REAL-TIME CALCULUS: REVISIT AND IMPROVEMENT

YUE TANG

PhD

The Hong Kong Polytechnic University

2020

The Hong Kong Polytechnic University

Department of Computing

Real-Time Calculus: Revisit and Improvement

Yue Tang

A Thesis Submitted in Partial Fulfillment of

the Requirements for the Degree of

Doctor of Philosophy

August 2019

CERTIFICATE OF ORIGINALITY

ABSTRACT

Real-time systems widely exist in our daily life, e.g., avionics, automotive electronics, wireless communications. In a real-time system, the correctness of the system behavior depends not only on the logical results of the computations, but also on the physical instant at which these results are produced. Satisfying the time constraints of a real-time system is equally or even more important than other aspects of performance requirements, thus developing analysis techniques to determine the whether time constraints are satisfied deserves the most concern. As a real-time performance analysis framework, Real-Time Calculus is proved to be appropriate for networked real-time systems, providing the sufficiently expressive and effective analysis for timing behaviors. This thesis revisits Real-Time Calculus, improves the analysis of fundamental abstract components and proves the correctness of basic properties, thus improving the overall timing analysis of real-time systems under the Real-Time Calculus framework. In detail, the work in this thesis includes

(1) Improving the analysis of GPC (Greedy Processing Component), which is a fundamental abstract component in Real-Time Calculus and mainly applied for modeling priority-based resource arbitration. The improvement involves two aspects: First, this thesis revises the misunderstanding part in the original proof of calculating output curves in GPC and confirms the correctness of existing calculations for output curves of GPC. Second, two different methods are proposed to derive more precise output arrival curves of GPC, thus improving the analysis preciseness of GPC.

(2) Improving the analysis of AND connector, another widely used abstract component for synchronization operations. This thesis first identifies a problem in the existing analysis of AND connector that may lead to negative values in the output curves, and presents corrections to the problem. Then the existing dual-input AND connector is generalized to

synchronize more than two input event streams.

(3) Proving the Pay-Burst-Only-Once property in Real-Time Calculus. Pay-Burst-Only-Once is one of the most important properties in Network Calculus, the root of Real-Time Calculus. Naturally, people would expect the Pay-Burst-Only-Once property to also hold in Real-Time Calculus since Real-Time Calculus builds up on and shares many similarities with Network Calculus. In fact, existing work has used it in some performance analysis problems. Unfortunately, the Pay-Burst-Only-Once property has never been proved in Real-Time Calculus. There are even some results seeming to be against the Pay-Burst-Only-Once property in Real-Time Calculus. As a result, it leaves an important open problem to find out whether the Pay-Burst-Only-Once property holds in RTC. This thesis answers this problem by proving that the property indeed holds.

Besides, this thesis improves global EDF scheduling by integrating analysis techniques of Network Calculus and real-time scheduling theory. Different from most existing analysis techniques analyzing sporadic tasks for global EDF, this thesis considers bursty tasks which have more general arrival patterns. In detail, shapers are adopted to eliminate burst in original system inputs and generate sporadic job sequences, and then the delay bound of each task is calculated. To further improve the proposed approach, a heuristic algorithm is designed to make as more tasks as possible to meet their deadlines by adjusting settings of shapers.

All the above theoretical results are implemented in Real-Time Calculus Toolbox and experiments show that the proposed methods and algorithms can lead to improvement of precision and efficiency.

PUBLICATIONS ARISING FROM THE THESIS

1. **Yue Tang**, Nan Guan, Weichen Liu, Linh Thi Xuan Phan and Wang Yi, “Revisiting GPC and AND Connector in Real-Time Calculus”, in *Proceedings of the 38th IEEE Real-Time Systems Symposium (RTSS)*, Paris, France, Dec. 5-8, 2017. CCF-A conference.

2. **Yue Tang**, Yuming Jiang, Xu Jiang and Nan Guan, “Improving Multiprocessor Real-Time Systems with Bursty Inputs under Global EDF using Shapers”, in *Proceedings of the 22nd IEEE International Symposium On Real-Time Computing (ISORC)*, Valencia, Spain, May 7-9, 2019. Best Paper Award.

3. **Yue Tang**, Yuming Jiang, Xu Jiang and Nan Guan, “Pay-Burst-Only-Once in Real-Time Calculus”, in *Proceedings of the 25th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, Hangzhou, China, Aug. 18-21, 2019.

4. **Yue Tang**, Yuming Jiang and Nan Guan, “Improving the Analysis of GPC in Real-Time Calculus”, in *Symposium on Dependable Software Engineering,Theories, Tools and Applications (SETTA)*, Shanghai, China, Nov. 27-29, 2019.

ACKNOWLEDGEMENTS

First and foremost, I want to express my gratitude to my supervisor, Dr. Guan Nan, without whom I would not have the chance to study at PolyU. Thanks for always giving me the most instructions ever since my postgraduate period. Thanks for not giving me up when my performance was really not satisfying. Thanks for always offering me great opportunities at the crossing of my life.

I must acknowledge Dr. Jiang Yuming at Norwegian University of Science and Technology for his guidance. I am really impressed by his patience, which gives me much encouragement. I feel the real, strong inner enthusiasm for research from him.

I also express my gratitude to the members of the research group in PolyU – Du He, Zhang Wei, He Qingqiang, Liu Di. Special thanks to Jiang Xu, who not only gives me much guidance in research (although not so patient and easily annoyed) but teaches me important lessons in life. Special thanks to Yang Tao, who helps me a lot with programming and shares with me the cute pictures of his little daughter.

Thanks to Zhang Ping, my best friend in Hong Kong. She is always by my side when I need help. I can not imagine my life in Hong Kong without her.

Last but not the least, I want to thank my parents. They have supported all my choices and they are my backup force. Each time I meet difficulty in life, they tell me not to worry and they will always support me. Now your little daughter has grown up, do not worry so much about me any more !

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1
INTRODUCTION

1.1 Real-Time systems and analysis

Real-time systems widely exist in our daily life, e.g., avionics, automotive electronics, wireless communications. In a real-time system, the correctness of the system behavior depends not only on the logical results of the computations, but also on the physical instant at which these results are produced. When real-time systems have interactions with external entities, e.g., triggered by the physical world and give response, they must react within a limited length of time intervals, which are the timing requirements and constraints they must satisfy. Violating the timing constraints may cause severe consequences even huge property damage or loss of human life. For example, when crash happens, the airbag system must detect the accidents and deploy the airbag in a timely manner, else passenger safety will be affected adversely.

Satisfying the timing constraints of a real-time system is equally or even more important than other aspects of performance requirements, thus developing techniques to determine the whether timing constraints are satisfied deserves the most concern. The timing analysis of real-time systems is never an easy job, and it becomes more difficult with distributed, parallel and heterogeneous designs, in which complex inputs, resource sharing, interferences among tasks and different arbitration strategies bring significant challenges to the analysis process. One possible solution is exhaustive testing. However, it is not applicable for complex and large-scale systems due to infinite or large possible execution scenarios. Instead, effective and efficient analysis techniques are strongly required.

An effective and efficient framework applicable to complex real-time systems must satisfy the following requirements:

(1) Correctness. This is the most fundamental requirement for an analysis framework. A direct consequence of incorrect analysis is that timing constraints are not guaranteed to satisfy, which may lead to operation failure, system crash and even more severe consequences such as loss of human life. For example, if the response time of an airbag control system is actually larger than the required threshold value, but is calculated to be smaller than that after incorrect analysis, then when the system is put into use, it will be a potential threaten and the safety of occupants will be severely affected when accident happens.

(2) Flexibility. Complex real-time systems are always dynamic, including network dynamic, e.g., changing network topology, switching of computation or transmission node, and input dynamic, e.g., changing users, applications and load. The analysis framework must be flexible to deal with dynamic characteristics. If the structure of framework is relatively fixed and difficult to modify, then the system framework needs to be rebuilt each time the system changes, which is time-consuming and fallible. Efficient interfaces among different modules should be maintained to implement the flexibility of system framework. For example, if the outputs of one module are the inputs for another module, then the involved inputs and outputs should be modeled in a unified manner, otherwise these two modules can not be connected and the target networked systems can not be analyzed as a whole.

(3) Precision. Precise analysis eliminates pessimism and provides the closest description of system behavior. During the design stage, precision is the basis for theoretical improvement such as fault detection and model optimization. Precise performance analysis has also profound practical significance. For example, imprecise analysis results may lead to deploying more processors than needed on chips, which not only causes resource waste and increases production cost but increases temperature rise and accelerates the component aging, damaging the benefits of both developers and users.

(4) Efficiency. The timing analysis must be at least conducted in acceptable time. The shorter the analysis time is, the better it is for the design and refinement of the whole system. For systems that require run-time analysis, the analysis efficiency is more critical to guarantee the overall operation of the system.

(5) Generality, which can be detailed in three aspects.

Inputs. First, various arrival patterns must be modeled. Since real-time systems interact with varieties of external entities, the inputs are of different types and contain many different arrival patterns. The model of inputs must possess sufficient expressiveness to deal with this kind of variety. Further, uncertain and uncontrolled factors in physical worlds cause the actual arrival patterns to be irregular and unpredictable, which requires the input model to correctly describe the uncertainty and variation. Second, multiple inputs and their inner relations should be considered and modeled. Thousands of entities may be involved with the operation of a real-time system at one time, and the number of inputs is always more than one, which must be considered in designing the input model of the analysis framework. Further, inner correlations exist among multiple inputs and appropriate modeling of correlations is required. For example, when a parallel task is executed on multiprocessors, the workload executed on each processor must confirm to the precedence constraints in the original structure of the task. If this type of correlation is ignored, then the analysis results may be incorrect.

Resources. In distributed and heterogeneous systems, several applications are deployed on one processor and one application can execute on different processors. As a result, different applications compete for and share resources, and the resource allocation no longer fits into simple specific patterns such as TDMA. So the analysis framework must be general enough to describe different complex allocation strategies and model the occupied resources for each individual task and application.

Execution-time characteristics. In real-time systems, many different components are designed and deployed to process the inputs, and the way in which inputs are processed by these components varies. For example, how many resources are used to process one input event, whether the event will be processed as soon as it arrives, or it will be processed until some of its subsequent events arrive. It is impractical to design a specific framework for each different module, and a general framework that can cover these features is needed.

1.2 Why RTC?

The strong requirements for satisfying timing constraints in real-time systems generate intense demands for precise and efficient timing analysis techniques. One possible method for timing analysis is simulation. The main advantage of simulation-based techniques is the large modeling scope, since various dynamic and complex interactions can be taken into account. However, most simulation-based performance estimation methods suffer from insufficient corner case coverage. As a result, formal analytic methods must be adopted to precisely predict the timing performance of real-time systems.

Existing formal analytic methods are mainly developed in two directions: (1) model checking and (2) real-time scheduling theory.

When adopting model checking to evaluate the performance of a real-time system, automata-based models such as timed automata [5] and temporal logics [32, 60] are used to the system behaviors, then exhaustive and automatic checking is conducted to check whether this model meets a given specification (e.g., absence of deadlocks and similar critical states that can cause the system to crash). Thanks to the strong expressiveness of automata-based models, complex system behaviors and various properties can be modeled and checked. However, this method suffers a lot from low efficiency, which is caused by its inherent high computation complexity. As a result, model checking can only be applied in the analysis of small-scale systems.

In real-time scheduling theory, system workload are described by task models of different abstraction levels, and dedicated techniques are developed to address a few specific problems (e.g., bounding response time and detection of deadline miss). Most of the analyzed task models are rather simple, such as the periodic/sporadic task models (A brief introduction of real-time task models are in Chapter 2). The direct benefit of simple task models is higher analysis efficiency. However, there are still some deficiencies that limit the applications of real-time scheduling theory to the analysis of complex real-time systems.

(1) Insufficient expressiveness. Although most of the analysis targeting simple task models is efficient, it pays. The expressiveness can only support the modeling of simple and

regular system behaviors and many timing behaviors in real-time systems are too complex to be characterized by these simple task models. For example, in the transmission of data flows, it is common that some bits are blocked at one node due to hardware breakdown or lack of bandwidth, and subsequent bits are accumulated at this node. When the node resumes transmitting, these accumulated bits will be emitted simultaneously, which is called 'burst'. However, this extremely common phenomenon can hardly be modeled by all widely used task models. Moreover, the task models considered for multiprocessor scheduling are even more simple, and most of them are periodic and sporadic task models. Besides, it is difficult for scheduling analysis techniques to be extended from simple to complex tasks. Consequently, real-time scheduling analysis can not cope with complex system behaviors.

(2) Lack of interfaces. Most of the work in scheduling analysis focuses on how to calculate the end-to-end delay under some specific scheduling strategies, and the timing behaviors of the output event sequence seem to be neglected and receive little attention. However, the output event sequences are as equally important as the delay analysis in a networked system. Only when information about the output sequences is obtained can it be possible to develop interfaces to connect different modules. Take multiprocessor scheduling as an example, where the execution of input events is not limited to one processor and the interchange of information among processors accounts for a lot in system performance. Lack of information about the output sequences makes it hard to establish relationships between one processing node and another, thus bringing difficulty to model networked structures.

(3) Lack of compatibility. In real-time scheduling analysis, the analysis techniques under various scheduling strategies greatly differentiate from each other. It is hard to establish a unified framework compatible with all kinds of scenarios and the hardiness significantly grows with the complexity of systems. As a simple example, one method to decide the schedulability on uniprocessors under fixed-priority strategy is to check the relative relation between the response time and relative deadline of each task, while the schedulability under Earliest Deadline First (EDF) scheduling is decided by comparing the total utilization of the task sets with 1. It is difficult to design a schedulability test applicable to both these two scheduling strategies.

Real-Time Calculus (RTC) can be viewed as a large extension along the line of real-time scheduling theory to analyze distributed systems. RTC uses variability characterization curves to model workload and resource, and analyzes workload flows through a network of computation and communication resources to derive performance characteristics of the system based on max plus/min plus algebra.

RTC has relatively high expressiveness. First, input workload and resources are modeled with both upper and lower curves, which supports tighter computation of the output curves and describes the worst-case and best-case of system performance. Any kind of workload and resources can be modeled in the RTC framework since there are no limitations to the shapes of upper and lower curves. Moreover, the input models in RTC are generalization of real-time task models. As an example, the workload generated by a periodic task can be modeled as a regular stair-based curve, and that of a DRT task can also be modeled with a little modification of curve shapes [42]. Second, RTC supports modeling different execution-time characteristics. The relations between the inputs and outputs at specific processing components are modeled as abstract components, and calculations for outputs of these abstract components are developed based on the semantics of related concrete components. As a direct consequence of introducing abstract components, inputs and outputs of one component are of unified formats, which greatly facilitate the network analysis. Further, RTC allows to model arbitrary connection of computation/communication components, which makes it easy to adapt to dynamic networks.

On the other hand, the analysis with RTC is relatively efficient compared with model checking method, and much work has been conducted to improve its analysis efficiency. For example, finitary RTC (component-based and operator-based) has been proposed to shorten the analyzed interval length for obtaining output curves in abstract components, which greatly accelerate the analysis of RTC network.

Due to these advantages mentioned above, RTC has drawn considerable attention from both academia and industry in recent years, and has been successfully applied to solve many realistic problems. As an example, RTC is applied for modeling control systems in automotive cars, where arrival curves model various input workload such as the sampling data through I/O interfaces and service curves model the available resource such as the computation resource on ECUs and communication resource on buses. The composition of abstract components models the workload processing behavior on different modules (e.g., ECUs, bus, FPGA), the allocation of resources and processing order of workload. The analysis of the modeled systems is based on max/min-plus algebra. The upper and lower bound of processed events and unused resources in different lengths of time intervals are obtained by calculating the output curves at each abstract component, and the end-to-end delay of the complete system is equivalent to that of abstract components corresponding to modeled modules.

1.3 Problems of existing RTC

RTC has been widely studied and applied since it was proposed. However, it is still not perfect and there is still space for further improvement.

(1) The theory foundation of RTC analysis framework is not complete. Since NC is the root of RTC, people sometimes directly borrow properties from NC and apply them in RTC framework without proof, taking for granted that these properties will certainly hold. As an example, the Pay-Burst-Only-Once, a basic property for obtaining tighter delay bounds in networked systems, has been applied in RTC without direct proof of its correctness. Moreover, there is some work seemingly against the correctness of Pay-Burst-Only-Once in RTC. As a result, it is essential and urgent to answer the question whether applying Pay-Burst-Only-Once in RTC to obtain tighter delay bounds is correct. If the answer is 'no', then the delay bound analysis of real-time systems will be greatly influenced.

(2) The analysis results are pessimistic. In RTC, calculation for outputs of abstract components is artificially designed based on the processing behaviors in real applications and it accounts for a lot in the overall performance analysis. However, after comparing the actual execution sequence with the calculated results, it is found that calculated outputs of some basic abstract components are not precise. The pessimism not only influences the analysis at

its direct successor component, but gradually spreads to all of the connected components in the network, and the extent of pessimism accumulates, degrading the analysis precision of the whole network.

(3) The application of RTC and its integration with other analysis techniques are limited. Benefiting from its expressiveness and analysis efficiency, RTC analysis framework has the potential to be applied to many different networked systems such as multi-processor systems on chip (MPSoC), Network on Chips (NoC) and Time-Sensitive Networking (TSN). However, most existing work of RTC mainly focuses on uniprocessors and techniques that support the extension of RTC have hardly been developed. On the other hand, although RTC is closely related to real-time scheduling theory, almost no work has completed the integration of these two frameworks, missing the potential advantages brought by the integration.

