed systems. The end-to-end flow model and how to apply our dGMF-PA model to end-to-end flows are presented in Chapter 3.3.1. At last, we conclude this chapter in Chapter 3.4.

3.1 The Generalized Multiframe Task Model

We mentioned in Chapter 1 that an operation is a unit of work. In real-time systems, a *job* is a unit of infinite recurring work done by computation and communication systems. The recurring jobs compose a *task*. A job has many properties, e.g., execution time, period, deadline, etc., that are introduced in Chapter 1. If we further explore the inner structure of a job, a job can be represented by more complex models such as the linear graph model or DAG (directed acyclic graph) model. The GMF model is a linear graph model in which sub-jobs[3] execute in sequences.

A GMF task [8] τ_i consists of a set of ordered frames and each frame ϕ_i^j has its own execution time E_i^j, relative deadline D_i^j, and frame separation time P_i^j. All frames of a task τ_i can be represented by the 3-vector tuple $(\overrightarrow{E_i}, \overrightarrow{D_i}, \overrightarrow{P_i})$ where $\overrightarrow{E_i}$=[E_i^0, E_i^1,..., $E_i^{N_i-1}$], $\overrightarrow{D_i}$=[D_i^0, D_i^1,..., $D_i^{N_i-1}$], and $\overrightarrow{P_i}$=[P_i^0, P_i^1,..., $P_i^{N_i-1}$]. The ℓ'th frame of task τ_i arrives at time a_i^ℓ, has deadline at $a_i^\ell + d_i^\ell$, and worst-case execution time e_i^ℓ. Since frames arrive in sequence, the ℓ'th frame corresponds to frame $\phi_i^{\ell \bmod N_i}$, and we have:

[3]We define a part of a job as a sub-job, and define a sub-job as a frame in the GMF model.

1. $a_i^{\ell+1} \geq a_i^{\ell} + P_i^{\ell \mod N_i}$

2. $d_i^{\ell} = D_i^{\ell \mod N_i}$

3. $e_i^{\ell} = E_i^{\ell \mod N_i}$

In this thesis, we consider that frame parameters (D_i^j and P_i^j) must satisfy the localized Monotonic Absolute Deadlines (l-MAD) property [8] to maintain frame execution order. That is, the absolute deadline of the j'th frame must be no later than the one of the $j+1$'th frame ($D_i^j \leq P_i^j + D_i^{(j+1) \mod N_i}$, $\forall i, j$). Figure 3 shows an example of the GMF model with the l-MAD property. The l-MAD property is widely used in systems which use first-in first-out (FIFO) scheduling for a shared resource. For example, a network can be seen as a shared resource and packets sent from a computational node to a network node follow FIFO scheduling. With this property, frames execute in sequence under EDF scheduling. In other words, $j+1$'th frame cannot start executing until the time when j'th frame finishes executing. We also require the l-MAD property in our GMF-PA model.

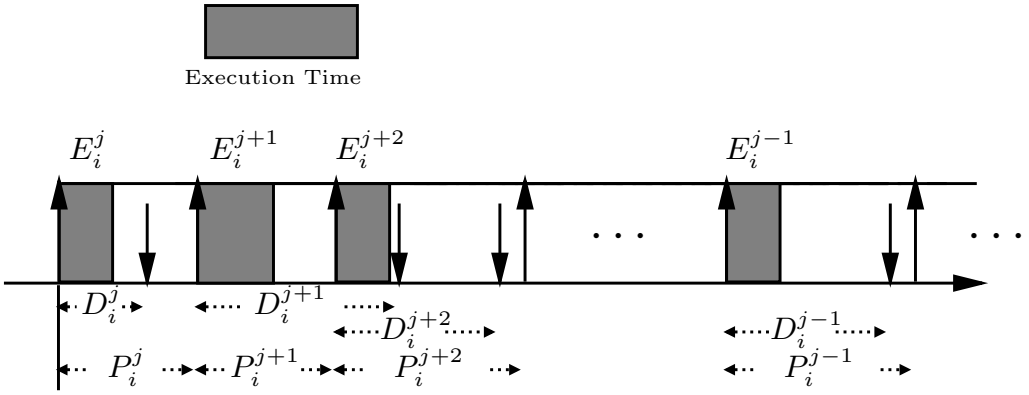


Figure 3: This figure contains all task τ_i's ordered frames from the j'th frame to the $(j-1) \mod N_i$'th frame (we omit "mod N_i" in this figure for simplicity). The starting frame can be any frame ϕ_i^j in an interval length t. Note that each frame deadline can be larger than frame separation time, e.g., $D_i^{j+1} \geq P_i^{j+1}$ in this figure. The details will be shown in Chapter 4.3.

3.2 The Generalized Multiframe Task Model with Parameter Adaptation for Uniprocessor Systems

Based on the GMF model, the GMF-PA model is derived to allow frame parameters to be assigned instead of fixing them during task specification time. Let $\mathcal{T}$ = {τ_0, τ_2,..., τ_{n-1}} be the task system of n GMF-PA tasks executing on one processor. The task τ_i=[ϕ_i^0, ϕ_i^1, ϕ_i^2,..., $\phi_i^{N_i-1}$] consists of N_i frames where ϕ_i^j=(E_i^j, $\underline{D}_i^j$, $\overline{D}_i^j$, $\underline{P}_i^j$, $\overline{P}_i^j$). The j'th frame execution time of the i'th task is E_i^j, and the i'th task-wise execution time is $E_i = \sum_{j=0}^{N_i-1} E_i^j$. The lower bound of relative deadline D_i^j (respectively, the minimum inter-arrival time between consecutive frames, P_i^j) is $\underline{D}_i^j$ (respectively, $\underline{P}_i^j$) and the upper bound of D_i^j (respectively, P_i^j) is $\overline{D}_i^j$ (respectively, $\overline{P}_i^j$). The frame parameters D_i^j and P_i^j can be flexibly assigned in the ranges $[\underline{D}_i^j, \overline{D}_i^j]$ and $[\underline{P}_i^j, \overline{P}_i^j]$, respectively. The frame distance $D_i^{j,k} = D_i^k + \sum_{p=0}^{(k-j-1) \mod N_i} P_i^{(j+p) \mod N_i}$ represents the relative time between the release of the j'th frame and the deadline D_i^k of the k'th frame. For example, $D_i^{2,4} = P_i^2 + P_i^3 + D_i^4$.

The N_i frames which execute in order can be seen as a cycle and the cycle can execute infinite times. This task/cycle deadline $\mathcal{D}_i$ is the upper bound[4] of $D_i^{N_i-1} + \sum_{j=0}^{N_i-2} P_i^j$, and the task period $\mathcal{P}_i$ is the upper bound of $\sum_{j=0}^{N_i-1} P_i^j$. The utilization of task τ_i is $U_i = E_i/\mathcal{P}_i$,

[4]Note that in our paper [61], we define the task/cycle deadline $\mathcal{D}_i$ is the upper bound of $\sum_{j=0}^{N_i-1} D_i^j$. We believe that the new definition is more appropriate for modeling the end-to-end constraint both for self-suspension tasks and end-to-end flows. The change will not affect the evaluation results because the previous work assumes $D_i^j = P_i^j$ (also set for our algorithms in evaluations) which make the two terms $\sum_{j=0}^{N_i-1} D_i^j$ and $D_i^{N_i-1} + \sum_{j=0}^{N_i-2} P_i^j$ be equal.

and the utilization of a task system is $U_{cap} = \sum_{i=0}^{n-1} U_i$. Frame parameters ($D_i^j$ and P_i^j) must satisfy the localized Monotonic Absolute Deadlines (l-MAD) property [8] to maintain frame execution order. That is, the absolute deadline of the j'th frame must be no later than the one of the $j+1$'th frame ($D_i^j \leq P_i^j + D_i^{(j+1) \mod N_i}, \forall i, j$).

With the frame deadlines and periods, we introduce the demand bound function and supply bound function which are used for schedulability analysis under EDF scheduling. Let $\mathsf{dbf}_i(t, \vec{F_i})$ be the *task* demand bound function of a GMF-PA task τ_i within the interval length t. Let $\vec{F_i}$ = [D_i^0, P_i^0, D_i^1, P_i^1,..., $D_i^{N_i-1}$, $P_i^{N_i-1}$] represent an assignment of values for all the task parameters (frame deadline and separations) of task τ_i. The task demand bound function $\mathsf{dbf}_i(t, \vec{F_i})$ accounts for task τ_i's accumulated execution time of frames which have both release times and deadlines inside the interval of length t. The supply bound function $\mathsf{sbf}(t)$ gives the lower bound of resources that the system can supply over an interval of length t. We use the notation $\mathsf{dbf}_i(t, D_i^{j,k})$ to represent the demand for the k'th frame when the first frame to arrive in the interval length t is the j'th frame. At any t-length interval, the total demand must be smaller than the supply to get an successful schedule.

We aim to optimally select frame relative deadlines (D_i^j) and minimum inter-arrival times/periods (P_i^j) in the GMF-PA model in uniprocessor systems, under the *basic requirements* as follows:

1. $E_i^j \leq \underline{D}_i^j \leq D_i^j \leq \overline{D}_i^j, \ \ \forall i, j$
2. $E_i^j \leq \underline{P}_i^j \leq P_i^j \leq \overline{P}_i^j, \ \ \forall i, j$
3. $D_i^j \leq P_i^j + D_i^{(j+1) \mod N_i}, \ \ \forall i, j$

4. $D_i^{N_i-1} + \sum_{j=0}^{N_i-2} P_i^j \leq \mathcal{D}_i, \ \ \forall i$

5. $\sum_{j=0}^{N_i-1} P_i^j \leq \mathcal{P}_i, \ \ \forall i$

Figure 3 is also an example of a GMF-PA model as long as the parameters satisfy the basic requirements and the frame deadlines and periods have their own ranges. A task system must obey the first two inequalities to be feasible. The third inequality is required by the l-MAD property. The last two inequalities check whether a system is feasible under the condition that all task deadlines and periods are assigned under their upper bounds $\mathcal{D}_i$ and $\mathcal{P}_i$, respectively. We call the first three constraints as *frame constraints* and the last two constraints as *task constraints* in the rest of this thesis.

3.2.1 The Multiple-Segment Self-Suspending Task Model and the GMF-PA Model

Scheduling tasks with self-suspensions has received renewed interests in recent years. A task suspends itself to communicate with external devices, I/O operations, computation offloading, etc. A typical model derived from such systems contains two computation frames separated by a self-suspending frame. After the first computational frame finishes, the job suspends executing the other computational frame until such an external operation completes. The multiple-segment self-suspending task model [39] allows a task to suspend many times. Huang and Chen [39] first identified the relationship between self-suspending tasks and the GMF model. They called a computational frame as a computation segment and a suspending frame as a suspension interval; we call both frames instead to be congruent with the GMF-PA model. Figure 4 shows an example of a multiple-segment self-suspending task model.

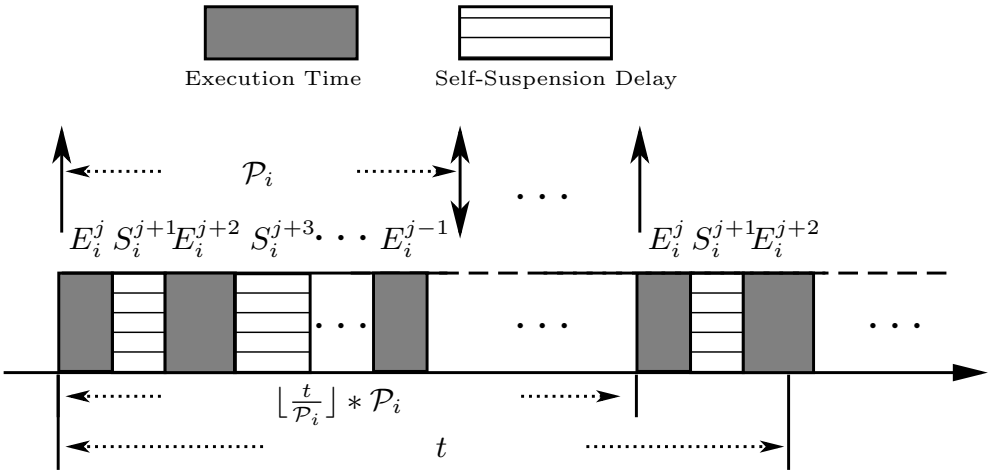


Figure 4: The computational frames are separated by a set of self-suspending frames. We analyze a worst-case release pattern in an interval length t and the details are analyzed in Chapter 4.3.

In the multiple-segment self-suspending task model [39], the task system $\mathcal{T} = \{\tau_0, \tau_2, ..., \tau_{n-1}\}$ consists of n sporadic self-suspending tasks executing on one processor. Each task $\tau_i = ((E_i^0, S_i^0, E_i^1, S_i^1, ...S_i^{m_i-2}, E_i^{m_i-1}), \mathcal{P}_i, D_i)$ has m_i computational frames and $m_i - 1$ suspending frames. The execution time of a computational frame is E_i^j and the self-suspending delay of a suspending frame is S_i^j. The frames work in order and $j+1$'th frame cannot start executing until the time when j'th frame finishes executing. The task period is $\mathcal{P}_i$ and task relative end-to-end deadline is D_i.

In order to incorporate with our GMF-PA model, each frame is rewritten as $\phi_i^j = (E_i^j, \underline{D}_i^j, \overline{D}_i^j, \underline{P}_i^j, \overline{P}_i^j)$. The total number of frames in a cycle of τ_i is $N_i = 2 * m_i - 1$. For a computational frame, E_i^j and the parameters in frame constraints consist of a frame $\phi_i^j = (E_i^j, \underline{D}_i^j, \overline{D}_i^j, \underline{P}_i^j, \overline{P}_i^j)$. For a suspending frame ϕ_i^k, $E_i^k = 0$, $\underline{D}_i^k = \overline{D}_i^k = \underline{P}_i^k = \overline{P}_i^k = S_i^k$. We change the superscripts of frames in $\tau_i = [\phi_i^0, \phi_i^1, \phi_i^2, ..., \phi_i^{N_i-1}]$ to match our GMF-PA

model. Frames are ordered such that a suspending frame executes after a computational frame. The minimum inter-arrival time for τ_i is $\mathcal{P}_i$, and the end-to-end deadline is $\mathcal{D}_i = D_i$ which is the upper bound of $D_i^{N_i-1} + \sum_{j=0}^{N_i-2} P_i^j$.

Our parameter-adaptation methods that are derived in Chapter 4.3 for the GMF-PA model are immediately applicable to the FRD-scheduling [23, 39] for multi-segment self-suspending tasks.

3.3 The Distributed Generalized Multiframe Model with Parameter Adaptation for Distributed Systems

In this chapter, we extend the GMF-PA model to define our distributed GMF-PA (dGMF-PA) model in distributed systems. Note that the dGMF-PA model can be reduced to the GMF-PA model when the number of processors is one. We review the end-to-end flow model and apply our dGMF-PA model to end-to-end flows in Chapter 3.3.1.

Let $\mathcal{T} = \{\tau_0, \tau_2, ..., \tau_{n-1}\}$ be the task system of n dGMF-PA tasks executing in a distributed system. Each task $\tau_i = [\tau_{i,0}, ..., \tau_{i,Q-1}]$ consists of Q virtual tasks on corresponding Q processors. Each virtual task $\tau_{i,p} = [\phi_{i,p}^0, \phi_{i,p}^1, \phi_{i,p}^2, ..., \phi_{i,p}^{N_i-1}]$ consists of N_i virtual frames and executes on processor p. Each virtual frame $\phi_{i,p}^j = (E_{i,p}^j, \underline{D}_{i,p}^j, \overline{D}_{i,p}^j, \underline{P}_{i,p}^j, \overline{P}_{i,p}^j)$ is similar to the frame in the GMF-PA model. In fact, there are only N_i frames in a dGMF-PA task τ_i and we require that each real frame must be statically assigned once on one processor. We call a virtual frame $\phi_{i,p}^k$ a real frame if the k'th frame of task τ_i is assigned on processor p, and we call a virtual frame $\phi_{i,p}^j$ an empty frame if the j'th frame is not assigned on processor p. Figure 5 shows an example of the dGMF-PA model (frame deadlines and periods are selected in their ranges). In an empty frame $\phi_{i,p}^j$, we set $E_{i,p}^j = \underline{D}_{i,p}^j = \overline{D}_{i,p}^j = \underline{P}_{i,p}^j = \overline{P}_{i,p}^j = 0$.

For simplicity, we also refer to a virtual frame as a frame (except when we specify a virtual frame as a real frame or an empty frame).

The cycle/task deadline $\mathcal{D}_i$ of task τ_i is the upper bound of $\sum_{p=0}^{Q-1}\left(D_{i,p}^{N_i-1}+\sum_{j=0}^{(N_i-2)}P_{i,p}^{j}\right)$ (Note that we summarize over all processors for all parameters because empty frames take zero deadlines), and the task/cycle period $\mathcal{P}_i$ is the upper bound of $\sum_{p=0}^{Q-1}\sum_{j=0}^{N_i-1}P_{i,p}^{j}$. The cycle deadline constraint intuitively represents the offset of the last frame's absolute deadline from the release time of the task and matches the traditional concept of an *end-to-end deadline*. The minimum execution time of the frame in τ_i is $E_i^{min}=\min\{\sum_{p=0}^{Q-1}E_{i,p}^{0},\sum_{p=0}^{Q-1}E_{i,p}^{1},...,\sum_{p=0}^{Q-1}E_{i,p}^{N_i-1}\}$, and the total execution time of task τ_i is $E_i=\sum_{p=0}^{Q-1}\sum_{j=0}^{N_i-1}E_{i,p}^{j}$. The utilization of task τ_i is $U_{i,p}=\sum_{j=0}^{N_i-1}E_{i,p}^{j}/\mathcal{P}_i$, and the utilization of a task system is $U_p=\sum_{i=0}^{n-1}U_{i,p}$ on processor p. The maximum utilization of a processor in the distributed system is $U_{cap}=\max_{p=0}^{Q-1}U_p$.

In this thesis, we consider that each frame of a task in the dGMF-PA model has its relative deadline constrained to be at most its period; that is, for all frames $\phi_{i,p}^{j}$, $D_{i,p}^{j}\leq P_{i,p}^{j}$. This assumption ensures that each frame has completed before the release time of the successive frame and simplifies the schedulability analysis for each processor.

With the frame deadlines and periods, we first present the demand bound function and supply bound function which are used for schedulability analysis on a uniprocessor. We then show that our schedulability analysis for distributed systems breaks down to the analysis for uniprocessor systems. The demand bound function $\mathsf{dbf}_i(t,\vec{F}_i,p)$ accounts for the task τ_i's accumulated execution time of jobs which have both release time and deadline inside any interval of length t on processor p, and the supply bound function $\mathsf{sbf}(t)$ gives the lower bound of resources that the system can supply over any interval of length t.

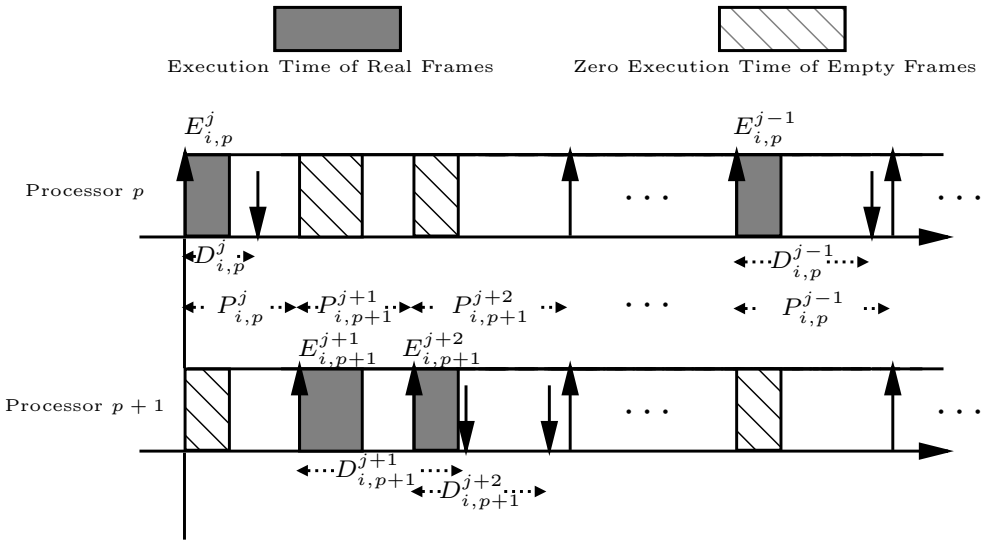


Figure 5: The dGMF-PA task τ_i executes in two processors p and $p+1$. The real frames $\phi^j_{i,p}$, $\phi^{j+1}_{i,p+1}$, $\phi^{j+2}_{i,p+1}$, and $\phi^{j-1}_{i,p}$ execute in an ordered sequence (the real frames between $j+2$'th frame and $j-1$'th frame are omitted here).

Note that our MILP algorithm can consider different supply bound functions $\mathsf{sbf}(t,p)$ for different processors. For simplicity, we consider the same supply bound functions $\mathsf{sbf}(t)$ over all processors since we use identical supply bound function for all processors. The demand must be smaller than the supply at any interval length t among all processors in the distributed system.

We aim to optimally select relative deadlines ($D^j_{i,p}$) and make scheduling decisions under the basic requirements as follows:

1. $E^j_{i,p} \le \underline{D}^j_{i,p} \le D^j_{i,p} \le \overline{D}^j_{i,p}, \ \ \forall i,j,p$

2. $E^j_{i,p} \le \underline{P}^j_{i,p} \le P^j_{i,p} \le \overline{P}^j_{i,p}, \ \ \forall i,j,p$

3. $D^j_{i,p} \le P^j_{i,p}, \ \ \forall i,j,p$

4. $\sum_{p=0}^{Q-1} \left(D^{N_i-1}_{i,p} + \sum_{j=0}^{(N_i-2)} P^j_{i,p} \right) \le \mathcal{D}_i, \ \ \forall i$

5. $\sum_{p=0}^{Q-1} \sum_{j=0}^{N_i-1} P_{i,p}^j \leq \mathcal{P}_i, \quad \forall i$

A task system must obey the first two inequalities to be feasible. The third inequality is the constrained frame deadline property. The last two inequalities check whether a system is feasible under the upper bounds $\mathcal{D}_i$ and $\mathcal{P}_i$.

3.3.1 Distributed End-to-End Flows and the dGMF-PA Model

In this chapter, we review the distributed end-to-end flow model [60, 67] and apply the dGMF-PA model to the flows where each processor is scheduled by EDF scheduling algorithm.

For distributed end-to-end flows, we use a tilde over task parameters to distinguish from the dGMF-PA model. A task system $\widetilde{\mathcal{T}} = \{\widetilde{\tau}_0, \widetilde{\tau}_1, ..., \widetilde{\tau}_{n-1}\}$ consists of n distributed end-to-end flows. Each task $\widetilde{\tau}_i = [\widetilde{\phi}_i^0, \widetilde{\phi}_i^1, \widetilde{\phi}_i^2, ..., \widetilde{\phi}_i^{N_i-1}]$ consists of N_i real frames. In each frame $\widetilde{\phi}_i^j = (\widetilde{E}_i^j, \widetilde{D}_i^j, \widetilde{O}_i^j, \widetilde{J}_i^j)$, $\widetilde{E}_i^j$ is the execution time, $\widetilde{D}_i^j$ is the global relative deadline which is relative to the activation time of the task, $\widetilde{O}_i^j$ is the offset between the release time of a flow and the activation time of the frame $\widetilde{\phi}_i^j$, and $\widetilde{J}_i^j$ is the maximum jitter between the activation time and release time of the frame $\widetilde{\phi}_i^j$. The end-to-end deadline of the task $\widetilde{\tau}_i$ is $\widetilde{\mathcal{D}}_i$ and period between invocations of the task is $\widetilde{\mathcal{P}}_i$. Frames can execute on different processors and each frame can only be activated when its preceding frame completes executing. Figure 6 shows an example of the end-to-end flow model.

Now we translate a task in the end-to-end flow model to one in the dGMF-PA model. For each end-to-end frame $\widetilde{\phi}_i^j$, we create Q virtual dGMF frames $\phi_{i,p}^j$ for $p = 0, 1, \ldots, Q-1$. If the original frame $\widetilde{\phi}_i^j$ is assigned to processor p, all virtual frames $\phi_{i,q}^j$ where $q \neq p$ are empty frames in the dGMF-PA model. For a real frame $\phi_{i,p}^j$, the manner in which we set the

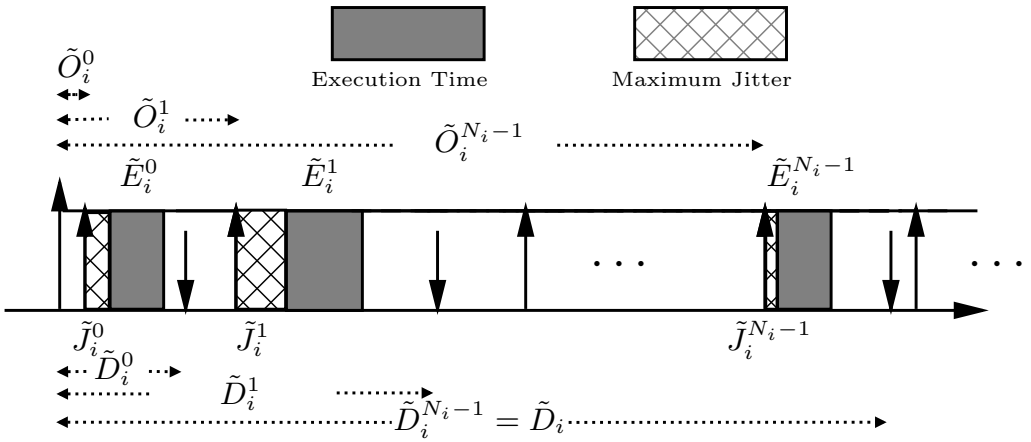


Figure 6: This end-to-end flow τ_i consists of N_i frames which can execute on different processors. The deadlines and jitters ensure the execution sequence of frames.

frame parameters depends upon the setting: 1) if the offsets and global relative deadlines are not fixed by the designer, then we can set trivial lower and upper bounds to the frame periods and relative deadlines (i.e., $\underline{D}_{i,p}^j = \underline{P}_{i,p}^j = \widetilde{E}_i^j$ and $\overline{P}_{i,p}^j = \overline{D}_{i,p}^j = \widetilde{D}_i$); or 2) if the offsets and/or deadlines are fixed by the designer, then the trivial lower and upper bounds can be used again for the frame period and relative deadline; however, two additional constraints must be added: $\widetilde{O}_i^j = \sum_{q=0}^{Q-1} \sum_{\ell=0}^{j-1} P_{i,q}^\ell$ and $\widetilde{D}_i^j = D_{i,p}^j + \sum_{q=0}^{Q-1} \sum_{\ell=0}^{j-1} P_{i,q}^\ell$. Clearly, we can always set the frame execution $E_{i,p}^j$ to be $\widetilde{E}_{i,p}^j$. A jitter $\widetilde{J}_i^j$ can be modeled as a new independent dGMF-PA frame $\phi_{i,p}^{j'}$ in which $E_{i,p}^{j'} = 0$ and $\underline{D}_{i,p}^{j'} = \overline{D}_{i,p}^{j'} = \underline{P}_{i,p}^{j'} = \overline{P}_{i,p}^{j'} = J_{i,p}^j$. This jitter frame is inserted before its corresponding frames (both empty and real) $\phi_{i,q}^j$ for all $q = 0, \ldots, Q-1$; once the jitter frame $\phi_{i,p}^{j'}$ "completes", then the frame $\phi_{i,p}^j$ is ready to execute. The period of task τ_i is $\mathcal{P}_i = \widetilde{\mathcal{P}}_i$, and the end-to-end deadline is $\mathcal{D}_i = \widetilde{\mathcal{D}}_i$.

Due to the hardness of the frame assignment in distributed systems [35], we assume that real frames of end-to-end flows are statically assigned on processors (each jitter frame is bundled with its real frame on a processor). Aside from real frames on each processor p, we assign the other frames of all tasks to be empty frames on each processor p. That is,

from the viewpoint of each processor p, all frames of tasks execute on processor p where some frames are empty.

In this thesis, we deviate from the typical end-to-end flow semantics; in particular, in the end-to-end flow model, it is often assumed that a frame is released as soon as the previous frame signals it is complete. Instead, in this thesis, we assume that a frame is eligible to execute only when its frame is released according to the period parameters of $P_{i,p}^{j}$. However, we conjecture that our schedulability results will continue to hold for the usual end-to-end semantics by permitting a frame to "early release" its job, but keeping its absolute deadline fixed to the same it would be in the dGMF-PA model (i.e., deadlines of frames do not shift when early released). The rationale is that fixing the deadlines but permitting early releases would only decrease the total execution demand and thus preserve schedulability.

Our combined parameter selection and schedulability test algorithms that are presented in Chapter 7 for the dGMF-PA model are thus applicable to distributed end-to-end flows.

3.4 Summary

In this chapter, we review the GMF model, and define our flexible models GMF-PA and dGMF-PA in uniprocessor and distributed systems, respectively. In the next, we introduce the schedulability analysis and parameter selection algorithms under the GMF-PA model in Chapters 4 to 6 and the dGMF-PA model in Chapter 7, respectively.

CHAPTER 4 PARAMETER ASSIGNMENT AND SCHEDULABILITY ANALYSIS IN UNIPROCESSORS

In this chapter, we introduce the parameter assignment and schedulability analysis (considered as a combined technique) in uniprocessor systems under the GMF-PA model. In Chapter 4.1, we introduce the GMF model (extended by the GMF-PA model) and self-suspension tasks (an application of the GMF-PA model). Utilizing the GMF-PA model, we state the problem statement of this chapter in Chapter 4.2. In Chapter 4.3, we introduce the combined technique using the MILP technique. Due to the technique is not a sufficient schedulability test in general, and does not scale well. We introduce the sufficient approximation algorithm in Chapter 4.4. Chapter 4.5 applies the parameter-adaptation method and its approximation algorithm to self-suspending tasks. Chapter 4.6 does extensive experiments compared with state-of-art results. We conclude this chapter in Chapter 4.7.

4.1 Introduction

In real-time systems, worst-case execution time (WCET) analysis calculates an upper bound for each task based on the total aggregate amount of execution required for a job. Such estimates derived from WCET analysis are used in real-time schedulability analysis to determine whether every job in a system can finish executing before its deadline. Therefore, the effectiveness of the schedulability analysis hinges upon the precision of WCET estimates. Unfortunately, many scheduler properties that simplify schedulability analysis often introduce pessimism into WCET analysis. For example, the oft-assumed property of a task is that the worst-case execution times are the same for all jobs. However, this assumption is inaccurate for tasks which produce a sequence of jobs with heterogeneous execution times. For example in multimedia systems, the execution time of a job containing

video-data is not necessarily the same as the execution time of the next job containing corresponding audio-data.