1.4 Contributions

Based on deep research about RTC basics and its existing problems, this thesis revisits RTC, improves the analysis of fundamental abstract components and proves the correctness of basic properties, thus improving the overall timing analysis of real-time systems under the RTC framework. In detail, this thesis completes the following work:

(1) Improving the analysis of GPC (Greedy Processing Component), which is a fundamental abstract component in RTC and mainly applied for modeling priority-based resource arbitration. The improvement involves two aspects:

First, this thesis revises the original proof of calculating output curves in GPC. Although the calculation of output curves of GPC has been widely used, its proof for calculation is not as elegant as supposed. Actually, the deduction of relations between two important parameters is not fully explained, which leads to misunderstanding and even doubt about the correctness of the calculation. This thesis complements the missing parts and verifies that the existing calculations are correct.

Second, this thesis proposes two methods to derive more precise output arrival curves for GPC. In the first method, a concrete component which has the same semantics as the concrete model corresponding to GPC is developed. Then it is proved which parts of resources are used for processing events while others are not used. Then this thesis proposes how to improve the analysis preciseness of output curves with the newly developed model: utilize the remaining service curve to refine the information about how much resource can be actually consumed, and exclude the unused resource when computing the output arrival curves. In the second method, the main idea is to explore the connections between output arrival curves and the number of accumulated events.

(2) Improving the analysis of AND connector, another widely used abstract component for synchronization operations. First, this thesis identifies a problem in the existing analysis of AND connector that may lead to negative values in the output curves, and presents corrections to the problem. Second, AND connector is generalized to support synchronization of more than two input event streams. While real-time systems generally involve several inputs for synchronization, the original AND connector can only deal with two inputs . A straightforward way for the generalization is to model a multi-input AND as several dual-input AND connectors cascaded together. However, this straightforward generalization is both imprecise and inefficient. This thesis presents a more elegant way to generalize AND to multiple inputs, which outperforms the straightforward generalization approach in terms of both precision and efficiency.

(3) Proving the Pay-Burst-Only-Once property in RTC. Pay-Burst-Only-Once property, originated in NC, is an extremely fundamental property and mainly used for the delay bound analysis for a network consisting of many nodes. Since RTC builds on NC and inherits many properties from it, people take for granted that the Pay-Burst-Only-Once property also holds in RTC, and some work based on the RTC framework has applied it for delay bound analysis. However, there has been no work directly proving the correctness of Pay-Burst-Only-Once in RTC. Moreover, there exists work proving that the concatenation property, which is the basis of Pay-Burst-Only-Once property in Network Calculus, does not hold in RTC. Therefore, it leaves an important open problem to find out whether the Pay-Burst-Only-Once property holds in RTC. This thesis proves that Pay-Burst-Only-Once does hold in RTC

by establishing the numerical relations between resource models in RTC and NC. Moreover, this thesis proposes how to calculate the outputs of a network system in a once-for-all way, rather than calculating it node by node.

Besides, this thesis extends NC, the root of RTC, to the analysis of multiprocessor scheduling. In detail, this thesis proposes a new approach to calculate delay bound for a multiprocessor system with bursty tasks under GEDF using shapers. Bursty inputs are modeled as bursty tasks and shapers are deployed for input bursty tasks. Once job sequences generated by bursty tasks enter the system, they first go through corresponding shapers, after which the behaviors of these jobs conform to sporadic patterns. Then these output jobs of shapers go into the scheduler and complete execution. With shapers, existing analysis techniques analyzing sporadic tasks can be applied, and the delay bound of each task is calculated. To further improve the proposed approach, this thesis designs a heuristic algorithm which can increase the number of tasks meeting their deadlines in a task set by adjusting settings of shapers.

1.5 Organization

The rest of this thesis is organized as follows. Chapter 2 briefly introduces representative results for scheduling analysis, including real-time scheduling theory and compositional analysis approaches. Chapter 3 reviews basic knowledge about NC and RTC. The improvement of fundamental components, GPC and AND connector, are shown in Chapter 4 and Chapter 5 respectively. In Chapter 6, the Pay-Burst-Only-Once property is proved to hold in RTC. The integration of analysis techniques in NC and classical scheduling analysis on multiprocessors is considered in Chapter 7. Chapter 8 concludes with a summary of the work presented in this thesis and a discussion of future work.

CHAPTER 2

RELATED WORK

Satisfying time constraints is of great importance in real-time systems, which greatly influences the overall system performance. To provide such guarantees, task models and precise analysis techniques are required to check the timing correctness of the system. Over the years, many different real-time task models and frameworks have been developed, which describe and analyze the timing behaviors of systems in different abstraction levels. This chapter first introduces real-time scheduling theory, including representative task models and the scheduling analysis methods, and then present compositional analysis approaches.

2.1 Real-Time scheduling theory

In real-time scheduling theory, system workload are modeled as simple task sets and specified techniques are developed to evaluate timing performance such as checking the schedulability and calculating response time. In this section, real-time task models and scheduling analysis methods are introduced.

2.1.1 Real-Time task models

Over the years, a large number of real-time task models have been developed to model different workload release patterns. Some model is as simple as a periodic pattern characterized by two parameters, while some of them are as expressive as Turing machine [75]. In general, the simpler the task model is, the easier to analyze it. This subsection briefly reviews some representative real-time task models, and only focuses on their expressiveness aspect.

Periodic/Sporadic task models. The periodic task model was first proposed by Liu and Layland [58], which describes the most basic timing behaviors of tasks in real-time systems [56, 57]. A released task generates an infinite sequence of jobs. A periodic task is characterized by worst-case execution time (WCET) C and period T. The first job of a task may be released at any time, and each subsequent job is released with a fixed separation time T from its predecessor job. Each job requires to execute for at most C time units and all jobs have the same WCET. Each task has a deadline denoting the amount of time within which each job should complete execution after its release. For a periodic task, its deadline equals its period. Since a periodic task is sequential, its jobs may not execute on multiple processors at the same time, i.e., parallelism is forbidden. The timing constraints of a periodic task are relaxed in sporadic task models [63], where the inter-release time between two consecutive jobs can be larger than its period and its deadline can be smaller than, larger than or equal to the period. If the deadline is equal to (smaller than) the period, the task is called implicit (explicit), else it is called an arbitrary task. The utilization of a task is the ratio between its WCET and period, and the utilization of a task set is the sum of utilizations of all its tasks.

Although the periodic/sporadic task models can capture the basic properties of real-time systems, their expressiveness is too limited to characterize the complex behaviors in real applications. To improve the expressiveness, the periodic/sporadic task models are further extended to graph-based real-time task models.

Graph-based real-time task models. One of the representative graph-based real-time task models is the Multiframe (MF) task model [64], which extends the periodic task model by modeling the execution of a sequence of jobs with different WCET. A further extension, the generalized multiframe (GMF) task model [10], not only allows tasks to have different WCETs, but different relative deadlines and minimum inter-released separations between two consecutive jobs. The release job sequence can be expressed by a set of triples (t_i, c_i, v_i), where t_i is the release time of the $i-th$ job v_i and c_i is its WCET. The GMF task model can be expressed by a directed loop graph, where each vertex denotes a job characterized by WCET and deadline, and each edge shows the minimum inter-release time between two jobs. All jobs are connected into a cycle. An example of a GMF task is shown in Figure 2.1. The task releases 4 jobs of different WCET, denoted by v_0, v_1, v_2, v_3, and the execution order of these 4 jobs is shown by the arrow directions of the edges. Suppose the task starts execution with v_0, then one of the possible released job sequences is:$\{(0,2,v_0),(3,3,v_1),(7,2,v_2),(12,2,v_3),(16,2,v_0),(19,3,v_1)\}$.

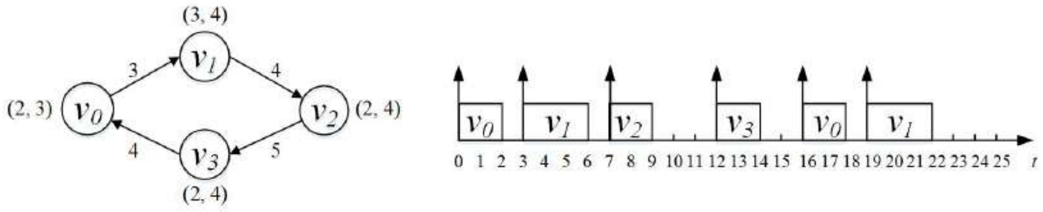


Figure 2.1: An example of GMF task.

A more expressive task model is the digraph real-time (DRT) task model [62, 76–80]. A DRT task is described by a directed graph, where the vertices denote jobs and the edges denote the execution order. There are no further restrictions to limit the expressiveness of the task model and any directed graph is applicable. Any vertex can be the first one to be executed. An example of DRT task and one of its possible job sequence $\{(0,3,v_1),(6,5,v_1),(19,5,v_2)\}$ are shown in Figure 2.2.

Parallel task models. Apart from the above task models for sequential workload, parallel task models are also developed for characterizing parallelism on multiprocessors (e.g. Gang task model [14, 37], parallel synchronous task model [70]), among which DAG parallel task model is one of the most widely analyzed and applied [20, 46, 47, 61, 71].

A DAG parallel task is characterized with $(\overrightarrow{C}, D, T)$, where D and T are relative deadlines and minimum inter-release separations as before. $\overrightarrow{C} = \{C_1, C_2, ..., C_i\}$ records the WCETs of each thread of the task. The DAG parallel task has an internal structure mod-

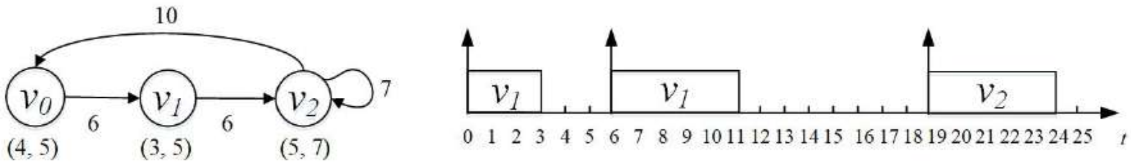


Figure 2.2: An example of DRT task.

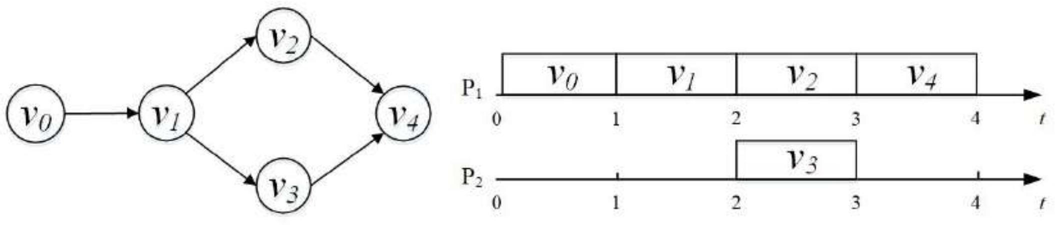


Figure 2.3: An example of DAG task.

eled by a directed acyclic graph, describing the precedence relationships among the threads. Each node represents a thread with worst-case execution time C_i and each edge represents the execution order of the two nodes it connects. A node is eligible for execution only when all its predecessors have completed execution. An example of the DAG parallel task model executed on two processors P_1 and P_2 is shown in Figure 2.3. v_2 and v_3 can execute simultaneously if v_1 completes and v_4 can start execution only when all of its predecessor nodes (v_2 and v_3) complete.

Although the DRT task model and DAG parallel task model are both modeled with directed graphs, they differ from each other in three aspects. First, loops are allowed in DRT task model, but not applicable in the DAG parallel task model. Second, in a DAG parallel task, all the successors of a vertex will be released and executed after the vertex is completed, while in a DRT task, only one of its successors will be executed. Third, the edges in DRT denote the minimum inter-release time of the two vertices connected, while the edges in a DAG task model only denote the relative execution order.

2.1.2 Real-Time scheduling analysis

Given a set of input workload and system resources, a scheduling algorithm must be decided, which specifies when each job can be executed and where it is executed. Scheduling algorithms are designed based on the timing constraints needed to satisfy. Before discussing the exact scheduling algorithms and related analysis techniques, we first introduce some terminologies for easier understanding.

Feasible: A task set is said to be feasible if there exists some way of scheduling and meeting all the deadlines of tasks in the task set.

Schedulable: A task is said to be schedulable on a hardware platform by some algorithm if all jobs can meet their deadlines. A task set is said to be schedulable if all its tasks are schedulable.

Optimal: A scheduling algorithm is said to be optimal if it can correctly schedule every feasible task set for every hardware platform.

In the following, we first introduce scheduling analysis techniques on uniprocessors[1] and then consider multiprocessors.

Analysis of uniprocessor scheduling

We classify the scheduling algorithms into two types and introduce scheduling analysis techniques respectively.

Static priority scheduling algorithms. Tasks are executed in an order that is decided a priori, for example, tasks with shorter periods are assigned higher priorities. All jobs generated by higher-priority tasks have precedence over jobs generated by lower-priority tasks.

A simple feasibility test for implicit deadline tasks was proposed in [58]: a task set is feasible if and only if the total utilization of the task set is no larger than 1. It also proved that assigning higher priorities to tasks with shorter periods generates an optimal algorithm for implicit deadline tasks. Further, a sufficient schedulability test was given which claims that a task set with n tasks is schedulable under rate-monotonic scheduling if $U(\tau) \leq n \times (2^{1/n} - 1)$.

For sporadic task sets with explicit deadlines, the concept 'critical instant' was proposed, which shows that the scenario where all tasks release jobs at the same time generates the worst-case response time for the analyzed task [7]. It is proved that the worst-case response time R_i for task τ_i is the smallest positive solution of the following recurrence

[1]We only introduce the analysis techniques for periodic and sporadic tasks since they are more related to our work.

relation,

$$R = C_i + \sum_{j<i} \lceil \frac{R}{P_j} \rceil \times C_j$$

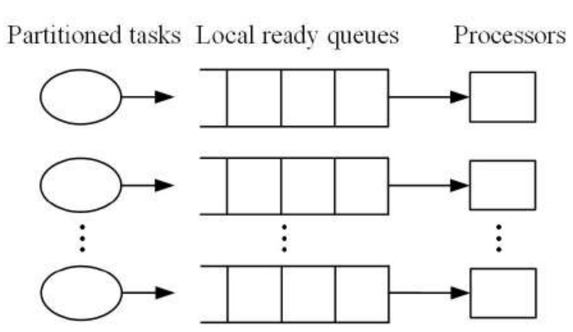
where P_j is smaller than P_i and $\sum_{j<i} \lceil \frac{R}{P_j} \rceil \times C_j$ denotes the interference from a higher-priority task.

Dynamic priority scheduling algorithms. The priorities of tasks can not be decided until execution and the priorities of jobs generated by the same tasks may change during execution. For example, in Earliest Deadline First (EDF) scheduling, the job with shorter absolute deadline has higher priority.

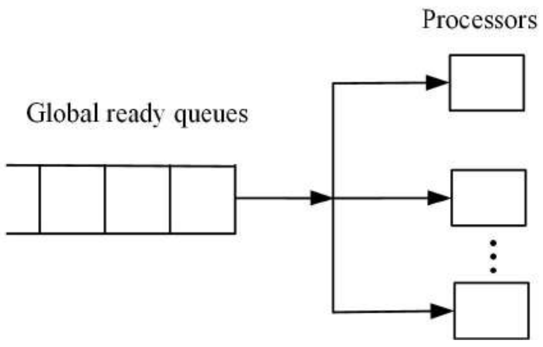
EDF is proved to be optimal for many workload models on uniprocessors. For the scheduling analysis of EDF, demand-bound function was proposed in [11]. The demand-bound function describes the accumulated worst-case execution time of jobs whose release times and deadlines are covered by the considered time interval. A task set is feasible if and only if for any time interval, the value of the accumulated workload does not exceed the length of time interval. Specially for sporadic tasks, a closed-form expression was derived based on the observation that the demand-bound function of periodic tasks are regular step-functions.

Analysis of multiprocessor scheduling

To deal with the increasing demands for system performance, the developers of real-time embedded systems have gradually switched from concentrating on the miniaturization required to increase processor clock speeds, to using multiprocessor platforms. Consequently, the scheduling analysis on multiprocessors has drawn more attention. The scheduling analysis on multiprocessors is not a simple extension of uniprocessor scenarios, and the extra dimension of different processors brings much challenges to it. Compared with scheduling analysis on uniprocessors which has already been reasonably mature, the scheduling analysis on multiprocessors has a late start and there is still great scope to explore, which is reflected by the fact that most scheduling analysis on multiprocessors considers simple task models such as sporadic tasks.



(a) Illustration for partitioned scheduling.



(b) Illustration for global scheduling.

Figure 2.4: Illustration for partitioned and global scheduling.

The scheduling algorithms on multiprocessors fall into two types: partitioned scheduling and global scheduling.

Partitioned scheduling. Under partitioned scheduling, tasks are allocated to the specific processor on which it can execute and no migration is permitted. Each processor acts as a uniprocessor for tasks allocated to it and a separate local ready queue is maintained for storing its ready jobs.The scheduling strategies on different processors can be same or different. An appropriate partitioning algorithm should ensure that for each processor the sum of the utilizations of tasks assigned to it does not exceed the utilization bound of its scheduler. Scheduling under this model is shown in Figure 2.4-(a).

Global scheduling. Under global scheduling, tasks are permitted to migrate from one processor to another and a global ready queue is used for storing ready jobs. At any instant, at most m ready jobs with the highest priority (in the global queue) are scheduled to be executed on the m processors. There are no restrictions about the processors a task may be executed on. Different jobs of a task can be executed on different processors, and a given job can be executed on different processors at different times (Note that a given job can only be executed on one processor at each time.) Figure 2.4-(b) illustrates global scheduling.

Compared with partitioned scheduling, global scheduling has the following advantages [33]:

(1) With global scheduling, there are typically fewer context switches and preemp-

tions are less likely to occur since a task will only be preempted when there are no processors idle [6].

(2) If a task executes for less than its worst-case execution time, then the remaining computation resources will be utilized by all other tasks, while with partitioned scheduling, only tasks allocated to the same processor can use the remaining computation resources.

(3) It is less possible for a task to overrun its worst-case execution time budget on all processors than on one single processor. As a result, global scheduling is likely to suffer less from deadline miss than partitioned scheduling.

(4) Since task allocation algorithms are not required when the set of tasks changes, global scheduling is more appropriate for open and dynamic systems.

In the following, we introduce some work for scheduling analysis on multiprocessors in two directions: schedulabiliy test and response time analysis.

Schedulability test. [8] developed a general strategy for determining the schedulability of sporadic task sets. The outline of this basic strategy is as follows. Suppose that a task τ is not schedulable under global EDF, and there exists one job sequence of τ on which more than one job misses its deadline. Suppose that the time instant when the first job misses its deadline is t_d, then the analysis is conducted over the interval $[t_0, t_d]$, where t_0 is the earliest time instant prior to this job's arrival time (Some conditions must be satisfied for this time instant, for detail, refer to [9]). Then the upper bound on the amount of work needed to be executed over the time interval $[t_0, t_d]$ is computed, and by setting this bound to be large enough to leave less than a job's worst-case execution time in its scheduling window, an unschedulability condition is obtained. There are two types of jobs whose workload needs to be executed during the interval $[t_0, t_d]$: jobs whose arrival time are in the interval, and one additional job that arrives before t_0 but has not completed execution by t_0, which is called a 'carry-in' job. It is proved that not all jobs whose scheduling windows cover t_0 have 'carry-in' workload in $[t_0, t_d]$. Actually, some jobs have completed their execution before t_0 although t_0 is included in its scheduling window.

The analysis in [8] suffers from the disadvantage that it assumes that every task in the considered system has 'carry-in' workload. This assumption brings pessimism when the number of tasks is much larger than the number of processors. This pessimism is improved in [12], which differing from [8] is a generalization of the known exact EDF-schedulability test on uniprocessors.

[12] derived a sufficient schedulability test for sporadic task sets with constrained deadlines scheduled under global EDF. This test adopts the same basic analysis framework as [8] but optimizes the analysis by considering t_0 as some point when at least one of the m processors is idle, which is an earlier time instant than considered in [8] , thus limiting the number of tasks with 'carry-in' workload to $m - 1$. Similar as [8] , the analysis in [12] considers each task τ_k separately, and ensures that τ_k does not miss any deadline. The maximum workload to be executed in $[t_0, t_d]$ is calculated by considering both arrived jobs and 'carry-in' jobs, and the conditions for schedulability test are obtained by ensuring that the length of time interval when each job of all tasks can execute is no less than its worst-case execution time.

While [8] and [12] analyze simple task patterns as sporadic tasks, [55] combines the analysis framework with RTC and focuses on more general workload and resources patterns. A schedulability test was proposed in [55] to check whether the given response-time bounds are satisfied. The conditions for the schedulability test follow the similar idea as [8] and [12]: it first computes the minimum amount of resources provided by all processors over the analyzed interval, and then gets the finite upper bound on the computing demand and the 'carry-in' workload. Finally, a sufficient test is obtained by checking whether for each task the upper bound of workload does not exceed the minimum offered resources.

Delay bound analysis. Another direction for evaluating the timing performance on multiprocessors is to compute the delay bound, which describes the maximum length of duration from task activation to completion.

An innovative technique was introduced in [34] for determining delay bound for global EDF, which has been widely applied to different scheduling algorithms, workload and processor models. The bound derived in [34] is as follows:

For each task τ_i, each job generated by τ_i is guaranteed to complete no later than

$$C_i + \frac{C_{sum} - C_{min}}{m - U_{sum}} + D_i$$

, where C_{sum} and U_{sum} denote the sum of the $m-1$ largest WCET's and utilizations respectively, D_i is the job's relative deadline and C_{min} is the smallest WCET of any task.

However, the delay bound in [34] suffers from pessimism since it calculates one common value for all tasks ($\frac{C_{sum}-C_{min}}{m-U_{sum}}$). The work was improved in [38], where the above mentioned expression is different for each task, thus reducing the pessimism. Further extension was proposed in [54], where the analysis is more general considering arbitrary job priorities, capacity restrictions on certain processors and non-preemptive regions.

More general analysis with the similar idea as [34] was presented in [55], where closed-form expressions of response time bounds are proposed for systems with general inputs.

There are also other work on scheduling analysis on multiprocessors, but they are less related to our work thus omitted. For further details, please refer to [33] and [36].

2.2 Compositional analysis approaches

The consistently increasing complexity of real-time systems raises demands for compositional analysis techniques, where timing analysis results of local components are composed to evaluate the timing performance of the overall system. The process of compositional analysis includes three major parts: implementing local analysis, deriving outputs, and connecting different components. This section introduces two representative compositional analysis approaches, SymTA/S and Real-Time Calculus.

2.2.1 The SymTA/S approach

SymTA/S (Symbolic Timing Analysis for Systems) is a system level analysis framework [2, 44] which supports compositional analysis. SymTA/S can be viewed as a good integration of various real-time scheduling analysis techniques. For each component, the analysis techniques based on the busy window technique [53] are adopted in SymTA/S. As a result, many scheduling analysis techniques are applicable.

The connections of different components are implemented with standard event models and their transformation [68]. First, standard event models are proposed to provide a comparatively general model to describe the timings of both input and output event sequences of components. Three parameters, period P, jitter J and minimum inter-release time D are adopted in standard event models. The most simple model is described by period P, corresponding to the event sequence where successive arrive with an separation time of J . Then the model is extended by introducing jitter, which describes the irregular release of events. Jitter is further limited by D, which constrains the maximum number of released events in any interval. In total, 6 different event models are proposed.

Based on standard event models, event model interfaces (EMIFs) and event adaptation function (EAF) are developed to transform event streams to a different event model [69]. For two connected components where the output event sequences of one component are the inputs for the other component, the event models corresponding to the output event sequences are transformed to another one, which is consistent with the requirements for inputs of the latter component. During the transformation, pessimism may be introduced but the correctness of analysis results is guaranteed. The mechanism of event model transformation can be illustrated by the following example. Suppose a periodic event sequence is scheduled under fixed-priority strategy on a sequence of processors, and all processors require periodic event sequences as inputs for analysis. During the scheduling process on the first scheduler, the original regular arrival patterns are interrupted, and jitter occurs in the output event sequence. As a result, the output event sequence no longer fits into the original event model and it does not satisfy the input requirements of the second processor. In this case, the event transformation functions are adopted to transform the irregular event sequence with jitter to another periodic sequence, and then this new periodic sequence is used as input of the second processor.

SymTA/S also supports synchronization of multi-inputs, which is implemented by AND-activator [45], and the calculation for output event sequences of AND-activator is developed.

2.2.2 Real-Time Calculus

RTC originates from NC and inherits many concepts and properties from it. It is necessary to present related work in NC as a foundation for RTC.

Network Calculus

NC is a theory analyzing queuing systems found in computer networks, whose focus is to provide performance guarantees [21, 59]. NC models traffic and service with arrival and service curves respectively, and performance bounds are derived with min-plus and max-plus algebra. NC has been widely and successfully used for modeling and analysis in networked systems such as in aeronautics industry for AFDX [41].

In the development of NC, two branches have appeared: algebraic NC [17, 19, 49], which derives delay bound based on the composition of operators such as min-plus and max-plus algebra [25], and optimization-based NC, which derives a linear program formulation whose solution bounds the end-to-end delay [16]. This thesis focuses on algebraic NC (called NC for short).

In NC, workload is modeled as arrival curves, which characterizes the upper bound of arrived bits in any time interval of certain length. Compared with the standard event models in SymTA/S, NC can model much more general workload patterns. The service guarantee in NC is described with three types of service curve: minimum service curve, strict service curve and maximum service curve. The definitions and applications of these three different types of service curves are different and the comparison is introduced in [22–24].

The traffic is regulated by greedy shaper, which is a fundamental component in NC. Each greedy shaper is characterized with a shaping curve. When an arrived bit enters a greedy shaper, it emits the bit only when doing this does not violate the time constraints specified by the shaping curve, otherwise the bit will be buffered until the time constraints are satisfied. Similar functions are realized with EMIFs and EAF in SymTA/S, while the greedy shaper supports more general types of output sequences. Greedy shaper has been adopted to improve system performance in real-time systems. In [67], shapers are used to control the traffic of higher-priority tasks so as to reduce interference to lower-priority ones under fixed-priority scheduling on uniprocessors, thus improving the system schedulability. Shapers are also used in EDF scheduling [48], where arrival patterns are changed to reduce peak temperature of chips. However, both of these two work assumes uniprocessors and the analysis on multiprocessors is not considered.

For a network consisting of many nodes, the concatenation property plays an important role in deriving tighter delay bound. The concatenation property proves how to derive the minimum service curve provided for the system based on the minimum service curve for each single node. The first well-known application of the concatenation property is the Pay-Burst-Only-Once property, which avoids considering the analyzed flow's burst at each node it traverses. Another application is the Pay-Multiplexing-Only-Once property [18, 74], which extends the Pay-Burst-Only-Once property to cross-flows and convolves as much service curves as possible before subtracting the cross traffic. The influence of the concatenation property is further explored in [73], which stated that concatenation brings pessimism to delay bound under arbitrary multiplexing. The reason is that the convolution operation is not sensitive to the operation order so that the topological information is ignored, making the burst paid at nodes that the flow does not even traverse.

Real-Time Calculus

RTC originates from NC and is also a queuing theory for performance analysis of real-time networked embedded systems [28, 81].

Compared with NC, RTC supports more precise characterization of real-time systems. First, RTC models input workload and resources with both upper and lower curves, while NC uses only upper curves for workload and lower curves for resources[2]. Second, RTC supports the description of various execution-time characteristics and models them as abstract components, which is less systematic in NC[3]. Moreover, both RTC and NC calculate closed-form analytical bounds for output curves based on max/min plus algebra, which provides a comparatively higher analysis efficiency compared with state-based modeling and analysis techniques such as model checking [29, 40]. However, NC only pays attention to the calculation of completed workload, but RTC considers both completed workload and remaining resources.

Based on the general input workload and closed-form analytical results, RTC supports compositional analysis, where local analysis is performed fo each component to derive the output event/resource models and afterwards the calculated output event models are propagated to the subsequent components. For the performance analysis of networked systems in RTC, the Pay-Burst-Only-Once property has been adopted to derive tighter delay bound, which is first proposed in [82]. Then it is used in [50] for the delay bound calculation under operator-finitary RTC, and also in [31] to minimize the energy consumption for pipelined multiprocessor embedded systems. However, none of these includes the proof of correctness.

There have been previous efforts to improve the analysis accuracy of the RTC theory. The first direction models the system using a state-based model, such as timed automata [3] or event count automata [29], which can then be analyzed using standard verification techniques. These approaches can provide tight bounds; however, as is with verification, they

[2]Here lower curves mean minimum service curve.

[3]Some abstract components in RTC are extension of counterparts in NC, such as greedy shaper [83].

suffer from the state explosion problem. To solve this efficiency issue, prior work has developed interfacing techniques [52,66] that combine RTC with state-based approaches. Beyond state-based approaches, the trajectorial approach has been developed [13] to bound the end-to-end analysis of distributed systems. This approach gives better accuracy than RTC does, but its fixed point computation is also much more expensive than RTC. Within the RTC context, Bouillard [26, 27] has developed a tighter analysis for blind multiplexing and FIFO networks using linear programming. This thesis focuses on improving the output bound of a single component (GPC and multi-input AND), and thus it is orthogonal to the work in [26, 27]. Moy and Altisen [4,65] identified the causality problem in the arrival curves and its solutions. The key insight is that interchanging the information between the upper and lower bounds in the arrival curve may exclude invalid approximation areas. Part of work about GPC in this thesis on a high level shares some similarity idea with their work, but explores the interaction between the arrival and service curves. A hybrid framework was proposed in [51] for analyzing real- time embedded systems combining RTC and Timed Automata. In principle, the timed automata component can be used to model synchronization operations. However, this approach is still limited by the low efficiency for model-checking the timed automata, although the state space has already been significantly reduced comparing with modeling the whole system with a timed automata network.

CHAPTER 3

PRELIMINARIES

This chapter briefly introduces basic background knowledge in NC, RTC and the system model for global EDF scheduling.

3.1 Basics in NC

NC is a theory analyzing queuing systems found in computer networks, whose focus is to provide performance guarantees [21, 59]. NC models traffic and service with arrival and service curves respectively, and performance bounds are derived with min-plus and max-plus algebra. NC has been widely and successfully used for modeling and analysis in networked systems such as in aeronautics industry for AFDX [41].

3.1.1 Curves in NC

In NC, the accumulated number of input bits on a flow from time 0 to t is denoted by input function $R(t)$, which by definition is wide-sense increasing. Correspondingly, the number of output bits during the same interval is denoted by output function $R^*(t)$. The limitation of bits is denoted by arrival curve, which describes an upper constraint of the flow.

Definition 3.1.1 (Arrival Curve in NC)**.** *Assume the input function of a flow is $R(t)$, then the flow has an arrival curve α^U iff*

$$\forall s \leq t, R(t) - R(s) \leq \alpha^U(t-s)$$

. which is equivalent to

$$R \leq R \otimes \alpha^U$$

. where $(f \otimes g)(\Delta) = \inf_{0 \leq \lambda \leq \Delta} \{f(\Delta - \lambda) + g(\lambda)\}$.

Figure 3.1 shows an input flow, its corresponding input function (blue full line) and arrival curve (black full line). The bits which should have been emitted at time 0 and 5 (shown with dashed up arrows) experience jitter of 10 and 5 respectively. As a result, three bits are emitted simultaneously at 10.

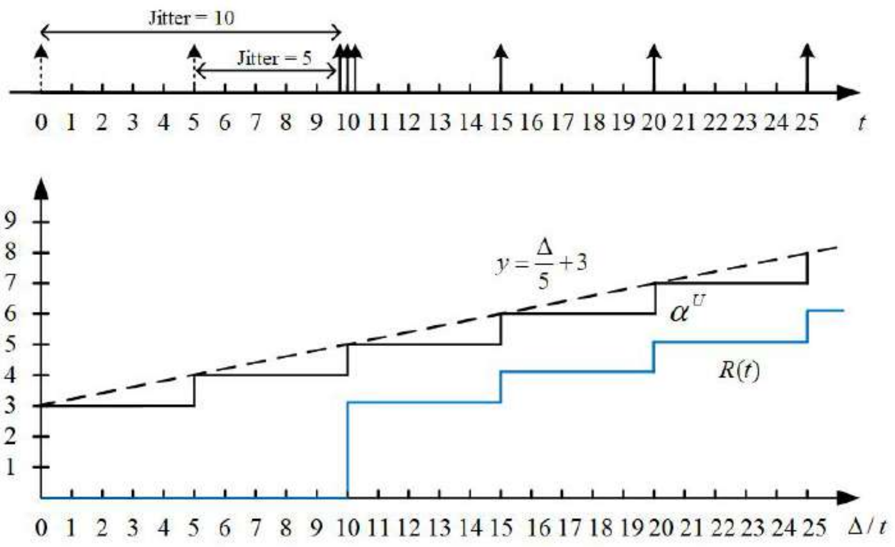


Figure 3.1: An input flow and corresponding $R(t)$ and α^U.

For each arrival curve α^U, there exists at least a pair of (η, Z) which satisfies:

$$\alpha^U(\Delta) \leq \eta * \Delta + Z \; for \; all \; \Delta \geq 0 \tag{3.1}$$

, where $\eta > 0$ and $Z \geq 0$.

In the remaining part of paper, set $\eta = lim_{\Delta \to +\infty} \frac{\alpha^U(\Delta)}{\Delta}$, which is the minimum value of η that satisfies formula (3.1). For example, in Figure 3.1, the arrival curve is $\alpha^U(\Delta) = \lceil \frac{\Delta+10}{5} \rceil$, corresponding to $\alpha(0^+) = 3$. The smallest η satisfying formula (3.1) is $\frac{1}{5}$, and with $\eta = \frac{1}{5}$ the smallest Z satisfying (3.1) is 3, that is, $y = \frac{\Delta}{5} + 3$ is the closest upper bound of $\alpha^U(\Delta)$, which is shown by the dashed black line.

While arrival curve constrains the input flows, service curve describes the service guarantees provided to these flows by network nodes. There are three types of service curve in NC: minimum service curve, strict service curve, and maximum service curve.

Definition 3.1.2 (Minimum Service Curve)**.** *The network node offers a minimum service curve β_{min} to a flow with input function $R(t)$ and output function $R^*(t)$ iff*

$$R^* \geq R \otimes \beta_{min}$$

The process of deciding the minimum service curve is different depending on whether the resource provided to the task can be specified. When the amount of resource a task occupies can be specified, the minimum service curve is derived by calculating the minimum number of execution units available over any time interval of length Δ. When it is hard to find out the exact amount of resource offered to each task such as in EDF scheduling, the minimum service curve is derived based on the delay bound of each task when scheduled in the system.