The multiframe task model [52] (MF) generalizes the periodic task model and reduces the pessimism by a set of finite recurring frames. The finite set of frames can be seen as a cycle. This cycle recurs an infinite number of times. Frames can have different WCETs instead of identical ones. The generalized multiframe task model [8] (GMF) further generalizes the sporadic task model and multiframe task model [52]. Instead of setting implicit deadlines and the same minimal inter-arrival times for each frame in the MF model, the GMF model assigns an individual deadline and a minimal inter-arrival time for each frame.

The GMF model increases flexibility compared to the sporadic task model and multiframe task model. Even with this, however, the parameters in the GMF model are fixed during task specification time. The schedulability will be decreased when the parameters are not flexible and dynamic, e.g., in multimedia and adaptive control systems. Buttazzo et al. [18] defined the elastic model in which each period has an upper bound. If a job misses a deadline, the period is allowed to increase under the upper bound. Chantem et al. [22] selected the deadlines and periods in the generalized elastic model. The elastic model is allowed to change parameters during runtime. In this thesis, we extend the GMF model to let frame periods and deadlines be selected under a set of upper bounds. In this flexible model, frame periods and deadlines are optimally assigned prior to runtime under our methods.

We also apply this flexible GMF model to schedule a set of self-suspending tasks under EDF scheduling. In order to address the pessimistic performance when considering the self-

suspension delay as a part of execution time, recent work [23, 39] presented fixed-relative-deadline (FRD) scheduling algorithm. FRD scheduling breaks a task into computation phases and suspensions phases; each computation phase can be viewed as a frame with its own relative deadline and execution. Under FRD scheduling, a simple deadline assignment approach is presented that equally assigns deadlines [23, 39] based on the difference between the self-suspension and minimum inter-arrival time. However, the proposed deadline assignment is very restrictive, and we will detail the restriction at the beginning of Chapter 4.5. We consider a more general, less restrictive deadline assignment strategy to improve schedulability for multi-segment self-suspending tasks. The system is also extended to an arbitrary-deadline system.

Our Contributions in this Chapter:

- We develop a parameter-adaptation algorithm that selects frame deadlines and minimum inter-arrival times in the GMF-PA model. We prove that the algorithm is a necessary schedulability test. When parameters are assumed to be integers, we also prove that the algorithm is a sufficient and necessary schedulability test, i.e., an exact test.

- We develop an approximation algorithm and prove that this method is a sufficient schedulability test.

- We apply our parameter adaptation algorithm and its approximation algorithm on multiple-segment self-suspending tasks. The system is also extended to an arbitrary-deadline system.

- The speed-up factor of our approximation algorithm is $1 + \epsilon$ with respect to the exact

schedulability test of GMF-PA tasks under EDF.

- We implement extensive experiments to show the improvement compared to previous work.

Next, we state the problem in Chapter 4.2 and introduce the combined technique (parameter selection and schedulability test) by mixed-integer linear programming (MILP) in Chapter 4.3.

4.2 Problem Statement

Let $\mathsf{dbf}_i(t, \vec{F_i})$ be the *task* demand bound function of a GMF-PA task τ_i within the interval length t. Let $\vec{F_i}$ = [D_i^0, P_i^0, D_i^1, P_i^1,..., $D_i^{N_i-1}$, $P_i^{N_i-1}$] represent an assignment of all task parameters (frame deadlines and periods) of task τ_i. The supply bound function $\mathsf{sbf}(t)$ gives the lower bound of resources that the system can supply over an interval of length t. In a uniprocessor system $\mathcal{T}$, the sufficient and necessary condition [8] for schedulability of a task set $\mathcal{T}$ is shown in Equation 4.1.

$$\sum_{\tau_i \in \mathcal{T}} \mathsf{dbf}_i(t, \vec{F_i}) \leq \mathsf{sbf}(t), \quad \forall t. \tag{4.1}$$

Problem Definition. *Given the above model, our goal is to find an optimal and valid assignment $\vec{F_i}$ of frame parameters of all tasks so that the worst-case demand $\sum_{\tau_i \in \mathcal{T}} \mathsf{dbf}_i(t, \vec{F_i})$ over all time intervals of length t is minimized.*

4.3 The Exact Frame Deadline/Period Assignment of Generalized Multiframe Tasks in the GMF-PA Model

In this chapter, we describe the selection of the deadline and period for each frame under our GMF-PA model by using mixed-integer linear programming (MILP) under EDF scheduling. The period and deadline of each frame are flexible to be chosen under the limit of the frame constraints and cycle constraints. Along with the selection, the algorithm provides a necessary feasibility test for arbitrary real-valued task parameters. We prove the sufficiency and necessity of the test when all frame deadlines and periods must be integers.

Mixed-integer linear programming (MILP) is a mathematical optimization model that contains three parts: an objective function, constraint functions, and ranges of variables. A subset of variables can be restricted to integers in MILP. An MILP problem aims to find the optimal value of the objective function under the restriction of constraint functions. We build our MILP-based algorithm to select the relative deadlines and periods for all frames. At the same time, the MILP-based algorithm gives a necessary feasibility test. However, the returned selection of real-valued deadlines and periods may not be feasible since the MILP algorithm is only necessary. Later, in Chapter 4.4, we give a sufficient approximation algorithm for the MILP that returns a feasible selection of frame periods and deadlines for the non-integer case.

In a system where parameters are fixed after task specification time, parameters are given as constants. However, in this thesis, we let frame deadlines and periods be variables and select such parameters using the MILP. The demand in this case is also treated as a variable. For instance, we determine the demand of frame ϕ_i^k over an interval of length t

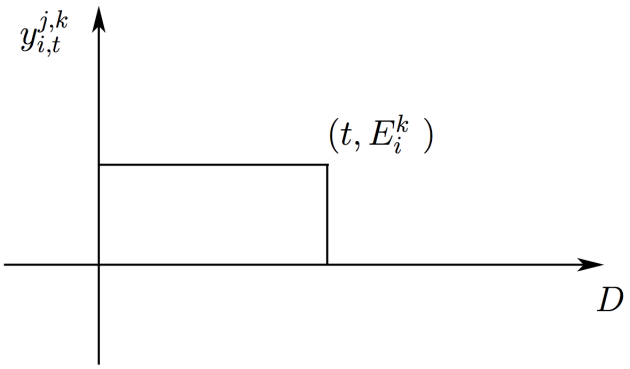


Figure 7: For simplicity, the frame demand $y_{i,t}^{j,k}$ in this figure is calculated when t is smaller than one period by Line 6 of the MILP algorithm in Figure 8.

.

and the frame arrives at the beginning of the interval. If the relative deadline D_i^k is set to be smaller or equal to t, the demand for this frame is E_i^k; otherwise the demand is zero. Figure 7 illustrates a staircase function of the frame demand over the interval of length t. The notations will be introduced later in our MILP-based algorithm. With the calculation of all task demand that summarized by all frame demands, Equation 4.1 becomes a set of constraint functions that a feasible system must obey to find a relative-deadline/period assignment. In our parameter-adaptation algorithm, the supply bound function is sbf$(t) = t$ and the length t of any interval is an integer. Our MILP algorithm can return an assignment if the system is schedulable. That is, we can determine the necessary feasibility of the system and select potential parameters at the same time.

The general steps of our algorithm are as follows. For a given sequence of frames and a time interval of length t, we calculate the demand contribution of each frame to that interval length. Adding the demands of all frames generates the demand of a task, and adding the demands of all tasks (over all possible sequences of frames) generates the total demand at the t-length interval. The system is schedulable at a time interval length if the

demand is smaller than the supply. We check all interval lengths, which are integers, in the algorithm.

For a given interval length t, we calculate the demand for every possible sequence of frames of task τ_i over any interval of length t. Assume that the first frame of τ_i to arrive in such an interval is ϕ_i^j (i.e., the j'th frame). The demand of any sequence starting with the j'th frame over a t-length interval is maximized if the j'th frame arrives exactly at the start of the interval and subsequent frames arrive as soon as possible (e.g., see Baruah et al. [8] for GMF schedulability). To calculate the demand from the k'th frame in such a t-length interval for the specified sequence, $y_{i,t}^{j,k}$ represents the frame demand of the task τ_i. We will calculate $y_{i,t}^{j,k}$ for all possible $i, j, k,$ and t. For simplicity, we use the "$\forall$" to represent the ranges of variables. The task index i ranges from zero to $n-1$. The superscripts j and k represent the starting frame and the current frame respectively, and both have the ranges from zero to $N_i - 1$. The range of any interval length t has been shown [8] that the maximum interval length is bounded by $O(\frac{U_{cap}}{1-U_{cap}} \cdot \max_{\tau_i \in \tau}(\mathcal{P}_i - D_i^0))$ in which $U_{cap} < 1$. We use $H = \lceil \frac{U_{cap}}{1-U_{cap}} \cdot \max_{\tau_i \in \tau}(\mathcal{P}_i - E_i^{min}) \rceil$ as the maximum integer length interval since we do not know frame deadlines beforehand in our GMF-PA model. We use such abbreviations across this thesis. The demand of the task τ_i started from j'th frame in time interval length t is $y_{i,t}^j$. The maximum demand of τ_i among all starting frames is $y_{i,t}$.

Figure 8 shows our *Parameter Selection and Exact Feasibility Test* algorithm, the notations in bold font are constants and the other notations are variables. Lines 3 to 5 are the basic constraints introduced in Chapter 3.2. Line 6 calculates the demand of $y_{i,t}^{j,k}$. The interval length $\lfloor \frac{t}{\mathcal{P}_i} \rfloor$ tracks the number of cycle periods in t, and $\lfloor \frac{t}{\mathcal{P}_i} \rfloor * E_i^k$ is the demand of ϕ_i^k in such cycle periods. The parameter $x_{i,t}^{j,k}$ is restricted to be an integer value and works as a

"flag" (either zero or one) to decide whether the demand E_i^k should be added in the interval length $t - \lfloor \frac{t}{\mathcal{P}_i} \rfloor * \mathcal{P}_i$, e.g., showed in Figure 4. Note that all frames release as soon as possible, the analysis of a demand in $[0, t - \lfloor \frac{t}{\mathcal{P}_i} \rfloor * \mathcal{P}_i]$ is equal to the one in $[\lfloor \frac{t}{\mathcal{P}_i} \rfloor * \mathcal{P}_i, t]$. The "flag" $x_{i,t}^{j,k}$ is decided by the constraints in Line 6. Line 6 is the constraint function that decides the value of $x_{i,t}^{j,k}$. The length t_b in Line 6 is the summation of the previous periods $\lfloor \frac{t}{\mathcal{P}_i} \rfloor * \mathcal{P}_i$ and the distance from the starting j'th frame to k'th frame $\sum_{p=0}^{(k-j-1) \mod N_i} P_i^{(j+p) \mod N_i} + D_i^k$. For example, the length $t_b = P_i^1 + P_i^2 + D_i^3 + \lfloor \frac{t}{\mathcal{P}_i} \rfloor * \mathcal{P}_i$ if we consider the interval starting with an arrival of the first frame and ending at the deadline of the third frame. In the inequality of Line 6, the lengths t_b and t decide whether the demand of k'th frame in $t - \lfloor \frac{t}{\mathcal{P}_i} \rfloor * \mathcal{P}_i$ will be added to $y_{i,t}^{j,k}$. The constant $realmin$ is the smallest representable positive number. When $t \geq t_b$, the flag $x_{i,t}^{j,k}$ must be one to let MILP feasible and the demand $x_{i,t}^{j,k} * E_i^k$ contributes to $y_{i,t}^{j,k}$. When $t < t_b$, the flag $x_{i,t}^{j,k}$ can be either zero or one. However, the demand $y_{i,t}^{j,k}$ is overestimated when $x_{i,t}^{j,k} = 1$. Our MILP algorithm tends to choose zero for $x_{i,t}^{j,k}$ because of the smaller demand, and the details are shown in Lemma 1. Figure 7 shows an example of the staircase function between frame deadline and demand. Note that the inequality in Line 6 is always correct when $x_{i,t}^{j,k}$ is one and $t \geq t_b$, and when $x_{i,t}^{j,k}$ is zero and $t < t_b$.

In Line 7, the demand $y_{i,t}^{j}$ of task τ_i starts from j'th frame. In Line 8, the demand $y_{i,t}$ is the maximum demand for τ_i over all possible starting frames. At last, the demand of all tasks $\sum_{i=0}^{n-1} y_{i,t}$ has to be less than the supply bound function for all t as showed in Equation 4.1; otherwise, the system is not schedulable. In Line 9, $\mathcal{L}$ is set to indicate how schedulable or not schedulable the system is. If the system is schedulable, then $\mathcal{L} \leq 1$.

In the setting of our MILP algorithm, the variables $D_i^k, P_i^k, t_b, y_{i,t}^{j,k}, y_{i,t}^{j}, y_{i,t}$, and $\mathcal{L}$ are free variables. The number of all variables is pseudo-polynomial bounded. The flag $x_{i,t}^{j,k}$ is

Parameter Selection and Exact Feasiblity Test

1 minimize: $\mathcal{L}$

2 subject to:

3 $\boldsymbol{E_i^k} \leq \underline{\boldsymbol{D_i^k}} \leq D_i^k \leq \overline{\boldsymbol{D}}_{\boldsymbol{i}}^{\boldsymbol{k}}, \quad \forall i, k.$

$\boldsymbol{E_i^k} \leq \underline{\boldsymbol{P_i^k}} \leq P_i^k \leq \overline{\boldsymbol{P}}_{\boldsymbol{i}}^{\boldsymbol{k}}, \quad \forall i, k.$

4 $D_i^k \leq P_i^k + D_i^{(k+1) \mod N_i}, \quad \forall i, k.$

5 $\sum_{k=0}^{N_i-1} P_i^k \leq \boldsymbol{\mathcal{P}_i}, D_i^{N_i-1} + \sum_{j=0}^{N_i-2} P_i^j \leq \boldsymbol{\mathcal{D}_i}, \quad \forall i.$

6 $\boxed{\begin{array}{l} y_{i,t}^{j,k} = x_{i,t}^{j,k} \cdot \boldsymbol{E_i^k} + \lfloor \frac{\boldsymbol{t}}{\boldsymbol{\mathcal{P}_i}} \rfloor \cdot \boldsymbol{E_i^k}, \quad \forall i, j, k, t. \\ \frac{\boldsymbol{t}-t_b}{\boldsymbol{\mathcal{P}_i}} \leq x_{i,t}^{j,k} - \frac{\boldsymbol{realmin}}{\boldsymbol{\mathcal{P}_i}}, \quad \forall i, j, k, t. \\ t_b = D_i^{j,k} + \lfloor \frac{\boldsymbol{t}}{\boldsymbol{\mathcal{P}_i}} \rfloor \cdot \boldsymbol{\mathcal{P}_i} \end{array}}$

7 $y_{i,t}^j = \sum_{k=0}^{N_i-1} y_{i,t}^{j,k}, \quad \forall i, j, t.$

8 $y_{i,t} \geq y_{i,t}^j, \quad \forall i, j.$

9 $\sum_{i=0}^{n-1} y_{i,t} \leq \mathcal{L} \cdot \boldsymbol{t} \quad \forall t.$

10 and: $D_i^k, P_i^k, y_{i,t}^{j,k}, y_{i,t}^j, y_{i,t}, \mathcal{L} \in \mathbb{R}^*, \boxed{x_{i,t}^{j,k} \in \{0, 1\}}.$

Figure 8: This figure shows our MILP algorithm. In the concave programming and LP-based algorithms (shown in Chapters 6.3 and 6.4), we only change the frame demand in Line 6 and remove all integer variables $x_{i,t}^{j,k}$.

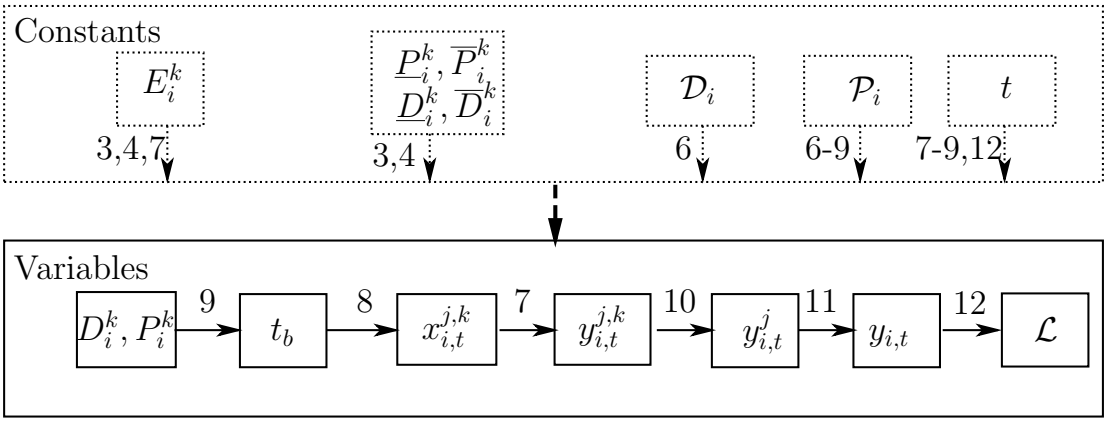


Figure 9: Relations of the parameters in the MILP algorithm.

restricted to be an integer variable that is either zero or one. The relationship among the variables is summarized in Figure 9. The boxes with solid lines contain free variables and the boxes with dotted lines contain constants. The arrows show the dependable relationships and the integers on the arrows indicate the number of lines in the MILP algorithm. For example, Lines 5 to 6 show that the constant $\mathcal{P}_i$ has an effect on the variables P_i^k, t_b, $x_{i,t}^{j,k}$ and $y_{i,t}^{j,k}$. All variables are connected and restrained in the MILP algorithm. Eventually, minimizing $\mathcal{L}$ also minimizes the total demand $\sum_{i=0}^{n-1} y_{i,t}$.

Next, we first prove that $y_{i,t}^{j,k}$ is an exact demand of k'th frame during the interval length t started from j'th frame in Lemma 1. Using Lemma 1, we can show in Theorem 1 that our task demand $y_{i,t}$ within the t-length interval is identical as the one in the GMF model [8]. Last, we prove that our MILP algorithm is a sufficient and necessary schedulability test for integer parameters (frame deadlines and periods) in Theorem 2, by showing that the demand only changes at integer-interval lengths in Lemma 2.

Lemma 1. *The value of $y_{i,t}^{j,k}$ in the MILP is the exact worst-case demand of frames ϕ_i^k over an interval of length t when the first frame of τ_i to arrive in the interval is ϕ_i^j (with respect to the deadline and periods assigned to each frame of τ_i by the MILP).*

Proof. We first prove that $y_{i,t}^{j,k}$ is an upper bound of the demand, then prove $y_{i,t}^{j,k}$ is the exact demand. Worst-case means that the interval length t starts at the release time of j'th frame and all succeeding frames release as soon as possible.

For an interval of length t, the demand $\lfloor \frac{t}{\mathcal{P}_i} \rfloor * E_i^k$ contributes to $y_{i,t}^{j,k}$ in the first $\lfloor \frac{t}{\mathcal{P}_i} \rfloor$ cycle periods. The remaining question is that whether the k'th frame in the $\lfloor \frac{t}{\mathcal{P}_i} \rfloor + 1$'th cycle will contribute to the interval length $t - \lfloor \frac{t}{\mathcal{P}_i} \rfloor * \mathcal{P}_i$. This is exactly what Line 6 in our MILP tells. The flag $x_{i,t}^{j,k}$ tells whether the k'th frame in the $\lfloor \frac{t}{\mathcal{P}_i} \rfloor + 1$'th cycle will contribute to the interval length $t - \lfloor \frac{t}{\mathcal{P}_i} \rfloor * \mathcal{P}_i$. The flag $x_{i,t}^{j,k}$ has the value which is either zero (not contribute) or one (contribute).

The inequality in Line 6 decides the value of $x_{i,t}^{j,k}$. Line 6 shows that t_b is the interval length that compares with the time interval length t. The time $\lfloor \frac{t}{\mathcal{P}_i} \rfloor * \mathcal{P}^i$ tracks $\lfloor \frac{t}{\mathcal{P}_i} \rfloor$ cycles. Since the starting frame of t is ϕ_i^j, the starting stage is still ϕ_i^j after $\lfloor \frac{t}{\mathcal{P}_i} \rfloor$ cycles. Starting from ϕ_i^j, the length $\sum_{p=0}^{(k-j-1) \mod N_i} P_i^{(j+p) \mod N_i} + D_i^k$ is compared with $t - \lfloor \frac{t}{\mathcal{P}_i} \rfloor * \mathcal{P}_i$ to check whether ϕ_i^k in the $\lfloor \frac{t}{\mathcal{P}_i} \rfloor + 1$'th cycle contributes. In other words, the inequality $t \geq t_b$ means that ϕ_i^k in the $\lfloor \frac{t}{\mathcal{P}_i} \rfloor + 1$'th cycle will contribute to $y_{i,t}^{j,k}$; Otherwise, the frame does not contribute when $t < t_b$. Note that we only need to consider the $\lfloor \frac{t}{\mathcal{P}_i} \rfloor + 1$'th cycle because $\lfloor \frac{t}{\mathcal{P}_i} \rfloor * \mathcal{P}_i \leq t < (\lfloor \frac{t}{\mathcal{P}_i} \rfloor + 1) * \mathcal{P}_i$. When $t \geq t_b$, $x_{i,t}^{j,k}$ is forced to be one showed in Line 6. When $t < t_b$, $x_{i,t}^{j,k}$ can be either zero or one. Thus, $y_{i,t}^{j,k}$ is an upper bound of the demand when $x_{i,t}^{j,k}$ is one.

We have proved that $y_{i,t}^{j,k}$ is an upper bound of the demand when $x_{i,t}^{j,k}$ is one. Now we prove that $y_{i,t}^{j,k}$ is an exact demand. When $t \geq t_b$, $x_{i,t}^{j,k}$ is forced to be one and $y_{i,t}^{j,k}$ is an exact demand. When $t < t_b$, the system always let $x_{i,t}^{j,k}$ be zero instead of one. The reason is that we minimize $\mathcal{L}$ in the objective function. In Line 9, minimizing $\mathcal{L}$ minimizes $y_{i,t}$. Minimizing

$y_{i,t}$ also minimizes $y^j_{i,t}$ and $y^{j,k}_{i,t}$. In all, $y^{j,k}_{i,t}$ is the exact worst-case demand for ϕ^k_i over the interval length t started from ϕ^j_i. □

Theorem 1. *The value of $y_{i,t}$ in the MILP over any interval of length t is exactly the value of* $\mathrm{dbf}_i(t, \vec{F_i})$ *for the GMF model when the parameters of τ_i are assigned the values of the deadline and period variables of the MILP.*

Proof. In Lemma 1, we have proved that $y^{j,k}_{i,t}$ in the MILP is the exact worst-case demand of frames ϕ^k_i over an interval of length t when the first frame of τ_i to arrive in the interval is the j'th frame. $y^j_{i,t}$ is thus the exact demand of task $\tau_{i,p}$ over length t starting from the j'th frame, and $y_{i,t}$ is the exact worst-case demand of τ_i over length t. We prove this by contradiction. Assume there exists a demand $y'_{i,t}$ which is larger than $y_{i,t}$, and the worst-case sequence of $y'_{i,t}$ ($y_{i,t}$) starts from the frame $\phi^{j'}_i$ (ϕ^j_i). If $j' = j$, $y'_{i,t} = y_{i,t}$ which incurs a contradiction with $y'_{i,t} > y_{i,t}$. If $j' \neq j$, $y^{j'}_{i,t} > y^j_{i,t}$ also incurs a contradiction since ϕ^j_i is the starting frame of the worst-case sequence (See Line 8 in MILP). By Lemma 1, we know that $y^j_{i,t}$ is the exact worst-case demand starting from frame ϕ^j_i, and $y_{i,t}$ is the exact worst-case demand of τ_i in the t-length interval. Thus, this theorem is proved. □

Note that our MILP in general is not a sufficient feasibility test when this integer constraint is removed since it does not check all real values in the range $[0, H]$. $\sum_{i=0}^{n-1} y_{i,t} \leq \boldsymbol{t}$ is a sufficient feasibility test when D^k_i, P^k_i, $t \in \mathbb{N}$ shown in Theorem 2. The algorithm is exact when frame deadlines and periods are integers, since it is shown in Lemma 2 that the demand changes value in this case only at integer times; thus, the MILP exactly checks all the relevant time intervals.

Lemma 2. *The demand of a task or a task system only increases at integer times when the*

interval length t increases, under D_i^k, $P_i^k \in \mathbb{N} \; \forall \; i, \; k$.

Proof. In the worst case of synchronous arrival pattern, all tasks release at the same time at the beginning of a t-length interval. Since frame periods P_i^k are integers, all frames release at integer times. Since frame deadlines D_i^k are also integers, the demand of a frame (a task) only changes at integer times according to the definition of the demand bound function. More specifically, at any integer-length (e.g., t') interval, $\mathsf{dbf}_i(t', \vec{F_i}) = \mathsf{dbf}_i(t' + \Delta, \vec{F_i})$ where $0 \leq \Delta < 1$. The demand of a task system thus only changes at integer times. In total, this lemma is proved. □

Theorem 2. *For arbitrary, real-valued parameters, our MILP is a necessary feasibility test. When the period and deadline parameters must be integers (i.e., D_i^k, $P_i^k \in \mathbb{N} \; \forall \; i, \; k$), then the MILP is an exact feasibility test.*

Proof. The necessity is straightforward to prove. That is, we need to show that our MILP is feasible if a task system is feasible. If the task system is feasible, the worst-case demand of the system at any interval length t should not be larger than t. In Theorem 1, we have proved that $y_{i,t}$ is the exact worst case demand of task τ_i in the t-length interval. The demand $\sum_{i=0}^{n-1} y_{i,t} \leq \mathcal{L} * t$ and $\mathcal{L} \leq 1$ in our MILP, and our MILP is thus feasible. The necessity is proved.

For integer-constrained values of frame periods and deadlines, we prove the sufficiency. That is, we prove that the task system is feasible if our MILP is feasible ($\mathcal{L} \leq 1$). We prove by contradiction, i.e., our MILP is feasible, but the task system is not feasible. Suppose our MILP is feasible and $\mathcal{L} \leq 1$, we have a parameter assignment $\mathcal{A} = (A_0^0, A_0^1, ..., A_0^{N_i-1}, A_1^0, A_1^1, ..., A_1^{N_i-1}, ..., A_{n-1}^0, ..., A_{n-1}^{N_i-1})$, where N_i is the maximum number

of frames for each task and n is the number of tasks. A pair $A_i^j = (D_i^j, P_i^j)$ indicates an assignment of frame deadline and period for ϕ_i^j. Such an assignment is feasible for our MILP but not feasible for the task system. Under this assignment, there exists an interval length t that is not larger than the demand of the system. Since we check all integer values of t, there exists two situations which make the system infeasible. The two situations are that the violation happens on integer length t or non-integer length t. For integer length t, we directly obtain a contradiction since our MILP calculates demand $\sum_{i=0}^{n-1} y_{i,t}$ for all integer lengths t, which by Theorem 1 corresponds to the demand within the t-length interval for a feasible system. For non-integer length t, we have shown in Lemma 2 that demand does not change at non-integer times when D_i^k and P_i^k are integers; thus, the MILP exactly calculates demand $\sum_{i=0}^{n-1} y_{i,t}$ for all the relevant time intervals. Similarly to the case of integer lengths t, we obtain a contradiction. In total, this theorem is proved.

□

4.4 The Approximation Algorithm Based on the MILP

In the previous chapter, we have built our MILP which can select the frame deadlines and periods of GMF-PA tasks under EDF scheduling. The method also indicates a necessary feasibility test at the same time. However, solving an MILP is NP-hard in general. Furthermore, the feasibility of our MILP is coNP-hard that trivially transformed from the feasibility test of sporadic tasks [33]. In this chapter, we modify the MILP to obtain an approximation based on reducing the number of time interval lengths being tested[5]. We also show that the speed-up factor of our approximation algorithm is $1 + \epsilon$ with respect to

[5]The approximation is still an MILP (and thus still potentially intractably), but a reduction in constraints leads to a significant improvement in time efficiency as shown in the evaluation chapter.

the exact schedulability test of GMF-PA tasks under EDF scheduling. We have introduced an approximation algorithm under the GMF-PA model and such similar technique can be traced back to admission control for the arbitrary demand curves [30].

The number of t-length intervals being tested in the MILP is H (defined in Chapter 4.3), and the number of time intervals being tested in the approximation algorithm is H_a. The set of test interval lengths in the MILP is $T = \{1, 2, 3, ..., H\}$ and the set in the approximation of the MILP algorithm is T_a. The supply bound function used in MILP is shown in Equation 4.2.

$$\mathsf{sbf}(t) = t. \tag{4.2}$$

Since the number of variables and equations in MILP depend on H, the size of the algorithm grows quickly when H grows. We propose an approximation method based on reducing the number of time intervals. We start from the initial time interval length t_0. The increasing rate is $\epsilon > 0$. We choose the time interval lengths by the increasing rate; thus, $T_a = \{t_0, t_0 * (1+\epsilon), t_0 * (1+\epsilon)^2, ..., t_0 * (1+\epsilon)^{H_a-2}, H\}$. Note that the $H_a - 2$'th element is not larger than H, and we add H at the end as the $H_a - 1$'th element. Note that the increasing rate between the last two elements is not larger than ϵ. For example, the set T_a is $[1, 1.5, 2.25, 3.375, ..., 17.0859375, 20]$ for $H = 20, t_0 = 1$ and $\epsilon = 0.5$. The supply $\mathsf{sbf}^a(t)$ in the approximation algorithm is showed in Equation 4.3.

$$\mathsf{sbf}^a(t) = \begin{cases} 0, & 0 \leq t \leq t_0 \\ t_0 * (1+\epsilon)^k, & t_0 * (1+\epsilon)^k < t \leq t_0 * (1+\epsilon)^{k+1} \\ H, & t = H \end{cases} \tag{4.3}$$

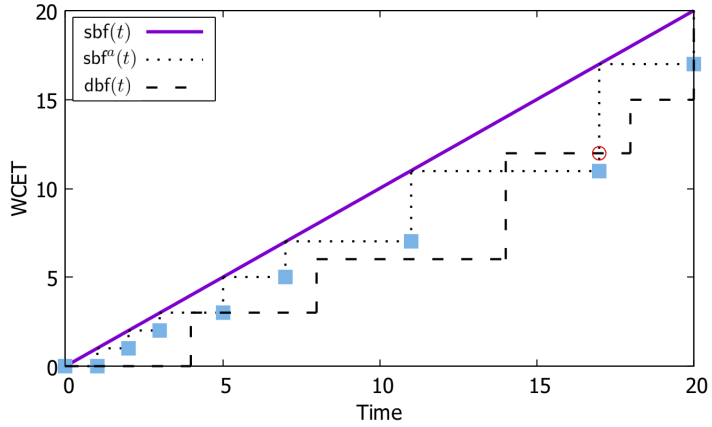


Figure 10: In this Figure, line y=t is the supply bound function $\mathsf{sbf}(t)$ of MILP. The stair case function drawn in dotted line is the supply bound function $\mathsf{sbf}^a(t)$ of an approximation algorithm. The staircase function drawn in dashed line is an example of a demand $\mathsf{dbf}(t) = \sum_{\tau_i \in \mathcal{T}} \mathsf{dbf}_i(t, \vec{F}_i)$. The square points on $\mathsf{sbf}^a(t)$ are the only required test intervals that is proved in Theorem 9. In this example, the total demand $\sum_{\tau_i \in \mathcal{T}} \mathsf{dbf}_i(t, \vec{F}_i) \leq \mathsf{sbf}(t)$ at all time interval length t except for the demand $\sum_{\tau_i \in \mathcal{T}} \mathsf{dbf}_i(t, \vec{F}_i) > \mathsf{sbf}^a(t)$ that is shown at the red circle.