Corollary 1. *[21] Suppose the delay bound experienced by a task when scheduled in a system is DLY, then the system provides a minimum service curve $\beta_{min}(\Delta)$ to the task where*

$$\beta_{min}(\Delta) = \delta_{DLY}(\Delta) = \begin{cases} 0, & 0 \leq \Delta \leq DLY \\ +\infty, & \Delta > DLY \end{cases}$$

δ_{DLY} is called the 'impulse function', and it has the property that for any wide-sense increasing function $\theta(t)$ defined with $t \geq 0$,

$$(\theta \otimes \delta_{DLY})(t) = \begin{cases} \theta(t - DLY), & t \geq DLY \\ \theta(0), & otherwise \end{cases}$$

Definition 3.1.3 (Strict Service Curve)**.** *The network node offers a strict service curve β_{strict} to a flow with output function $R^*(t)$ iff during any backlogged period from s to t, it holds*

$$R^*(t) - R^*(s) \geq \beta_{strict}(t - s),$$

Definition 3.1.4 (Maximum Service Curve)**.** *The network node offers a maximum service curve β_{max} to a flow with input function $R(t)$ and output function $R^*(t)$ iff it holds*

$$R^* \leq R \otimes \beta_{max}$$

Next introduce how to obtain performance bounds in NC.

Assume that a flow with input function $R(t)$, output function $R^*(t)$, arrival curve α^U, traverses a node that offers a minimum service curve β_{min} and a maximum service curve β_{max}. The virtual delay $d(t)$ for all t satisfies:

$$d(t) = inf\{\tau \geq 0 : R(t) \leq R^*(t+\tau)\} \leq H(\alpha^U, \beta_{min})$$

the backlog $b(t)$ for all t satisfies:

$$b(t) = R(t) - R^*(t) \leq V(\alpha^U, \beta_{min})$$

where

$$H(f,g) = \sup_{s\geq 0}\{\inf\{\tau \geq 0 : f(s) \leq g(s+\tau)\}\}$$
$$V(f,g) = \sup_{s\geq 0}\{f(s) - g(s)\}$$

The output flow is constrained by output arrival curve

$$\alpha'^U = (\alpha^U \otimes \beta_{max}) \oslash \beta_{min}$$

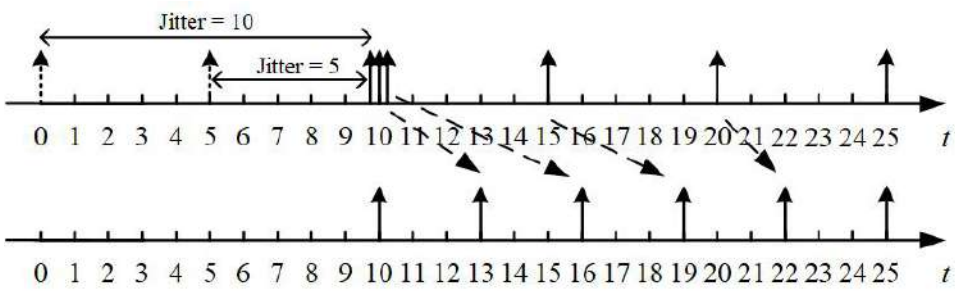
or

$$\alpha'^U = \alpha^U \oslash \beta_{min}(when\ \beta_{max}\ is\ unknown)$$

where $(f \oslash g)(\Delta) = \sup_{\lambda\geq 0}\{f(\Delta+\lambda) - g(\lambda)\}$.

3.1.2 Shapers

A greedy shaper S processes input flows and forces its output flows to conform to some time constraints which generally set the minimum separation time between two consecutive bits.



Jitter = 10
Jitter = 5
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 t
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 t

Figure 3.2: The input flow and output flow of a shaper.

A shaper works as follows: when a bit arrives at a shaper, the shaper first checks whether outputting the bit is consistent with the supposed time constraint. If so, the bit will leave the shaper immediately. Otherwise, the shaper will buffer the bit and output it as soon as outputting the bit satisfies the constraint.

In NC, a greedy shaper is modeled as an abstract component. Each shaper S is designed with a shaping curve σ, which uniquely decides the behavior of the shaper, and thus its output flow. For example, when the shaping curve is designed as $\sigma(\Delta) = \lceil \frac{\Delta}{p_s} \rceil$, the output flow has minimum separation time p_s between any two consecutive bits. In this special case of generating sporadic arrival patterns of bits, p_s is called the period of the shaper.

Assume that the input flow with arrival curve α^U passes a shaper with shaping curve σ, then the arrival curve of output flow is calculated by:

$$\alpha'^U = \alpha^U \otimes \sigma = min(\alpha^U, \sigma) \tag{3.2}$$

And the delay bound of the flow experienced at the shaper is calculated by

$$DLY_s = H(\alpha^U, \sigma)$$

An example. Assume that the input flow shown in upper part of Figure 3.2 passes a shaper, whose shaping curve is shown by σ' in Figure 3.3. The output flow of the shaper is shown in the lower part of Figure 3.2. Based on σ', there can not be more than 1 output bit in any time interval of length 3, so the three bits at time 10 can not leave the shaper at

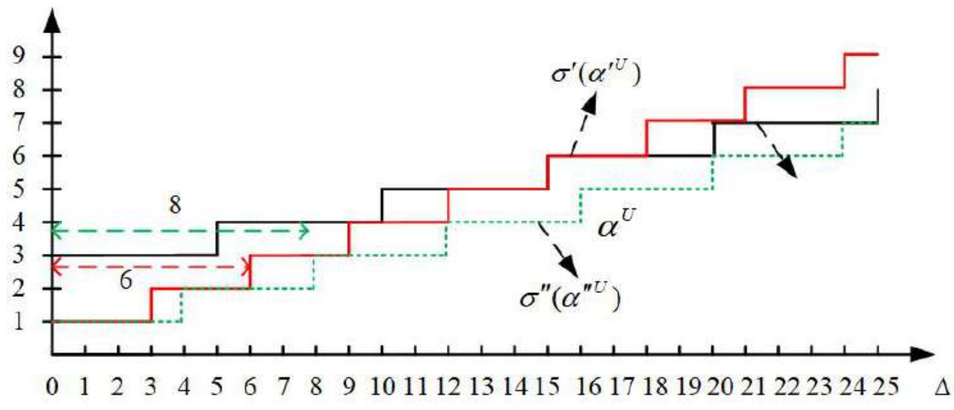


Figure 3.3: Different shaping curves and corresponding output arrival curves.

the same time. While the first one of these 3 bits is output exactly at 10, the other two bits are delayed in the shaper until time 13 and 16, respectively. The delay of release time is shown by the dashed arrows. The arrival curve of output flow, denoted by α'^U, is same as σ' according to formula (3.2). And the maximum delay experienced is 6, corresponding to the delay experienced by the third bit at time 10. When changing the shaping curve to σ'', the output arrival curve is changed to α''^U correspondingly, and the maximum delay is 8. It can be seen that given the same input curve, the lower the shaping curve is, the larger the delay bound is.

3.1.3 Network analysis

Next we consider the analysis of networked nodes. One of the important properties is concatenation, which concerns the calculation of service curve when a flow traverses several nodes.

Property 3.1.1 (Concatenation)**.** *Assume that a flow traverses m nodes $S_1, S_2, ..., S_m$ in sequence, each offering a minimum service curve $\beta_{min,i}$, a maximum service curve $\beta_{max,i}$, $i = 1, 2, ...m$ to the flow respectively. Then the concatenation of the m nodes offers a minimum service curve $\beta_{min,1} \otimes \beta_{min,2} ... \otimes \beta_{min,m}$ and a maximum service curve $\beta_{max,1} \otimes \beta_{max,2} ... \otimes \beta_{max,m}$ to the flow.*

According to the concatenation property, the overall minimum service curve of networked nodes equals the convolution of minimum service curve offered by each node in the network. As a result, the output arrival curves and delay bound can be calculated in the same way as that of a single node, except replacing the minimum service curve with the overall service curve. Specially, the delay bound derived in this way is tighter than that by simply adding up delay bound at each traversed node, which is known as the Pay-Burst-Only-Once property.

Property 3.1.2 (Pay-Burst-Only-Once)**.** *Assume that a flow with arrival curve α^U traverses m nodes $S_1, S_2, ..., S_m$ in sequence, each offering a minimum service curve $\beta_{min,i}, i = 1, 2, ...m$ to the flow respectively. Then*

$$H(\alpha^U, \beta_{min,1} \otimes \beta_{min,2} ... \otimes \beta_{min,m}) \leq H(\alpha_0'^U, \beta_{min,1}) + H(\alpha_1'^U, \beta_{min,2}) + ... + H(\alpha_{m-1}'^U, \beta_{min,m})$$

where $\alpha_i'^U$ is the output arrival curve of the $i - th$ node and the input arrival curve of the $(i + 1) - th$ node, and $\alpha_0'^U = \alpha^U$. The proof of the Pay-Burst-Only-Once property can be found in [21].

3.2 Basics in RTC

RTC is a theoretical framework for performance analysis of networked embedded systems, which is rooted in NC . This section introduces basic knowledge in RTC.

3.2.1 Curves in RTC

RTC uses arrival curves and service curves to describe timing properties of event streams and available resource.

Definition 3.2.1 (Arrival Curve)**.** *Let $R[s, t)$ denote the total number of arrived events in time interval $[s, t)$, where s and t are two arbitrary non-negative real numbers. Then, the* *corresponding upper and lower arrival curves are denoted as* $\alpha^u(t-s)$ *and* $\alpha^l(t-s)$*, respectively, and satisfy:*

$$\forall s < t,\ \ \alpha^l(t-s) \leq R[s,\, t)^1 \leq \alpha^u(t-s),$$

where $\alpha^u(0) = \alpha^l(0) = 0$.

Comparing the definitions of arrival curves in RTC and NC, the upper arrival curve in RTC is equivalent to the arrival curve in NC.

Intuitively, arrival curves represent, for each Δ, the maximum and minimum number of events arrived in any time interval of length Δ. While cumulative function corresponds to one concrete event stream trace, arrival curves models the common worst-case/best-case timing behavior of a set of event stream traces whose cumulative functions are bounded in certain ranges. That is, in any time interval of length Δ, there are at most $\alpha^u(\Delta)$ events arrived and at least $\alpha^l(\Delta)$ events arrived.

Definition 3.2.2 (Service Curve)**.** *Let* $C[s,\, t)$ *denote the total available resource in time interval* $[s,\, t)$*. Then, the corresponding upper and lower service curves are denoted as* β^u *and* β^l*, respectively, and satisfy:*

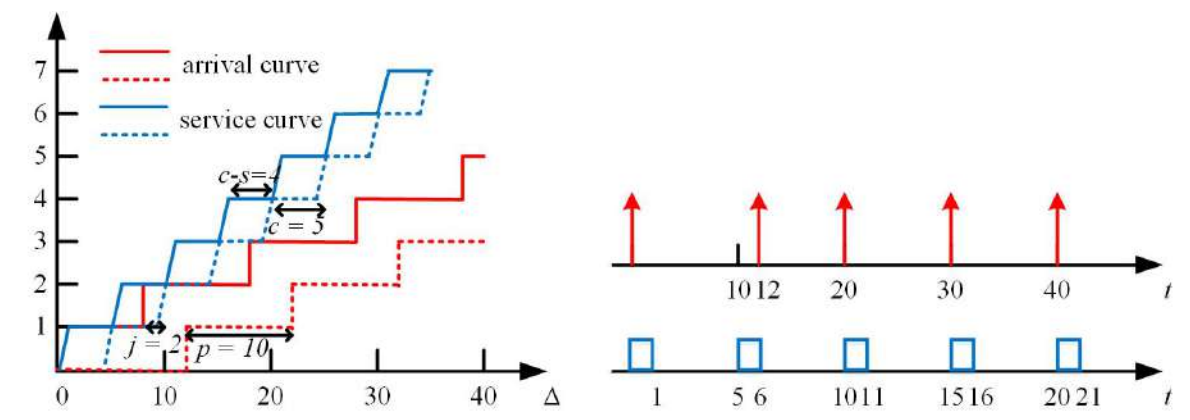
$$\forall s < t,\ \ \beta^l(t-s) \leq C[s,\, t) \leq \beta^u(t-s),$$

where $\beta^u(0) = \beta^l(0) = 0$.

This thesis adopts the PJD workload model for arrival curve and TDMA model for service curve in [82]. In detail, the input arrival curves are characterized by (p, j, d), where p denotes the period, j the jitter, and d the minimum inter-arrival distance of events in the modeled stream:

$$\alpha^u(\Delta) = \min\left(\left\lceil \frac{\Delta + j}{p} \right\rceil, \left\lceil \frac{\Delta}{d} \right\rceil\right),\ \ \alpha^l(\Delta) = \left\lfloor \frac{\Delta - j}{p} \right\rfloor.$$

^{1}Both left-open and right-open are feasible.



(a) Input arrival/service curve. (b) A possible event stream and TDMA resource.

Figure 3.4: An example of input workload and resources.

The service curves are characterized by (s, c, b):

$$\beta^u(\Delta) = \left(\left\lfloor \frac{\Delta}{c} \right\rfloor \cdot s + \min(\Delta \bmod c, s)\right) \cdot b, \beta^l(\Delta) = \left(\left\lfloor \frac{\Delta'}{c} \right\rfloor \cdot s + \min(\Delta' \bmod c, s)\right) \cdot b$$

, where $\Delta' = \max(\Delta - c + s, 0)$.

Generally, arrival curves can be any sub-additive curves and can model different types of job sequences[2]. The more complicated the expression of $\alpha^u(\alpha^l)$ is, the more complex job sequences it can model. For example, when $\alpha^u(\Delta) = \lceil \frac{\Delta}{p} \rceil$, it models job sequences generated by a sporadic task with period p_i. When $\alpha^u(\Delta) = \lceil \frac{\Delta+j}{p} \rceil$, it models sporadic job sequences with period p and jitter j, where the difference between supposed and actual release time of each job is at most j and at most $\lfloor \frac{j}{p} \rfloor + 1$ jobs can be released at the same time. When $j \geq p$, arrival curve models a bursty input. This thesis does not place any constraints on the expressions of arrival functions.

The definitions of arrival curves and service curves are further explained in Figure 3.4. Figure 3.4-(a) shows arrival curves with $p = 10, j = 2, d = 0$ and service curves with $s = 1, c = 5, b = 1$, and Figure 3.4-(b) shows a possible sequence of workload and TDMA resource corresponding to Figure 3.4-(a).

[2]We do not consider dynamic inputs in this work.

For simplicity of presentation, a curve pair $f = (f^u; f^l)$ to represent both the upper and lower curves.

3.2.2 GPC (Greedy processing component)

A GPC processes events from input in a greedy fashion, as long as it complies with the availability of resources. When an event arrives at the input event port, it will be backlogged in a FIFO buffer if currently no resource is available at the input resource port. When the available resource arrives, the first event in the buffer is processed and an output event is generated and sent to the output event port. If the buffer is empty, the resource will be sent to the output resource port and passed to the subsequent component. That is, a GPC behaves as if there was a built-in controller to decide whether the input resource would process input events or directly go to output event port. Slightly abusing the notations, use $(R', C') = GPC(R, C)$ to denote the function mapping the input event and resource traces to the output event and resource traces that defines the operational semantics of GPC:

$$R'[s,t) = C[s,t) - C'[s,t) \tag{3.3}$$

$$C'[s,t) = \sup_{s \le u \le t} \{C[s,u) - R[s,u) - B(s), 0\} \tag{3.4}$$

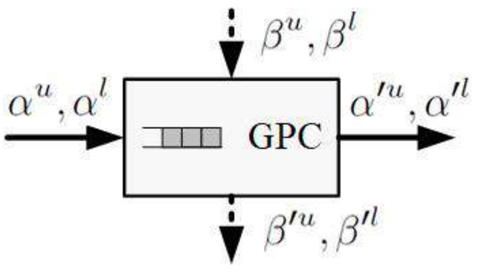
The output event stream produced by GPC is described by arrival curve α'^u, α'^l, and the remaining resource is described by service curve β'^u, β'^l.

$$\alpha'^u = \min((\alpha^u \otimes \beta^u) \oslash \beta^l, \beta^u), \tag{3.5}$$

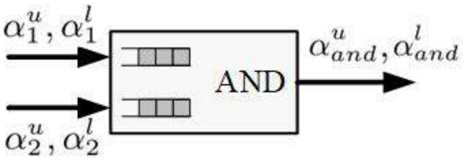
$$\alpha'^l = \min((\alpha^l \oslash \beta^u) \otimes \beta^l, \beta^l), \tag{3.6}$$

$$\beta'^u = (\beta^u - \alpha^l)\overline{\oslash}0, \tag{3.7}$$

$$\beta'^l = (\beta^l - \alpha^u)\overline{\otimes}0, \tag{3.8}$$



(a) GPC.



(b) AND.

Figure 3.5: Schematic of GPC and AND, where solid lines denote event streams and dashed lines denote resource streams.

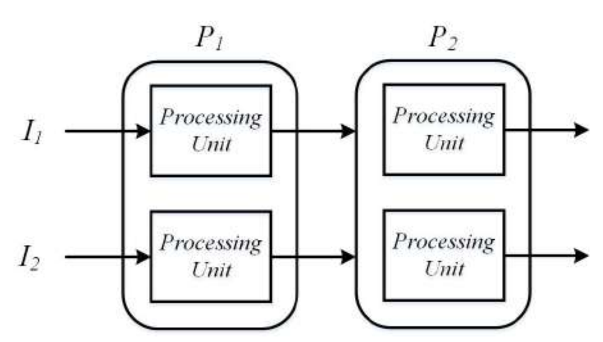
where

$$\begin{aligned}(f \otimes g)(\Delta) &= \inf_{0\le\lambda\le\Delta}\{f(\Delta-\lambda)+g(\lambda)\},\\ (f\overline{\otimes}g)(\Delta) &= \sup_{0\le\lambda\le\Delta}\{f(\Delta-\lambda)+g(\lambda)\},\\ (f \oslash g)(\Delta) &= \sup_{\lambda\ge0}\{f(\Delta+\lambda)-g(\lambda)\},\\ (f\overline{\oslash}g)(\Delta) &= \inf_{\lambda\ge0}\{f(\Delta+\lambda)-g(\lambda)\}.\end{aligned}$$

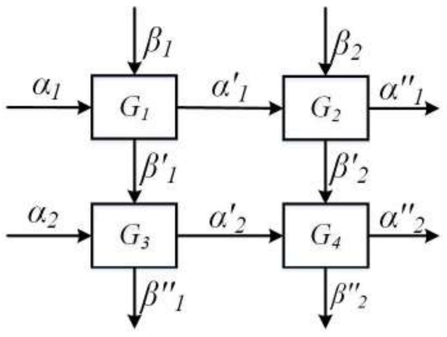
The maximum delay d_{max} experienced by any event on the event stream and the amount of events in the input buffer b_{max}, i.e., the backlog, are respectively bounded by

$$\begin{aligned}d_{max} &\le \sup_{\lambda\ge0}\left\{\inf\{\delta\ge0:\alpha^u(\lambda)\le\beta^l(\lambda+\delta)\}\right\} = H(\alpha^u,\beta^l)\\ b_{max} &\le \sup_{\lambda\ge0}\{\alpha^u(\lambda)-\beta^l(\lambda)\} = V(\alpha^u,\beta^l)\end{aligned}$$

Notes on delay and backlog bound. Note that the calculation of delay and backlog bound originates from NC. It is known that the delay (backlog) bound in NC is calculated as the maximum horizontal (vertical) distance between the arrival curve and minimum service curve. Comparing the definitions of arrival curves and service curves in NC and RTC, it is directly obtained that the the arrival curve in NC is equivalent to the upper arrival curve in RTC. Meanwhile, it can be proved that a lower service curve in RTC satisfies the conditions to be a minimum service curve in NC (to be proved in Chapter 6). Then the delay (backlog) bound is derived by getting the maximum horizontal (vertical) distance between the upper



(a) A networked system.



(b) A GPC network.

Figure 3.6: An example for GPC network.

arrival curve and lower service curve in RTC, which actually maps to the arrival curve and minimum service curve in NC.

GPCs can be connected together to model systems with various scheduling and arbitration policies. One of the typical scenarios is to model fixed-priority scheduling, where the resource allocation for tasks with different priorities is described by the resource stream direction.

An example. Suppose two event streams I_1, I_2 are processed on two successive processors. I_1 has higher priority than I_2 and the execution of I_2 will be preempted by I_1 as long as an event of I_1 arrives on each processor. Such a system (Figure 3.6-(a)) can be modeled by a GPC network (Figure 3.6-(b)). The processing of each stream on each processor is modeled as a GPC component. The output arrival curves of G_1 are the input arrival curves of G_2, corresponding to the system behavior that the events of I_1 completing execution on P_1 are further processed on P_2. And the remaining service curve of $G_1(G_2)$ is the input service curve $G_3(G_4)$, since I_1 has higher priority on both processors.

3.2.3 AND connector

Another useful abstract component is the AND connector [82], which models synchronization behaviors that are ubiquitous in distributed sensing and processing systems. Such systems usually combine data from different sources with the same timestamp to infer useful information. The structure of AND connector is shown in Figure 3.5-(b).

Data arriving on one input stream must be buffered until partner events arrive on the other input stream. Partnering events join together and immediately pass the AND connector, and consequently either of the internal buffers is empty at any point of time. Let $R_1(t)$ and $R_2(t)$ denote the accumulated number of events arrived to port 1 and port 2 respectively and $R'(t)$ the accumulated number of output events, B_1 and B_2 denote the initial buffer levels of the two input ports. The operational semantics of AND connector can be described by

$$R'(t) = \min(R_1(t) + B_1, R_2(t) + B_2).$$

If the two input event streams are characterized by arrival curves α_1^u, α_1^l and α_2^u, α_2^l, the following output curves $\alpha_{1,2}^u$ and $\alpha_{1,2}^l$ are used to upper and lower bound the combined event streams at the output [82]:

$$\alpha_{1,2}^u = \max(\min(\alpha_1^u \oslash \alpha_2^l + B_1 - B_2, \alpha_2^u), \min(\alpha_2^u \oslash \alpha_1^l + B_2 - B_1, \alpha_1^u)), \tag{3.9}$$

$$\alpha_{1,2}^l = \max(\min(\alpha_1^l \overline{\oslash} \alpha_2^u + B_1 - B_2, \alpha_2^l), \min(\alpha_2^l \overline{\oslash} \alpha_1^u + B_2 - B_1, \alpha_1^l)). \tag{3.10}$$

The delay of events at two inputs is bounded by

$$d_1 \leq H(\alpha_1^u + B_1, \alpha_2^l + B_2),$$
$$d_2 \leq H(\alpha_2^u + B_2, \alpha_1^l + B_1),$$

and the backlog at the two input buffers are bounded by

$$b_1 \leq \max(0, V(\alpha_1^u + B_1, \alpha_2^l + B_2)),$$
$$b_2 \leq \max(0, V(\alpha_2^u + B_2, \alpha_1^l + B_1)).$$

3.3 Basics for global EDF scheduling

This section introduces the system model and analysis basics when extending techniques in NC to improve the analysis for global EDF scheduling on multiprocessors.

3.3.1 System model

Scheduling strategy and workload model. The system inputs are modeled as a set of n bursty tasks $\tau = \{\tau_1, \tau_2, ..., \tau_n\}$, and they are considered to be scheduled under global EDF on $m \geq 2$ identical processors. In generally considered EDF scheduling, the relative deadline not only expresses the constraint for delay bound, but acts as a priority indicator. A job's execution priority is decided by its absolute deadline (release time plus the relative deadline of the task releasing it). The earlier the absolute deadline is, the higher priority a job has. This thesis considers EDF of a more general type where the relative deadline only constrains a task's delay bound, and there exists another variable deciding the execution priority of each job. The relative deadline and the priority indicator are set as two independent values and they are not necessarily the same. Consistent with this general EDF scheduling, bursty task models are as follows:

Each bursty task τ_i defines an infinite sequence of jobs. Jobs released by one task are assumed to be sequential and at any time may execute on at most one processor. It is also assumed that more than one job can be released simultaneously by the same task. A task τ_i is characterized by $(C_i, D_i, \lambda_i, \alpha_i^U)$, where C_i, D_i and λ_i are non-negative integers denoting worst-case execution time (WCET), relative deadline and priority level respectively. Since this thesis considers a general-style EDF as introduced above, D_i and λ_i are independent. α_i^U is the arrival curve in NC modeling the job sequence of task τ_i.

For each task τ_i, the maximum amount of execution units it requires in any time interval of length Δ is $C_i * \alpha_i^U(\Delta)$. Its utilization is denoted by $U_i = C_i * \eta_i$, and the total utilization of task set τ is denoted by $U = \sum_{i=1}^{n} C_i * \eta_i$. Based on the analysis in [35], the following two conditions must be satisfied to guarantee that the system is not overloaded.

$$\eta_i * C_i \leq 1$$

$$\sum_{i=1}^{n} \eta_i * C_i \leq m$$

A task is schedulable if it meets its deadline, and a task set is schedualable when all tasks in it meet their deadlines.

3.3.2 Delay analysis for bursty tasks

Assume that a task τ_i with arrival curve (in NC) α_i^U and WCET C_i is executed in a system providing minimum service curve $\beta_{min,i}$, then C_i, α_i^U and $\beta_{min,i}$ make up inputs of the corresponding component, based on which the delay bound DLY_i of τ_i is calculated as:

$$DLY_i = Del(C_i, \alpha_i^U, \beta_{min,i}) \tag{3.11}$$

where

$$Del(C, \alpha^U(\Delta), \beta_{min}(\Delta)) = \sup_{\lambda \geq 0}\{inf\{d \geq 0 : C * \alpha^U(\lambda) \leq \beta_{min}(\lambda + d)\}$$

Note that $Del(C, f, g)$ is a generalization of $H(f, g)$ by considering the workload of each job. Similarly, the Pay-Burst-Only-Once property is generalized for the delay bound calculation of bursty tasks, and the delay bound experienced by task i is calculated as:

$$DLY_{cont,i} = Del(C_i, \alpha_i^U, \beta_{min}^{cont}) \tag{3.12}$$

, where

$$\beta_{min}^{cont} = \beta_{min,1} \otimes \beta_{min,2} \cdots \otimes \beta_{min,m} \tag{3.13}$$

Assume that the job sequence of a task with upper arrival curve α^U passes a shaper with shaping function σ, then the delay bound of the task experienced at the shaper is calculated by [3]

$$DLY_s = Del(1, \alpha^U, \sigma)$$

[3]Since this thesis only concerns about the number of jobs that can be output during certain length of intervals, the WCET of the task can be viewed as 1.

CHAPTER 4

IMPROVING THE ANALYSIS OF GPC

4.1 Introduction

RTC uses different abstract components to model different resource arbitration schemes or operational semantics. GPC [28] is one of the fundamental abstract components in RTC, which essentially models priority-based resource arbitration among multiple workload streams sharing the same hardware platform. GPC is the most widely used building block in RTC due to the universal use of fixed-priority scheduling in practice.

This section makes a revisit to GPC in RTC. Our work includes two aspects: First, revise the proof of output curves in GPC and complement the missing deduction steps, which further clarifies the correctness of the original results. Second, derive tighter output arrival curves to more precisely bound the timing behavior of output event streams with two methods. The original output arrival curves of GPC were developed about 15 years ago [28, 81], as a major foundational result of the RTC framework. Since then, no improvement has ever been made, although it is widely known that these bounds are not tight. Our work for the first time makes these bounds tighter. The key idea of the first method is to utilize the remaining service curve to refine the information about how much resource can be actually consumed, and exclude the unused resource when computing the output arrival curves. The key idea of the second method is to find the connection between output arrival curves and the number of accumulated events. When modeling and analyzing a networked system, the output arrival curves of one component are the input arrival curves of its successor component, and are the necessities for delay bound analysis. Benefiting from the improvement brought by these two new methods, a tighter delay bound can be obtained when GPCs are adopted to model the target system.

Experiments are conducted to evaluate our new results in different aspects, with randomly generated system models under different configurations. Experiment results show significant improvement of our new methods in analysis precision and efficiency.

4.2 Revised proof of output curves

Although how to calculate bounds for output curves in GPC has been proved in [82], some important deduction steps are missing. In detail, the process of specifying the numerical relations between some critical parameters is too simple to be convincing. This section adds these missing deduction parts and gives revised proof of calculating output curves in GPC.

Theorem 4.2.1. *Given a GPC with arrival curve α^u, α^l and service curve β^u, β^l, then its remaining service curves are bounded by*

$$\beta'^u = ((\beta^u - \alpha^l)\overline{\oslash}0)^+$$

$$\beta'^l = (\beta^l - \alpha^u)\overline{\otimes}0$$

where for a function $f(\Delta)$, $(f(\Delta))^+ = max(f(\Delta), 0)$.

Proof. (1) First prove β'^l. Suppose p is an arbitrarily small time such that the backlog satisfies $B(p) = 0$.

Then for all $p \le s \le t$,

$$C'[s,t) = C'[p,t) - C'[p,s) = \sup_{p \le a \le t} \{C[p,a) - R[p,a)\}^+ - \sup_{p \le b \le s} \{C[p,b) - R[p,b)\}^+$$

Since $C[p,p) = R[p,p) = C[p,p) - R[p,p) = 0$, the suprema are nonnegative and we have

$$\begin{aligned}C'[s,t) &= \sup_{p\le a\le t}\{C[p,a)-R[p,a)\} - \sup_{p\le b\le s}\{C[p,b)-R[p,b)\}\\ &= \inf_{p\le b\le s}\{\sup_{p\le a\le t}\{C[b,a)-R[b,a)\}\}\\ &= \inf_{p\le b\le s}\{max\{\sup_{b\le a\le t}\{C[b,a)-R[b,a)\}, \sup_{p\le a\le b}\{C[b,a)-R[b,a)\}\}\}\end{aligned}$$

Let $\chi_1(b) = \sup_{b\le a\le t}\{C[b,a)-R[b,a)\} = max\{C[b,b)-R[b,b), C[b,b+1)-R[b,b+1), ..., C[b,t)-R[b,t)\} \ge 0.$ [1]

$\chi_2(b) = \sup_{p\le a\le b}\{C[b,a)-R[b,a)\} = max\{C[b,p)-R[b,p), C[b,p+1)-R[b,p+1), ..., C[b,b-1)-R[b,b-1), C[b,b)-R[b,b)\} \ge 0.$

Next we prove that[2] $C'[s,t) = \inf_{p\le b\le s}\{max\{\chi_1(b),\chi_2(b)\}\} = \inf_{p\le b\le s}\{\chi_1(b)\}$.

We consider two cases:

1) For any $i \in [p,s]$, $\chi_1(i) \ge \chi_2(i)$, then $C'[s,t) = \inf_{p\le b\le s}\{max\{\chi_1(b),\chi_2(b)\} = \inf_{p\le b\le s}\{\chi_1(b)\}$, then $C'[s,t) = \inf_{p\le b\le s}\{\chi_1(b)\}$.

2) There exists at least one $i \in [p,s]$ that $\chi_1(i) < \chi_2(i)$, that is, there exists one $x \in [p,i]$ that

$C[b,a) - R[b,a) = C[i,x) - R[i,x) = max\{\chi_1(i),\chi_2(i)\} = max\{C[i,p) - R[i,p), ..., C[i,x-1) - R[i,x-1), ..., C[i,x+1) - R[i,x+1), ..., C[i,t) - R[i,t)\}$

Then we have

$$R[x,i) - C[x,i) \ge R[p,i) - C[p,i) \Rightarrow C[p,x) \ge R[p,x)$$

[1]For ease of presentation assume s,t,a,b,p to be integer in following proofs.

[2]This part is just briefly explained as $'a \ge b$ since $t \ge s'$ in the existing proof [82].

$$R[x,i) - C[x,i) \geq R[p+1,i) - C[p+1,i) \Rightarrow C[p+1,x) \geq R[p+1,x)$$

$$...$$

$$R[x,i) - C[x,i) \geq R[x-1,i) - C[x-1,i) \Rightarrow C[x-1,x) \geq R[x-1,x)$$

$$R[x,i) - C[x,i) \geq R[x+1,i) - C[x+1,i) \Rightarrow R[x,x+1) \geq C[x,x+1)$$

$$R[x,i) - C[x,i) \geq C[i,i+1) - R[i,i+1) \Rightarrow R[x,i+1) \geq C[x,i+1)$$

$$R[x,i) - C[x,i) \geq C[i,t) - R[i,t) \Rightarrow R[x,t) \geq C[x,t)$$

Then when $b = x$, we have

$\chi_1(x) = max\{C[x,x) - R[x,x), C[x,x+1) - R[x,x+1), ..., C[x,t) - R[x,t)\} = 0,$

$\chi_2(x) = max\{C[x,p) - R[x,p), C[x,p+1) - R[x,p+1), ..., C[x,b-1) - R[x,b-1), C[x,x) - R[x,x)\} = 0.$

So $max\{\chi_1(x), \chi_2(x)\} = 0$.

Then

$C'[s,t) = \inf\limits_{p \leq b \leq s}\{max\{\chi_1(b), \chi_2(b)\}\} = max\{\chi_1(x), \chi_2(x)\} = 0$ (since $C'[s,t) \geq 0$).

On the other hand,

$\inf\limits_{p \leq b \leq s}\{\chi_1(b)\} = min\{\chi_1(p), \chi_1(p+1), ..., \chi_1(x), ..., \chi_1(s)\} = 0$, then we have

$C'[s,t) = \inf\limits_{p \leq b \leq s}\{\chi_1(b)\} = 0.$

So in both cases, $C'[s,t) = \inf\limits_{p \leq b \leq s}\{\chi_1(b)\}$, which implies that removing the cases when $a < b$ does not influence the final result. Note that $a \geq b$ is a consequence of above deduction, but not simply a direct result of $s \leq t$ as implied in [82].