.

In our $\mathsf{sbf}^a(t)$, the starting time interval length is $t_0 = \min_{\tau_i \in \mathcal{T}} E_i^{min}$ and the range of integer k is $[1, H_{a-2}]$ in our approximation algorithm. Figure 10 shows an example of the relationship among $\sum_{\tau_i \in \mathcal{T}} \mathsf{dbf}_i(t, \vec{F}_i)$, $\mathsf{sbf}(t)$ and $\mathsf{sbf}^a(t)$. It is straightforward to show that the number of elements in T_a is $O(\log_{1+\epsilon} H)$.

Next, we modify the general schedulability condition of Equation 4.1 with respect to the reduced set of testing points T_a.

Theorem 3. *Consider any task system composed of tasks $\mathcal{T}$ (e.g., GMF tasks) where the $\mathsf{dbf}_i(t, \vec{F}_i)$ is computable (e.g., see Baruah [8]) for any $\tau_i \in \mathcal{T}$. Then, by checking the following modified condition:*

$$\sum_{\tau_i \in \mathcal{T}} \mathsf{dbf}_i(t, \vec{F}_i) \leq \mathsf{sbf}^a(t), \quad \forall t \in T_a, \tag{4.4}$$

where t_0 of $\mathsf{sbf}^a(t_0)$ must not be larger than the $\min_{i,j}\{D_i^j\}$.

We have the following guarantee:

1. *If* $\sum_{\tau_i \in \mathcal{T}} \mathsf{dbf}_i(t, \vec{F_i}) \leq \mathsf{sbf}^a(t),\ \forall t \in T_a$*, the system is EDF-schedulable on a unit-speed processor.*

2. *If* $\exists t \in T_a, \quad \sum_{\tau_i \in \mathcal{T}} \mathsf{dbf}_i(t, \vec{F_i}) > \mathsf{sbf}^a(t)$*, the system is EDF-infeasible on a* $\frac{1}{1+\epsilon}$*-speed processor.*

Proof. We first show the sufficiency on unit-speed processors for Equation 4.4 by contradiction.

Assume the task system satisfies Equation 4.4 but is infeasible. This means that there exists a time interval length t' when Equation 4.1 is violated (i.e., $\sum_{\tau_i \in \mathcal{T}} \mathsf{dbf}_i(t', \vec{F_i}) > \mathsf{sbf}(t')$), since it is a necessary and sufficient condition. Assume that $t' \in \left(t_0 * (1+\epsilon)^k, t_0 * (1+\epsilon)^{k+1}\right)$ for some $k \in \mathbb{N}$. Note that $\mathsf{sbf}^a(t')$ equals $\mathsf{sbf}(t_0 * (1+\epsilon)^{k+1})$ which is $t_0 * (1+\epsilon)^k$. However, note that it is known that the demand function is monotonically non-decreasing [8]. Thus, if $\sum_{\tau_i \in \mathcal{T}} \mathsf{dbf}_i(t', \vec{F_i}) > \mathsf{sbf}(t') > \mathsf{sbf}^a(t')$, then $\sum_{\tau_i \in \mathcal{T}} \mathsf{dbf}_i(t_0 * (1+\epsilon)^{k+1}, \vec{F_i}) > \mathsf{sbf}^a(t_0 * (1+\epsilon)^{k+1})$ which is a contradiction of Equation 4.4.

We now proof the infeasibility on a slower processor when Equation 4.4 is not satisfied. In order to prove the "speed-up factor", assume $\sum_{\tau_i \in \mathcal{T}} \mathsf{dbf}_i(t^*, \vec{F_i}) > \mathsf{sbf}^a(t^*)$ at time t^*. It must be that $t^* > t_0$ since for all values of $t \leq t_0$ the $\mathsf{dbf}_i(t, \vec{F_i})$ is zero by supposition that t_0 exceeds the minimum frame relative deadline. Furthermore, it is easy to observe that for all $t \geq t_0$, the $\mathsf{sbf}(t)$ is at most $(1+\epsilon)$ times larger than $\mathsf{sbf}^a(t)$. From this, we have:

$$\begin{aligned}
\max_{t>0} \frac{\sum_{\tau_i \in \mathcal{T}} \mathsf{dbf}_i(t, \vec{F}_i)}{\mathsf{sbf}(t)} &\geq \frac{\sum_{\tau_i \in \mathcal{T}} \mathsf{dbf}_i(t^*, \vec{F}_i)}{\mathsf{sbf}(t^*)} \\
&\geq \frac{\frac{\sum_{\tau_i \in \mathcal{T}} \mathsf{dbf}_{i(t^*, \vec{F}_i)}}{(1+\epsilon)}}{\frac{\mathsf{sbf}(t^*)}{(1+\epsilon)}} \\
&\geq \frac{\sum_{\tau_i \in \mathcal{T}} \mathsf{dbf}_i(t^*, \vec{F}_i)}{\mathsf{sbf}^a(t^*) * (1+\epsilon)} \text{ (By Equation 4.3)} \\
&\geq \frac{1}{1+\epsilon} \text{ (By assumption).}
\end{aligned}$$

Thus, we have proved that the speed-up factor is $1 + \epsilon$. □

We can now apply Theorem 3 to modify the MILP to create a sufficient approximate feasibility test for the GMF-PA task model with arbitrary, real-valued parameters. To do so, we simply limit the range of t to now be T_a for all constraints that depend upon t, and modify Line 9 of MILP to be $\sum_{i=0}^{n-1} y_{i,t} \leq \mathcal{L} * \frac{\boldsymbol{t}}{1+\epsilon}$. Clearly, this reduces the number of constraints by a logarithmic factor (dependent upon our choice of ϵ). We refer to this approximate assignment algorithm as MILP-ϵ.

In all, the approximate MILP is a sufficient feasibility test. The number of the time interval lengths being tested is reduced from $O(H)$ to $O(\log_{1+\epsilon} H)$. Since the number of variables and number of equations depend on the number of intervals, the running time is greatly reduced. We have done exhaustive experiments in Chapter 4.6. Our MILP algorithm and approximation algorithm also work for the multi-segment self-suspending tasks represented by our GMF-PA model. The transformation is presented in the next chapter.

4.5 Fixed-Relative-Deadline Assignment for Multi-Segment Self-Suspension Tasks

Before applying the GMF-PA model to multiple-segment self-suspending tasks, we first state the limitation of the deadline assignment in the previous papers [23] [39]. Equal-deadline assignment (EDA) assigns each computational frame of τ_i the same deadline $D_i^j = \frac{\mathcal{P}_i - \sum_{j=0}^{m_i-2} S_i^j}{m_i}$. Such assignment is restrictive because the system is already infeasible if $C_i^j > D_i^j$. The situation may happen in several applications. For example, in computation offloading of a robot car [57], the cost of the first computational frame (image capture) and the suspending frame occupy most of the time in one period, and the cost of the second computation frame (path planning) is relatively small.

In order to generate a fair comparison with the recent results [23, 39], our MILP will be transferred to schedule the self-suspending tasks under implicit-deadline system. We set $D_i^k = P_i^k$ (Line 4 in MILP will be automatically satisfied) and $\mathcal{D}_i = \mathcal{P}_i$. The variables $D_i^k, P_i^k, \mathcal{D}_i$, and $\mathcal{P}_i$ are reduced to D_i^k and $\mathcal{P}_i$ for all i and k. In this case, the end-to-end deadline of τ_i is $\mathcal{P}_i$. Because the previous work [23, 39] have no frame constraints, we set $\underline{D}_i^k = E_i^k$ and $\overline{D}_i^k = \mathcal{P}_i$ for any computational frame ϕ_i^k. Note that $D_i^j = S_i^j$ in any suspending frame ϕ_i^j due to the setting introduced in Chapter 3.2.1. The constraints from Lines 3 to 5 thus become:

1. $\boldsymbol{E_i^k} \leq D_i^k, \ \ \forall i, k.$
2. $\sum_{k=0}^{N_i-1} D_i^k \leq \boldsymbol{\mathcal{P}_i}, \ \ \forall i.$

Since the evaluations [23] contain the situation when $U_{cap} = 1$, the range of the interval

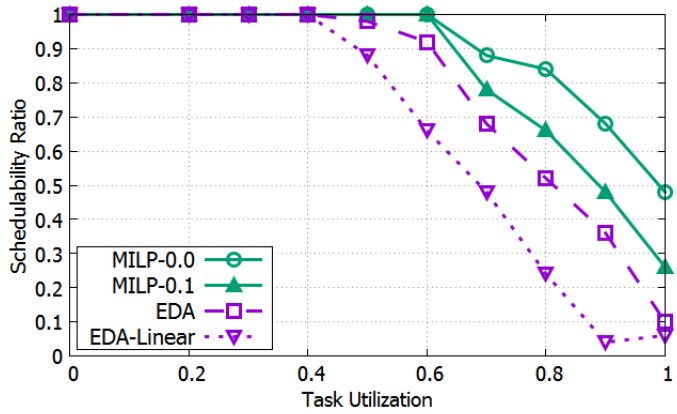
length t becomes $\lceil \min\left(\frac{U_{cap}}{1-U_{cap}} \cdot \max_{\tau_i \in \tau}(\mathcal{P}_i - E_i^{min}), hyperperiod\right) \rceil$. The task system hyperperiod $hyperperiod = lcm\{\mathcal{P}_0, \mathcal{P}_1, ..., \mathcal{P}_{n-1}\}$ is the least common multiple of the n tasks' cycle periods. The other lines in our MILP and approximation algorithms remain the same.

4.6 Evaluation

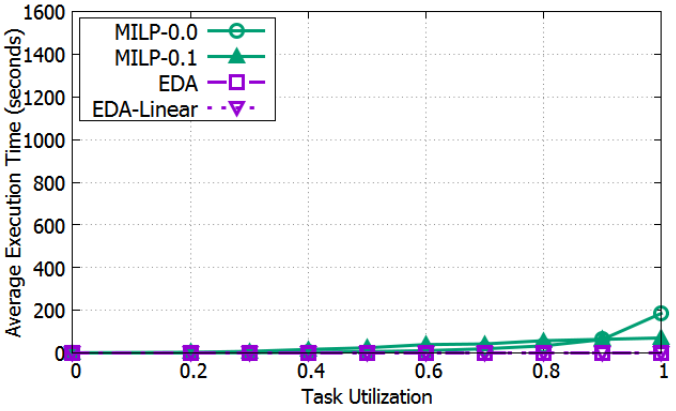
We have implemented our MILP-based algorithm and its approximation algorithm MILP-ϵ using the commercial solver GUROBI [1] in MATLAB. GUROBI is a state-of-the-art mathematical programming solver that has great performance in solving linear and mixed-integer programming problems.

As far as we know, there is no previous work to directly compare with the flexible generalized multiframe model, and we compare our work with the application to self-suspending tasks [23, 39] on uniprocessor systems. We restrict our model to compare with one-segment self-suspending tasks using EDA under EDF scheduling [23]. We also compare our work with multiple-segment self-suspending tasks [39] using EDA under EDF scheduling. Since EDF scheduling is optimal under uniprocessor systems, we compare EDA and our algorithms for multiple-segment self-suspending tasks under EDF. MILP-0.0 is our necessary feasibility test and MILP-ϵ ($\epsilon > 0$) is our approximation algorithm. The algorithms used in the previous paper [23, 39] are EDA and EDA-linear [23]. The algorithm EDA assigns each task the identical deadline for each computational frame and EDA-linear is EDA's approximation algorithm.

For one-segment self-suspending tasks, we follow the similar setting of the previous paper [23]. Tasks are randomly generated. There are three ranges for task utilizations and the length of self-suspensions. The ranges for task utilization U_i are [0.01, 0.1] (l,



(a) mm-ratio,$\mathcal{P}_i \in [1, 10]$

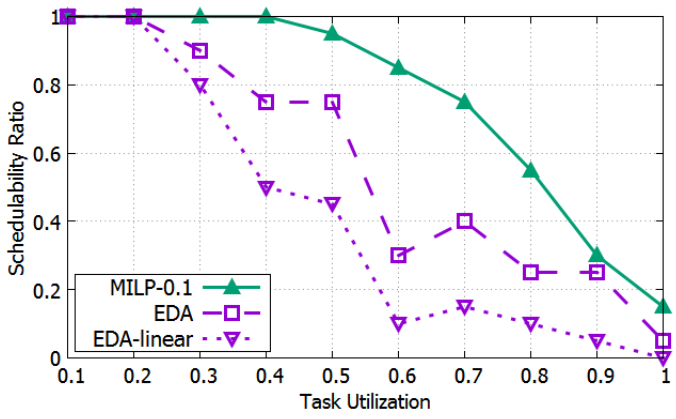


(b) mm-time,$\mathcal{P}_i \in [1, 10]$

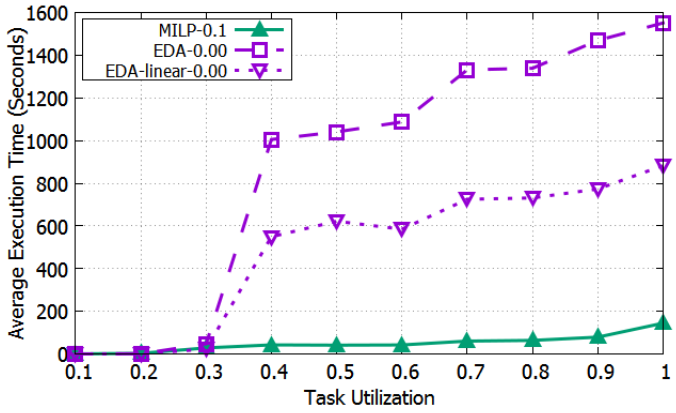
short for “light”), [0.1, 0.3] (m short for “medium”), and [0.3, 0.6] (h, short for “heavy”). The ranges for the length of suspensions are $[0.01 * (1 - U_i) * T_i, 0.1 * (1 - U_i) * T_i]$ (s, short for “short”), $[0.1 * (1 - U_i) * T_i, 0.3 * (1 - U_i) * T_i]$ (m, short for “medium”), and $[0.3 * (1 - U_i) * T_i, 0.6 * (1 - U_i) * T_i]$ (l, short for “long”). Tasks are randomly generated until the total utilization is equal to U_{cap}. The utilization of the last task is reduced if the total utilization is larger than the cap.

We show the results for medium utilization and medium suspension length in two different settings. The results of the other combinations follow a similar pattern. For the reason that the least common multiple of periods could be very large, e.g., $2.3 * 10^8$ for the period range $[1, 20]$. We first generate Figure 11(a) and 11(b) in which the tasks have small cycle periods that randomly chosen in [1,10]. Such task systems are randomly generated 50 times under each utilization cap and the running time in the figure is the average running time. The schedulability ratio is the number of feasible systems over the total systems generated. MILP and MILP-0.1 have higher schedulability ratio than EDA and EDA-linear, but have longer running times as shown in Figure 11(b). MILP-0.1 can schedule at most 16 (44) % more than EDA (EDA-linear) when $U_{cap} = 1$ (0.9). MILP (MILP-0.1) uses at most around 184(70) seconds more than EDA and EDA-linear when $U_{cap} = 1$. In order to generate a set of tasks with larger hyperperiods, we have the experiments shown in Figure 11(c) and 11(d). The cycle period of each task is randomly chosen in [1,50] and has lower and upper bounds. Such task system is randomly generated 20 times at each utilization cap. In the figure, MILP-0.1 behaves better on both schedulability ratio and running time.

For multiple-segment self-suspending tasks, we follow the similar setting of the previous paper [39]. Due to high execution times for both EDA and MILP, we apply the same approximation of Theorem 3 to EDA and compare with our MILP-ϵ. The cycle periods $\mathcal{P}_i$ are randomly generated from $[10, 1000]$. Under each U_{cap}, ten tasks are randomly generated. The UUniFast algorithm [12] is used to divide the utilizations U_i of the ten tasks under U_{cap}. The total execution time $C_i = \mathcal{P}_i * U_i$, and the total suspension delay is generated under the three ranges which are the same as the ones in one-segment self-suspending tasks [23]. The



(c) mm-ratio,$\mathcal{P}_i \in [1, 50]$

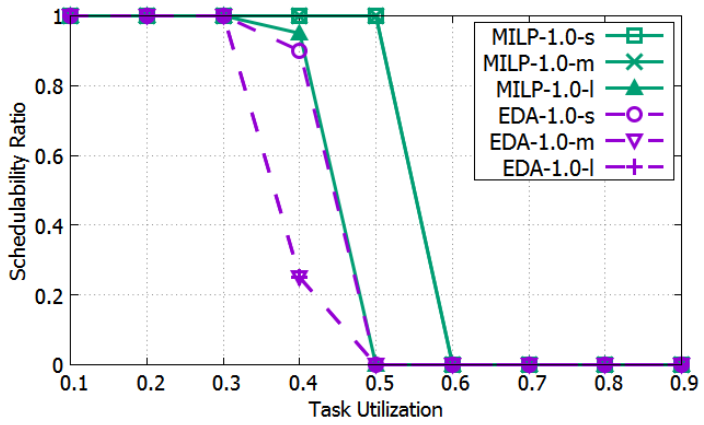


(d) mm-time,$\mathcal{P}_i \in [1, 50]$

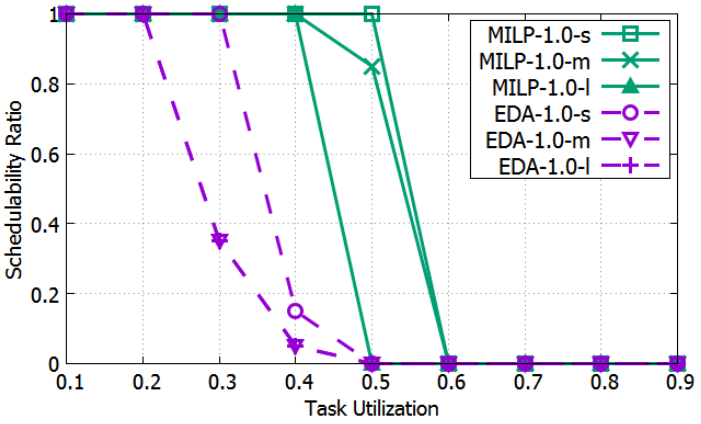
Figure 11: The utilization of tasks is in medium (m) range and the suspending lengths are in medium (m) range. In Figures 11(a) and 11(b), task periods $\mathcal{P}_i$ are in the range $[1, 10]$. In Figures 11(c) and 11(d), task periods $\mathcal{P}_i$ are in the range $[1, 50]$. In Figures 11(c) and 11(d), let the $maxTaskNum(U_{cap}, l_{uti}) = \lceil \frac{U_{cap}}{l_{uti}} \rceil$ be the maximum number of tasks under the utilization cap U_{cap} and the minimum utilization l_{uti} in a utilization range (For example, $U_{cap} = 0.3$ and $l_{uti} = 0.1$ under the medium utilization and $maxTaskNum(0.3, 0.1) = 3$). We let the upper bound of the hyperperiod be $hp_u = min(2 * 10^6, 50^{maxTaskNum(U_{cap}, l_{uti})})$ and the lower bound of the hyperperiod be $hp_l = 0.5 * hp_u$.

UUniFast algorithm is also used to divide total execution times and suspension lengths to the ones for multiple frames. Figure 11 shows that our MILP-ϵ is better than EDA-ϵ for both two and five suspending frames. Note that the utilization, execution time and suspending length are uniformly distributed. The result of EDA in other distributions may be even worse since the deadlines are equally assigned.

In all, our MILP and MILP-ϵ algorithms always yield higher schedulability ratio on both one-segment and multi-segment self-suspending tasks. MILP-ϵ has the lowest running time when the hyperperiod is large.



(a) 2 s/m/l-ratio



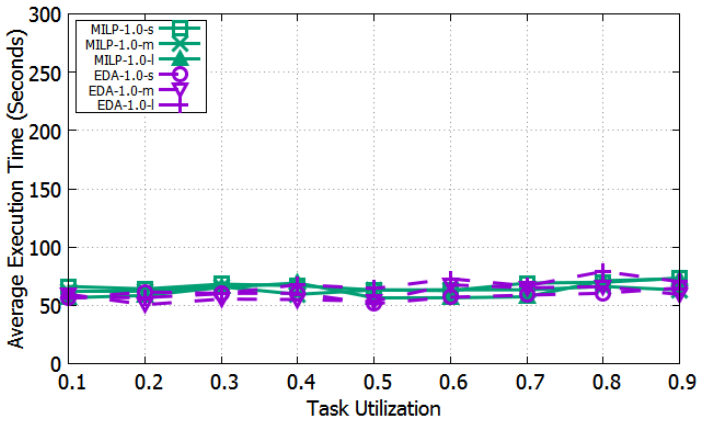
(b) 5 s/m/l-ratio

4.7 Summary

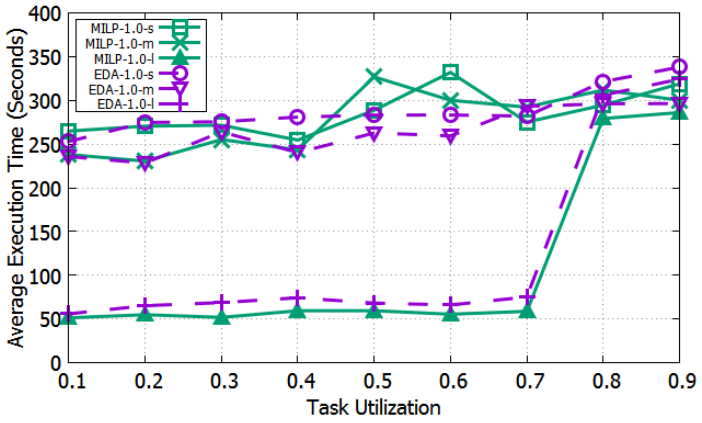
Under the flexible GMF-PA model, we propose a mixed-integer linear programming (MILP) based algorithm that can select deadlines and periods. Our MILP-based algorithm is an exact feasibility test when parameters are integers, and a necessary feasibility test in general. In order to reduce the running time of the MILP algorithm, we propose an approximation algorithm MILP-ϵ based on the supply bound function. The number of time intervals being tested is bounded by a logarithmic function of the task system parameters. We prove that the MILP-ϵ is a sufficient feasibility test.

We apply our MILP and MILP-ϵ to self-suspending tasks. We remove the assumption that

the deadlines are fixedly equally assigned in the previous work. Exhaustive experiments for both one-segment and multiple-segment self-suspending tasks have shown that our algorithms have improved the schedulability ratio and running time compared to the previous results.



(c) 2 s/m/l-time



(d) 5 s/m/l-time

Figure 11: Figures 12(a) and 12(b) compare the schedulability ratio for the tasks that have two-suspending and five-suspending frames, respectively. Three ranges of the self-suspending length are considered. For instance, MILP-1.0-s shows the ratio from the approximation algorithm MILP-1.0 on scheduling short self-suspending tasks. Since there are ten tasks in a system under each U_{cap}, the average execution times of a system under different U_{cap} are around the same. Figures 12(c) and 12(d) show the corresponding average execution time. The average execution time for the task system with two and five suspending frames is around 60 and 240 seconds, respectively.

CHAPTER 5 A CASE STUDY OF A ROBOT CAR TRACKING SYSTEM IN UNIPROCESSORS

In this chapter, we give a case study of a robot car tracking system [57]. This case study is based on the settings/parameters of a real robot and a server[6] [57] and we run our MILP-based algorithms on such systems. We use this case study for two reasons:

1. We first relax an assumption made in previous work [57, 75, 79]. The papers assumed the stages of the image analysis are independent. The stages should be dependent [70] (details are shown in Chapter 5.2), and we then map the image analysis procedures to GMF-PA tasks.

2. We map the image analysis of the tracking robot to GMF-PA tasks and compare our parameter-assignment algorithm MILP-ϵ algorithm with EDA. Using the data generated by [57], MILP-ϵ can schedule more systems than EDA (shown in Chapter 5.3).

We state the components of a tracking robot car in Chapter 5.1. Based on current components, we state the assumption made in previous papers and first propose its solution for scheduling dependent components in Chapter 5.2. We give the parameter setting of our case study and experiment results in Chapters 5.3 and 5.4.

5.1 Components of a tracking robot

In this subchapter, we introduce the basic components of a tracking robot car [57]. A typical tracking robot consists of image capture, image analysis, and path planning procedures. Image capture and path planning procedures which execute in robot cars

[6]We reuse the data generated by [57]. The systems they used are a robot car and a Linux server. The car is a pioneer 3DX robot with an Intel core2-duo 2 GHz processor and two cameras. The resolution of both cameras is 640×480. The server is an Intel Xeon Linux server with eight quad-core 2.33 GHz processors and 8 GB RAM.

usually take small execution time relative to the image analysis procedure. We focus on the image analysis procedure since it takes relatively longer execution time and sometimes offloads a portion of execution to a remote server. When a portion of execution is offloaded to a server, the time for transmitting the execution to the server and later back to the car, and the time for executing on a server can be considered as self-suspension lengths on a car.

Image analysis consists of three modules that are motion detection, stereovision, and object recognition. Motion detection is a module that detects the motion of a target object. The paper [57] used background subtraction to detect the differences of two adjacent pictures taken by cameras. The module binocular stereovision is to detect the distance between the target object and the robot car, by a left and a right cameras on the robot car. The binocular stereovision [57] used the disparity map to compute the distance of the object. Basically, the distance is calculated by the shifted amount of pixels (disparity) between two adjacent images and current known distance. Object recognition consists of feature extraction and the search through the feature database. The running time of feature extraction depends on the number of features which are effected by the background complexity of the image. The running time of the search depends on the number of features and the size of target object database.

5.2 Assumption in current research

In a tracking robot car, [57] assumed that the modules of image analysis are independent, because the data exchanges between modules are small and there is only one moving object which is tracked by the robot car. Based on this case study, multiple works, e.g., server resource reservation and computation offloading by using timing unreliable components [75,

79] make the same assumption that modules are independent in their case studies. The modules should be dependent [70] in a real more complex situation especially when there are multiple moving objects. As far as we know, we are the first to relax this assumption in real-time systems and the modules should not be independent. The execution sequence is object recognition, motion detection and stereovision. Note that this sequence is also a valid sequence[7] in the previous work [57]. That is, we should first detect the target object from images taken by cameras, and detect its motion and the distance to the robot car. In all, we model an image analysis procedure as a GMF-PA task. Each GMF-PA task represents a target object and this model can be extended to multiple-object recognition and tracking.

In this chapter, we relax the assumption that the modules of image analysis are independent for the purpose of evaluating our schedulability analysis. Note that we are only using the data from the original experiment and not implementing the robot car system. Due to this assumption is applied to different areas and system setup [57, 75, 79], we do not compare experiments with them. For example, the goal of the paper [57] is to minimize the total execution time of modules, but not to analyze the schedulability in hard real-time systems. The papers consider each module as an independent task and each task can be considered a one-segment self-suspending task. We instead extend and map the modules of image analysis to a multiple-segment self-suspending/GMF-PA task, then compare our algorithm with EDA on scheduling self-suspending tasks.

[7] In the work [57], the execution of modules are represented by a graph, we choose a reasonable valid sequence from the graph. The aim is to relax the independence of the modules. This sequence is also compatible with the sequences in a more general context [70].

5.3 Parameter calculations and computation offloading decisions

In this subchapter, we present how to calculate parameters (i.e., running time, suspensions, deadline, etc.) of each module in image analysis. With these parameters, we present the computation offloading decisions and how we map image analysis procedures to GMF-PA tasks.

Each module of image analysis can execute on a car, a server, or both (partial offloading). The decision to offload computation to a server (or not) depends on network bandwidth and execution times on the server and car. If the execution time of a module is smaller on a car, we do not offload; otherwise, we offload the computation to a server. If we offload the computation to a server, we consider the total time (transmission, and execution on a server) as self-suspension lengths on a car. In this case, we simplify the execution on two-processor systems to uniprocessor systems. Our GMF-PA model and the MILP-based algorithm thus can be applied to such applications. This offloading decision is based on the previous work [57], and we reuse the data [57] to investigate a more extensive test.

The running times of motion detection ($E^{md,r}$)[8] and stereovision ($E^{sv,r}$) tested on a robot are 0.06 seconds and 0.2 seconds, respectively. The running times of motion detection ($S^{md,s}$) and stereovision ($S^{sv,s}$) on a server and transmission (suspensions on a car) are calculated by Equation 5.1.

$$\begin{aligned} S^{md,s} &= \frac{E^{md,r}}{\eta} + \frac{d_I}{\beta} \\ S^{sv,s} &= \frac{E^{sv,r}}{\eta} + 2 * \frac{d_I}{\beta} \end{aligned} \tag{5.1}$$

[8] In the GMF-PA model, E_i^j is the j'th frame execution time of task τ_i. In this subchapter, we omit subscript for simplicity.

The processing speed ratio η between a server and a robot can be up to a factor of 20. I.e., the sever processor has higher processor speed and 20 times than the robot processor speed. Each file size d_I is 10 kB, and wireless bandwidth β is up to 140 kbps (according to the setup of [57]). $\frac{E^{md,r}}{\eta}$ is the running time of motion detection on the server. $\frac{d_I}{\beta}$ is the transmission time for one image. The summation of the two is the considered as the suspension length $S^{md,s}$ on the robot car. The suspension of the module stereovision is similar. In the module stereovision, the robot sends two images captured by the left and right cameras on the robot, and the two images are used to accurately compute the distance of the object so that the transmission time is $2 * \frac{d_I}{\beta}$. In our case study, we vary server processing speed ratio η and bandwidth β.

The computation[9] of object recognition c_{or} is given by Equation 5.2.

$$
\begin{aligned}
c_{or} &= c_f + c_s \\
c_f &= k_f * n_f \\
c_s &= k_s * n_f * \gamma \\
n_f &= -16089\alpha^4 + 30411\alpha^3 - 14268\alpha^2 + 3692\alpha + 1135
\end{aligned}
\tag{5.2}
$$

Object recognition consists of feature extraction and the search through a feature database. The computations of extraction c_f and search c_s depend on their algorithm-specific constants k_f and k_s, respectively. The number n_f of features varies on images and background complexity α. Complexity α ranges between 0 and 1. By regression, the equation between n_f and α obtains a recognition accuracy of at least 90%. For example, when background

[9]The paper [57] refer to computations as cycles. For example, c_f is the total number of cycles needed to extract the features of a picture.

complexity $\alpha = 0.5$, $n_f = 2210$ features ensure accuracy at least 90%. γ is the size of the object database. The running time and suspension of object recognition are calculated in Equation 5.3 below:

$$\begin{aligned} E^{or,r} &= \frac{c_{or}}{f_r} \\ S^{or,s} &= \frac{E^{or,r}}{\eta} + \frac{d_I}{\beta} \\ E^{or,p} &= \frac{c_f}{f_r} \\ S^{or,s} &= \frac{c_s}{f_r \eta} + \frac{d_f}{\beta} \end{aligned} \tag{5.3}$$

f_r is the processor speed of the car and d_f is the file size of features. The partial offloading consists of the feature extraction time $E^{or,p} = \frac{c_f}{f_r}$ on the robot and search time $S^{or,s} = \frac{c_s}{f_r \eta}$ on the server. We vary α, β, and γ to evaluate this case study. For example, $E^{or,r} = \frac{k_f * n_f}{f_r} = 1.7$ when $n_f = 2210$ in the base case, we vary α to calculate n_f and $E^{or,r}$ in our evaluation.

For each module in the image analysis, the decision to offload to a server (or not) depends on Equation 5.4 below. This equation is affected by the network bandwidth and execution times on the server and car. If the execution time of a module is smaller in a car than the server, we do not offload; otherwise, we offload the computation to a server. For example, if $E^{md,r} > S^{md,s}$ for a motion detection module, we offload the execution to the server. The running time of this module is 0, and the self-suspension length is $S^{md,s}$. Similarly, for E^{or} and S^{or}, we calculate $\min\{E^{or,r}, S^{or,s}, E^{or,p} + S^{or,p}\}$ to make the offloading decision. For example, if $E^{or,p} + S^{or,p}$ is the smallest of the three values, we partially offload

and set $E^{or} = E^{or,p}, S^{or} = S^{or,p}$.

$$
\begin{aligned}
E^{md} &= \begin{cases} 0, & \text{if } E^{md,r} > S^{md,s} \\ E^{md,r}, & \text{otherwise} \end{cases} \\
S^{md} &= \begin{cases} S^{md,s}, & \text{if } E^{md,r} > S^{md,s} \\ 0, & \text{otherwise} \end{cases}
\end{aligned}
\tag{5.4}
$$

$$
\begin{aligned}
E^{sv} &= \begin{cases} 0, & \text{if } E^{sv,r} > S^{sv,s} \\ E^{sv,r}, & \text{otherwise} \end{cases} \\
S^{sv} &= \begin{cases} S^{sv,s}, & \text{if } E^{sv,r} > S^{sv,s} \\ 0, & \text{otherwise} \end{cases}
\end{aligned}
\tag{5.5}
$$

$$E^{or} = \begin{cases} E^{or,r}, & \text{if } E^{or,r} = \min\{E^{or,r}, S^{or,s}, E^{or,p} + S^{or,p}\} \\ E^{or,p}, & \text{else if } E^{or,p} + S^{or,p} = \min\{E^{or,r}, S^{or,s}, E^{or,p} + S^{or,p}\} \\ 0, & \text{otherwise} \end{cases}$$

$$S^{or} = \begin{cases} 0, & \text{if } E^{or,r} = \min E^{or,r}, S^{or,s}, E^{or,p} + S^{or,p} \\ S^{or,p}, & \text{else if } E^{or,p} + S^{or,p} = \min\{E^{or,r}, S^{or,s}, E^{or,p} + S^{or,p}\} \\ S^{or,s}, & \text{otherwise} \end{cases} \tag{5.6}$$

Until now, we have stated how to calculate the execution times and self-suspensions of three modules in image analysis and in what situations to offload computation. If the execution of a module is offloaded to a server, we consider the execution time on the server as a self-suspension length or a new frame with zero execution time on the robot car. The end-to-end/task deadline is calculated by $\frac{d \sin \theta}{v_t}$. The distance between the robot and the object is d, camera's angle view is θ, and the speed of the target object is v_t. We assume an implicit deadline system in which each task deadline equals task period.

Now, we introduce the mapping of an image analysis procedure to a GMF-PA task. For simplicity, we first map an image analysis procedure to a multiple-segment self-suspending task. The mapping from a multiple-segment self-suspending task to a GMF-PA task is introduced in Chapters 3.2.1 and 4.5. In the image analysis procedure, each module is represented by a computational frame and a self-suspending frame. Modules of image analysis are dependent and execute in sequence. Each self-suspending task $\tau = ((E^{md}, S^{md}, E^{sv}, S^{sv}, E^{or}, S^{or}), D, D)$. That is, the frame execution sequence of this self-suspending tasks on the robot car is motion detection, stereovision, and object recognition. The calculation of

Table 4: Notations.

Symbols	Definitions
E^{md}, E^{sv}, E^{or}	frame execution time
$E^{md,r}, E^{sv,r}, E^{or,r}, E^{or,p}$	robot execution time
S^{md}, S^{sv}, S^{or}	frame suspension length
$S^{md,s}, S^{sv,s}, S^{or,s}, S^{or,p}$	robot suspension length
D	task deadline and period (implicit system)
α	background complexity
β	wireless bandwidth
γ	database size
d_I	file size
η	speed ratio between a server and a robot
c_f, c_s, c_{or}	computations of feature extraction, search database, and object recognition
θ, v_t, d	camera angle, object speed, and distance to the object
n_f	number of features

frame executions and suspensions (offloading options) depends on Equation 5.4. Due to the implicit system, task periods equal task deadlines ($D = \frac{d \sin\theta}{v_t}$). This multiple-segment self-suspending task then can be transferred to a GMF-PA task as shown in Chapters 3.2.1 and 4.5. In all, the notations can be referred in Table 4.

5.4 Experiments

In this subchapter, we compare our algorithm MILP-ϵ ($\epsilon = 0.1$) and EDA on scheduling GMF-PA tasks. In the experiment of this case study, the car keeps track of three objects, i.e., three GMF-PA tasks. The distances d of the objects and the robot are 2, 3 and 4 meters, respectively[10]. The angle view θ is $90°$, and the speeds of the objects are 1.5 m/sec. The task deadlines are thus 1.3, 2, and 2.6 seconds, respectively.

We divide α, γ, and η into three ranges as shown in Table 5. There are three ranges (small, medium, and high) divided by the upper bounds of real data [57]. For instance,

[10]In this case, it is assumed that the three objects have same speed and move parallel, this can be applied in the scenario of tracking cars in different lanes [70].

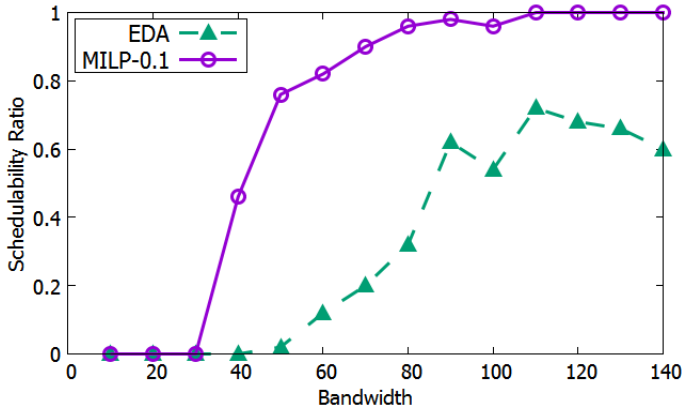
Table 5: Ranges of α, γ, and η.

	α	γ	η
small	$0.1 \sim 0.4$	$1000 \sim 2000$	$1 \sim 7$
medium	$0.4 \sim 0.7$	$2000 \sim 3000$	$7 \sim 14$
high	$0.7 \sim 1.0$	$3000 \sim 4000$	$14 \sim 20$

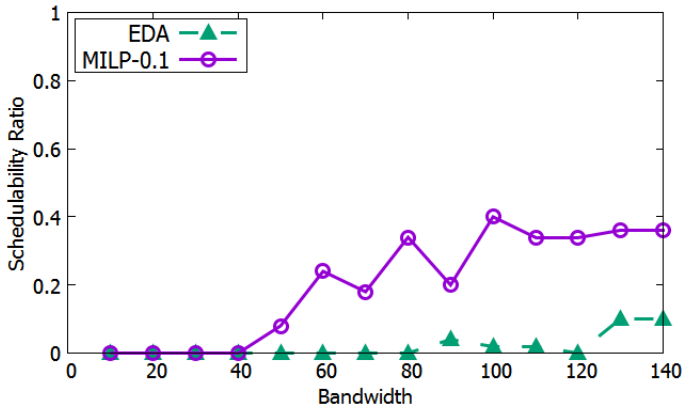
the processing speed ratio η can be at most 20 in the setting of the server and network bandwidth β can be at most 140kbps [57]. We vary β from 10 to 140 with a step 10. At each value of β, we test 50 systems with randomly generated α, γ, and η in their corresponding ranges. There are multiple range combinations of α, γ, and η. We show two representative results since the pattern of figures is similar.

We test high range of the background complexity α and the database size γ which represent a high workload of object recognition, and evaluate all ranges of speed ratio η and network bandwidth β which represent the capability of resources. For small and medium workloads (α and γ), the lines in Figure 12 will be shifted left compared to high workload, which means algorithms can schedule more systems. We test our sufficient schedulability test MILP-0.1 and EDA[11]. Figure 12 shows our algorithm MILP-0.1 has high schedualbility ratio than EDA on medium and high ranges of bandwidth. Since no algorithms can schedule any system in any bandwidth at the low range of bandwidth, we do not show the figure. There are two main reasons that our algorithm can schedule more systems. First, MILP-0.1 jointly consider the deadline assignment with the schedulability test. Second, object recognition takes more running time than motion detection and stereovision so that the identical deadlines assigned by EDA cannot reflect the needs of the modules. Some frame

[11]Note that the EDA algorithm in the previous paper [23] only considers one-segment self-suspending tasks, we extend EDA using our MILP. In MILP, we add one more constraint to let the frame deadlines of each task be equal.



(e) high η



(f) medium η

Figure 12: Under different network bandwidths β, this figure shows the schedulability ratio under high workload with high speed ratio η in Figure 12(e) and medium η in Figure 12(f).

execution time is larger than the assigned deadline by EDA. Since there are three GMF-PA tasks, the running time difference between MILP-0.1 (around 4 seconds) and EDA (around 0.7 seconds) is small.

CHAPTER 6 PARAMETER ASSIGNMENT BASED ON LINEAR PROGRAMMING VIA CONCAVE APPROXIMATIONS OF DEMAND

In Chapter 4, we present an MILP-based approximation algorithm. The algorithm runs efficient but still is MILP-based. In general, an MILP-based algorithm does not scale well. In order to solve this problem, we give an efficient linear programming (LP) based algorithm in this chapter. Other than that, the MILP-based approximation algorithm is based on the supply bound function while the LP-based algorithm presented in this chapter is based on the demand bound function.

6.1 Introduction

We introduce our GMF-PA model and its application self-suspension tasks that can be represented using the GMF-PA model but the problem size can be very large, e.g., in automotive systems. In the keynote [19] of ECRTS 2012, Buttle has shown many scheduling challenges as the number of ECUs in vehicles increases rapidly each year; there are more than 100 ECUs nowadays and each task can easily have 50-300 functions. In such complex systems, there are several self-suspension tasks (each consisting of multiple functions) and their end-to-end latencies need to be maintained in distributed settings.

We analyze the scheduling of suspension tasks between a robot car and a remote server in Chapter 5. We also analyze the scheduling of self-suspension tasks in distributed systems in Chapter 7. The MILP-based techniques can give successful schedules, but not time-efficient in general. More time-efficient methods are needed as the needs of more complex functions in distributed/multi-core environments required in complex systems such as car systems. Due to this reason, this chapter analyzes an approximation scheme based on linear programming to improve time efficiency.

Although the GMF-PA model is more flexible than the GMF model, it has been shown that both the feasibility and the parameter selection problems are very hard to solve. On the feasibility side, Ekberg and Yi [33] proved that the feasibility of sporadic task systems remains coNP-complete under bounded utilization. On the parameter selection side, the priority assignment of subtasks in end-to-end task systems (originally the classical job-shop scheduling algorithm) has been shown to be NP-hard [36]. The scheduling of self-suspending tasks (even for self-suspending tasks with at most two frames) is NP-hard in the strong sense [66].

In order to address the feasibility test and parameter selection problem, We gave an exact schedulability test of GMF-PA tasks when frame parameters are integers. The test is based on mixed-integer linear programming (MILP) under EDF scheduling in uniprocessor systems. A sufficient, MILP-based schedulability test was also developed. Although this sufficient approximation algorithm is quite efficient, it is still MILP-based and thus may require exponential-time to solve in general. The goal and contribution of this chapter are to give an efficient linear programming-based algorithm that can determine the feasibility and select the frame parameters of GMF-PA tasks.

The MILP-based algorithm contains a set of integer variables which form a set of staircase functions/constraints (detailed in Chapter 4). To transform the MILP-based algorithm into a LP-based algorithm, our idea is to use a set of linear functions to approximate all staircase functions. As such, the selection of the slope values of the linear functions is directly related to the schedulability of a system; if the slope values are not properly set, the linear functions can grossly over-approximate, resulting in low schedulability ratio (the number of successfully scheduled systems over the total tested).

In order to get a close approximation, we first use a set of concave functions that very closely tracks the demand staircase functions to incur only a very small speed-up factor compared to the MILP algorithm. Since there exist no known efficient methods to solve concave programming problems, we use the concave functions to guide the slope assignment of linear functions in our iterative LP-based algorithm. That is, the LP algorithm runs multiple times during which the algorithm adjusts the slopes of the linear functions based on the concave functions. According to experiments, after a small number of iterations, the LP-based algorithm can approach (or reach) the local optimal[12]. We apply the LP-based algorithms to schedule self-suspending tasks under EDF scheduling in uniprocessor systems as a test case.

Our Contributions in this Chapter:

- We give a concave approximation algorithm based on the MILP algorithm and prove the speed-up factor of the algorithm is $(1+\delta)^2$ with respect to the exact schedulability test of GMF-PA tasks under EDF scheduling on uniprocessors. The positive constant δ is a user-defined constant which can be made arbitrarily close to zero.

- Since there is no known tractable way to solve a concave programming problem, we develop a LP-based heuristic algorithm based on the concave approximation algorithm for GMF-PA tasks. The LP-based algorithm is an efficient schedulability test and can select frame parameters at the same time.

- We apply the LP-based algorithm to schedule multiple-suspending tasks. To exploit the unique property of one-suspending tasks, as opposed to multi-suspending tasks,

[12]The local optimal of the iterative LP-based algorithm is reached when all variables converge.

we present an improved heuristic algorithm for GMF-PA tasks.

- We conduct extensive experiments and show that the LP-based algorithms with fixed numbers of iterations outperform previous work in terms of schedulability and average running time. The fixed numbers of iterations make the LP-based algorithms pseudo-polynomial (the input size depends on the maximum interval length [8]), which is more efficient than the MILP-based approach.

We state the goal of this chapter in Chapter 6.2. The concave approximation algorithm based on the MILP algorithm is presented in Chapter 6.3. Since concave programming algorithm does not scale well, two iterative LP-based algorithms are presented in Chapter 6.4. After applying the LP-based algorithms to self-suspending tasks, Chapter 6.5 provides extensive experimental results compared to state-of-the-art results. At last, Chapter 6.6 concludes this chapter.

6.2 Problem Statement

Let $\mathsf{dbf}_i^{\text{linear}}(t, \vec{F}_i)$ be the *task* demand bound function of a GMF-PA task τ_i within the interval length t generated by our approximation LP-based algorithm. Let $\vec{F}_i$ = [D_i^0, P_i^0, D_i^1, P_i^1,..., $D_i^{N_i-1}$, $P_i^{N_i-1}$] represent an assignment of values for all the task parameters (frame deadlines and periods) of task τ_i. We let $\mathsf{sbf}(t) = t$ since we focus on the demand bound function in this chapter. In a uniprocessor system $\mathcal{T}$, the sufficient condition for schedulability of a task set $\mathcal{T}$ is shown in Equation 6.1.

$$\sum_{\tau_i \in \mathcal{T}} \mathsf{dbf}_i^{\text{linear}}(t, \vec{F}_i) \leq t, \quad \forall t. \tag{6.1}$$

Problem Definition. *Given the above model, our goal is to find fast and effective assignment $\vec{F_i}$ of frame parameters of all tasks so that the worst-case demand $\sum_{\tau_i \in \mathcal{T}} \mathsf{dbf}_i^{linear}(t, \vec{F_i})$ over all time intervals of length t closely approximates the exact demand $\sum_{\tau_i \in \mathcal{T}} \mathsf{dbf}_i(t, \vec{F_i})$ under the MILP.*

6.3 The Concave Approximation Algorithm

For the ease of references, we also show the MILP algorithm in Chapter 4 in Figure 13. In this subchapter, we give a concave approximation algorithm for the MILP algorithm and prove the speed-up factor of the concave approximation algorithm (compared to the optimal FRD/the MILP algorithm) can approach one. Although there is no known efficient way to solve a concave programming problem, our concave approximation algorithm plays a key role in the LP-based algorithms presented in the next subchapter.

6.3.1 The Concave Functions

We first use the concave function (Equation 6.2) (illustrated by the blue dashed curve of Figure 14) to approximate the exact frame demand determined by the MILP in Line 6 of Figure 13.

$$\mathsf{dbf}_i^{\text{concave}}(t, D_i^{j,k}) = \max\left\{0, E_i^k \cdot (1+\delta) - E_i^k \cdot \delta \cdot e^{\mu \cdot (D_i^{j,k} + \lfloor \frac{t}{\mathcal{P}_i} \rfloor \cdot \mathcal{P}_i - t)}\right\} + \lfloor \frac{t}{\mathcal{P}_i} \rfloor \cdot E_i^k \tag{6.2}$$

The concave programming algorithm is constructed by replacing all staircase functions in Line 6 of Figure 13 with $y_{i,t}^{j,k} = \mathsf{dbf}_i^{\text{concave}}(t, D_i^{j,k})$ and removing all integer variables. The other lines in Figure 13 remain the same.

Equation 6.2 shows our proposed concave approximation function $\mathsf{dbf}_i^{\text{concave}}(t, D_i^{j,k})$ (e.g., the blue dashed curve in Figure 14) for the k'th frame demand of task τ_i during the

Parameter Selection and Exact Feasiblity Test

1 minimize: $\mathcal{L}$

2 subject to:

3 $\boldsymbol{E_i^k} \leq \underline{\boldsymbol{D_i^k}} \leq D_i^k \leq \overline{\boldsymbol{D}}_{\boldsymbol{i}}^{\boldsymbol{k}}, \quad \forall i, k.$

$\boldsymbol{E_i^k} \leq \underline{\boldsymbol{P_i^k}} \leq P_i^k \leq \overline{\boldsymbol{P}}_{\boldsymbol{i}}^{\boldsymbol{k}}, \quad \forall i, k.$

4 $D_i^k \leq P_i^k + D_i^{(k+1) \mod N_i}, \quad \forall i, k.$

5 $\sum_{k=0}^{N_i-1} P_i^k \leq \boldsymbol{\mathcal{P}_i}, D_i^{N_i-1} + \sum_{j=0}^{N_i-2} P_i^j \leq \boldsymbol{\mathcal{D}_i}, \quad \forall i.$

6 $\boxed{\begin{array}{l} y_{i,t}^{j,k} = x_{i,t}^{j,k} \cdot \boldsymbol{E_i^k} + \lfloor \frac{\boldsymbol{t}}{\boldsymbol{\mathcal{P}_i}} \rfloor \cdot \boldsymbol{E_i^k}, \quad \forall i, j, k, t. \\ \frac{\boldsymbol{t}-t_b}{\boldsymbol{\mathcal{P}_i}} \leq x_{i,t}^{j,k} - \frac{\boldsymbol{realmin}}{\boldsymbol{\mathcal{P}_i}}, \quad \forall i, j, k, t. \\ t_b = D_i^{j,k} + \lfloor \frac{\boldsymbol{t}}{\boldsymbol{\mathcal{P}_i}} \rfloor \cdot \boldsymbol{\mathcal{P}_i} \end{array}}$

7 $y_{i,t}^j = \sum_{k=0}^{N_i-1} y_{i,t}^{j,k}, \quad \forall i, j, t.$

8 $y_{i,t} \geq y_{i,t}^j, \quad \forall i, j.$

9 $\sum_{i=0}^{n-1} y_{i,t} \leq \mathcal{L} \cdot \boldsymbol{t} \quad \forall t.$

10 and: $D_i^k, P_i^k, y_{i,t}^{j,k}, y_{i,t}^j, y_{i,t}, \mathcal{L} \in \mathbb{R}^*, \boxed{x_{i,t}^{j,k} \in \{0, 1\}}.$

Figure 13: This figure shows the MILP algorithm. In the concave programming and LP-based algorithms (shown in Chapters 6.3 and 6.4), we only change the frame demand in Line 6 and remove all integer variables $x_{i,t}^{j,k}$.

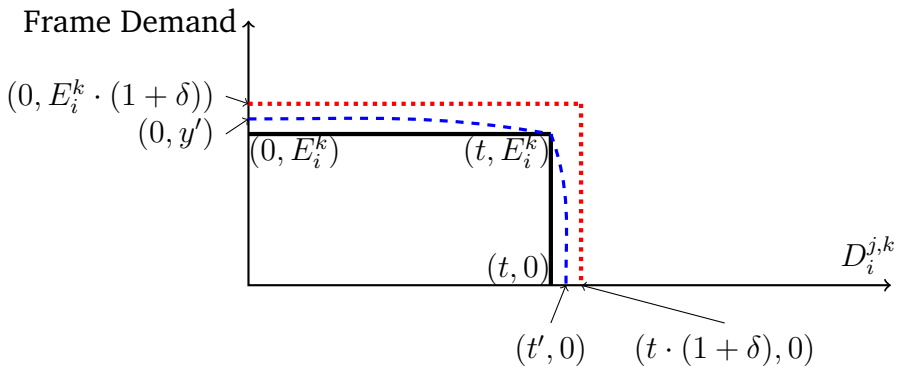


Figure 14: This example shows the frame demand within interval length $t < \mathcal{P}_i$. The blue dashed curve is a concave function and the staircase function in black solid line represents the exact frame demand in the MILP. The red dotted staircase line with error rates δ on both axes represents an upper bound on the concave function.

t-length interval in which the starting frame is the j'th frame. We define the system-wide maximum error rate[13] δ. The rate δ must be larger than zero to ensure the demand of any approximation function be larger than the staircase function for any given deadline. We set δ as a designer-defined constant in the system, and set the constant $\mu = \frac{1}{\delta} \cdot \ln\left(1 + \frac{1}{\delta}\right)$ as shown in Lemma 3. In Lemma 3, we prove that the maximum error rate of the concave function is smaller than the system maximum error rate δ, and the concave function approaches the staircase function when δ decreases.

Lemma 3. *The demand of the concave function in Equation 6.2 over-approximates the one in the MILP algorithm, and the error rate of the concave function is smaller than the system error constant δ when we set μ in Equation 6.2 as follows,*

$$\mu = \frac{1}{\delta} \cdot \ln\left(1 + \frac{1}{\delta}\right). \tag{6.3}$$

[13]The error rate (with respect to the exact frame demand function) of an approximation function is its percentage increase in the y-axis direction for $t \leq D_i^{j,k}$ or its percentage increase in the x-axis dimension if $t > D_i^{j,k}$. The maximum error rate is the largest error rate over all $t > 0$. E.g., the error rate on the x-axis of the point $(t \cdot (1+\delta), 0)$ in Figure 14 is δ. The maximum error rate of any approximation function must be smaller than the system-wide maximum error rate δ.

Proof. Let δ_y and δ_d be the worst-case error rates on the demand (on y-axis) and deadline (on x-axis) directions of concave functions, respectively. Let $t_b = D_i^{j,k} + \lfloor \frac{t}{\mathcal{P}_i} \rfloor \cdot \mathcal{P}_i - t$. The worst rates happen when, in Figure 14 for example, $E_i^k \cdot (1 + \delta_y) = y'$ and $t \cdot (1 + \delta_d) = t'$. We will prove that $\delta \geq \delta_y$ and $\delta \geq \delta_d$.

When $0 \leq t_b \leq t$, the largest demand of the concave function happens at $t_b = 0$. By substituting $E_i^k \cdot (1 + \delta_y)$ (respectively, 0) for $y_{i,t}^{j,k}$ (respectively, t_b), the concave function becomes $E_i^k \cdot (1+\delta_y) = E_i^k \cdot (1+\delta) - E_i^k \cdot \delta \cdot e^{\mu \cdot (0-t)}$. After simplification, we get $\delta_y = \delta - \delta \cdot e^{\mu \cdot (-t)}$. Thus, $\delta \geq \delta_y$ and δ is an upper bound of δ_y. Since the concave function is a decreasing function and it passes the points $(0, E_i^k \cdot (1 + \delta_y))$ and (t, E_i^k), the concave function over-approximates the corresponding demand in MILP when $0 \leq t_b \leq t$.

When $t_b > t$, the maximum error on the deadline direction happens at $t_b = t \cdot (1 + \delta_d)$. By substituting 0 (respectively, $t \cdot (1 + \delta_d)$) for $y_{i,t}^{j,k}$ (respectively, t_b), we have $0 = E_i^k \cdot (1 + \delta) - E_i^k \cdot \delta \cdot e^{\mu \cdot (t \cdot (1+\delta_d) - t)}$. After simplification, we have $\delta_d = \frac{1}{t \cdot \mu} \cdot \ln(1 + \frac{1}{\delta})$. We set $\mu = \frac{1}{\delta} \cdot \ln(1 + \frac{1}{\delta})$, and $\delta_d = \frac{\delta}{t}$ after replacing μ in δ_d. Since $t \geq 1$, $\delta \geq \delta_d$.

□

6.3.2 Speed-Up Factor Analysis

A speedup factor is a value that quantifies the quality of an approximation algorithm with respect to the optimal scheduling algorithm. A speedup factor $\mathcal{S} > 1$ [9] means that an approximation algorithm can schedule a task system at a speed-$\mathcal{S}$ processor if an optimal algorithm can schedule the system at a speed-one processor.

Let $\mathcal{L}_{MILP}$ be the value of the objective function returned by the MILP algorithm and $\mathcal{L}_{concave}$ be the value returned by the concave programming algorithm. We will prove

that $\mathcal{L}_{MILP} < \mathcal{L}_{concave} < \mathcal{L}_{MILP} \cdot (1+\delta)^2$. $\mathcal{L}_{MILP} < \mathcal{L}_{concave}$ indicates that a task system will be deemed schedulable by the MILP algorithm if the system is schedulable by the concave programming algorithm (which means $\mathcal{L}_{MILP} < \mathcal{L}_{concave} \leq 1$). By the definition of the speed-up factor, $\mathcal{L}_{concave} < \mathcal{L}_{MILP} \cdot (1+\delta)^2$ indicates that the speed-up factor of our concave programming algorithm is $(1+\delta)^2$ with respect to the MILP algorithm. In other words, $\mathcal{L}_{concave}/(1+\delta)^2 < \mathcal{L}_{MILP}$ indicates a task system can be scheduled by the concave programming algorithm under a $(1+\delta)^2$-speed processor if the system can be scheduled by the MILP algorithm under the corresponding one-speed processor.

We prove $\mathcal{L}_{MILP} < \mathcal{L}_{concave}$ in Lemma 4, and $\mathcal{L}_{concave} < \mathcal{L}_{MILP} \cdot (1+\delta)^2$ from Lemma 5 to Lemma 8. By Lemmas 4 and 8, we prove that the speed-up factor of our concave programming algorithm is $(1+\delta)^2$ with respect to the MILP algorithm in Theorem 4.

Lemma 4. *Let $\mathcal{L}_{MILP}$ and $\mathcal{L}_{concave}$ be the values returned by the MILP and concave programming algorithms (assume they exist), respectively. We have:*

$$\mathcal{L}_{MILP} < \mathcal{L}_{concave}. \tag{6.4}$$

Proof. Let $\mathcal{L}'_{MILP}$ be the value calculated as follows. Assume there exists such a solver that can solve the concave programming algorithm and return $\mathcal{L}_{concave}$, frame deadlines and separations. We assign the returned frame parameters from the concave programming algorithm to the formulation of the MILP algorithm and get the value of $\mathcal{L}'_{MILP}$.

Under the same values of frame parameters, any frame demand of concave programming algorithm is larger than its corresponding demand of the MILP algorithm, as shown in Lemma 3. The task demands of concave programming algorithm with the preassigned

frame parameters are thus also larger than the ones from the MILP approach. When we summarize task demands over any interval length, $\mathcal{L}'_{MILP}$ is thus always less than $\mathcal{L}_{concave}$. Since $\mathcal{L}'_{MILP}$ is calculated by preassigned frame parameters, $\mathcal{L}'_{MILP}$ must not be smaller than $\mathcal{L}_{MILP}$. If the frame parameters returned by the MILP and concave programming algorithms are all identical, $\mathcal{L}'_{MILP} = \mathcal{L}_{MILP}$. In all, $\mathcal{L}_{MILP} \leq \mathcal{L}'_{MILP} < \mathcal{L}_{concave}$ and this lemma is proved. □

In order to prove $\mathcal{L}_{concave} < \mathcal{L}_{MILP} \cdot (1+\delta)^2$, we first define $\mathcal{L}'_{concave}$. Let the MILP algorithm return $\mathcal{L}_{MILP}$, frame deadlines and separations. If we fix the deadline and separation variables of the concave programming formulation to be the values returned by the MILP, we calculate the value of $\mathcal{L}'_{concave}$. We will prove $\mathcal{L}_{concave} \leq \mathcal{L}'_{concave} < \mathcal{L}_{MILP} \cdot (1+\delta)^2$. $\mathcal{L}_{concave} \leq \mathcal{L}'_{concave}$ is proved in Lemma 5. Based on the demand bound functions defined in Equations 6.8 and 6.9, we prove $\mathcal{L}'_{concave} < \mathcal{L}_{MILP} \cdot (1+\delta)^2$ in Lemma 8.