Then continue to lower bound $C'[s,t)$.

$$\begin{aligned} C'[s,t) &= \inf_{p\leq b\leq s}\{\sup_{b\leq a\leq t}\{C[b,a)-R[b,a)\} \\ &= \inf_{p\leq b\leq s}\{\sup_{0\leq a-b\leq t-b}\{C[b,a)-R[b,a)\} \\ &\geq \inf_{p\leq b\leq s}\{\sup_{0\leq \lambda\leq t-b}\{\beta^l(\lambda)-\alpha^u(\lambda)\} \\ &\geq \sup_{0\leq \lambda\leq t-s}\{\beta^l(\lambda)-\alpha^u(\lambda)\} \\ &= (\beta^l-\alpha^u)\overline{\otimes}0 \end{aligned}$$

(2) Next prove β'^u.

$$\begin{aligned} C'[s,t) &= C'[p,t)-C'[p,s) \\ &= \sup_{p\leq a\leq t}\{\inf_{p\leq b\leq s}\{C[b,a)-R[b,a)\}\} \\ &= max\{\sup_{p\leq a\leq s}\{\inf_{p\leq b\leq s}\{C[b,a)-R[b,a)\}\}, \sup_{s\leq a\leq t}\{\inf_{p\leq b\leq s}\{C[b,a)-R[b,a)\}\}\} \\ &= max\{\sup_{p\leq a\leq s}\{min\{\inf_{a\leq b\leq s}\{C[b,a)-R[b,a)\}, \inf_{b\leq a\leq s}\{C[b,a)-R[b,a)\}\}\}, \\ &\quad \sup_{s\leq a\leq t}\{\inf_{p\leq b\leq s}\{C[b,a)-R[b,a)\}\}\} \end{aligned}$$

Similar as above, define $\psi=\sup_{p\leq a\leq s}\{\inf_{p\leq b\leq s}\{C[b,a)-R[b,a)\}\}$, $\psi_1(a)=\inf_{a\leq b\leq s}\{C[b,a)-R[b,a)\}$, and $\psi_2(a)=\inf_{b\leq a\leq s}\{C[b,a)-R[b,a)\}$. Next prove $\psi\leq 0$.

For any $a\in[p,s]$, consider three cases:

1) $\psi_1(a)=\psi_2(a)$. Then there must exist $b_{inf}=a$ such that

$$C[b_{inf},a)-R[b_{inf},a)=0=min\{\psi_1(a),\psi_2(a)\}.$$

2) $\psi_1(a)>\psi_2(a)$. Then there must exist $b_{inf}<a$ such that

$$C[b_{inf},a)-R[b_{inf},a)=min\{\psi_1(a),\psi_2(a)\}<\psi_1(a)\leq 0.$$

3) $\psi_1(a) < \psi_2(a)$. Then there must exist $b_{inf} > a$ such that

$$C[b_{inf}, a) - R[b_{inf}, a) = min\{\psi_1(a), \psi_2(a)\} < \psi_2(a) \leq 0.$$

Then combining the above three cases, for each $a \in [p, s]$, $min\{\psi_1(a), \psi_2(a)\} \leq 0$, so $\psi \leq 0$.

By now we have

$$\begin{aligned} C'[s,t) &= \sup_{p\leq a\leq t}\{\inf_{p\leq b\leq s}\{C[b,a) - R[b,a)\}\} \\ &= \sup_{s\leq a\leq t}\{\inf_{p\leq b\leq s}\{C[b,a) - R[b,a)\}\}^+ \\ &= \sup_{s\leq a\leq t}\{\inf_{a-s\leq a-b\leq a-p}\{C[b,a) - R[b,a)\}\}^+ \end{aligned}$$

Note that $a \geq b$ is a consequence of $a \geq s$, but not a direct result of $s \leq t$.

Then with substitution λ we have

$$\begin{aligned} C'[s,t) &\leq \sup_{s\leq a\leq t}\{\inf_{a-s\leq a-b\leq a-p}\{\beta^u(\lambda) - \alpha^l(\lambda)\}\}^+ \\ &\leq \sup_{s\leq a\leq t}\{\inf_{t-s\leq a-b\leq p}\{\beta^u(\lambda) - \alpha^l(\lambda)\}\}^+ \\ &= \inf_{t-s\leq\lambda\leq p}\{\beta^u(\lambda) - \alpha^l(\lambda)\}\}^+ \\ &= \inf_{t-s\leq\lambda}\{\beta^u(\lambda) - \alpha^l(\lambda)\}\}^+ \\ &= ((\beta^u - \alpha^l)\overline{\oslash}0)^+ \end{aligned}$$

$\square$

Theorem 4.2.2. *Given a GPC with arrival curve α^u, α^l and service curve β^u, β^l, then its output arrival curves are bounded by*

$$\alpha'^u = \min((\alpha^u \otimes \beta^u) \oslash \beta^l, \beta^u)$$

$$\alpha'^l = \min((\alpha^l \oslash \beta^u) \otimes \beta^l, \beta^l)$$

Proof. (1) We first prove α'^u. Suppose p is an arbitrarily small time such that the backlog satisfies $B(p) = 0$.

Then for all $p \le s \le t$,

$$R'[s,t) = R'[p,t) - R'[p,s)$$

$$= \sup_{p \le b \le s} \{C[p,b) - R[p,b)\}^+ - \sup_{p \le a \le t} \{C[p,a) - R[p,a)\}^+ + C[p,t) - C[p,s)$$

Since $C[p,p) = R[p,p) = C[p,p) - R[p,p) = 0$, the suprema are nonnegative and we have

$$R'[s,t) = \sup_{p \le b \le s} \{C[p,b) - R[p,b)\} - \sup_{p \le a \le t} \{C[p,a) - R[p,a)\} + C[p,t) - C[p,s)$$

$$= \sup_{p \le b \le s} \{ \inf_{p \le a \le t} \{R[b,a) + C[a,t) - C[b,s)\}\}$$

$$= \sup_{p \le b \le s} \{ \inf_{p \le a \le t} \{C[s,t) + C[a,b) - R[a,b)\}\}$$

$$= C[s,t) + \sup_{p \le b \le s} \{ \inf_{p \le a \le t} \{C[a,b) - R[a,b)\}\}$$

$$= C[s,t) + \sup_{p \le b \le s} \{min\{ \inf_{b \le a \le t} \{C[a,b) - R[a,b)\}, \inf_{p \le a \le b} \{C[a,b) - R[a,b)\}\}\}$$

Let $\chi = \sup_{p \le b \le s} \{min\{ \inf_{b \le a \le t} \{C[a,b) - R[a,b)\}, \inf_{p \le a \le b} \{C[a,b) - R[a,b)\}\}\}$,

$\chi_1(b) = \inf_{b \le a \le t} \{C[a,b) - R[a,b)\} = min\{C[b,b) - R[b,b), C[b+1,b) - R[b+1,b), ..., C[t,b) - R[t,b)\} \le 0$,

$\chi_2(b) = \inf_{p \le a \le b} \{C[a,b) - R[a,b)\} = min\{C[p,b) - R[p,b), C[p+1,b) - R[p+1,b), ..., C[b-1,b) - R[b-1,b), C[b,b) - R[b,b)\} \le 0$.

Next we prove $\chi = \sup_{p \le b \le s} \{\chi_1(b)\}$[3].

We consider two cases:

1) For any $i \in [p,s]$, $\chi_1(i) \le \chi_2(i)$, then $\chi = \sup_{p \le b \le s} \{\chi_1(b)\}$.

[3]This is not detailed in [82].

2) There exists at least one $i \in [p, s]$ satisfying $\chi_1(i) > \chi_2(i)$, then

$min\{\chi_1(i), \chi_2(i)\} = \chi_2(i) < 0,$

Similar as the proof for β'^{l}, there exists $x \in [p, s]$ such that $\min\{\chi_1(x), \chi_2(x)\} = \chi_1(x) = \chi_2(x) = 0$. Then

$$\begin{aligned}
\chi = &\sup_{p \le b \le s} \{min\{\chi_1(b), \chi_2(b)\}\} \\
&= max\{min\{\chi_1(p), \chi_2(p)\}, ..., min\{\chi_1(x), \chi_2(x)\}, ..., min\{\chi_1(s), \chi_2(s)\}\} \\
&= \sup_{b \in \phi}\{min\{\chi_1(b), \chi_2(b)\}\} = \sup_{b \in \phi}\{\chi_1(b)\}
\end{aligned}$$

where ψ is the set of values in $[p, s]$ which satisfy for any $b \in \phi$, $\chi_1(b) \le \chi_2(b)$.

On the other hand, $\sup_{p \le b \le s} \{\chi_1(b)\} = \sup_{b \in \psi}\{\chi_1(b)\}$, since when $b \in ([p, s] - \psi)$, $\chi_1(b) < 0$. Then we have $\chi = \sup_{p \le b \le s} \{\chi_1(b)\}$.

So in both two cases we have $\chi = \sup_{p \le b \le s} \{\chi_1(b)\}$. Then

$$\begin{aligned}
R'[s, t) &= C[s, t) + \sup_{p \le b \le s} \{\inf_{p \le a \le t} \{C[a, b) - R[a, b)\}\} \\
&= C[s, t) + \sup_{p \le b \le s} \{\inf_{b \le a \le t} \{C[a, b) - R[a, b)\}\} \\
&= \sup_{p \le b \le s} \{\inf_{b \le a \le t} \{C[s, t) + C[a, b) - R[a, b)\}\} \\
&= \sup_{p \le b \le s} \{\inf_{b \le a \le t} \{R[b, a) + C[a, t) - C[b, s)\}\}
\end{aligned}$$

Then with $\lambda = s - b$ and $\mu = a + \lambda - s$, we have

$$\begin{aligned}
R'[s,t) &= \sup_{p \le b \le s} \{ \inf_{b \le a \le t} \{R[b,a) + C[a,t) - C[b,s)\}\} \\
&= \sup_{0 \le \lambda \le s-p} \{ \inf_{0 \le \mu \le \lambda + (t-s)} \{R[s-\lambda, \mu - \lambda + s) + C[\mu - \lambda + s, t) - C[s-\lambda, s)\}\} \\
&\le \sup_{0 \le \lambda \le s-p} \{ \inf_{0 \le \mu \le \lambda + (t-s)} \{\alpha^u(\mu) + \beta^u(\lambda + (t-s) - \mu) - \beta^l(\lambda)\}\} \\
&\le \sup_{0 \le \lambda} \{ \inf_{0 \le \mu \le \lambda + (t-s)} \{\alpha^u(\mu) + \beta^u(\lambda + (t-s) - \mu) - \beta^l(\lambda)\}\} \\
&= (\alpha^u \otimes \beta^u) \oslash \beta^l
\end{aligned}$$

Since the number of processed events can not be larger than the available resource, $R'[s,t) \le \beta^u(t-s)$, then we have $R'[s,t) \le \min((\alpha^u \otimes \beta^u) \oslash \beta^l, \beta^u)$.

(2) The results for α'^l can be proved as with a combination of β'^u and α'^u. $\square$

An example. Take the calculation of β'^l as an example to show $C'[s,t) = \inf_{p \le b \le s} \{\chi_1(b)\}$. Let $p = 0$, $s = 3$, $t = 5$. Then the value of $C'[s,t)$ with different values of a, b is shown in Table 4.1.

	$b = 0$	$b = 1$	$b = 2$	$b = 3$
$a = 0$	0	$R[0,1) - C[0,1)$	$R[0,2) - C[0,2)$	$R[0,3) - C[0,3)$
$a = 1$	$C[0,1) - R[0,1)$	0	$R[1,2) - C[1,2)$	$R[1,3) - C[1,3)$
$a = 2$	$C[0,2) - R[0,2)$	$C[1,2) - R[1,2)$	0	$R[2,3) - C[2,3)$
$a = 3$	$C[0,3) - R[0,3)$	$C[1,3) - R[1,3)$	$C[2,3) - R[2,3)$	0
$a = 4$	$C[0,4) - C[0,4)$	$C[1,4) - R[1,4)$	$C[2,4) - R[2,4)$	$C[3,4) - R[3,4)$
$a = 5$	$C[0,5) - R[0,5)$	$C[1,5) - R[1,5)$	$C[2,5) - R[2,5)$	$C[3,5) - R[3,5)$

Table 4.1: An example for part of Theorem 1.

Assume that when $b = 2$, $\chi_1(2) < \chi_2(2)$ and $\chi_2(2) = R[1,2) - C[1,2)$. Then we have

$$R[1,2) - C[1,2) \geq R[0,2) - C[0,2) \Rightarrow C[0,1) \geq R[0,1)$$

$$R[1,2) - C[1,2) \geq C[2,3) - R[2,3) \Rightarrow R[1,3) \geq C[1,3)$$

$$R[1,2) - C[1,2) \geq C[2,4) - R[2,4) \Rightarrow R[1,4) \geq C[1,4)$$

$$R[1,2) - C[1,2) \geq C[2,5) - R[2,5) \Rightarrow R[1,5) \geq C[1,5)$$

Then when $b = 1$, $max\{\chi_1(1), \chi_2(1)\} = \chi_1(1) = 0$, and

$C'[0,3) = \inf\limits_{0 \leq b \leq 3}\{max\{\chi_1(b), \chi_2(b)\}\} = 0 = min\{\chi_1(0), \chi_1(1), \chi_1(2), \chi_1(3)\} = \inf\limits_{0 \leq b \leq 3}\{\chi_1(b)\}$.

4.3 Improving the precision of GPC: method 1

The calculation of upper output arrival curves in existing RTC is pessimistic, and this section introduces one method to improve the precision. The overall idea is shown in Figure 4.1, where we use GPC (GPC*) when concerning more about semantics, and abstract GPC (GPC*) for analysis part. Recall that the calculation of output curves in existing analysis is developed based on the semantics of GPC. To obtain tighter results, this thesis proposes a model with equivalent semantics as original GPC, and develops the calculation based on the new model. With the adoption of new model, the analysis process can partly exclude the resource that is not actually consumed to process any events, and avoid mistaking them as available for processing events, which causes pessimism in existing analysis.

The method starts by analyzing the processing behavior in GPC and proposes the equivalent model.

In GPC, when the buffer is empty, the available resource is not consumed, but directly sent to the output resource port. In other words, only a portion of the input resource is actually consumed to process the events. If classifying the input resource into two types, (i) those actually consumed to process events and (ii) those are not consumed, then eliminating

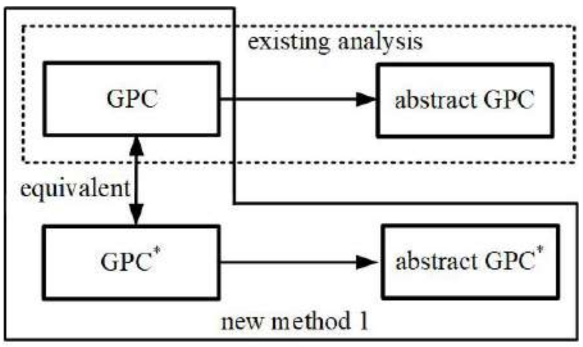
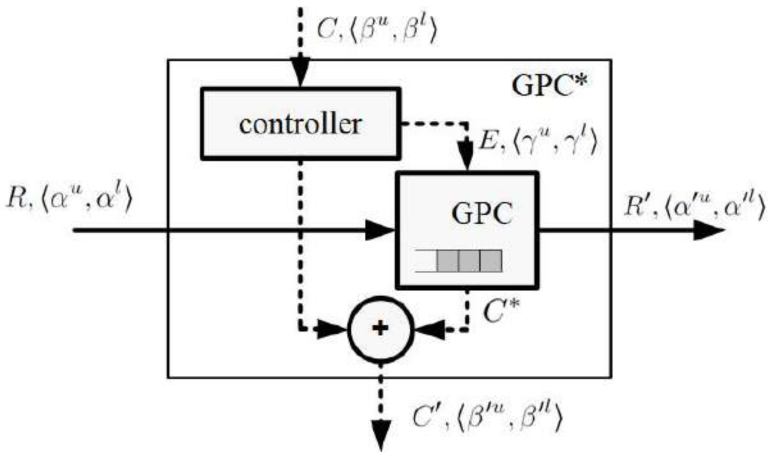


Figure 4.1: The overall idea of method 1.

type (ii) resource from the input resource stream will not affect the timing behavior of output event stream. Based on these observations, a new way to model the behavior of GPC is introduced. The new model, denoted by GPC*, is shown in Figure 4.2, and its behavior can be described as follows:

- An event arriving at the *input event port* is immediately stored in the buffer of its internal GPC in FIFO order.
- When resource is available at the input resource port, the internal controller checks whether the buffer of its internal GPC is empty (this checking is instantaneous).
  - If the buffer is not empty, the resource is immediately sent to the input resource port of the internal GPC.
  - If the buffer is empty, the resource is immediately sent to the output resource port.
- Events emitted from the output event port of the internal GPC are immediately sent to the output event port.
- The remaining resource sent out by the internal GPC is immediately set to the output resource port.

According to the semantics of GPC and GPC*, given same input event trace, the resource that is sent to the output resource port in GPC* is equivalent to that in GPC, since they are



$C, \langle \beta^u, \beta^l \rangle$
GPC*
controller
$E, \langle \gamma^u, \gamma^l \rangle$
$R, \langle \alpha^u, \alpha^l \rangle$
GPC
$R', \langle \alpha'^u, \alpha'^l \rangle$
C^*
+
$C', \langle \beta'^u, \beta'^l \rangle$

Figure 4.2: The refined model of GPC.

both corresponding to the situation when the buffer of GPC (internal GPC of GPC*) is empty. That is, the C' in Figure 4.2 and that in Figure 3.5-(a) are equivalent when they have same input R.

In the following, this thesis will prove GPC* is equivalent to GPC*. First, a key lemma showing that this new modeling method is equivalent to the original one is proved. (Refer to Figure 3.5-(a) and 4.2).

Lemma 4.3.1. *Let* $(R', C') = GPC(R, C)$ *and* $(R'', C^*) = GPC(R, C - C')$*, then* $R' = R''$ *and* $C' = C''$*.*

Proof. Let $R' = \mathcal{F}(C, C')$ denote the function defined in (3.3) and $C' = \mathcal{G}(R, C)$ the function defined in (3.4).