Lemma 5. *Let $\mathcal{L}_{concave}$ be the optimal value returned by the concave programming algorithm, and $\mathcal{L}'_{concave}$ be the value calculated by the concave programming algorithm using the frame parameters returned by the MILP. We have:*

$$\mathcal{L}_{concave} \leq \mathcal{L}'_{concave}. \tag{6.5}$$

Proof. Since the concave programming algorithm minimizes $\mathcal{L}_{concave}$, $\mathcal{L}_{concave}$ must be the smallest value over all feasible-assigned/preassigned frame parameters, and $\mathcal{L}_{concave} < \mathcal{L}'_{concave}$. If frame parameters returned by the MILP and concave programming algorithms are same, $\mathcal{L}_{concave} = \mathcal{L}'_{concave}$. In all, $\mathcal{L}_{concave} \leq \mathcal{L}'_{concave}$. □

For ease of proof, we consider a staircase approximation function $\text{dbf}_i^a(t, D_i^{j,k})$ illustrated

by the red dotted line in Figure 14 for task τ_i over the t-length interval, and the solid line shows an example of the staircase demand $\mathsf{dbf}_i(t, D_i^{j,k})$.

Equation 6.6 shows $\mathsf{dbf}_i(t, D_i^{j,k})$ as the k'th frame-demand function of task τ_i over the t-length interval that starts with the j'th frame. The corresponding task demand $\mathsf{dbf}_i(t, \vec{F}_i)$ is shown in Equation 6.8, and the reasoning is same as to the relationship between $y_{i,t}$ and $y_{i,t}^{j,k}$ in the MILP algorithm. I.e., we take the maximum demand over all sequences as the task demand. The approximate frame-demand $\mathsf{dbf}_i^a(t, D_i^{j,k})$ and task-demand $\mathsf{dbf}_i^a(t, \vec{F}_i)$ (for $\mathsf{dbf}_i(t, D_i^{j,k})$ and $\mathsf{dbf}_i(t, \vec{F}_i)$, respectively) are defined in Equations 6.7 and 6.9, respectively. We prove that the approximation demand over-approximates the concave demand in Lemma 6.

$$\mathsf{dbf}_i(t, D_i^{j,k}) = \begin{cases} 0, & 0 \le t < D_i^{j,k} \\ E_i^k, & D_i^{j,k} \le t \le \mathcal{P}_i \\ E_i^k \cdot \lfloor \frac{t}{\mathcal{P}_i} \rfloor + \mathsf{dbf}_i(t - \mathcal{P}_i \cdot \lfloor \frac{t}{\mathcal{P}_i} \rfloor, D_i^{j,k}), & t > \mathcal{P}_i \end{cases} \tag{6.6}$$

$$\mathsf{dbf}_i^a(t, D_i^{j,k}) = \begin{cases} 0, & 0 \le t < \frac{D_i^{j,k}}{(1+\delta)} \\ (1+\delta) \cdot E_i^k, & \frac{D_i^{j,k}}{(1+\delta)} \le t \le \mathcal{P}_i \\ E_i^k \cdot \lfloor \frac{t}{\mathcal{P}_i} \rfloor + \mathsf{dbf}_i^a(t - \mathcal{P}_i \cdot \lfloor \frac{t}{\mathcal{P}_i} \rfloor, D_i^{j,k}), & t > \mathcal{P}_i \end{cases} \tag{6.7}$$

$$\mathsf{dbf}_i(t, \vec{F}_i) = \max_{j=0}^{N_i-1} \{ \sum_{k=0}^{N_i-1} \mathsf{dbf}_i(t, D_i^{j,k}) \} \tag{6.8}$$

$$\mathsf{dbf}_i^a(t, \vec{F}_i) = \max_{j=0}^{N_i-1} \{ \sum_{k=0}^{N_i-1} \mathsf{dbf}_i^a(t, D_i^{j,k}) \} \tag{6.9}$$

Lemma 6. *The demand of task τ_i over any interval length t in Equation 6.9 is an upper bound of its corresponding concave approximation demand.*

Proof. In Lemma 3, we proved that $\delta_d \leq \delta$. Let $t_\Delta = t - \mathcal{P}_i \cdot \lfloor \frac{t}{\mathcal{P}_i} \rfloor$. From Equation 6.2 and the definition of δ_d, the concave demand with any value assigned for $D_i^{j,k} \in [0, t_\Delta \cdot (1+\delta_d)]$ is smaller than $E_i^k \cdot (1+\delta)$, and the demand is zero when $D_i^{j,k} > t_\Delta \cdot (1+\delta_d)$. Since $\mathsf{dbf}_i^a(t, D_i^{j,k}) = E_i^k \cdot (1+\delta)$ when $D_i^{j,k} \leq t_\Delta \cdot (1+\delta)$ and $\delta_d \leq \delta$, the demand function $\mathsf{dbf}_i^a(t, D_i^{j,k})$ over approximates the concave demand. For task-wise demand $\mathsf{dbf}_i^a(t, \vec{F}_i)$, we take the summation of all frame demand $\mathsf{dbf}_i^a(t, D_i^{j,k})$ of task τ_i over all sequences (sequences differ from the starting j'th frame in the t-length interval), and take the maximum demand over all sequences as the task demand. The task demand $\mathsf{dbf}_i^a(t, \vec{F}_i)$ also over approximates the corresponding concave demand. In all, we have proved this lemma.

□

With the demand bound functions shown in Equations 6.8-6.9, we prove $\mathcal{L}'_{concave} < \mathcal{L}_{MILP} \cdot (1+\delta)^2$ in Lemmas 7-8.

Lemma 7. *For the task τ_i's demand $\mathsf{dbf}_i(t, \vec{F}_i)$ and its approximation demand $\mathsf{dbf}_i^a(t, \vec{F}_i)$ in the t-length time interval, we have: $\mathsf{dbf}_i((1+\delta) \cdot t, \vec{F}_i) \cdot (1+\delta) \geq \mathsf{dbf}_i^a(t, \vec{F}_i)$.*

Proof. We first prove $\mathsf{dbf}_i((1+\delta) \cdot t, D_i^{j,k}) \cdot (1+\delta) \geq \mathsf{dbf}_i^a(t, D_i^{j,k})$, and $\mathsf{dbf}_i((1+\delta) \cdot t, \vec{F}_i) \cdot (1+\delta) \geq \mathsf{dbf}_i^a(t, \vec{F}_i)$ can be extended by Equations 6.8 and 6.9. We classify all interval lengths t in three sets:

$T_1 : 0 \leq t < D_i^{j,k}/(1+\delta)$,

$T_2 : D_i^{j,k}/(1+\delta) \leq t \leq \mathcal{P}_i$,

T_3 : otherwise.

When $t \in T_1$, $\mathsf{dbf}_i(t, D_i^{j,k}) = \mathsf{dbf}_i^a(t, D_i^{j,k}) = 0$. Since demand bound functions are monotonically increasing functions, $\mathsf{dbf}_i((1+\delta)\cdot t, D_i^{j,k})\cdot(1+\delta) \geq \mathsf{dbf}_i(t, D_i^{j,k}) = \mathsf{dbf}_i^a(t, D_i^{j,k})$.

When $t \in T_2$, we know that $\mathsf{dbf}_i(t^*, D_i^{j,k}) = E_i^k$ at $D_i^{j,k} \leq t^* \leq \mathcal{P}_i$ from Equation 6.6. Let $t^* = t\cdot(1+\delta)$, we have $\mathsf{dbf}_i(t\cdot(1+\delta), D_i^{j,k}) = E_i^k$ at $D_i^{j,k}/(1+\delta) \leq t \leq \mathcal{P}_i$. From Equations 6.6 and 6.7, we know that $\mathsf{dbf}_i(t\cdot(1+\delta), D_i^{j,k})\cdot(1+\delta) = \mathsf{dbf}_i^a(t, D_i^{j,k})$ at $D_i^{j,k}/(1+\delta) \leq t \leq \mathcal{P}_i$.

When $t \in T_3$, it is trivial to see the fact that $\mathsf{dbf}_i((1+\delta)\cdot t, D_i^{j,k})\cdot(1+\delta) \geq \mathsf{dbf}_i^a(t, D_i^{j,k})$ since the demand is iteratively calculated from the demand when $t \in T_1 \cup T_2$. □

Lemma 8. *Let $\mathcal{L}_{MILP}$ be the optimal value returned by the MILP algorithm, and $\mathcal{L}_{concave}^{'}$ be the value calculated by the frame parameters returned by the MILP. We have:*

$$\mathcal{L}_{concave}^{'} < \mathcal{L}_{MILP}\cdot(1+\delta)^2. \tag{6.10}$$

Proof. Line 9 of Figure 13 shows that $\mathcal{L}$ is the largest value of $\frac{\sum_{i=0}^{n-1} y_{i,t}}{t}$ for all values of t in the MILP algorithm (can be derived from Lemma 1). We also require this line in the concave programming algorithm. From Lemma 7, we know that $\mathsf{dbf}_i((1+\delta)\cdot t, \vec{F_i})\cdot(1+\delta) \geq \mathsf{dbf}_i^a(t, \vec{F_i})$ for any task τ_i over any t-length interval. Let $t = (1+\delta)\cdot t^*$, we have:

$$\begin{aligned}
\mathcal{L}_{MILP} &= \max_{t>0} \frac{\sum_{\tau_i\in\mathcal{T}} \mathsf{dbf}_i(t, \vec{F_i})}{t} \text{ By Lemma 1} \\
&= \frac{\sum_{\tau_i\in\mathcal{T}} \mathsf{dbf}_i((1+\delta)\cdot t^*, \vec{F_i})}{(1+\delta)\cdot t^*} \\
&= \frac{\sum_{\tau_i\in\mathcal{T}} \mathsf{dbf}_i((1+\delta)\cdot t^*, \vec{F_i})\cdot(1+\delta)}{(1+\delta)^2\cdot t^*} \\
&\geq \frac{\sum_{\tau_i\in\mathcal{T}} \mathsf{dbf}_i^a(t^*, \vec{F_i})}{(1+\delta)^2\cdot t^*} \text{ By Lemma 7} \\
&\geq \frac{\mathcal{L}_{concave}^{'}}{(1+\delta)^2} \text{ By Lemma 6}
\end{aligned} \tag{6.11}$$

□

Theorem 4. *When the concave programming algorithm returns integer frame deadlines and separation times, the speed-up factor of our concave programming algorithm with respect to the MILP algorithm is* $(1+\delta)^2$.

Proof. In Lemmas 4, 5, and 8, we have proved that $\mathcal{L}_{MILP} < \mathcal{L}_{concave} < \mathcal{L}_{MILP} \cdot (1+\delta)^2$. $\mathcal{L}_{MILP} < \mathcal{L}_{concave}$ indicates that a task system is deemed schedulable (with integer frame parameters) by the MILP if the task system is deemed schedulable (with integer parameters) by the concave programming algorithm. $\mathcal{L}_{MILP} < \mathcal{L}_{concave}$ shows our concave programming algorithm is an approximation algorithm for the MILP.

We divide $(1+\delta)^2$ on both sides of the inequality $\mathcal{L}_{concave} < \mathcal{L}_{MILP} \cdot (1+\delta)^2$ to get $\mathcal{L}_{concave}/(1+\delta)^2 < \mathcal{L}_{MILP}$. $\mathcal{L}_{concave}/(1+\delta)^2$ represents that we change the speed of the processor from one to $(1+\delta)^2$. Thus, a task system must be scheduled by the concave programming algorithm with a $(1+\delta)^2$-speed processor if the task system is scheduled by the MILP on a single speed processor. From the definition of the speed-up factor, we have proved that the speed-up factor of our concave programming algorithm with respect to the MILP is $(1+\delta)^2$. □

6.4 The Linear Programming-Based Heuristic Algorithm and its Application to One-Suspension Self-Suspending Tasks

Until now, we have constructed the concave programming approximation algorithm for the MILP-based algorithm. Due to the difficulties in solving concave programming (or non-convex optimization) problems in general, we use a heuristic LP-based scheme to efficiently select frame parameters of GMF-PA tasks, and apply it to self-suspending tasks.

The LP-based Algorithm for GMF-PA tasks

Initialize D as $D_i^k \leftarrow (E_i^k/E_i) \cdot \mathcal{P}_i$, $\mathcal{L}^{\text{last}} \leftarrow \infty$, and $\mathcal{L}^{\text{cur}} \leftarrow \infty$
repeat
    $\mathcal{L}^{\text{last}} \leftarrow \mathcal{L}^{\text{cur}}$
    $S \leftarrow$ *computeSlope(D)*
    $[D, \mathcal{L}^{\text{cur}}] \leftarrow Heuristic\text{-}LP(D, S)$
  until $\mathcal{L}^{\text{last}} - \mathcal{L}^{\text{cur}} < \epsilon$
Process frame deadlines D to integers.
$[\mathcal{L}^{\text{cur}}] \leftarrow Heuristic\text{-}LP\text{-}fixedDeadline(D, S)$
if $\mathcal{L}^{\text{cur}} \leq 1$
  then return $schedulable$
else return $unschedulable$

Figure 15: The LP-based algorithm for GMF-PA tasks.

For ease of presentation, we let $\underline{D}_i^k = E_i^k$, $\overline{D}_i^k = \mathcal{P}_i$, and $P_i^k = D_i^k$. In this case, frames deadlines are constrained by frame execution time and the l-MAD property. We present the LP-based heuristic algorithm in Chapter 6.4.1, and further to optimize the LP-based algorithm to schedule one-suspension self-suspending tasks in Chapter 6.4.2.

6.4.1 The Linear Programming-Based Heuristic Algorithm

The general routine of the LP-based scheme for GMF-PA tasks is: 1) We initialize frame parameters of GMF-PA tasks. 2) Given the frame parameters, we recalculate a set of linear functions, which approximate the staircase functions for frame demands in the MILP, guided by the concave programming algorithm. 3) We run the LP algorithm (shown later) based on the assigned linear functions, and receive frame parameters as outputs. If the difference in $\mathcal{L}$ values between the current and the last iterations is no smaller than some threshold, the program goes back to Step 2. 4) We round frame parameters to integers and run the LP algorithm with the fixed integer-valued parameters to get the final assignment.

computeSlope(D)

1 Calculate all $D_i^{j,k'}$ from D
2 $t' \leftarrow t - \lfloor \frac{t}{\mathcal{P}_i} \rfloor \cdot \mathcal{P}_i$
3 $y_{i,t'}^{j,k'} \leftarrow E_i^k \cdot (1+\delta) - E_i^k \cdot \delta \cdot e^{\mu \cdot (D_i^{j,k'} - t')}$
4 **if** $D_i^{j,k'} > t'$
5 **then** $s_{i,t}^{j,k} \leftarrow (0 - E_i^k)/(\frac{1}{\mu} \cdot \ln(1+\frac{1}{\delta}) + t' - t')$
6 **elseif** $D_i^{j,k'} == t'$
7 **then** $s_{i,t}^{j,k} \leftarrow \frac{\partial}{\partial D_i^{j,k}} \left[\mathsf{dbf}_i^{\text{concave}}(t', D_i^{j,k}) \right]$
8 **else** $s_{i,t}^{j,k} \leftarrow (y_{i,t'}^{j,k'} - E_i^k)/(D_i^{j,k'} - t')$
9 **return** S
10 ▷ S is the matrix that stores all slopes $s_{i,t}^{j,k}$.

Figure 16: This algorithm calculates all slopes given all frame deadlines.

$$\mathsf{dbf}_i^{\text{linear}}(t, D_i^{j,k}) = \max\{0, s_{i,t}^{j,k} \cdot (D_i^{j,k} - t') + E_i^k\} + \lfloor \frac{t}{\mathcal{P}_i} \rfloor \cdot E_i^k, \;\; t' = t - \lfloor \frac{t}{\mathcal{P}_i} \rfloor \cdot \mathcal{P}_i \tag{6.12}$$

In *The LP-based Algorithm for GMF-PA Tasks* (Figure 15), we initialize frame deadlines by proportional deadline assignment (PDA [47]) to $D_i^k = (E_i^k / E_i) \cdot \mathcal{P}_i$. Given the deadline matrix D which stores all D_i^k, we calculate all slopes and store them in matrix S. We replace Line 6 of Figure 13 with Equation 6.12 to transform the algorithm into a LP algorithm $Heuristic\text{-}LP(D, S)$ (Line 5 of Figure 15). The slope element $s_{i,t}^{j,k}$ of S, which corresponds to $y_{i,t}^{j,k}$, is calculated in the algorithm shown in Figure 16 and all lines pass the point (t', E_i^k). The linear functions are illustrated by the red lines in Figures 17-19. If the deadline $D_i^{j,k'}$ (generated from the previous iteration) is smaller than $t' = t - \lfloor \frac{t}{\mathcal{P}_i} \rfloor \cdot \mathcal{P}_i$, we calculate the demand $y_{i,t'}^{j,k'}$ of the concave function at $D_i^{j,k'}$. The slope of the line is calculated by two points $(D_i^{j,k'}, y_{i,t'}^{j,k'})$ and (t', E_i^k) illustrated in Figure 17. If the deadline $D_i^{j,k'}$ equals t', we calculate the slope by taking the tangent of the concave function at the

point (t', E_i^k) shown in Figure 18. If the deadline is larger than t', we use two points (t', E_i^k) and $\left(\frac{1}{\mu} \cdot \ln(1 + \frac{1}{\delta}) + t', 0\right)$, which is the cross point of the x-axis and the concave function, to calculate the slope, and the line with the slope is shown in Figure 19. The slope matrix S is adjusted in each iteration of the loop in Figure 15.

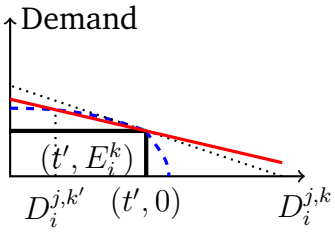


Figure 17: The frame deadline $D_i^{j,k'}$ of the last iteration is smaller than t' in this case.

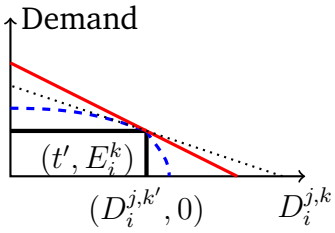


Figure 18: The frame deadline $D_i^{j,k'}$ of the last iteration equals t' in this case.

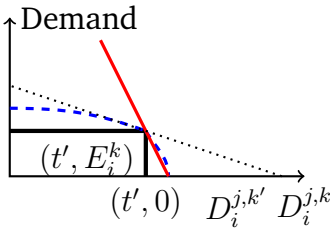


Figure 19: The frame deadline $D_i^{j,k'}$ of the last iteration is larger than t' in this case.

The loop in Figure 15 will not stop recursively calling function $Heuristic\text{-}LP(D, S)$ until the difference of the $\mathcal{L}$ values in two consecutive iterations is smaller than the positive threshold ϵ. $\mathcal{L}^{\text{last}}$ and $\mathcal{L}^{\text{cur}}$ represent the $\mathcal{L}$ values of the last and current iterations, respectively. The $Heuristic\text{-}LP\text{-}fixedDeadline(D, S)$ algorithm (Line 8 of Figure 15) uses the integer deadlines to maintain sufficiency for schedulability, which is proved in Theorem 6. We first round up frame deadlines to be integers. For each task, we keep reducing the largest frame deadline by one until the summation of them equals to its task deadline/period. We assign the deadline variables to the integer values in Line 7 of Figure 15 and the other parts are the same as in the $Heuristic\text{-}LP(D, S)$ algorithm. The system is schedulable if $\mathcal{L} \leq 1$.

We prove in Theorem 5 that the `while` loop of the algorithm *The LP-based Algorithm for GMF-PA Tasks* function stops after a finite number of iterations. The sufficiency of the LP-based algorithm for schedulability is proved in Theorem 6.

Theorem 5. *The `while` loop of the function* The LP-based Algorithm for GMF-PA Tasks *stops*

in a finite number of iterations.

Proof. We first prove that $\mathcal{L}$ decreases from one iteration to the next. Before each iteration of the algorithm $Heuristic\text{-}LP(D, S)$, we use the deadline assignment D' from the last iteration to calculate the slopes S of frame functions in the current iteration. Let $\mathcal{L}^{\text{last}}$ be the value of $\mathcal{L}$ in the last iteration. In the current iteration, let us assume that we use the same set of the deadlines D' to calculated the value $\mathcal{L}^{\text{cur}}$.

In the first and third cases shown in Figures 17 and 19, the frame demand is either smaller (if the last iteration is the first iteration) or equal to the one in the last iteration. In the second case, the frame demand is the same as the one in the last iteration. From all cases, we know that the same set of deadlines causes $\mathcal{L}^{\text{cur}} \leq \mathcal{L}^{\text{last}}$. Since we minimize $\mathcal{L}$ in the algorithm, the returned deadlines by the algorithm $Heuristic\text{-}LP(D, S)$ must generate a value of $\mathcal{L}$ that is smaller than $\mathcal{L}^{\text{cur}}$. Thus, we have proved that $\mathcal{L}$ decreases from one iteration to the next. We also set a threshold to be the difference of the $\mathcal{L}$ values in two consecutive iterations, and we know that the lower bound of $\mathcal{L}$ equals $\sum_{i=1}^{n} U_i$. In either cases, the loop of the function *The LP-based Algorithm for GMF-PA Tasks* stops in a finite number of iterations. □

Theorem 6. *The LP-based algorithm is a sufficient schedulability test when* $\mathcal{L} \leq 1$.

Proof. This proof is similar to Theorem 2. The sufficiency of any approximation/heuristic algorithm (w.r.t. the MILP algorithm) for schedulability requires two conditions: 1) the demand of the algorithm over any t-length interval is larger than the one in the MILP. 2) frame parameters must take integer values. The first condition ensures that the demand over approximates on any t, and the second condition ensures that the demand only changes

at integer values. We require the second condition since all lengths (represented by t) can only be integers in the MILP algorithm. The LP-based algorithm over approximates system demand among all t, and the algorithm adjusts frame deadlines to be integers in the last iteration. □

6.4.2 The Application of the LP-Based Algorithm to One-Suspension Self-Suspending Tasks

The LP-based scheme can be applied to multiple-segment self-suspending tasks directly. In this chapter, we further optimize the algorithm for one-suspension self-suspending tasks by reducing the number of free variables and equations. Given that n is the number of tasks and H is the maximum interval length, the algorithm uses $8 \cdot n \cdot H + n$ fewer variables and $15 \cdot n \cdot H + n$ fewer number of constraints than the ones in the standard LP-based scheme.

For each task τ_i, we use variables D_i^1 and $\mathcal{P}_i - S_i - D_i^1$ (instead of D_i^1 and D_i^2) to denote frame deadlines to reduce the number of variables and constraints. S_i is the suspension length of task τ_i. In this case, the demand bound function only relies on D_i^1 and t and $\vec{F_i} = [D_i^1, D_i^1, \mathcal{P}_i - S_i - D_i^1, \mathcal{P}_i - S_i - D_i^1]$ since we let $P_i^k = D_i^k$. A task demand falls in four cases which are shown and proved in Theorem 7.

Theorem 7. *The demand bound function of a task τ_i lies in one of the following four cases:*

$$
\mathsf{dbf}_i(t,\vec{F}_i) = \begin{cases}
\mathsf{dbf}_i^1(t,\vec{F}_i) & = \begin{cases} E_i^1, & 0 < D_i^1 \le t \\ 0, & t < D_i^1 < \mathcal{P}_i - S_i - t \\ E_i^2, & \mathcal{P}_i - S_i - t < D_i^1 \le \mathcal{P}_i - S_i \end{cases} \\
& \textit{when } 0 < t < (\mathcal{P}_i - S_i)/2 \\
\mathsf{dbf}_i^2(t,\vec{F}_i) & = \begin{cases} E_i^1, & 0 < D_i^1 < \mathcal{P}_i - S_i - t \\ \max\{E_i^1, E_i^2\}, & \mathcal{P}_i - S_i - t \le D_i^1 \le t \\ E_i^2, & t < D_i^1 < \mathcal{P}_i - S_i \end{cases} \\
& \textit{when } (\mathcal{P}_i - S_i)/2 \le t < \mathcal{P}_i - S_i \\
\mathsf{dbf}_i^3(t,\vec{F}_i) & = E_i^1 + E_i^2, \\
& \textit{when } \mathcal{P}_i - S_i \le t \le \mathcal{P}_i \\
\mathsf{dbf}_i^4(t,\vec{F}_i) & = \lfloor \tfrac{t}{\mathcal{P}_i} \rfloor \cdot (E_i^1 + E_i^2) + \mathsf{dbf}_i(t - \lfloor \tfrac{t}{\mathcal{P}_i} \rfloor \cdot \mathcal{P}_i, D_i^1), \\
& \textit{when } t > \mathcal{P}_i
\end{cases}
\tag{6.13}
$$

Proof. Figures 20-21 show an example of the staircase demand of $\mathsf{dbf}_i^1(t,\vec{F}_i)$ and $\mathsf{dbf}_i^2(t,\vec{F}_i)$ with black solid lines, respectively. Roughly, the two staircase/concave demand curves head toward each other when t increases. The first two cases differ when the two staircase functions meet as t increases. The demand $\mathsf{dbf}_i^3(t,\vec{F}_i)$ considers the total task demand and $\mathsf{dbf}_i^4(t,\vec{F}_i)$ iterates over the first three cases.

For the demand $\mathsf{dbf}_i^1(t,\vec{F}_i)$ in the first case, when $0 < t < (\mathcal{P}_i - S_i)/2$, we know that

$t < \mathcal{P}_i - S_i - t$ by simple mathematical transformation. In this case, we have two separate staircase functions as shown in Figure 20. When $D_i^1 \leq t$, the demand of the first frame is E_i^1, the demand of the second frame is zero because $D_i^1 \leq t < \mathcal{P}_i - S_i - t$. $D_i^1 < \mathcal{P}_i - S_i - t$ means $t < \mathcal{P}_i - S_i - D_i^1$ which indicates the deadline of the second frame is larger than t. Thus, $\mathsf{dbf}_i^1(t, \vec{F}_i) = E_i^1$ when $D_i^1 \leq t$. When $t < D_i^1 < \mathcal{P}_i - S_i - t$, $\mathsf{dbf}_i^1(t, \vec{F}_i) = 0$ because $t < D_i^1$ and $t < \mathcal{P}_i - S_i - D_i^1$. When $D_i^1 \geq \mathcal{P}_i - S_i - t$, i.e., $t \geq \mathcal{P}_i - S_i - D_i^1$, the demand $\mathsf{dbf}_i^1(t, \vec{F}_i)$ equals E_i^2. Thus, we have proved that the demand of task τ_i is this case when $0 < t < (\mathcal{P}_i - S_i)/2$.

For the demand $\mathsf{dbf}_i^2(t, \vec{F}_i)$, the proof is similar to the one of the demand $\mathsf{dbf}_i^1(t, \vec{F}_i)$. We know $\mathcal{P}_i - S_i - t \leq t$ since $(\mathcal{P}_i - S_i)/2 \leq t$. By comparing the deadline and length t, $\mathsf{dbf}_i^2(t, \vec{F}_i) = E_i^1$ when $0 < D_i^1 < \mathcal{P}_i - S_i - t$ and $\mathsf{dbf}_i^2(t, \vec{F}_i) = E_i^2$ when $t < D_i^1 < \mathcal{P}_i - S_i$. When $\mathcal{P}_i - S_i \leq t \leq \mathcal{P}_i$, we know that either frame can contribute to the demand. However, the two frames cannot contribute together since $t < \mathcal{P}_i - S_i$. In other words, the interval length t cannot fit both frames. Thus, we take the maximum execution of the two frames as the demand when $\mathcal{P}_i - S_i \leq t \leq \mathcal{P}_i$.

It is trivial to see that $\mathsf{dbf}_i^3(t, \vec{F}_i) = E_i^1 + E_i^2$ when $\mathcal{P}_i - S_i \leq t \leq \mathcal{P}_i$, and the fourth case iterates over the first three cases. In all, we have proved this theorem. □

The LP-based algorithm for one-suspension tasks is based on approximating the exact demand in Theorem 7 and the algorithm *The LP-based Algorithm for GMF-PA Tasks* in Figure 15. We replace Lines 6-8 in the MILP algorithm with the linear functions shown in Equation 6.16 to get the LP algorithm $Heuristic\text{-}LP(D, S)$ in Line 5 of Figure 15. The linear functions shown in Equations 6.14-6.15 are to approximate the two concave portions

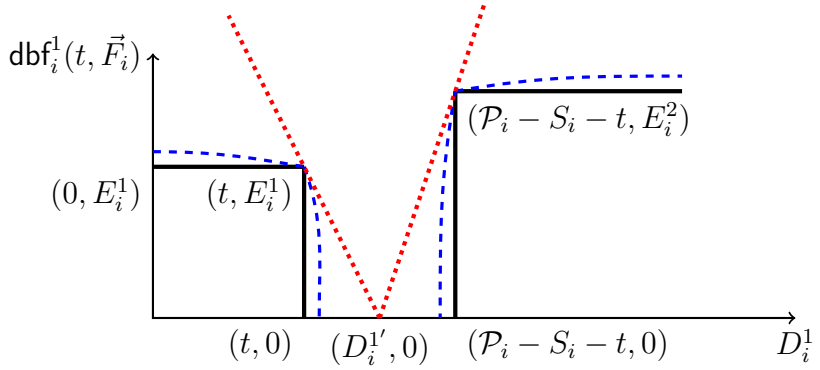


Figure 20: The black solid line shows the demand $\mathsf{dbf}_i^1(t, \vec{F}_i)$, the blue dashed line shows its concave approximation, and the red dotted line shows its linear function when the deadline $D_i^{1'}$ of the last iteration lies between $(t, 0)$ and $(\mathcal{P}_i - S_i - t, 0)$.

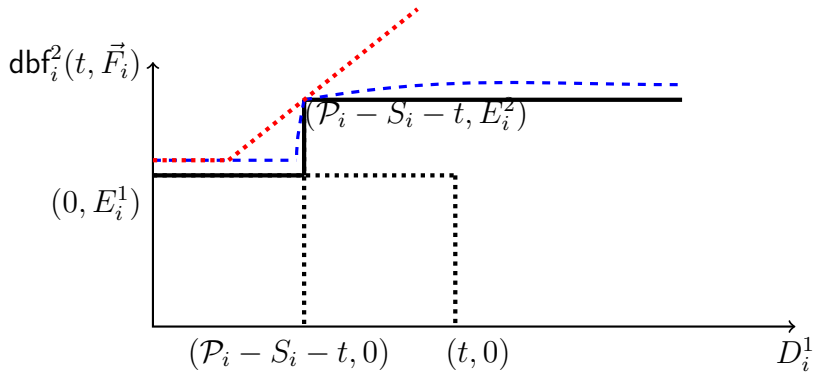


Figure 21: Similar to Figure 20, the dashed and dotted lines show the concave and linear functions of the demand $\mathsf{dbf}_i^2(t, \vec{F}_i)$, shown with the solid line, respectively. The black dotted line shows the frame-wise demand.

of the task demand for $\mathsf{dbf}_i^1(t, \vec{F}_i)$ and $\mathsf{dbf}_i^2(t, \vec{F}_i)$, respectively, illustrated by the red dotted lines in Figures 20-21.

$$\mathsf{dbf}_i^{1,\text{linear}}(t, \vec{F}_i) = \begin{cases} \max \begin{cases} \mathsf{dbf}_i^{\text{linear}}(t, D_i^1) \\ \mathsf{dbf}_i^{\text{linear}}(t, \mathcal{P}_i - S_i - D_i^1) \end{cases} \\ \quad \text{when } 0 < D_i^{1'} \leq t \\ \max \begin{cases} \frac{0-E_i^1}{D_i^{1'}-t'} \cdot (D_i^1 - t) + E_i^1 \\ \frac{0-E_i^2}{D_i^{1'}-(\mathcal{P}_i-S_i-t')} \cdot (D_i^1 - (\mathcal{P}_i - S_i - t)) + E_i^2 \\ 0 \end{cases} \\ \quad \text{when } t < D_i^{1'} < \mathcal{P}_i - S_i - t \\ \max \begin{cases} \mathsf{dbf}_i^{\text{linear}}(t, D_i^1) \\ \mathsf{dbf}_i^{\text{linear}}(t, \mathcal{P}_i - S_i - D_i^1) \end{cases} \\ \quad \text{when } \mathcal{P}_i - S_i - t \leq D_i^{1'} < \mathcal{P}_i - S_i \end{cases} \tag{6.14}$$

$$\mathsf{dbf}_i^{2,\text{linear}}(t, \vec{F}_i) = \begin{cases} \max \begin{cases} \mathsf{dbf}_i^{\text{linear}}(t, D_i^1) \\ E_i^2 \end{cases} \\ \quad \text{when } E_i^1 \geq E_i^2 \\ \max \begin{cases} \mathsf{dbf}_i^{\text{linear}}(t, \mathcal{P}_i - S_i - D_i^1) \\ E_i^1 \end{cases} \\ \quad \text{when } E_i^1 < E_i^2 \end{cases} \tag{6.15}$$

$$\mathsf{dbf}_i^{\text{linear}}(t,\vec{F}_i) = \begin{cases} \mathsf{dbf}_i^{1,\text{linear}}(t,\vec{F}_i) & \text{when } 0 < t < (\mathcal{P}_i - S_i)/2 \\ \mathsf{dbf}_i^{2,\text{linear}}(t,\vec{F}_i) & \text{when } (\mathcal{P}_i - S_i)/2 \le t < \mathcal{P}_i - S_i \\ \mathsf{dbf}_i^{3,\text{linear}}(t,\vec{F}_i) & = E_i^1 + E_i^2, \\ & \text{when } \mathcal{P}_i - S_i \le t \le \mathcal{P}_i \\ \mathsf{dbf}_i^{4,\text{linear}}(t,\vec{F}_i) & = \lfloor \frac{t}{\mathcal{P}_i} \rfloor \cdot (E_i^1 + E_i^2) + \mathsf{dbf}_i(t - \lfloor \frac{t}{\mathcal{P}_i} \rfloor \cdot \mathcal{P}_i, D_i^1), \\ & \text{when } t > \mathcal{P}_i \end{cases} \tag{6.16}$$

The approximation demand $\mathsf{dbf}_i^{\text{linear}}(t,\vec{F}_i)$ is calculated based on the t-length interval. Equation 6.14 shows that the task demand is approximated when $0 < t < (\mathcal{P}_i - S_i)/2$. This case is illustrated by the red dashed lines shown in Figure 20. The functions are also based on the LP-based iterative process and the initial deadline D_i^1 is assigned by PDA $(\mathcal{P}_i - S_i) \cdot \frac{E_i^1}{E_i^1 + E_i^2}$. The slope of the linear function depends on the frame deadline $D_i^{1'}$ from the last iteration. If the deadline $D_i^{1'}$ lies in the region $(t, \mathcal{P}_i - S_i - t)$, we use the two red dotted lines shown in Figure 20 to approximate the staircase demand. The first line passes the points (t, E_i^1) and $(D_i^{1'}, 0)$, and the second line passes the points $(D_i^{1'}, 0)$ and $(\mathcal{P}_i - S_i - t, E_i^2)$. When the frame deadline $D_i^{1'}$ lies in the region $(0, t]$ or $[\mathcal{P}_i - S_i - t, \mathcal{P}_i - S_i)$, we reuse the linear function $\mathsf{dbf}_i^{\text{linear}}(t, D_i^1)$ shown in Equation 6.12 to calculate the slopes.

Equation 6.15 shows the task demand when $(\mathcal{P}_i - S_i)/2 \le t < \mathcal{P}_i - S_i$, the demand functions differ by the values of E_i^1 and E_i^2. In the case of the demand $\mathsf{dbf}_i^2(t,\vec{F}_i)$, the first line equals $\min\{E_i^1, E_i^2\}$, and the second line uses the previous method *computeSlope(D)* to adjust the slope of the linear function as shown in Figure 21. Figure 21 shows the approximate lines when $E_i^1 < E_i^2$, and the case is similar when $E_i^1 \ge E_i^2$. When $t \ge \mathcal{P}_i - S_i$,

the demand $\mathsf{dbf}_i^{3,\text{linear}}(t, \vec{F}_i)$ and $\mathsf{dbf}_i^{4,\text{linear}}(t, \vec{F}_i)$ are identical to $\mathsf{dbf}_i^{3}(t, \vec{F}_i)$ and $\mathsf{dbf}_i^{4}(t, \vec{F}_i)$ of Equation 6.13, respectively. Thus, we have created the LP-based algorithm for one-suspension tasks.

6.5 Experiments

We implement our LP-based algorithms using the commercial solver GUROBI [1] in MATLAB on a 2 GHz Intel Core i5 processor and 8 GB memory machine. We compare our LP-based algorithm with the MILP algorithm and its application to self-suspending tasks [23, 39] on uniprocessor systems. The algorithm LP-δ is the LP-based schedulability test given the maximum error δ of the concave programming algorithm. The algorithm n_{iter}-LP-δ limits the number of iterations to be n_{iter}. Note that we set $\delta = 0.1$, as the constant $\mu = \frac{1}{\delta} \cdot \ln\left(1 + \frac{1}{\delta}\right)$ (e.g. the exponential constants in Equation 6.2) will be out of range if δ is too small.

The MILP algorithm is introduced in Chapter 4. The algorithm EDA (equal deadline assignment [8, 23]) assigns each frame the same deadline ($D_i^k = (\mathcal{P}_i - \sum_{i=0}^{N_i-1} S_i^k)/N_i$), and the algorithm PDA [8, 47] assigns frame deadlines proportional to frame execution time ($D_i^k = (\mathcal{P}_i - \sum_{i=0}^{N_i-1} S_i^k) \cdot E_i^k/E_i$). Note that we use the schedulability test in the GMF model [8] with the EDA and PDA deadline assignment, since the upper bound of the maximum interval length is bounded [8]. The details of application from GMF-PA to self-suspending tasks is introduced in Chapter 4.5. Comparative results on tasks with one suspension and multiple suspensions are shown in Chapters 6.5.1 and 6.5.2, respectively.

6.5.1 The Experiments for One-Suspension Self-Suspending Tasks

For one-suspension self-suspending tasks, we compare schedulability ratio and total running time among the algorithms in Figures 22 and 23, respectively. Since the MILP algorithm does not scale well with an increasing number of tasks (Figure 25) and task periods, we test multiple-suspension self-suspending tasks in Figures 29 and 32 without the MILP algorithm. The schedulability ratio is the number of feasible systems over the total systems. The total running time consists of matrix building time and solver running time.

In the task systems, task periods $\mathcal{P}_i$ are randomly generated in the range $[P_{low}, P_{high}]$. P_{low} and P_{high} are the low and high bounds of the task periods. The UUniFast algorithm [12] divides the utilizations U_i of n tasks under system utilization U_{cap}. The total execution time is $E_i = \mathcal{P}_i \cdot U_i$, and the suspension delay is generated from $[S_{low} \cdot (1-U_i) \cdot \mathcal{P}_i, S_{high} \cdot (1-U_i) \cdot \mathcal{P}_i]$. S_{low} and S_{high} in suspension range [S_{low}, S_{high}] are the low and high suspension index bounds, respectively. The UUniFast algorithm also divides the total execution time into frame execution times. ϵ represents the threshold in the LP-based algorithm shown in Figure 15 and is set to be 0.01. Since all algorithms perform well under small system utilization U_{cap}, we focus on the experiments whose system utilization $U_{cap} \geq 0.5$.

In Figures 22 and 23, the x-axes represent the system utilization $U_{cap} \in [0.5, 0.9]$ with a step size of 0.05. Each task system contains five tasks. The task configuration parameters are $P_{low} = 10$, $P_{high} = 100$, $S_{low} = 0.3$, and $S_{high} = 0.6$. The y-axes represent the schedulability ratio and total running time in Figures 22 and 23, respectively. The data are the average numbers of 500 runs on each U_{cap}. Figure 22 shows that our LP-δ is better than PDA and EDA algorithms in terms of schedulability ratio. The iteration numbers of all tested LP-δ

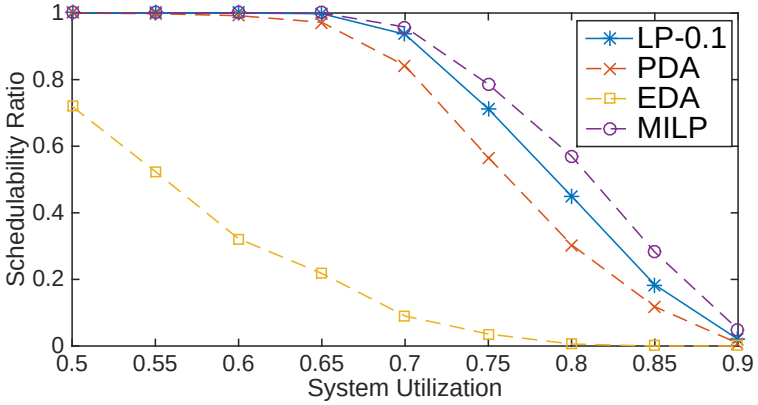


Figure 22: The schedulability ratio of the algorithms at system utilization [0.5, 0.9].

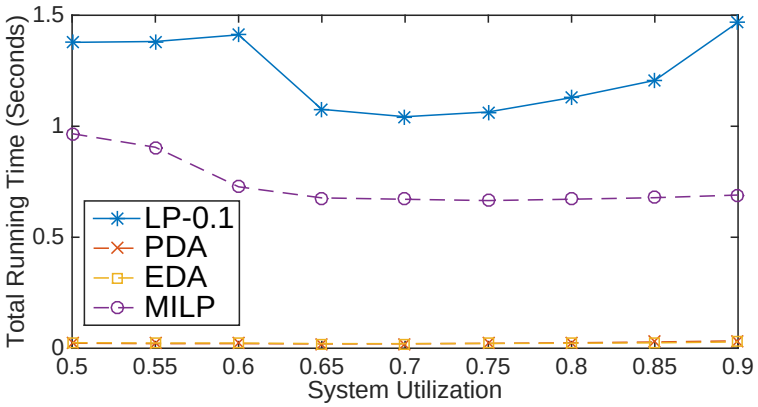


Figure 23: The average running time of the algorithms at system utilization [0.5, 0.9].

Figure 24: The comparison of our LP-based algorithm with the MILP and other polynomial-time algorithms on schedulability ratio and average running time.

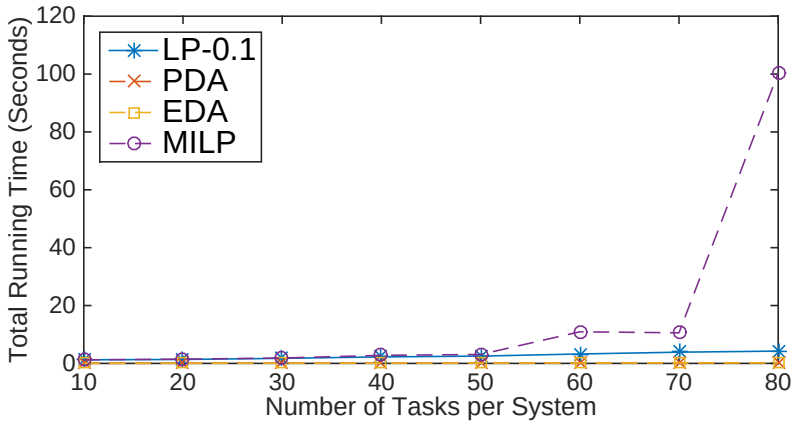


Figure 25: The average running time of the algorithms as the number of tasks increases.

algorithm are smaller than five. The multiple runs of the LP algorithm make the LP-δ algorithm take slightly longer than the MILP algorithm shown in Figure 23. The MILP can be relatively efficient for small enough task systems; however, as the number of tasks/frames increases, the MILP running time increases exponentially. Note that in Figure 24, we focus on a small system where we can gauge the effectiveness of the LP in comparison with the MILP and other algorithms. With $U_{cap} = 0.5$, Figure 25 shows that the execution time of the MILP algorithm increases dramatically when the number of tasks increases. Multiple input dimensions affect the execution time of the MILP algorithm, e.g., the task periods. Task periods directly affect the number of integer variables of the MILP algorithm and the running time is longer with higher task periods even when the number of tasks in the system is small. The running time of the LP-based algorithm scales relatively well.

Since we use the concave programming algorithm to guide the LP-based algorithm and have not proved a speed-up factor for the LP-based algorithm, we perform experiments on $\mathcal{L}$ value and maximum error ($\sqrt{\mathcal{L}/\mathcal{L}_{\text{MILP}}} - 1$ by the transformation of Theorem 4). $\mathcal{L}$ shows how close the value of the heuristic algorithm is to the MILP algorithm. $\mathcal{L}$ indicates the minimization of the maximum demands over all tested intervals. E.g., assume there

exist two heuristic algorithms that generate $\mathcal{L} = 0.2$ and 0.9, respectively. Both algorithms will give successful schedules in the schedulability ratio test, but the one with $\mathcal{L} = 0.2$ is a tighter schedule compared to the other one. If $\mathcal{L} > 1$, the system is not schedulable. We also compare the maximum error of the LP-δ algorithm since the error can be larger than δ.

Figure 26 shows the average $\mathcal{L}$ value of the algorithms among all system utilization points. The LP-0.1 algorithm returns the closest values to the MILP algorithm. The maximum error values shown in Figure 27 take the maximum values among 500 runs in each utilization point. Our LP-based algorithm returns the smallest error across all algorithms.

6.5.2 The Experiments for Multiple-Suspension Self-Suspending Tasks

Among the shown experiments on self-suspending tasks with one suspension frame, the average number of iterations of the LP-based algorithm is smaller than five among all system utilization U_{cap}. Since we believe that the algorithms can approach local optimal with a small number of iterations, we fix the number of iterations to five and test on multiple-suspending tasks.

In Figures 31 and 34, the data for each system utilization point is based on 100 runs. Each run of the system contains 30 tasks and each task contains six execution frames separated by five suspending frames (11 frames in total). $P_{low} = 10$ and $P_{high} = 100$. Since the MILP-based approach in this setting takes much longer than the LP-based algorithm, we do not include the MILP-based approach in this experiment. The MILP-based approach takes more than $1.5 * 10^3$ (respectively, $3.0 * 10^3$) seconds with optimality gap (the gap between the lower and upper objective bounds) which is larger than 10% (respectively, 5%).

In Figure 29, the system utilization $U_{cap} \in [0.8, 0.96]$ with step size of 0.02 is shown on the x-axis. Figure 29 has the suspension range with $S_{low} = 0.1$ and $S_{high} = 0.3$. In

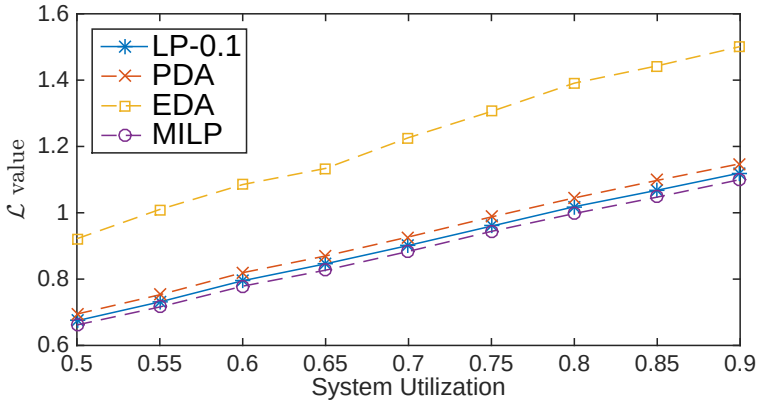


Figure 26: The $\mathcal{L}$ value of the algorithms at system utilization [0.5, 0.9].

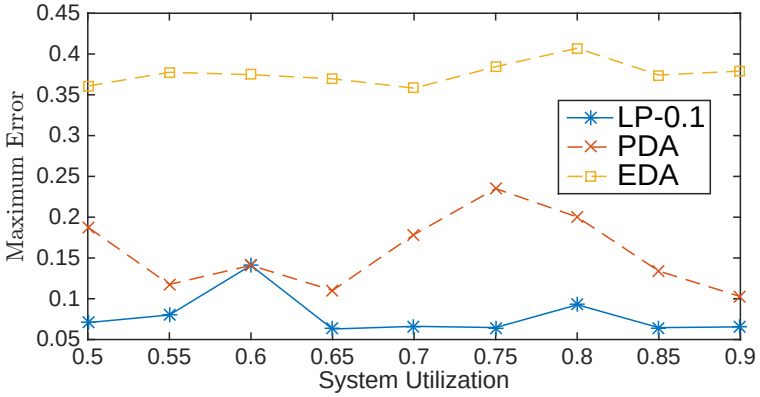


Figure 27: The maximum error of the algorithms compared with the MILP algorithm.

Figure 28: The quality of the LP-based algorithm on the $\mathcal{L}$ value and the maximum system error.

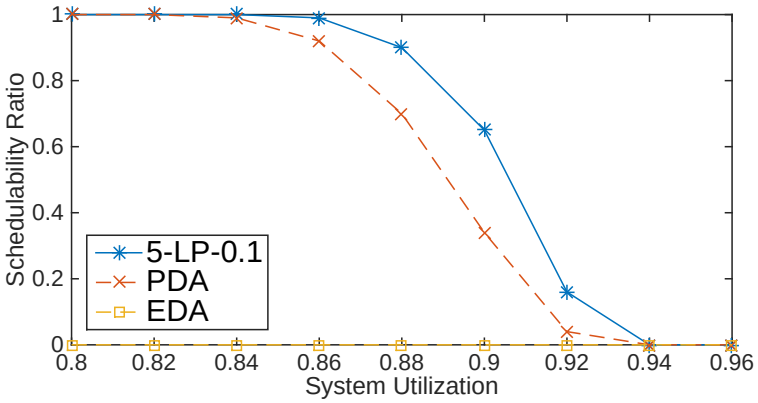


Figure 29: The schedulability ratio of the algorithms at system utilization [0.8, 0.96].

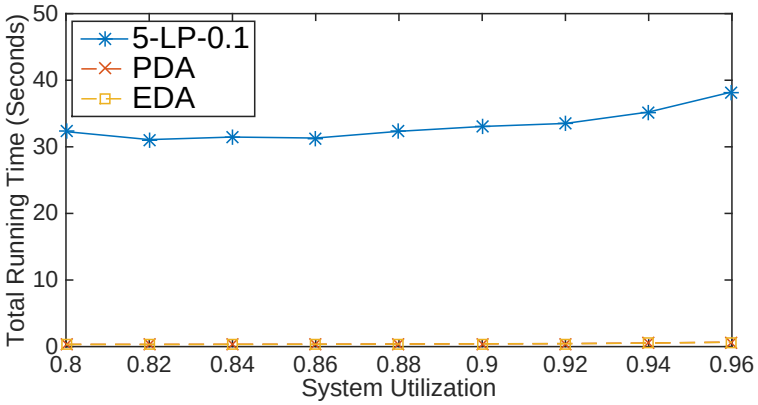


Figure 30: The average running time of the algorithms at system utilization [0.8, 0.96].

Figure 31: Comparison of our LP-based algorithm with other polynomial-time algorithms on the schedulability ratio and average running time.

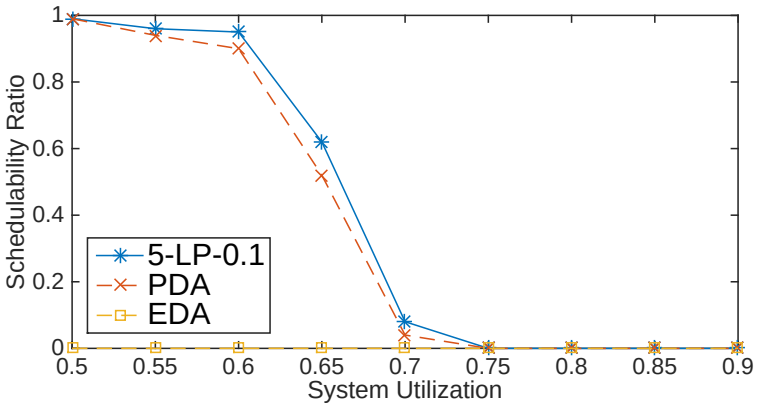


Figure 32: The schedulability ratio of the algorithms at system utilization [0.5, 0.9].

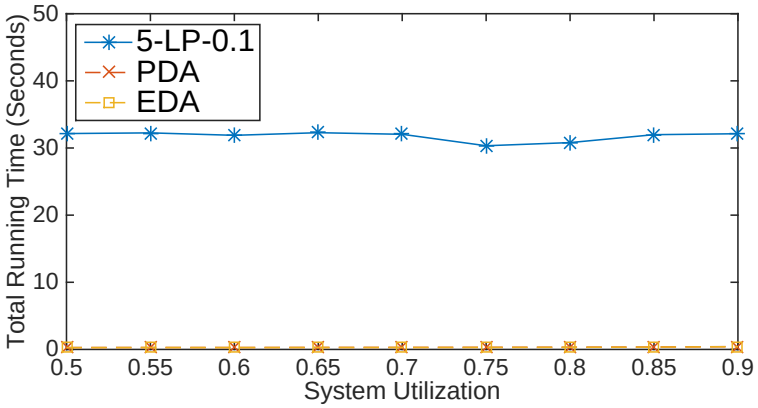


Figure 33: The average running time of the algorithms at system utilization [0.5, 0.9].

Figure 34: The comparison of our LP-based algorithm with other polynomial-time algorithms on schedulability ratio and average running time.

Figure 32, the system utilization $U_{cap} \in [0.5, 0.9]$ with a step size of 0.05 is shown on the x-axis. Figure 32 has the suspension range with $S_{low} = 0.3$ and $S_{high} = 0.6$. Figures 29 and 32 show that our LP-δ is the best among all polynomial-time algorithms in terms of schedulability ratio. The running times in Figures 30 and 30 reveal that LP-δ also scales well. The improvements for low suspension range $[0.1, 0.3]$ are better than the one with long range $[0.3, 0.6]$. The reason is that when the system specification has more slack time (small frame execution time and short suspending length), the LP-based algorithms can be "trained" to get near optimal parameters during the five iterations. In other words, e.g., the frames deadlines will be equal to their corresponding execution times if there are no slacks for all tasks, and all algorithms will return identical frame deadlines.

Our LP-based algorithm always yields higher schedulability ratio compared to other polynomial-time algorithms. The average running time is competitive overall even when compared with non-mathematical-programming based algorithms such as EDA/PDA.

6.6 Conclusions

In this chapter, we propose a concave programming approximation algorithm and prove its speed-up factor (can approach one) compared to the optimal MILP algorithm. Under the guidance of the tunable small speed-up factor, we present the general LP-based scheme to schedule GMF-PA tasks. We further optimize the LP-based algorithm and apply it to schedule one-suspension tasks. Extensive experiments show that our algorithms improve the schedulability ratio and have competitive running time compared to the previous results.

CHAPTER 7 PARAMETER ASSIGNMENT AND SCHEDULABILITY ANALYSIS IN DISTRIBUTED SYSTEMS

In this chapter, we introduce the parameter assignment and schedulability analysis (considered as a combined technique) in distributed systems under the dGMF-PA model. In Chapter 7.1, we introduce end-to-end flows and their schedulability analysis. Chapter 7.2 states the problem of this chapter. Chapter 7.3 presents our combined algorithm which uses mixed-integer linear programming (MILP) to get a necessary schedulability test under EDF scheduling. An approximation algorithm of MILP is presented in Chapter 7.4. In Chapter 7.5, we conduct extensive experiments and compare them with state-of-the-art results. At last, Chapter 7.6 concludes this work.

7.1 Introduction

A job in the sporadic task model has an individual continuous unit of work. Sporadic tasks are independent except for resource contention. Such simple models are concise and able to represent many applications in uniprocessor systems, but not precise enough to represent complex tasks in distributed systems. In practice, a multimedia function [51] or a network service [40] usually consists of subtasks which may have precedence constraints. An end-to-end flow models a precedence graph as a chain in which a subtask becomes ready to execute when its preceding subtasks on the chain complete. In distributed real-time systems, subtasks of end-to-end flows can be (sometimes have to be) assigned to execute on different processors. For instance, in common video applications, data needs to be transformed from analog signals to digital signals. The digital signals are transmitted over the networks and transformed back to analog signals at the client side. These three steps have precedence constraints and can be modeled as an end-to-end flow.

The schedulability analysis for such distributed real-time systems is drawing increased attention, as real-time applications are becoming more and more complex. Since the problem of optimal task assignment in distributed real-time systems is NP-hard [35], we assume that subtasks are statically assigned to processors before a schedulability test is performed and focus instead on the priority assignment of subtasks. Due to the NP-hardness [47] of priority assignment for subtasks in end-to-end flows, many heuristics have been presented. The schedulability analysis of most heuristics consists of two independent steps. The first step is priority assignment and the second step utilizes the assignment to make scheduling decisions. A priority assignment such as PD (Proportional Deadline Algorithm) [47] can be efficient. PD assigns subtasks relative deadlines that are proportional to their execution times. However, such analysis often introduces pessimism as schedulability hinges upon the effectiveness of the priority assignment of subtasks. Iterative-based methods [67] have been considered to improve schedulability ratio. In such methods, the current iteration of priority assignment is calculated based on the parameters of the system in preceding iterations, and the assignment can affect the parameters in the next iterations. The iterations stop when the system is schedulable or some stopping criterion is met. However, pessimism also exists in iterative-based methods since the priority assignment and schedulability analysis of a system are not considered jointly in each iteration.

In order to reduce potential pessimism, we combine priority assignment under EDF scheduling and schedulability analysis together into one framework which utilizes mathematical programming. Two combined algorithms are developed under our dGMF-PA (distributed generalized multiframe tasks with parameter adaptation) model which extends the GMF-PA (generalized multiframe tasks with parameter adaptation) model. The dGMF-PA

model can represent end-to-end flows in distributed systems. We refer to the minimum inter-arrival time among frames as a period for simplicity. We refer to an end-to-end flow as a task and a subtask as a frame to be congruent with the dGMF-PA model. The insight of our work is that deadlines and periods of frames are flexibly chosen to increase the schedulability of distributed systems.

The first algorithm presented is a necessary schedulability test (in general) under EDF scheduling. The algorithm becomes an exact schedulability test and can select relative deadlines and periods of frames when parameters are integers. An approximation algorithm, which is proved to be (in general) a sufficient schedulability test, can reduce the running time and select its frame parameters. We also prove the speed-up factor is $1 + \epsilon$ where ϵ can be arbitrarily small, with respect to the exact schedulability test of dGMF-PA tasks under EDF scheduling. Note that the two algorithms are both offline algorithms. In other words, parameters are fixed once parameter assignment and schedulability test have been completed.

Our Contributions in this Chapter:

- In distributed systems, we extend the exact and approximation algorithms used in uniprocessor systems to be capable of selecting frame deadlines and periods for the dGMF-PA tasks.

- We apply our parameter selection algorithm and its approximation algorithm on end-to-end flows in distributed systems.

- We prove that the speed-up factor of the MILP-based approximation algorithm is $1 + \epsilon$ with respect to the exact schedulability test of dGMF-PA tasks under EDF scheduling.

- We conduct extensive experiments to show that our algorithms outperform previous work in terms of schedulability and average running time.

Next, we state the problem in Chapter 7.2 and introduce the combined technique (parameter selection and schedulability test) by mixed-integer linear programming (MILP) in Chapter 7.3.

7.2 Problem Statement

Let $\mathsf{dbf}_i(t, \vec{F}_i, p)$ be the *task* demand bound function of a dGMF-PA task τ_i within the interval length t on processor p. Let $\vec{F}_i$ = [D_i^0, P_i^0, D_i^1, P_i^1,..., $D_i^{N_i-1}$, $P_i^{N_i-1}$] represent an assignment of all task parameters (frame deadlines and periods) of task τ_i. The supply bound function $\mathsf{sbf}(t)$ gives the lower bound of resources that the system can supply over an interval of length t. In general, the sufficient and necessary condition [8] for a uniprocessor feasible system is given in Equation 7.1, and all processors in a distributed systems must satisfy this condition.

$$\sum_{\tau_i \in \mathcal{T}} \mathsf{dbf}_i(t, \vec{F}_i, p) \leq \mathsf{sbf}(t), \quad \forall t, p. \tag{7.1}$$

Problem Definition. *Given the above model, our goal is to find an optimal and valid assignment $\vec{F}_i$ of frame parameters of all tasks so that the worst-case demand $\sum_{\tau_i \in \mathcal{T}} \mathsf{dbf}_i(t, \vec{F}_i, p)$ over all time intervals of length t and processors p is minimized.*

7.3 The Exact Deadline Assignment of End-to-End Flows in the dGMF-PA Model

In this chapter, we describe the combined technique of deadline selection for each frame and schedulability analysis under our dGMF-PA model by using mixed-integer linear

programming (MILP) under EDF scheduling which utilizes demand and supply bound functions. The deadline of each frame is flexible subject to the frame and cycle constraints. Along with the selection, the schedulability analysis provides a necessary feasibility test for arbitrary real-valued task parameters. We prove the sufficiency and necessity of the schedulability test when task parameters are integers.