Since $R'' = \mathcal{F}(C - C', C^*)$, $C^* = \mathcal{G}(R, C - C')$, we have $R'' = \mathcal{F}(C - C', \mathcal{G}(R, C - C'))$, then by substituting C' by $\mathcal{G}(R, C)$, we get

$$R'' = \mathcal{F}\left(C - \mathcal{G}(R, C), \mathcal{G}(R, C - \mathcal{G}(R, C))\right), \tag{4.1}$$

then by applying the definition of $\mathcal{F}$ and $\mathcal{G}$ in (3.3) and (3.4) we have $\forall s < t$:

$$\begin{aligned} R''[s,t) =& C[s,t) - \sup_{s\le u\le t}\{C[s,u) - R[s,u) - B(s), 0\} \\ & - \sup_{s\le u\le t}\{C[s,u) - \sup_{s\le v\le u}\{C[s,v) - R[s,v) \\ & - B(s), 0\} - R[s,u) - B(s), 0\}. \end{aligned}$$

Let $f(u) = C[s,u) - R[s,u) - B(s)$, so the above equation can be rewritten as

$$R''[s,t) = C[s,t) - \sup_{s\le u\le t}\{f(u), 0\} - X, \tag{4.2}$$

where

$$X = \sup_{s\le u\le t}\{f(u) - \sup_{s\le v\le u}\{f(v), 0\}, 0\}.$$

Since

$$\forall u \in [s,t] : f(u) \le \sup_{s\le v\le u}\{f(v)\} \le \sup_{s\le v\le u}\{f(v), 0\},$$

we know $X \le 0$. On the other hand, by the definition of X, we can easily see $X \ge 0$ (since $X = \sup\{\cdots, 0\}$). Therefore we can conclude $X = 0$, which together with (4.2) gives

$$\begin{aligned} R''[s,t) =& C[s,t) - \sup_{s\le u\le t}\{C[s,u) - R[s,u) - B(s), 0\} \\ =& C[s,t) - C'[s,t) = R'[s,t). \end{aligned}$$

Moreover, since $R'' = C - C' - C^*$, and $R' = C - C'$ and $R' = R''$, we have $C^* = 0$, i.e., the internal GPC does not emit any available resource. Thus we know $C'' = C'$. □

The above lemma indicates that the output event trace produced by the original GPC and the internal GPC are the same, and so are the output resource traces. Then it can be concluded that the new model of GPC is equivalent to the original GPC.

Then the effective resource trace can be defined:

Definition 4.3.1 (Effective Resource Trace)**.** *For any time interval $[s,t)$, the effective resource trace $E[s,t)$ is defined as*

$$E[s,t) = C[s,t) - C'[s,t). \tag{4.3}$$

where C and C' are the input and output resource trace of a GPC.

Intuitively, E is the input resource trace of the internal GPC.

In the following, this thesis will derive output arrival curves for GPC*, which also apply to GPC as their output event traces are equivalent. The resource sent to the internal GPC in the time interval domain is modeled by *effective service curve*:

Definition 4.3.2 (Effective Service Curve)**.** *Let $E[s,t)$ denote the amount of resource actually sent to the internal GPC of GPC* to process the events in time interval $[s,t)$, then the corresponding upper and lower effective service curves are denoted as γ^u and γ^l, respectively, and satisfy:*

$$\forall s < t,\;\; \gamma^l(t-s) \leq E[s,\, t) \leq \gamma^u(t-s), \tag{4.4}$$

where $\gamma^u(0) = \gamma^l(0) = 0$.

It is easy to know the following lemma:

Lemma 4.3.2. *Let $C[s,t)$ denote the amount of available resource in time interval $[s,t)$, $C'[s,t)$ denote the amount of resource not used to process events in $[s,t)$. Let $E[s,t)$ denote the amount of resource actually sent to the internal GPC to process the events in $[s,t)$. For any time interval $[s,t)$ it holds:*

$$C[s,t) = E[s,t) + C'[s,t).$$

Proof. Any available resource available in $[s,t)$ is either used to process the events (included in $E[s,t)$) or not (included in $C'[s,t)$). □

By transferring this relation to the time interval domain, γ^u can be computed by β and β' as follows:

Lemma 4.3.3. *For any time interval* $[s,t)$ *with* $t = s + \Delta$ $(\Delta > 0)$*,* $E[s,t)$ *is upper and lower bounded by*

$$\gamma^u(\Delta) = \inf_{\lambda \geq \Delta}\{\beta^u(\lambda) - \beta'^l(\lambda)\}, \tag{4.5}$$

$$\gamma^l(\Delta) = \sup_{0 \leq \lambda \leq \Delta}\{\beta^l(\lambda) - \beta'^u(\lambda)\}. \tag{4.6}$$

Proof. First prove (4.5). For any time interval $[s - \varepsilon_1, t + \varepsilon_2)$ with $\varepsilon_1 \geq 0$ and $\varepsilon_2 \geq 0$, it holds $E[s,t) \leq E[s - \varepsilon_1, t + \varepsilon_2)$, so we have

$$\begin{aligned}
E[s,t) &= \inf_{\varepsilon_1 \geq 0 \wedge \varepsilon_2 \geq 0}\{E[s - \varepsilon_1, t + \varepsilon_2)\} \\
&= \inf_{\varepsilon_1 \geq 0 \wedge \varepsilon_2 \geq 0}\{C[s - \varepsilon_1, t + \varepsilon_2) - C'[s - \varepsilon_1, t + \varepsilon_2)\} \\
&\leq \inf_{\varepsilon_1 \geq 0 \wedge \varepsilon_2 \geq 0}\{\beta^u(\Delta + \varepsilon_1 + \varepsilon_2) - \beta'^l(\Delta + \varepsilon_1 + \varepsilon_2)\} \\
&= \inf_{\varepsilon \geq 0}\{\beta^u(\Delta + \varepsilon) - \beta'^l(\Delta + \varepsilon)\} \\
&= \inf_{\lambda \geq \Delta}\{\beta^u(\lambda) - \beta'^l(\lambda)\}.
\end{aligned}$$

Then prove (4.6). For any time interval $[s+\varepsilon_1, t-\varepsilon_2)$ with $\varepsilon_1 \geq 0$, $\varepsilon_2 \geq 0$ and $\varepsilon_1+\varepsilon_2 \leq t-s$, it holds $E[s,t) \geq E[s + \varepsilon_1, t - \varepsilon_2)$, so we have

$$\begin{aligned}
E[s,t) &= \sup_{\varepsilon_1 \geq 0 \wedge \varepsilon_2 \geq 0 \wedge \varepsilon_1 + \varepsilon_2 \leq t-s}\{E[s + \varepsilon_1, t - \varepsilon_2)\} \\
&= \sup_{\varepsilon_1 \geq 0 \wedge \varepsilon_2 \geq 0 \wedge \varepsilon_1 + \varepsilon_2 \leq t-s}\{C[s+\varepsilon_1, t-\varepsilon_2) - C'[s+\varepsilon_1, t-\varepsilon_2)\} \\
&\geq \sup_{\varepsilon_1 \geq 0 \wedge \varepsilon_2 \geq 0 \wedge \varepsilon_1 + \varepsilon_2 \leq \Delta}\{\beta^l(\Delta-\varepsilon_1-\varepsilon_2) - \beta'^u(\Delta-\varepsilon_1-\varepsilon_2)\} \\
&= \sup_{0 \leq \varepsilon \leq \Delta}\{\beta^l(\Delta - \varepsilon) - \beta'^u(\Delta - \varepsilon)\} \\
&= \sup_{0 \leq \lambda \leq \Delta}\{\beta^l(\lambda) - \beta'^u(\lambda)\}.
\end{aligned}$$

□

With the γ^u and γ^l obtained from the above lemma, and the fact that GPC and GPC* are equivalent, now the output arrival curves for GPC can be computed:

Lemma 4.3.4. *Given a GPC with input arrival curves α^u, α^l and service curves β^u, β^l, and γ^u, γ^l computed by Lemma 4.3.3, the output events can be upper bounded by:*

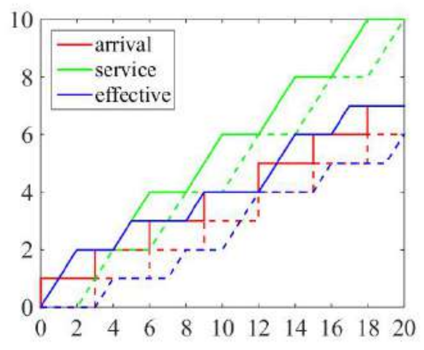
$$\alpha'^u = [(\alpha^u \otimes \gamma^u) \oslash \gamma^l] \wedge \gamma^u, \tag{4.7}$$

where $f \wedge g$ is defined as $\forall \delta, (f \wedge g)(\delta) = \min(f(\delta), g(\delta))$. The new upper output arrival curve in the above lemma is generally incomparable with the original one: sometimes our new curve is tighter, sometimes the original one is tighter, and sometimes the new curve and the original curve do not dominate each other (one curve is tighter for some parts and the other curve is tighter for some other parts). Therefore, combining our new curve with the original one gives the best result as stated in the following theorem:

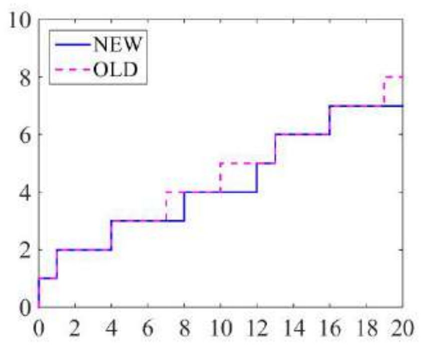
Theorem 4.3.1. *Given a GPC with input arrival curves α^u, α^l and service curves β^u, β^l, and γ^u, γ^l computed by Lemma 4.3.3, the output events can be upper bounded by:*

$$\alpha'^u = [(\alpha^u \otimes \gamma^u) \oslash \gamma^l] \wedge [(\alpha^u \otimes \beta^u) \oslash \beta^l] \wedge \gamma^u, \tag{4.8}$$

Similarly, by using γ^u and γ^l to replace β^u and β^l, we can also get a new *lower* output arrival curve. However, this yields a looser lower output arrival curve than the original one. Therefore, for computing lower output arrival curve, we should still use (3.6). Finally, the re-modeling of GPC into GPC* which contains an internal GPC would apply recursively. It is an interesting question to ask what is the condition for such a recursion to reach the fixed point. This thesis has conducted intensive experiments to test this, with all of them the fixed point is reached after the first round (i.e., further substituting the internal GPC in GPC* does not bring further benefits). However it is yet an open problem to find the general convergence conditions.



(a) Input curves.



(b) Upper output arrival curves.

Figure 4.3: An example comparing the original and new upper output arrival curves for GPC.

An example. We use a simple example to demonstrate our new output arrival curve bounds. Suppose we have a strictly periodic event arriving pattern with a period of 3 time units (the processing of each event takes one time unit) and a TDMA resource provides two time units of resource with a period of 4 time units, the corresponding arrival curves, service curves and effective service curves of which are shown in Figure 4.3. The original and our new output upper arrival curves, denoted by OLD and NEW respectively, are shown in Figure 4.3, where we can see our new curve is tighter than the original one.

4.4 Improving the precision of GPC: method 2

This section adopts the idea in [15] and derives a new upper bound for output arrival curves, which combined with the original result (3.5) generates a tighter upper bound, as shown in Section 4.5.2.

Lemma 4.4.1. *Given an event stream with input function $R(t)$, output function $R'(t)$, arrival curve α^u, α^l, service curve β^u, β^l, the output arrival curve of a GPC is upper bounded by $\alpha'^u(\Delta) = \alpha^u(\Delta) + V(\alpha^u, \beta^l) - \alpha^u(0^+)$.*

Proof. Use $B(s)$ to denote the backlog at time s. For all $s \leq t$, we have $B(t) - B(s) = R[s,t) - R'[s,t)$. Suppose p is an arbitrarily small time satisfying $B(p) = 0$, then substituting

t with p gives $B(s) = R[p, s) - R'[p, s)$. Based on the behaviors of GPC, it holds that

$$C'[p, s) = \sup_{p \le u \le s} \{C[p, u) - R[p, u) - B(p), 0\}$$

Then

$$R'[p, s) = C[p, s) - C'[p, s) = C[p, s) - \sup_{p \le u \le s} \{C[p, u) - R[p, u) - B(p), 0\}$$

Since $B(p) = 0$ and $C[p, p) - R[p, p) = 0$, we have

$$R'[p, s) = C[p, s) - C'[p, s) = C[p, s) - \sup_{p \le u \le s} \{C[p, u) - R[p, u)\}$$

Since $\alpha^u(\Delta)$ is concave, there exists $\widetilde{\alpha^u}(\Delta)$ satisfying that for any $\Delta > 0$, $\alpha^u(\Delta) = \widetilde{\alpha^u}(\Delta) + \alpha^u(0^+)$. Then

$$\begin{aligned}
R'[s, t) &= R[s, t) + B(s) - B(t) \\
&\le R[s, t) + B(s) \\
&= R[p, t) - R[p, s) + R[p, s) - \{C[p, s) - \sup_{p \le u \le s} \{C[p, u) - R[p, u)\}\} \\
&= R[p, t) - C[p, s) + \sup_{p \le u \le s} \{C[p, u) - R[p, u)\} \\
&= \sup_{p \le u \le s} \{R[u, t) - C[u, s)\} \\
&\le \sup_{p \le u \le s} \{\alpha^u(t - u) - \beta^l(s - u)\} \\
&= \sup_{p \le u \le s} \{\widetilde{\alpha^u}(t - u) + \alpha^u(0^+) - \beta^l(s - u)\} \\
&\le \sup_{p \le u \le s} \{\widetilde{\alpha^u}(s - u) + \widetilde{\alpha^u}(t - s) + \alpha^u(0^+) - \beta^l(s - u)\} \\
&= \widetilde{\alpha^u}(t - s) + \sup_{p \le u \le s} \{\alpha^u(s - u) - \beta^l(s - u)\} \\
&\le \widetilde{\alpha^u}(t - s) + V(\alpha^u, \beta^l) \\
&= \alpha^u(t - s) + V(\alpha^u, \beta^l) - \alpha^u(0^+)
\end{aligned}$$

Then the lemma is proved. □

Theorem 4.4.1. *Given a GPC with input arrival curves α^u, α^l and service curves β^u, β^l, the output events can be upper bounded by:*

$$\alpha'^u = ((\alpha^u \otimes \beta^u) \oslash \beta^l) \wedge \beta^u \wedge (\alpha^u + V(\alpha^u, \beta^l) - \alpha^u(0^+)).$$

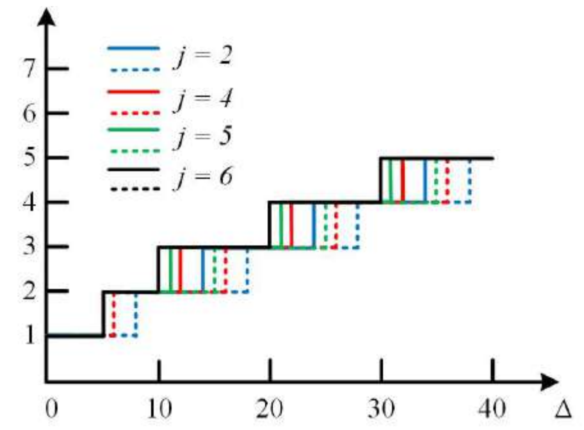
An example. Suppose a GPC has arrival curves and service curves as in Figure 3.4, then the output arrival curves calculated by Theorem 4.4.1 (blue dashed lines) and the original result (blue full lines) are shown in Figure 4.4(a). Next we show the influence of j, s to the results calculated with these two methods, where the result of original method is denoted with full lines, and that of our new method (Theorem 4.4.1) is denoted with dashed lines.

Figure 4.4-(a) shows the influence of j with 4 different sets of inputs. All these 4 inputs are generated with $p = 10, d = 0, s = 1, c = 5, b = 1$, and they differ in j which equals $2, 4, 5, 6$ respectively. Comparing the cases when $j = 2, 4, 5$ and that when $j = 6$, it implies when $p - j > c - s$, the new method is more possible to outperform the original one[4]. Focusing on the cases when $j = 2, 4, 5$, it is observed that the new method performs better when j is smaller.

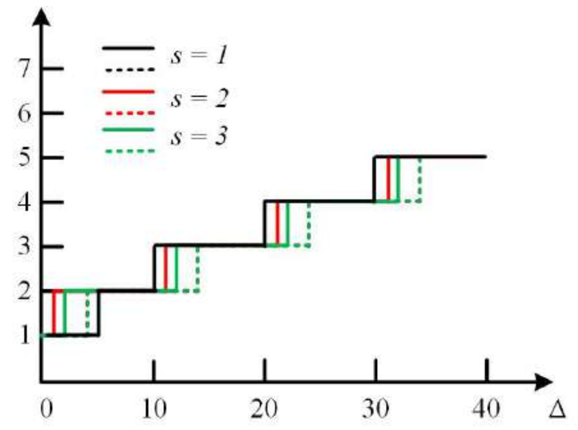
Figure 4.4-(b) shows the influence of s with 3 different sets of inputs. All these 4 inputs are generated with $p = 10, j = 6, d = 0, c = 5, b = 1$, and they differ in s which equals $1, 2, 3$ respectively. Our method outperforms the original one when $s = 2, 3$ with $p - j > c - s$, which is consistent with the trend in Figure 4.4(a). Considering the inputs with $s = 2, 3$, the difference between our method and the original one grows larger when s is smaller.