Similar to the algorithms in uniprocessor systems, we build our MILP to select relative deadlines for all frames. At the same time, MILP gives a necessary feasibility test. However, note that for non-integer parameters, since the MILP is only necessary, the returned selection of deadlines may not be feasible. Later, in Chapter 7.4, we will give an approximation algorithm for the MILP that returns a feasible selection of relative deadlines for the non-integer case.

Since frame relative deadlines and periods are variables which are selected by MILP, Equation 7.1 becomes a set of constraint functions that a feasible system must obey. Note that the demand of an empty frame $\phi_{i,p}^j$ is always zero since $E_{i,p}^j = 0$. Figure 35 illustrates a graph stating that frame demand is a stair function of deadlines in an interval of length t. The detailed notations will be introduced later. In our algorithm, the supply bound function is $\mathsf{sbf}(t) = t$ and the length t of any interval length is an integer. Our MILP can return an assignment if the system is schedulable. That is, we can determine the necessary feasibility of the system and select potential deadlines at the same time.

The general steps of our algorithm are as follows. For a given sequence of frames and a time interval of length t, we calculate the demand contribution of each frame to that interval length. Adding the demands of all frames generates the demand of a task, and adding the demands of all tasks (over all possible sequences of frames) generates the total

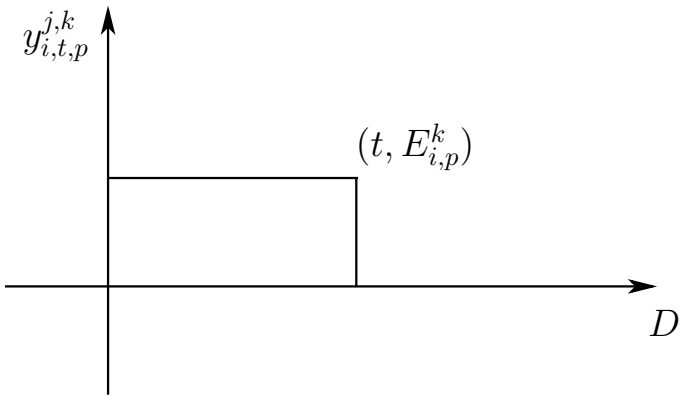


Figure 35: The demand $y^{j,k}_{i,t,p}$ in this figure is calculated when t is smaller than one cycle period. When the deadline of frame $\phi^k_{i,p}$ ends inside the interval length t, the demand $y^{j,k}_{i,t,p}$ is $E^k_{i,p}$. Otherwise, the demand $y^{j,k}_{i,t,p}$ is zero.

demand in each processor at the time interval length t. The system is schedulable at a time interval length if the demand is no larger than the supply in all processors. We check all interval lengths, which are integers, in the algorithm.

For a given interval length t, we need to calculate the demand for every possible sequence of frames of task $\tau_{i,p}$ over any interval of length t and processor p. Assume that the first frame of $\tau_{i,p}$ to arrive in such an interval is $\phi^j_{i,p}$ (i.e., the j'th frame on processor p). The demand of any sequence starting with the j'th frame over a t-length interval is maximized if the j'th frame arrives exactly at the start of the interval and subsequent frames arrive as soon as possible (e.g., see Baruah et al. [8] for GMF schedulability). To calculate the demand from the k'th frame in such an interval for the specified sequence, $y^{j,k}_{i,t,p}$ represents the demand of this frame. We will calculate $y^{j,k}_{i,t,p}$ for all possible $i, j, k, p,$ and t. For simplicity, we use "$\forall$" to represent the ranges of variables. The task index i ranges from zero to $n-1$. The superscripts j and k represent the starting frame and the current frame respectively, and both have the ranges from zero to $N_i - 1$. A processor p has the ranges

from zero to $Q-1$. It has been shown [8] that the maximum interval length is bounded by $O(\frac{U_{cap}}{1-U_{cap}} \cdot \max_{\tau_i \in \tau}(\mathcal{P}_i - D_i^0))$ where $U_{cap} < 1$. We use $H = \lceil \frac{U_{cap}}{1-U_{cap}} \cdot \max_{\tau_{i,p} \in \tau}(\mathcal{P}_i - E_i^{min}) \rceil$ as the maximum integer length interval since we do not know deadlines beforehand in our dGMF-PA model. Note that the range of any interval length t is shown in uniprocessor systems [8]. We choose the maximum utilization U_{cap} among processors to calculate the maximum interval length H. We use such abbreviations across this thesis. The demand of the task $\tau_{i,p}$ starting from the j'th frame in time interval length t is $y_{i,t,p}^{j}$. The maximum demand of $\tau_{i,p}$ among all starting frames is $y_{i,t,p}$.

Figure 36 shows our *Deadline Assignment and Feasibility Test* MILP, notations in bold font are constants and the other notations are variables. Lines 3 to 6 present the basic constraints introduced in Chapter 3. Line 7 calculates the demand of $y_{i,p,t}^{j,k}$. The interval length $\lfloor \frac{t}{\mathcal{P}_i} \rfloor$ tracks the number of cycle periods in t, and $\lfloor \frac{t}{\mathcal{P}_i} \rfloor * E_{i,p}^{k}$ is the demand of $\phi_{i,p}^{k}$ in such cycle periods. The parameter $x_{i,t,p}^{j,k}$ is restricted to be an integer value and works as a "flag" (either zero or one) to decide whether the demand $E_{i,p}^{k}$ should be added in the interval length $t - \lfloor \frac{t}{\mathcal{P}_i} \rfloor * \mathcal{P}_i$ as shown in Figure 35. Note that all frames are released as soon as possible. The analysis of a demand in $[0, t - \lfloor \frac{t}{\mathcal{P}_i} \rfloor * \mathcal{P}_i]$ is equal to the one in $[\lfloor \frac{t}{\mathcal{P}_i} \rfloor * \mathcal{P}_i, t]$. The "flag" $x_{i,t,p}^{j,k}$ is decided by the constraints in Lines 8 and 9. Line 8 is the constraint function that decides the value of $x_{i,t,p}^{j,k}$. The length t_b in Line 9 is the summation of the previous periods $\lfloor \frac{t}{\mathcal{P}_i} \rfloor * \mathcal{P}_i$ and the distance from the starting j'th frame to k'th frame $\left(\sum_{p=0}^{Q-1} \left(\sum_{q=0}^{(k-j-1) \mod N_i} P_{i,p}^{(j+q) \mod N_i} \right) \right) + D_{i,p}^{k}$.

The length t_b ensures the sequence of real frames in distributed systems. That is, since the frames before $\phi_{i,p}^{k}$ may be empty frames on the processor p, we add all the

Deadline Assignment and Feasibility Test

Initialization: $E_{i,p}^k = D_{i,p}^k = P_{i,p}^k = 0$ if $\phi_{i,p}^k$ is an empty frame.

minimize: $\mathcal{L}$

subject to:

$\boldsymbol{E_{i,p}^k} \leq \boldsymbol{\underline{D}_{i,p}^k} \leq D_{i,p}^k \leq \boldsymbol{\overline{D}_{i,p}^k}, \quad \forall i, k, p.$

$\boldsymbol{E_{i,p}^k} \leq \boldsymbol{\underline{P}_{i,p}^k} \leq P_{i,p}^k \leq \boldsymbol{\overline{P}_{i,p}^k}, \quad \forall i, k, p.$

$D_{i,p}^k \leq P_{i,p}^k, \quad \forall i, k, p.$

$$\sum_{p=0}^{Q-1} \sum_{k=0}^{N_i-1} P_{i,p}^k \leq \boldsymbol{\mathcal{P}_i}, \sum_{p=0}^{Q-1} \left(D_{i,p}^{N_i-1} + \sum_{j=0}^{(N_i-2)} P_{i,p}^j \right) \leq \boldsymbol{\mathcal{D}_i}, \forall i.$$

$y_{i,t,p}^{j,k} = x_{i,t,p}^{j,k} * \boldsymbol{E_{i,p}^k} + \lfloor \frac{\boldsymbol{t}}{\boldsymbol{\mathcal{P}_i}} \rfloor * \boldsymbol{E_{i,p}^k}, \quad \forall i, j, k, t, p.$

$\frac{\boldsymbol{t}-t_b}{\boldsymbol{\mathcal{P}_i}} \leq x_{i,t,p}^{j,k} - \frac{\boldsymbol{realmin}}{\boldsymbol{\mathcal{P}_i}}, \quad \forall i, j, k, t, p.$

$$t_b = \left(\sum_{p=0}^{Q-1} \left(\sum_{q=0}^{(k-j-1) \mod N_i} P_{i,p}^{(j+q) \mod N_i} \right) \right) + D_{i,p}^k + \lfloor \frac{\boldsymbol{t}}{\boldsymbol{\mathcal{P}_i}} \rfloor * \boldsymbol{\mathcal{P}_i}$$

$$y_{i,t,p}^j = \sum_{k=0}^{N_i-1} y_{i,t,p}^{j,k}, \quad \forall i, j, t, p.$$

$y_{i,t,p} \geq y_{i,t,p}^j, \quad \forall i, j, t, p.$

$$\sum_{i=0}^{n-1} y_{i,t,p} \leq \boldsymbol{\mathcal{L}} * \boldsymbol{t} \quad \forall t, p.$$

and:

$D_{i,p}^k, P_{i,p}^k, t_b, y_{i,t,p}^{j,k}, y_{i,t,p}^j, y_{i,t,p}, \boldsymbol{\mathcal{L}} \in \mathbb{R}^*, x_{i,t,p}^{j,k} \in \{0, 1\}.$

Figure 36: Our MILP algorithm.

periods of the j'th frame to $k-1'$th frame in all processors. For example, the length $t_b = \left(\sum_{p=0}^{Q-1} P_{i,p}^1 + \sum_{p=0}^{Q-1} P_{i,p}^2\right) + D_{i,p}^3 + \lfloor \frac{t}{\mathcal{P}_i} \rfloor * \mathcal{P}_i$ if we consider the interval starting with the arrival of the first frame and ending with the deadline of the third frame in the end-to-end flow τ_i. In the inequality of Line 8, the lengths t_b and t decide whether the demand of the k'th frame in length $t - \lfloor \frac{t}{\mathcal{P}_i} \rfloor * \mathcal{P}_i$ will be added to $y_{i,t,p}^{j,k}$. The constant $realmin$ is the smallest representable positive number. When $t \geq t_b$, the flag $x_{i,t,p}^{j,k}$ must be one and the demand $x_{i,t,p}^{j,k} * E_{i,p}^k$ contributes to $y_{i,t,p}^{j,k}$. When $t < t_b$, the flag $x_{i,t,p}^{j,k}$ can be either zero or ones. However, the demand $y_{i,t,p}^{j,k}$ is overestimated when $x_{i,t,p}^{j,k} = 1$. The solver MILP tends to choose zero for $x_{i,t,p}^{j,k}$ because of the smaller demand, and the details are shown in Lemma 9. Note that the inequality in Line 8 is always correct when $x_{i,t,p}^{j,k}$ is one and $t \geq t_b$, and when $x_{i,t}^{j,k}$ is zero and $t < t_b$.

In Line 10, the demand $y_{i,t,p}^{j}$ of task τ_i starts from the j'th frame. In Line 11, the demand $y_{i,t,p}$ is the maximum demand for $\tau_{i,p}$ over all possible starting frames. At last, the demand of all tasks $\sum_{i=0}^{n-1} y_{i,t,p}$ has to be less than the supply bound function for all interval lengths t and processors p as shown in Equation 7.1; otherwise, the system is not schedulable. In Line 12, $\mathcal{L}$ is set to indicate the degree of schedulability of the system. If the system is schedulable, then $\mathcal{L} \leq 1$.

In the setting of our MILP, the variables $D_{i,p}^k$, $P_{i,p}^k$, t_b, $y_{i,t,p}^{j,k}$, $y_{i,t,p}^{j}$, $y_{i,t,p}$, and $\mathcal{L}$ are free variables. The number of all variables is pseudo-polynomial bounded. The flag $x_{i,t,p}^{j,k}$ is restricted to be an integer variable that is either zero or one. The relationship among the variables is summarized in Figure 37. The boxes with solid lines contain free variables and the boxes with dotted lines contain constants. The arrows show the dependable relationships and the integers on the arrows indicate the number of lines in the MILP. For

example, Lines 6 to 9 show that the constant $\mathcal{P}_i$ has an effect on the variables $P^k_{i,p}$, t_b, $x^{j,k}_{i,t,p}$ and $y^{j,k}_{i,t,p}$. All variables are connected and constrained in MILP. Eventually, minimizing $\mathcal{L}$ also minimizes the total demand $\sum_{i=0}^{n-1} y_{i,t,p}$.

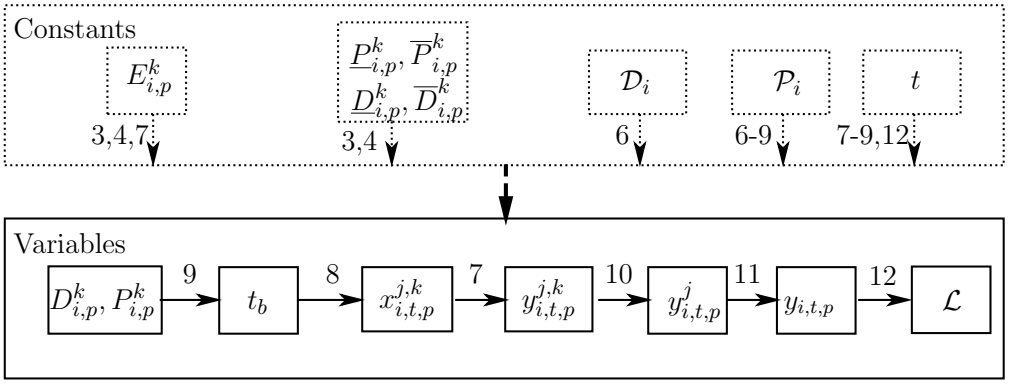


Figure 37: Relationship among the parameters.

In our dGMF-PA model for distributed systems, we prove that our MILP is a necessary schedulability test in general, and the MILP is also a sufficient and necessary schedulability test for integer parameters in Lemma 9 and Theorem 8.

Lemma 9. *The value of $y^{j,k}_{i,t,p}$ in the MILP is the exact worst-case demand of frames $\phi^k_{i,p}$ over an interval of length t when the first frame of τ_i to arrive in the interval is j'th frame. (with respect to the deadline assigned to each frame of $\tau_{i,p}$ by the MILP).*

Proof. If the j'th frame is not assigned on the processor p, the demand $y^{j,k}_{i,t,p}$ is the exact worst-case demand which is zero. The proof is straightforward since the frame does not execute on the processor p.

When the j'th frame is assigned on the processor p, it is trivial that $y^{j,k}_{i,\lfloor \frac{t}{\mathcal{P}_i} \rfloor * \mathcal{P}_i,p}$ is the exact demand $\lfloor \frac{t}{\mathcal{P}_i} \rfloor * E^k_i$ in the time interval length $\lfloor \frac{t}{\mathcal{P}_i} \rfloor * \mathcal{P}_i$. We will prove that the worst-case demand $y^{j,k}_{i,t',p} = x^{j,k}_{i,t,p} * E^k_{i,p}$ is exact in the interval length $t' = t - \lfloor \frac{t}{\mathcal{P}_i} \rfloor * \mathcal{P}_i$. Worst-case means that the interval length t starts at the release time of the j'th frame and all succeeding

frames are released as soon as possible. We will show that $y^{j,k}_{i,t',p}$ is an upper bound and a lower bound on the demand. That is, the demand $y^{j,k}_{i,t,p}$ is exact. For simplicity, we refer to $y^{j,k}_{i,t',p}$ ($x^{j,k}_{i,t',p}$) as y (x).

Assume that $t'' = \left(\sum_{p=0}^{Q-1}\left(\sum_{q=0}^{(k-j-1) \mod N_i} P_{i,p}^{(j+q) \mod N_i}\right)\right) + D^k_{i,p}$, there are also two situations: when $0 \leq t' < t''$ and $t'' \leq t' < \mathcal{P}_i$. Note that t' is smaller than $\mathcal{P}_i$ from definition. When $0 \leq t' < t''$, x can be zero or one from the MILP. Since we minimize $\mathcal{L}$ in MILP, y is also minimized to take the value zero (by $x = 0$). When $t'' \leq t' < \mathcal{P}_i$, y has to be $E^k_{i,p}$ to satisfy the constraints in Lines 7 to 9 of our MILP.

When $0 \leq t' < t''$, $y = 0$. The demand y is a lower bound since no demand takes negative values. We prove that y is an upper bound by contradiction. If there exist $y' > y$, $y' = E^k_{i,p}$ since x can only take an integer value one or zero. In this case, $t' \geq t''$ and get a contradiction with $0 \leq t' < t''$. y is an upper bound and a lower bound when $0 \leq t' < t''$.

When $t'' \leq t' < \mathcal{P}_i$, $y = E^k_{i,p}$ since $x = 1$. The proof of the lower bound is similar to the proof of the upper bound when $t'' \leq t' < \mathcal{P}_i$. That is the demand cannot be smaller than y; otherwise, t' will be smaller than t''. We prove that the demand y is an upper bound by contradiction. Assume that the demand y' is the upper bound which is larger than the demand $y = E^k_{i,p}$. If $y' > E^k_{i,p}$, the corresponding interval length t' has to be larger than $\mathcal{P}_i$. This is a contradiction since $t'' \leq t' < \mathcal{P}_i$. y is an upper bound and a lower bound when $t'' \leq t' < \mathcal{P}_i$.

In total, the demand $y^{j,k}_{i,t,p}$ is the exact worst-case demand for the frame $\phi^k_{i,p}$ over an interval of length t when the first frame of $\tau_{i,p}$ to arrive in the interval is the j'th frame. □

Theorem 8. *For arbitrary, real-valued parameters, our MILP is a necessary feasibility test.*

When the period and deadline parameters are integers (i.e., $D_{i,p}^k$, $P_{i,p}^k \in \mathbb{N}$, $\forall$ i, k and p), the MILP is an exact feasibility test.

Proof. It is straightforward to prove that our MILP is a necessary feasibility test in general. If a distributed system is feasible, the worst-case demand ($\sum_{i=0}^{n-1} y_{i,t,p}$) of all tasks over any interval length t must be smaller than t in any processor p.

In Lemma 1, we have proved that $y_{i,t,p}^{j,k}$ in the MILP is the exact worst-case demand of frames $\phi_{i,p}^k$ over an interval of length t when the first frame of τ_i to arrive in the interval is the j'th frame. $y_{i,t,p}^{j}$ is thus the exact demand of task $\tau_{i,p}$ over length t starting from the j'th frame, and $y_{i,t,p}$ is the exact worst-case demand of $\tau_{i,p}$ over length t. $\sum_{i=0}^{n-1} y_{i,t,p} \leq \boldsymbol{t}$ is a sufficient feasibility test when $D_{i,p}^k$, $P_{i,p}^k$, $t \in \mathbb{N}$. The algorithm is exact when the frame deadline and period parameters are integers, since it can be easily shown that the demand function changes value in this case only at integer times; thus, the MILP exactly checks all the relevant time intervals. Note that our MILP in general is not a sufficient feasibility test when this integer constraint is removed since it does not check all real values in the range $[0, H]$. □

Due to the fact that our MILP is not an exact feasibility test in general and does not scale well, we introduce a sufficient feasibility test in general in Chapter 7.4. The sufficient feasibility test is an approximation algorithm based on our MILP, where the running time is greatly reduced.

7.4 The Approximation Algorithm Based on our MILP

In the previous chapter, we have built our MILP which can select frame relative deadlines of dGMF-PA tasks under EDF scheduling. The method also indicates a necessary feasibility

test at the same time. However, solving an MILP is NP-hard in general. Furthermore, the feasibility of selecting deadlines in the dGMF-PA model is coNP-hard as the problem can be trivially transformed from the feasibility test of sporadic tasks [33]. In this chapter, we modify the MILP to obtain an approximation algorithm based on a reduction in the number of time interval lengths being tested[14]. We also show that the speed-up factor of our approximation algorithm is $1+\epsilon$ with respect to the exact schedulability test of dGMF-PA tasks under EDF scheduling. We have introduced an approximation algorithm under the GMF-PA model and such similar technique can be traced back to admission control for the arbitrary demand curves [30].

In Chapter 4.4, we have introduced the supply bound function $\mathsf{sbf}^a(t)$ that based on the smallest interval t_0, increasing rate ϵ, and the maximum interval length H. We will not repeat redefining the terms. Instead, we directly modify the general schedulability condition of Equation 6.1 with respect to the reduced set of test intervals.

Theorem 9. *Consider any distributed task system composed of tasks $\mathcal{T}$ (e.g., dGMF-PA tasks) where the $\mathsf{dbf}_i(t, \vec{F}_i, p)$ is computable for any $\tau_i \in \mathcal{T}$ and $p \in \mathcal{Q}$ ($\mathcal{Q}$ is the index set of processors). Then, by checking the following modified condition:*

$$\sum_{\tau_i \in \mathcal{T}} \mathsf{dbf}_i(t, \vec{F}_i, p) \leq \mathsf{sbf}^a(t), \quad \forall t \in T_a,\ p \in \mathcal{Q}, \tag{7.2}$$

when t_0 of $\mathsf{sbf}^a(t)$ must not be larger than $\min_{i,j,p}\{\sum_{p=0}^{Q-1} D_{i,p}^j\}$.

We have the following guarantee:

1. If $\sum_{\tau_i \in \mathcal{T}} \mathsf{dbf}_i(t, \vec{F}_i, p) \leq \mathsf{sbf}^a(t),\ \forall t \in T_a,\ p \in \mathcal{Q}$, the distributed system is EDF-

[14]The approximation is still an MILP (and thus still potentially intractable), but a reduction in constraints leads to a significant improvement in efficiency as shown in the evaluation chapter.

schedulable on unit-speed processors.

2. *If* $\exists t \in T_a \; and \; p \in \mathcal{Q}, \quad \sum_{\tau_i \in \mathcal{T}} \mathsf{dbf}_i(t, \vec{F}_i, p) > \mathsf{sbf}^a(t)$*, the system is EDF-infeasible when each processor is* $\frac{1}{1+\epsilon}$*-speed.*

Proof. We first prove the sufficiency. If $\sum_{\tau_i \in \mathcal{T}} \mathsf{dbf}_i(t', \vec{F}_i, p) \leq \mathsf{sbf}^a(t')$ at an interval length $t' = t_0 * (1+\epsilon)^k$ for any $k \in H_a$ in Equation 4.3, all demands over the range of intervals $(\frac{t'}{1+\epsilon}, t']$ are also smaller than $\mathsf{sbf}^a(t')$ since the demand bound function is a monotonically increasing function. In other words, the system is schedulable on any interval length in $(\frac{t'}{1+\epsilon}, t']$ if the system is schedulable on interval length t'. The test intervals are thus reduced to the set T_a. If $\sum_{\tau_i \in \mathcal{T}} \mathsf{dbf}_i(t', \vec{F}_i, p) \leq \mathsf{sbf}^a(t)$ for all $t \in T_a$ and $p \in \mathcal{Q}$, all unit-speed processors are EDF-schedulable. The distributed system composed of the processors is also EDF-schedulable, which indicates the sufficiency in the distributed system.

We prove the infeasibility on a slower processor when Equation 7.2 is not satisfied (equal to the proof of the "speed-up factor"). Assume $\sum_{\tau_i \in \mathcal{T}} \mathsf{dbf}_i(t^*, \vec{F}_i, p) > \mathsf{sbf}^a(t^*)$ at time interval length t^* and processor p. It must be that $t^* > t_0$ since for all values of $t \leq t_0$, $\mathsf{dbf}_i(t, \vec{F}_i, p)$ is zero by supposition that t_0 exceeds the minimum frame relative deadline. Furthermore, it is easy to observe that for all $t \geq t_0$, the $\mathsf{sbf}(t)$ is at most $(1+\epsilon)$ times larger than $\mathsf{sbf}^a(t)$. From this, we have:

$$
\begin{aligned}
\max_{t>0} \frac{\sum_{\tau_i \in \mathcal{T}} \mathsf{dbf}_i(t, \vec{F}_i, p)}{\mathsf{sbf}(t)} &\geq \frac{\sum_{\tau_i \in \mathcal{T}} \mathsf{dbf}_i(t^*, \vec{F}_i, p)}{\mathsf{sbf}(t^*)} \\
&\geq \frac{\frac{\sum_{\tau_i \in \mathcal{T}} \mathsf{dbf}_{i(t^*, \vec{F}_i, p)}}{(1+\epsilon)}}{\frac{\mathsf{sbf}(t^*)}{(1+\epsilon)}} \\
&\geq \frac{\sum_{\tau_i \in \mathcal{T}} \mathsf{dbf}_i(t^*, \vec{F}_i, p)}{\mathsf{sbf}^a(t^*) * (1+\epsilon)} \quad \text{(By Equation 4.3)} \\
&\geq \frac{1}{1+\epsilon} \quad \text{(By assumption).}
\end{aligned}
$$

Summing both sides of the above-derived inequality, we get:

$$\sum_{p\in\mathcal{Q}}\left(\max_{t>0}\frac{\sum_{\tau_i\in\mathcal{T}}\mathsf{dbf}_i(t,\vec{F}_i,p)}{\mathsf{sbf}(t)}\right)\geq\frac{Q}{1+\epsilon}$$

$$Q*\max_{t>0,p\in\mathcal{Q}}\frac{\sum_{\tau_i\in\mathcal{T}}\mathsf{dbf}_i(t,\vec{F}_i,p)}{\mathsf{sbf}(t)}\geq\frac{Q}{1+\epsilon}$$

$$\max_{t>0,p\in\mathcal{Q}}\frac{\sum_{\tau_i\in\mathcal{T}}\mathsf{dbf}_i(t,\vec{F}_i,p)}{\mathsf{sbf}(t)}\geq\frac{1}{1+\epsilon}$$

Thus, we have proved that the speed-up factor is $1+\epsilon$ over all processors in distributed systems, with respect to the exact schedulability test of dGMF-PA tasks under EDF scheduling.

□

We can now apply Theorem 9 to modify the MILP to create a sufficient approximate feasibility test for the dGMF-PA task model with arbitrary, real-valued parameters. To do so, we simply limit the range of t to now be T_a for all constraints that depend upon t, and modify Line 9 of MILP to be $\sum_{i=0}^{n-1} y_{i,t,p} \leq \mathcal{L} * \frac{\boldsymbol{t}}{1+\epsilon}$. Clearly, this reduces the number of constraints by a logarithmic factor (dependent upon our choice of ϵ). We refer to this approximate assignment algorithm as MILP-ϵ.

In all, the approximate MILP is a sufficient feasibility test. The number of the time interval lengths is reduced from $O(H)$ to $O(\log_{1+\epsilon} H)$. Since the number of variables and number of equations depend on the number of time interval lengths, the running time is greatly reduced.

7.5 Evaluation

We have implemented our MILP and approximation algorithm MILP-ϵ ($\epsilon > 0$) using the commercial solver GUROBI [1] in MATLAB. We compare our work with the combination

(represented by HOSPA-Offset) of the deadline assignment HOSPA [67,68] and offset-based analysis under EDF scheduling [59] in the MAST suite [2].

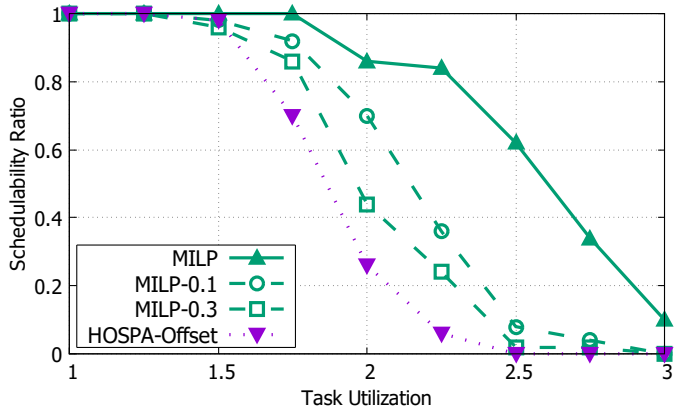
In order to generate a fair comparison with HOSPA-Offset, we set $D_{i,p}^k = P_{i,p}^k$ (Line 5 in MILP will be automatically satisfied) and $\mathcal{D}_i = \mathcal{P}_i$. The variables $D_{i,p}^k, P_{i,p}^k, \mathcal{D}_i$, and $\mathcal{P}_i$ are reduced to $D_{i,p}^k$ and $\mathcal{P}_i$ for all i, k and p. In this case, the end-to-end deadline of $\tau_{i,p}$ is $\mathcal{P}_i$. Because HOSPA-Offset has no frame constraints, we set $\underline{D}_{i,p}^k = E_{i,p}^k$ and $\overline{D}_{i,p}^k = \mathcal{P}_i$ for all frames. The constraints from Lines 3 to 6 of our MILP and MILP-ϵ thus become:

1. $\boldsymbol{E_{i,p}^k} \leq D_{i,p}^k, \quad \forall i, k, p.$
2. $\sum_{p=0}^{Q-1} \sum_{k=0}^{N_i-1} D_{i,p}^k \leq \boldsymbol{\mathcal{P}_i}, \quad \forall i.$

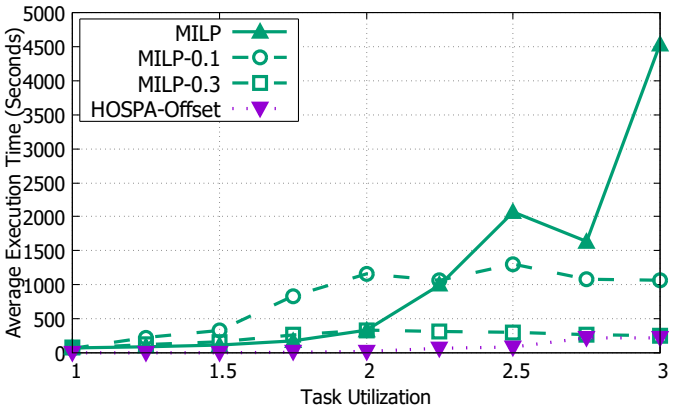
We follow the similar setting of the previous paper [67] to randomly generate end-to-end flows (tasks). There are five processors and eight tasks in the distributed system. Each task contains five frames which are randomly assigned on the processors in the distributed system. There are nine system utilization levels (100%, 125%, 150%,..., 300%) each of which contains fifty distributed systems. In each system, we use the UUniFast algorithm [12] to generate the utilization of each task and the execution time of each frame. We record the schedulability ratio (the number of schedulable systems over the total number of systems) and average running time of the task sets for each utilization level.

For the reason that our MILP is not scalable in general, we first generate Figure 38(a) and 38(b) in which tasks have small cycle periods that are randomly chosen in [1,10]. Note that each curve of MILP is a characterization of an “upper bound" on the best we can hope for in our model. MILP-0.1 and MILP-0.3 have higher schedulability ratio than HOSPA-Offset as shown in Figure 38(a), but have longer running time as shown in Figure 38(b). MILP-0.1

can schedule at most 44 % more than HOSPA-Offset when $U_{cap} = 2$, and MILP-0.3 can schedule at most 18 % more than HOSPA-Offset when $U_{cap} = 2$. MILP is at most 19.1 times slower than HOSPA-Offset, MILP-0.1 is at most 4.8 times slower than HOSPA-Offset, and MILP-0.3 is at most 0.5 times slower than HOSPA-Offset.

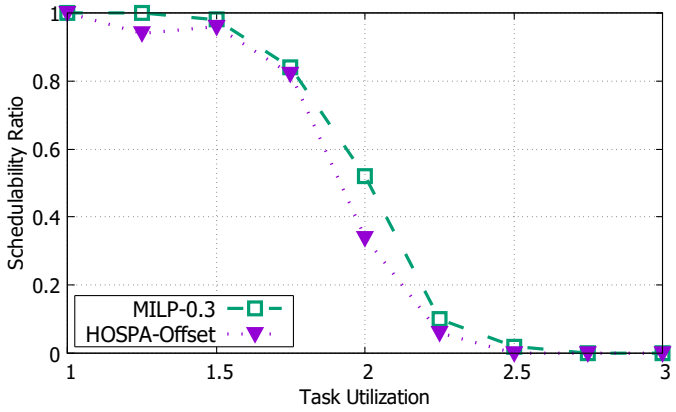


(a) $\mathcal{P}_i \in [1, 10]$

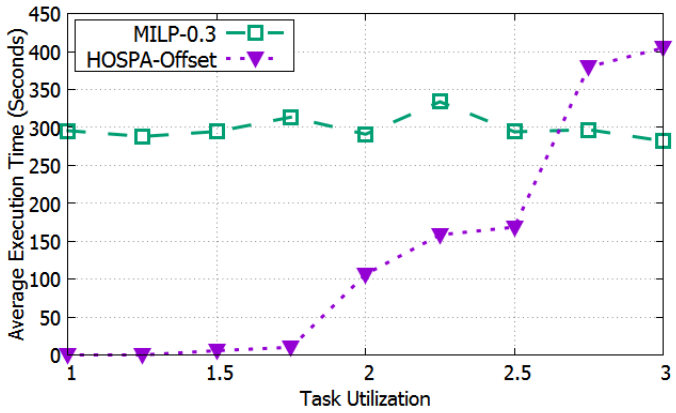


(b) $\mathcal{P}_i \in [1, 10]$

In order to generate a set of tasks with larger cycle periods, we have the experiments shown in Figure 38(c) and 38(d). The cycle period of each task is randomly chosen in [1,1000]. Figure 38(c) shows that any experiment with $\epsilon \leq 0.3$ will generate better schedulability ratio than HOSPA-Offset. MILP-0.3 can schedule at most 18 % more than HOSPA-Offset when $U_{cap} = 2$, and MILP-0.3 uses at most around 303 seconds more than



(c) $\mathcal{P}_i \in [1, 1000]$



(d) $\mathcal{P}_i \in [1, 1000]$

Figure 37: The figures show the schedulability ratio and average running time over task utilization from one to three. $\mathcal{P}_i \in [1, 10]$ is in Figures 38(a) and 38(b), and $\mathcal{P}_i \in [1, 1000]$ is in Figures 38(c) and 38(d).

HOSPA-Offset when $U_{cap} = 1.75$.

In all, our MILP and MILP-ϵ algorithms always yield higher schedulability ratio. The running time of the combined technique HOSPA-Offset is shorter in general; however, our MILP-ϵ is not worse by much and still efficient enough.

7.6 Summary

Upon the flexible GMF-PA model in uniprocessor systems, we propose the dGMF-PA model in distributed real-time systems. The relative deadlines of frames in end-to-end flows

can be flexibly chosen in our dGMF-PA model, using mixed-integer linear programming (MILP). Our MILP-based algorithm is an exact feasibility test when parameters are integers, and a necessary feasibility test in general. In order to reduce the running time of the MILP algorithm and give a sufficient schedulability test (in general), we propose an approximation algorithm MILP-ϵ based on the supply bound function. The number of time interval lengths is bounded by a logarithmic function of the task system parameters. Extensive experiments have shown that our algorithms improve the schedulability ratio of task sets when compared to the previous results.

CHAPTER 8 CONCLUSION

In uniprocessor systems, upon the GMF model, we propose the GMF-PA model which has frame constraints and task/cycle constraints to let the deadline and period of each frame be flexible. Using the mixed-integer linear programming (MILP), we propose an algorithm that can select frame deadlines and periods in the GMF-PA model. Our MILP-based algorithm is an exact feasibility test when parameters are integers, and a necessary feasibility test in general. In order to reduce the running time of the MILP algorithm and get a sufficient schedulability test in general, we propose an approximation algorithm MILP-ϵ based on the supply bound function. The number of time intervals being tested is bounded by a logarithmic function of task parameters. We prove that the MILP-ϵ is a sufficient feasibility test and the speed-up factor is $1 + \epsilon$, with respect to the exact schedulability test of GMF-PA tasks under EDF scheduling.

We apply our MILP and MILP-ϵ to self-suspending tasks. We remove the assumption that the deadlines are fixedly equally assigned in the previous work. Exhaustive experiments for both one-segment and multiple-segment self-suspending tasks have shown that our algorithms have improved the schedulability ratio and running time compared to the previous results. We also apply the MILP-ϵ algorithm to a robot car and a remote server. By transforming the tracking tasks to GMF-PA tasks under different environment conditions (e.g., network bandwidth, processing speed ratio between car and server, etc.), the MILP-ϵ algorithm also schedule more systems than EDA.

In distributed systems, we extend the GMF-PA model and propose the dGMF-PA model. The relative deadlines of frames in end-to-end flows can be flexibly chosen in our dGMF-PA

model, using mixed-integer linear programming (MILP). Our MILP-based algorithm is an exact feasibility test when parameters are integers, and a necessary feasibility test in general. In order to reduce the running time of the MILP algorithm and give a sufficient schedulability test (in general), we propose an approximation algorithm MILP-ϵ based on the supply bound function. The number of time interval lengths is bounded by a logarithmic function of the task system parameters. We prove that the MILP-ϵ is a sufficient feasibility test and the speed-up factor is $1 + \epsilon$, with respect to the exact schedulability test of dGMF-PA tasks under EDF scheduling. Extensive experiments have shown that our algorithms improve the schedulability ratio of task sets when compared to the previous results.

In order to further improve the efficiency of our MILP-based algorithms by considering other optimization techniques that remove the integer requirements, we give a recursive LP-based algorithm. Guided by the concave programming algorithm which closely approximates the exact task demand, the LP-based algorithm runs efficiently in a small number of iteration and yields high schedulability ratio. The advantage of the MILP-ϵ algorithm is that it has a proved small speed-up factor. The advantage of the LP-based algorithms is that they have lower running time complexity.

Our overall contribution of this thesis is to give efficient and effective algorithms that can be used as a general optimization technique for determining parameters in an interactive real-time uniprocessor and distributed systems. We believe that this thesis can apply to many chainlike tasks, e.g., flows in multi-hop networks, messages in CAN-based system message queues, tasks with shared resources, etc. As far as we know, we are the first to use mathematical programming techniques to schedule such complex task systems towards system schedulability. The approximation algorithms based on mathematical programming

are optimized in this thesis so that we believe large-scale systems can also benefit from our methods. Since our methods are universal and flexible using mathematical programming, they are extendable for future work. The tasks with graphical models (e.g., the directed acyclic graph model) can benefit from our methods and the tasks in multi-processor systems can benefit from our methods. Since our methods based on the LP which has polynomial time complexity, the methods can be further improved to compare with non-mathematical programming methods so that the online scheduling algorithms can benefit from our algorithms.

CHAPTER 9 LIST OF PUBLICATIONS

CONFERENCE

Published

(a) **Bo Peng**, Nathan Fisher, and Thidapat Chantem. Fast and Effective Multiframe-Task Parameter Assignment Via Concave Approximations of Demand, *Proc. of the Euromicro Conference on Real-Time Systems (ECRTS)*, Stuttgart, Germany, 2019.

(b) **Bo Peng**, Nathan Fisher, and Thidapat Chantem. MILP-based Deadline Assignment for End-to-End Flows in Distributed Real-Time Systems [63], *Proc. of the Real-time Networks and Systems (RTNS)*, Brest, France, 2016.

(c) **Bo Peng** and Nathan Fisher. Parameter Adaptation for Generalized Multiframe Tasks and Applications to Self-Suspending Tasks [61], *Proc. of the Embedded and Real-Time Computing Systems and Applications (RTCSA)*, Daegu, Korea, 2016. (**Best Student Paper Candidate**)

(d) **Bo Peng**, Nathan Fisher, and Marko Bertogna. Explicit Preemption Placement for Real-Time Conditional Code, *Proc. of the Euromicro Conference on Real-Time Systems (ECRTS)*, Madrid, Spain, 2014.

JOURNAL

Published

(a) **Bo Peng** and Nathan Fisher. Parameter Adaptation for Generalized Multiframe Tasks: Schedulability Analysis, Case Study, and Applications to

Self-Suspending Tasks [62], *ACM Transactions on Real-Time Systems, 2017.*

WORKSHOP

Published

(a) Masud Ahmed, Camille Williams, **Bo Peng**, and Nathan Fisher. Real-Time Multi-Modal Implementation of a Robotic Toy Car.,*Proc. of the RTSS@Work: Open Demo Session of Real-Time Systems (RTSS)*, Puerto Rico, 2012.

REFERENCES

[1] Gurobi: The state-of-the-art mathematical programming solver. http://www.gurobi.com/.

[2] Mast: Modeling and analysis suite for real-time applications. http://mast.unican.es/.

[3] The openmp api specification for parallel programming. http://www.openmp.org.

[4] E. Ahmed, A. Akhunzada, M. Whaiduzzaman, A. Gani, S. Hamid, and R. Buyya. Network-centric performance analysis of runtime application migration in mobile cloud computing. *Simulation Modelling Practice and Theory*, pages 42 – 56, 2015. Special Issue on Resource Management in Mobile Clouds.

[5] B. Andersson. Schedulability analysis of generalized multiframe traffic on multihop-networks comprising software-implemented ethernet-switches. In *Proceedings of the IEEE International Symposium on Parallel and Distributed Processing*, pages 1–8, April 2008.

[6] S. K. B., R. R. H., and L. R. Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor. *Real-Time Systems*, 2:301–324, 1990.

[7] S. Baruah. The non-cyclic recurring real-time task model. In *Proceedings of the 31st IEEE Real-Time Systems Symposium*, pages 173–182, Nov 2010.

[8] S. Baruah, D. Chen, S. Gorinsky, and A. Mok. Generalized multiframe tasks. *Real-Time Systems*, pages 5–22, 1999.

[9] S. Baruah and N. Fisher. The partitioned multiprocessor scheduling of sporadic task systems. In *Proceedings of the 26th Real-Time Systems Symposium*, pages 321–329,

2005.

[10] S. K. Baruah. Dynamic- and static-priority scheduling of recurring real-time tasks. *Real-Time Syst.*, 24(1):93–128, Jan. 2003.

[11] R. Bettati. *End-to-end scheduling to meet deadlines in distributed system*. PhD thesis, University of Illinois at Urbana-Champaign, 1994.

[12] E. Bini and G. Buttazzo. Measuring the performance of schedulability tests. *Real-Time Systems*, pages 129–154, 2005.

[13] A. Biondi, G. Buttazzo, and S. Simoncelli. Feasibility analysis of engine control tasks under edf scheduling. In *Proceedings of the 27th Euromicro Conference on Real-Time Systems*, pages 139–148, July 2015.

[14] K. Bletsas, N. Audsley, W.-H. Huang, J.-J. Chen, and G. Nelissen. Errata for three papers (2004-05) on fixed-priority scheduling with self-suspensions. *Leibniz Transactions on Embedded Systems*, 5(1):02–1, 2018.

[15] G. Buttazzo, E. Bini, and Y. Wu. Partitioning real-time applications over multicore reservations. *IEEE Transactions on Industrial Informatics*, 7(2):302–315, May 2011.

[16] G. C. Buttazzo. *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*. Springer Publishing Company, Incorporated, 3rd edition, 2011.

[17] G. C. Buttazzo, E. Bini, and D. Buttle. Rate-adaptive tasks: Model, analysis, and design issues. In *Proceedings of the Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1–6, March 2014.

[18] G. C. Buttazzo, G. Lipari, M. Caccamo, and L. Abeni. Elastic scheduling for flexible workload management. *IEEE Transactions on Compututers*, pages 289–302, 2002.

[19] D. Buttle. Real-time in the prime-time. In *Proceedings of the 24th Euromicro Conference on Real-Time Systems*, pages xii–xiii, July 2012.

[20] D. Casini, A. Biondi, G. Nelissen, and G. Buttazzo. Partitioned fixed-priority scheduling of parallel tasks without preemptions. In *2018 IEEE Real-Time Systems Symposium (RTSS)*, pages 421–433. IEEE, 2018.

[21] D. Casini, T. Blaß, I. Lütkebohle, and B. B. Brandenburg. Response-time analysis of ros 2 processing chains under reservation-based scheduling. In *Proceedings of the 31st Euromicro Conference on Real-Time Systems (ECRTS)*, 2019.

[22] T. Chantem, X. Wang, M. Lemmon, and X. Hu. Period and deadline selection for schedulability in real-time systems. In *Proceedings of the Euromicro Conference on Real-Time Systems (ECRTS)*, pages 168–177, July 2008.

[23] C. Chen, J.-J.and Liu. Fixed-relative-deadline scheduling of hard real-time tasks with self-suspensions. In *Proceedings of the Real Time Systems Symposium (RTSS)*, December 2014.

[24] J. Chen. Computational complexity and speedup factors analyses for self-suspending tasks. In *2016 IEEE Real-Time Systems Symposium (RTSS)*, pages 327–338, Nov 2016.

[25] J. Chen, G. Nelissen, and W. Huang. A unifying response time analysis framework for dynamic self-suspending tasks. In *2016 28th Euromicro Conference on Real-Time Systems (ECRTS)*, pages 61–71, July 2016.

[26] J.-J. Chen and B. Brandenburg. A note on the period enforcer algorithm for self-suspending tasks. *CoRR*, abs/1606.04386, 2016.

[27] J. J. Chen, G. Nelissen, and W. H. Huang. A unifying response time analysis framework for dynamic self-suspending tasks. In *Proceedings of the 28th Euromicro Conference on*

Real-Time Systems (ECRTS), pages 61–71, July 2016.

[28] J.-J. Chen, G. Nelissen, W.-H. Huang, M. Yang, B. Brandenburg, K. Bletsas, C. Liu, P. Richard, F. Ridouard, N. Audsley, R. Rajkumar, D. de Niz, and G. von der Brüggen. Many suspensions, many problems: a review of self-suspending tasks in real-time systems. *Real-Time Systems*, 55(1):144–207, Jan 2019.

[29] J.-J. Chen, G. von der Brüggen, W.-H. Huang, and C. Liu. State of the art for scheduling and analyzing self-suspending sporadic real-time tasks. In *2017 IEEE 23rd International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pages 1–10. IEEE, 2017.

[30] F. Dewan and N. Fisher. Efficient admission control for enforcing arbitrary real-time demand-curve interfaces. In *Proceedings of the 33rd IEEE Real-Time Systems Symposium*, pages 127–136, Washington, DC, USA, 2012. IEEE Computer Society.

[31] S. Ding, H. Tomiyama, and H. Takada. Scheduling algorithms for i/o blockings with a multi-frame task model. In *Proceedings of the 13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, Aug 2007.

[32] P. Ekberg, N. Guan, M. Stigge, and W. Yi. An optimal resource sharing protocol for generalized multiframe tasks. *Journal of Logical and Algebraic Methods in Programming*, 84(1):92 – 105, 2015.

[33] P. Ekberg and W. Yi. Uniprocessor feasibility of sporadic tasks remains coNP-complete under bounded utilization. In *Proceedings of the 36th IEEE Real-Time Systems Symposium (RTSS)*, 2015.

[34] P. Ekberg and W. Yi. Fixed-priority schedulability of sporadic tasks on uniprocessors is np-hard. In *2017 IEEE Real-Time Systems Symposium (RTSS)*, pages 139–146. IEEE,

2017.

[35] M. Garey and D. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., NY, USA, 1990.

[36] M. R. Garey, D. S. Johnson, and R. Sethi. The complexity of flowshop and jobshop scheduling. *Math. Oper. Res.*, 1(2):117–129, May 1976.

[37] S. Hong, T. Chantem, and X. S. Hu. Meeting end-to-end deadlines through distributed local deadline assignments. In *Proceedings of the IEEE 32nd Real-Time Systems Symposium (RTSS)*, pages 183–192, Nov 2011.

[38] S. Hong, T. Chantem, and X. S. Hu. Local-deadline assignment for distributed real-time systems. *IEEE Transactions on Computers*, 64(7):1983–1997, July 2015.

[39] W.-H. Huang and J.-J. Chen. Self-suspension real-time tasks under fixed-relative-deadline fixed-priority scheduling. In *Proceedings of the Design, Automation, and Test in Europe (DATE)*, March 2016.

[40] P. Jayachandran and T. Abdelzaher. Delay composition in preemptive and non-preemptive real-time pipelines. *Real-Time Syst.*, 40(3):290–320, Dec. 2008.

[41] J. Kim, B. Andersson, D. d. Niz, J.-J. Chen, W.-H. Huang, and G. Nelissen. Segment-fixed priority scheduling for self-suspending real-time tasks. Technical report, Technical Report CMU/SEI-2016-TR-002, CMU/SEI, 2016.

[42] J. Kim, B. Andersson, D. d. Niz, and R. R. Rajkumar. Segment-fixed priority scheduling for self-suspending real-time tasks. In *Proceedings of the 34th Real-Time Systems Symposium*, pages 246–257, Dec 2013.

[43] S. Li, S. Rubini, F. Singhoff, and M. Bourdelles. Applying holistic schedulability tests to industrial systems: Experience and lessons learned. In *Proceedings of the*

5th International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems, volume 200, 2014.

[44] G. Lipari and E. Bini. On the Problem of Allocating Multicore Virtual Resources to Real-Time Task Pipelines. In *4th Workshop on Compositional Theory and Technology for RealTime Embedded Systems*, Nov 2011.

[45] C. Liu and J. H. Anderson. Task scheduling with self-suspensions in soft real-time multiprocessor systems. In *Proceedings of the 30th IEEE Real-Time Systems Symposium*, pages 425–436, Dec 2009.

[46] C. Liu and J. H. Anderson. Suspension-aware analysis for hard real-time multiprocessor scheduling. In *Proceedings of the 25th Euromicro Conference on Real-Time Systems*, pages 271–281, July 2013.

[47] J. Liu. Real-Time Systems. *Prentice Hall*, 2000.

[48] M. Liu, M. Behnam, and T. Nolte. Schedulability analysis of multi-frame messages over controller area networks with mixed-queues. In *Proceedings of the 18th Emerging Technologies Factory Automation (ETFA)*, pages 1–6, Sept 2013.

[49] R. M. R. Garey, D. S. Johnson. The complexity of flowshop and jobshop scheduling. *Mathematics of Operations Research*, 1(2):117–129, 1976.

[50] J. Mäki-Turja and M. Nolin. Efficient implementation of tight response-times for tasks with offsets. *Real-Time Systems*, 40(1):77–116, 2008.

[51] S. Matic and T. A. Henzinger. Trading end-to-end latency for composability. In *Proceedings of the 26th IEEE International Real-Time Systems Symposium (RTSS)*, pages 12 pp.–110, Dec 2005.

[52] A. Mok and D. Chen. A multiframe model for real-time tasks. In *Proceedings of the 17th IEEE Real-Time Systems Symposium*, pages 22–29, Dec 1996.

[53] N. Moyo, E. Nicollet, F. Lafaye, and C. Moy. Real time scheduling analysis for DSP base band processing in multi-channel SDR set. In *Proceedings of the SDR Forum Technical Conference*, Washington, United States, Dec. 2009.

[54] N. Moyo, E. Nicollet, F. Lafaye, and C. Moy. On schedulability analysis of non-cyclic generalized multiframe tasks. In *Proceedings of the 22nd Euromicro Conference Real-Time Systems (ECRTS)*, pages 271–278, July 2010.

[55] G. Nelissen, J. Fonseca, G. Raravi, and V. Nelis. Timing analysis of fixed priority self-suspending sporadic tasks. In *Proceedings of the 27th Euromicro Conference on Real-Time Systems (ECRTS)*, pages 80–89, July 2015.

[56] G. Nelissen, J. Fonseca, G. Raravi, and V. Nélis. Errata: Timing analysis of fixed priority self-suspending sporadic tasks. *Technical Report CISTER-TR-170205, CISTER, ISEP, INESC-TEC*, 2017.

[57] Y. Nimmagadda, K. Kumar, Y. Lu, and C. Lee. Real-time moving object recognition and tracking using computation offloading. In *Proceedings of the IEEE/RSJ Intelligent Robots and Systems (IROS 2010)*, Oct 2010.

[58] J. C. Palencia and M. G. Harbour. Schedulability analysis for tasks with static and dynamic offsets. In *Proceedings of the 19th IEEE Real-Time Systems Symposium*, pages 26–37, Dec 1998.

[59] J. C. Palencia and M. G. Harbour. Offset-based response time analysis of distributed systems scheduled under edf. In *Proceedings of 15th Euromicro Conference on Real-Time Systems*, pages 3–12, July 2003.

[60] R. Pellizzoni and G. Lipari. Improved schedulability analysis of real-time transactions with earliest deadline scheduling. In *Proceedings of the 11th IEEE Real Time and Embedded Technology and Applications Symposium*, pages 66–75, March 2005.

[61] B. Peng and N. Fisher. Parameter adaptation for generalized multiframe tasks and applications to self-suspending tasks. In *Proceedings of the 22nd Embedded and Real-Time Computing Systems and Applications (RTCSA)*, August 2016.

[62] B. Peng and N. Fisher. Parameter adaptation for generalized multiframe tasks: schedulability analysis, case study, and applications to self-suspending tasks. *Real-Time Systems*, 2017.

[63] B. Peng, N. Fisher, and T. Chantem. Milp-based deadline assignment for end-to-end flows in distributed real-time systems. In *Proceedings of the 24th International Conference on Real-Time Networks and Systems*, RTNS '16, pages 13–22, New York, NY, USA, 2016. ACM.

[64] A. Rahni, E. Grolleau, M. Richard, and P. Richard. Feasibility analysis of real-time transactions. *Real-Time Systems*, 48(3):320–358, 2012.

[65] R. Rajkumar. Dealing with suspending periodic tasks. 1991.

[66] F. Ridouard, P. Richard, and F. Cottet. Negative results for scheduling independent hard real-time tasks with self-suspensions. In *Proceedings of the 25th Real-Time Systems Symposium*, pages 47–56, Dec 2004.

[67] J. M. Rivas, J. J. Gutiérrez, J. C. Palencia, and M. G. Harbour. Schedulability analysis and optimization of heterogeneous edf and fp distributed real-time systems. In *Proceedings of the 23rd Euromicro Conference on Real-Time Systems (ECRTS)*, pages 195–204, July 2011.

[68] J. M. Rivas, J. J. Gutiérrez, J. C. Palencia, and M. G. Harbour. Deadline assignment in edf schedulers for real-time distributed systems. *IEEE Transactions on Parallel and Distributed Systems*, Oct 2015.

[69] J. Schlatow and R. Ernst. Response-time analysis for task chains in communicating threads. In *Proceedings of the 2016 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 1–10. IEEE, 2016.

[70] S. Sivaraman and M. M. Trivedi. Looking at vehicles on the road: A survey of vision-based vehicle detection, tracking, and behavior analysis. *IEEE Transactions on Intelligent Transportation Systems*, 14(4):1773–1795, Dec 2013.

[71] M. Stigge, P. Ekberg, N. Guan, and W. Yi. The digraph real-time task model. In *Proceedings of the 17th IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 71–80, April 2011.

[72] M. Stigge and W. Yi. Graph-based models for real-time workload: A survey. *Real-Time Syst.*, 51(5):602–636, Sept. 2015.

[73] J. Sun, N. Guan, Y. Wang, Q. He, and W. Yi. Real-time scheduling and analysis of openmp task systems with tied tasks. In *2017 IEEE Real-Time Systems Symposium (RTSS)*, pages 92–103. IEEE, 2017.

[74] K. Tindell and J. Clark. Holistic schedulability analysis for distributed hard real-time systems. *Microprocessing and Microprogramming - Parallel processing in embedded real-time systems.*, 40(2-3):117–134, Apr. 1994.

[75] A. Toma and J. J. Chen. Server resource reservations for computation offloading in real-time embedded systems. In *Proceedings of the 11th IEEE Symposium on Embedded Systems for Real-time Multimedia*, pages 31–39, Oct 2013.

[76] G. von der Brüggen, W.-H. Huang, and J.-J. Chen. Hybrid self-suspension models in real-time embedded systems. In *2017 IEEE 23rd International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pages 1–9. IEEE, 2017.

[77] G. von der Brüggen, W.-H. Huang, J.-J. Chen, and C. Liu. Uniprocessor scheduling strategies for self-suspending task systems. In *Proceedings of the 24th International Conference on Real-Time Networks and Systems*, RTNS '16, pages 119–128, New York, NY, USA, 2016. ACM.

[78] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Mueller, I. Puaut, P. Puschner, J. Staschulat, and P. Stenström. The worst-case execution-time problem—overview of methods and survey of tools. *ACM Trans. Embed. Comput. Syst.*, 7(3):36:1–36:53, May 2008.

[79] W.Liu, J.-J. Chen, A. T. nd T.W. Kuo, and Q. Deng. Computation offloading by using timing unreliable components in real-time systems. In *Proceedings of the 51st Design Automation Conference (DAC)*, pages 1–6, June 2014.

ABSTRACT

PARAMETER ASSIGNMENT AND SCHEDULABILITY ANALYSIS FOR REAL-TIME MULTIFRAME TASK SYSTEMS

by

BO PENG

August 2019

Advisor: Dr. Nathan Fisher

Major: Computer Science

Degree: Doctor of Philosophy

Schedulability analysis has been considered as one of the most important subjects in real-time systems. Schedulability analysis decides whether all tasks work correctly and safely in a system. For example, the schedulability analysis of an Air Traffic Control (ATC) system should ensure that all airplanes do not have conflicts on departure lanes and are scheduled on time. In a modern car system, it has been shown that there are more than one hundred engine control units (ECUs), and more than twenty million lines of code in a typical modern car [19]. The scheduling of such complex systems is required to be well developed. As more sensors and functions (e.g., self-driving) will be added in a car, the scheduling of a car system faces more challenges that are caused by the dependent behaviors of functions, the suspending behaviors of functions, and randomness of parameters [19]. We tackle these challenges in this thesis, and we believe that the scheduling of similar large systems can also benefit from the techniques in this thesis.

The input of schedulability analysis is the information of task parameters such as execution times, deadlines, periods, etc. Parameters in a real-time system are typically

immutable and assigned before the launch of schedulability analysis. Such immutable parameters lack flexibility and may lead to the failure of schedulability analysis. In order to tackle this problem, we let a subset of parameters be flexible to be chosen, specifically in tasks each of which contains dependable subtasks/frames. In this thesis, we introduce new flexible models GMF-PA (the generalized multiframe model with parameter adaptation) and dGMF-PA (the distributed GMF-PA) which let frame periods and deadlines be flexible to be chosen under certain constraints.

The GMF-PA and dGMF-PA models generalize the GMF model which extends the sporadic task model and multiframe task model. Each frame in the GMF model contains an execution time, a relative deadline, and a period (minimum inter-arrival time). These parameters are fixed after task specification time in the GMF model. However, systems such as multimedia and adaptive control systems may be overloaded and no longer stabilized when the task parameters in such systems are not flexible. In order to address this problem, task deadlines and periods may change to alleviate temporal overload, for example in the parameter adaptation and elastic scheduling model.

Our GMF-PA (dGMF-PA) model allows frame periods and deadlines to be flexible in arbitrary (constrained) -deadline systems. A necessary schedulability test based on mixed-integer linear programming (MILP) is given to check the schedulability under EDF scheduling and optimally assign frame deadlines and periods at the same time. We also prove that the test is a sufficient and necessary schedulability test when task parameters must be integers. A MILP-based approximation algorithm is also deployed to reduce computational running time and is a sufficient schedulability test in general. The speed-up factor of our approximation algorithm is $1 + \epsilon$ where ϵ can be arbitrarily small, with respect to the exact

schedulability test of GMF-PA (dGMF-PA) tasks under EDF scheduling.

We also present a pseudo-polynomial linear programming (LP)-based heuristic algorithm guided by a concave approximation algorithm to achieve a feasible parameter assignment at a fraction of the time overhead of the MILP-based approach. The concave programming approximation algorithm closely approximates the MILP algorithm, and we prove its speed-up factor is $(1+\delta)^2$ where $\delta > 0$ can be arbitrarily small, with respect to the exact schedulability test of GMF-PA tasks under EDF. The LP-based heuristic algorithm takes shorter running time than the MILP-based heuristic algorithm, but the MILP-based heuristic algorithm has a lower speed-up factor in general.

In uniprocessor systems, we apply the GMF-PA model to self-suspending tasks. By extending the work on scheduling self-suspending tasks, we remove the assumption that deadlines are fixed after system specification time in self-suspending tasks, and the system is extended from constrained-deadline systems to arbitrary-deadline systems. We have done extensive experiments to show that the schedulability ratio is improved using our techniques in our GMF-PA model. We also analyze a case study on a robot car to show the effectiveness of the algorithms.

In distributed systems, we apply the dGMF-PA model to transactions (end-to-end tasks). By applying our algorithms on scheduling transactions in distributed systems, the schedulability ratio is improved compared to state-of-the-art algorithms. Since our parameter assignment is jointly considered with schedulability analysis, this combined technique dominates the previous parameter assignment algorithms based on trial and error.

AUTOBIOGRAPHICAL STATEMENT

Bo Peng received his Bachelor's degree in Electrical Engineering in 2012 from Xidian University, Xi'an, Shaanxi, China. He received a Master's degree in Computer Science in 2014 from Wayne State University, Detroit, Michigan, USA. His research interests include real-time scheduling, mathematical optimization, distributed systems, algorithm design and analysis, embedded systems, etc.