Comparison between method 1 and method 2. Method 1 and method 2 improve the precision of upper output arrival curves in GPC based on two different insights: method 1 focuses more on the amount of resource that is actually consumed to process events, while method 2 explores the connection between the output arrival curves and the accumulated

[4]Note that this is not applicable to all task sets.



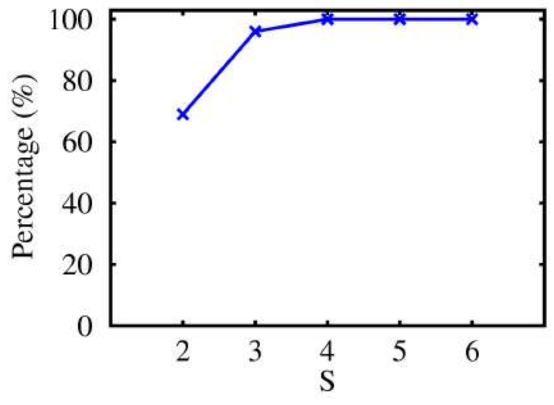
(a) Different j.



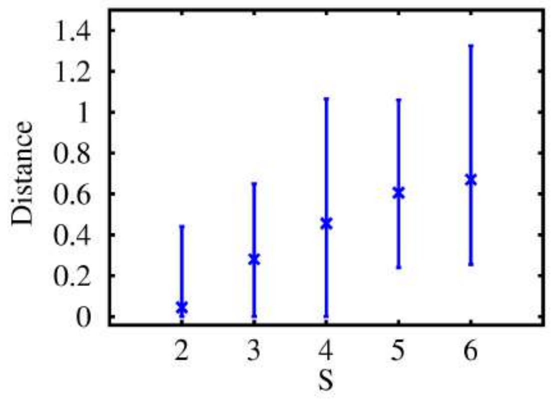
(b) Different s.

Figure 4.4: Intuitive observations about the influence of j and s.

events. With regards to the performance, both method 1 and method 2 improve the precision compared with the original results in GPC, but neither of the two dominates the other, which is further discussed in experiments.



(a) Percentage.

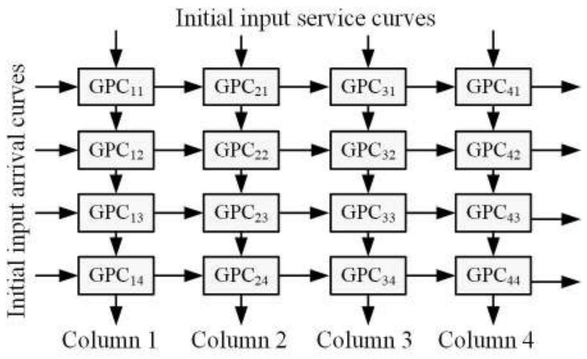


(b) Distance.

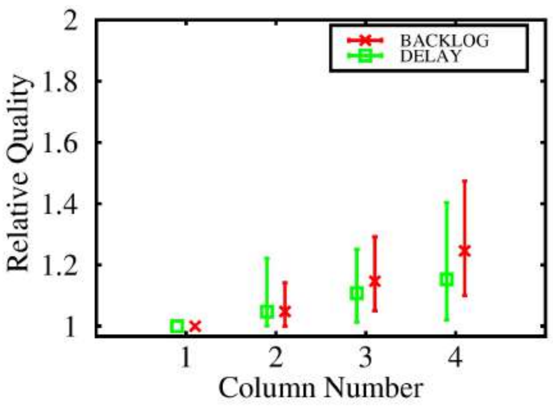
Figure 4.5: Experiment results for single GPC.

4.5 Experimental evaluation

New theoretical results are implemented in RTC Toolbox [1] and experiments are conducted to evaluate their performance.



(a) The 4×4 RTC network.



(b) Relative Quality.

Figure 4.6: Experiment results for GPC network.

This section compares the analysis results by method 1 (denoted by m1), method 2 (denoted by m2) and the original GPC (3.5) (denoted by org). These three methods are compared in the following aspects:

- Percentage, which is the ratio between the number of improved GPCs and the total number of generated GPCs in each setting, where improved GPCs are the ones where the calculated results (including output curves and delay/backlog bound) of one method are more precise than the other one.

- Distance. The distance of two curves f and g is defined as follows:

$$dist(f, g) = \frac{\sum_{\Delta=1}^{n} |(f(\Delta) - g(\Delta))|}{n}.$$

- Relative quality, which is the ratio between the delay (backlog) bounds obtained using two different methods.

The comparison between method 1 (method 2) with original GPC is shown in Section 4.5.1 (4.5.2), then the comparison between method 1 and method 2 is shown in Section 4.5.3.

4.5.1 Comparing method 1 with original GPC

This subsection compares the analysis results by method 1 and the original calculation of GPC (3.5).

Single GPC. The input arrival curves are randomly generated by selecting the p, j and d values in the following ranges: $p \in [20, 50]$, $j \in [10, 100]$ and $d \in [1, 10]$. The input service curves have a fixed $c = 60$ and $b = 1$, and s varies for different groups of experiments (corresponding to the x-axis). With each s value, we perform 1000 experiments with both methods.

Figure 4.5-(a) reports the percentage of generated GPC for which the output curve obtained by method 1 is more precise than original GPC. We start from $s = 2$ (the system is overloaded if $s = 1$). As s increases, the resource becomes more sufficient, and the percentage of experiments in which method 1 is more precise becomes higher. Eventually, when $s \geq 4$, the long-term slope of resource curves is always larger than the arrival curves, and our new method always yields more precise results than original GPC.

Figure 4.5-(b) reports the distance between the upper output arrival curves obtained by method 1 and original GPC $dist(\alpha^u_{old}, \alpha^u_{new})$ with $n = 200$ and different s. For each s, the upper and lower ends of the vertical segment represent the maximal and minimal distance, and the cross symbol in the middle represents the average distance of all the experiments with this s value. In general, the distance between method 1 and original GPC is larger when s increases. In summary, Figure 4.5-(a) and (b) show that the precision improvement of new upper output arrival curves is more significant with more sufficient resource.

GPC network. In RTC, the final goal is to compute the backlog and delay bounds of the event streams. Therefore, the precision improvement in the output curves is meaningful only if it leads to more precise backlog and delay bounds. Therefore, we evaluate the backlog and delay bounds of a 4×4 RTC network, as shown in Figure 4.6. The initial input arrival curves are randomly generated in the same way as the above experiments, and the initial input service curves are generated with $s = 20$, $c = 60$ and $b = 1$. Figure 4.6-(b) shows the ratio between the delay (backlog) bounds obtained using original GPC and those obtained by method 1. Each result for x-axis value i is the average of the delay (backlog) relative quality of components in the i^{th} column. Similar to Figure 4.5-(b), the results include the minimal, maximal and average relative quality for each group of experiments. We can see that the precision improvement for delay (backlog) bounds using new output arrival curves is more significant for the downstream components. This is because the precision gain in output curves by method 1 will be accumulated as the curves propagate in the network.

4.5.2 Comparing method 2 with original GPC

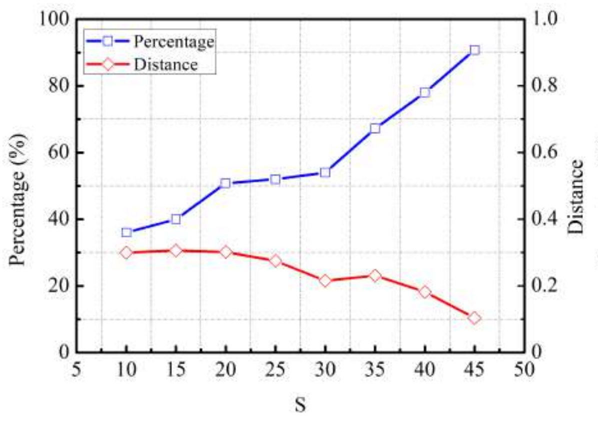
This subsection compares method 2 (Theorem 4.4.1) with the original calculation (3.5) for output arrival curves.

Single GPC. For each GPC, its arrival curves and service curves are generated as: $p \in [20, 100]$, $j \in [1, x]$, $d \in [1, 20]$, $s = y$, $c = 50$, $b = 1$, where x and y vary in each setting[5]. For each setting, we generate 1000 task sets.

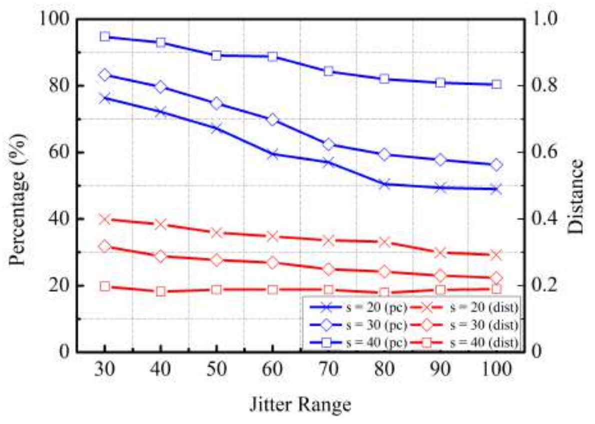
Figure 4.7-(a) shows the results with different s (X-axis) with $x = 100, n = 500$, and y-axis shows the percentage of task sets when method 2 generates tighter results than original GPC and the distance between the results. Figure 4.7-(b) shows the results with different jitter range (X-axis) with $y = 20, 30, 40$ respectively, and the definition of y-axis is the same as Figure 4.7-(a). In both settings, the output arrival curves can be improved by method 2. Specially, the percentage of improved GPCs increases with larger s. This is consistent with the observation of the example in Section 4.4: when s is larger, it is more possible that $p - j > c - s$ can be satisfied. Besides, it can be observed in Figure 4.7-(b) that the percentage of improved GPCs decreases with increasing jitter. The reason is that when jitter is large, $\alpha^+(0)$ becomes large, and it causes pessimism to the results.

GPC network. We generate $3 \times 3, 4 \times 4, 5 \times 5$ GPC networks, and compare method 2 with original calculation. The generation of inputs is similar as for single GPC and we set

[5]Note that p and c are relatively small since larger values will cause computation exception with larger-scale GPC networks.



(a) Different s.



(b) Different jitter range.

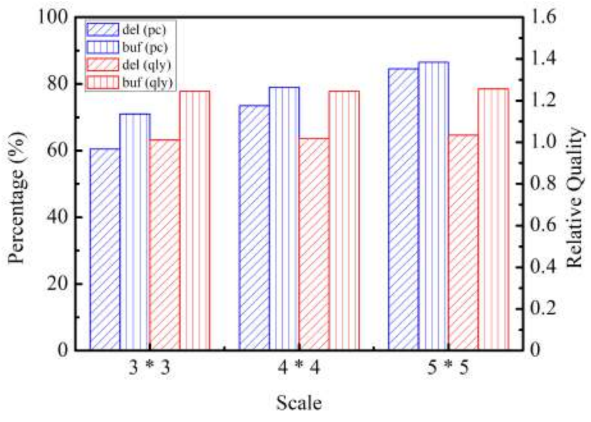
Figure 4.7: Experiment results for single GPC.

$x = 100$, $y = 100$. The results with $s = 20, 30, 40$ are shown in Figure 4.8 respectively. The percentage of networks where method 2 generates tighter delay bound (backlog bound) than the original method is shown by $del(pc)$ and $buf(pc)$, and the relative quality of delay bound (backlog bound) calculated by these two methods is shown by $del(qly)$ and $buf(qly)$. The improvement of method 2 is more obvious when the network scale is larger since the improvement of output arrival curves accumulates.

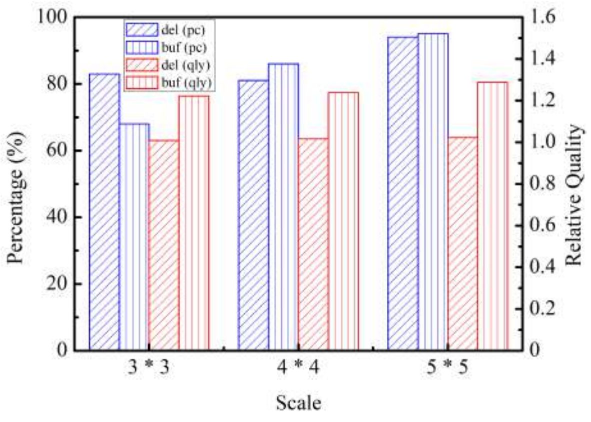
4.5.3 Comparing method 1 with method 2

This subsection compares the analysis improvement of method 1 and method 2 with both the parameter settings in last two subsections.

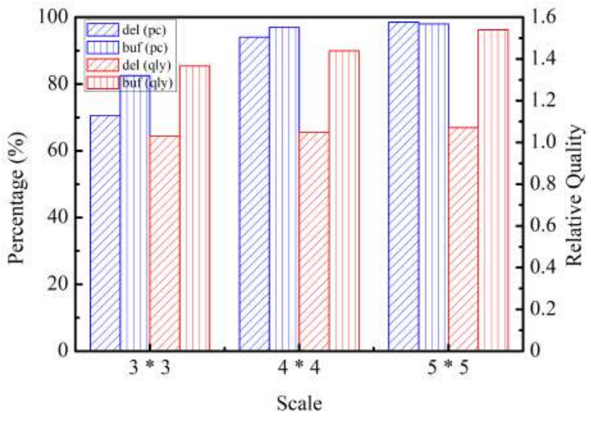
Single GPC. Figure 4.9 shows the comparison of method 1 and method 2 in the same settings as Section 4.5.1. Figure 4.9-(a) compares the percentage of GPCs for which the calculation results of A are more precise than B, denoted by $pc(A, B)$, and Figure 4.9-(b) shows the distance between calculated results of the related two methods, denoted by $dist(A, B)$. For example, the line labeled by $pc(m2, org)$ shows the percentage of improved GPCs where the results calculated by method 2 are more precise than the original GPC, and $dist(m2, org)$ describes the distance between upper output arrival curves calculated with



(a) $s = 20$.



(b) $s = 30$.



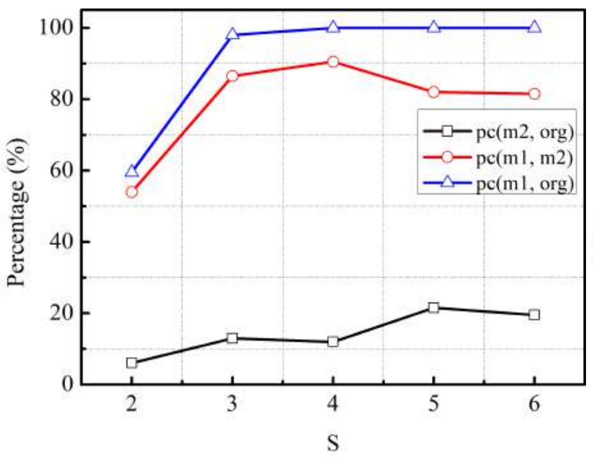
(c) $s = 40$.

Figure 4.8: Experiment results for GPC network.

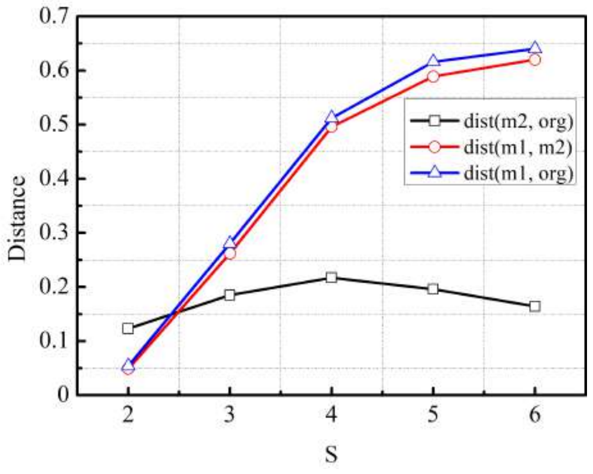
these two methods. Under this setting, method 1 has better performance than method 2.

Figure 4.10 and 4.11 show the comparison of method 1 and method 2 with the same settings as Section 4.5.2. In detail, Figure 4.10-(a) and (b) share the same settings with Figure 4.7-(a) and show results with different s. With increasing s, the percentage of GPCs for which method 2 generates more precise results than method 1 is increasing. Figure 4.11-(a) to (f) share the same settings with Figure 4.7-(b) and show the results with different j. With larger jitter range, the improvement of method 1 over original GPC stays relatively stable, while improvement of method 2 becomes smaller.

GPC network. Figure 4.12 shows the comparison of different methods with regards to calculated delay bound in a GPC network. In each setting, we evaluate the delay bound

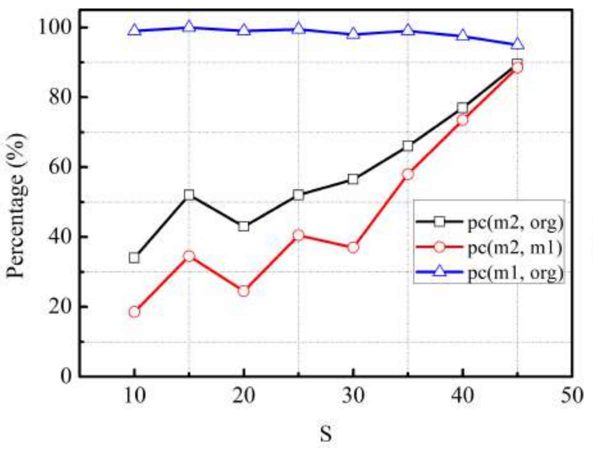


(a) The percentage with different s.

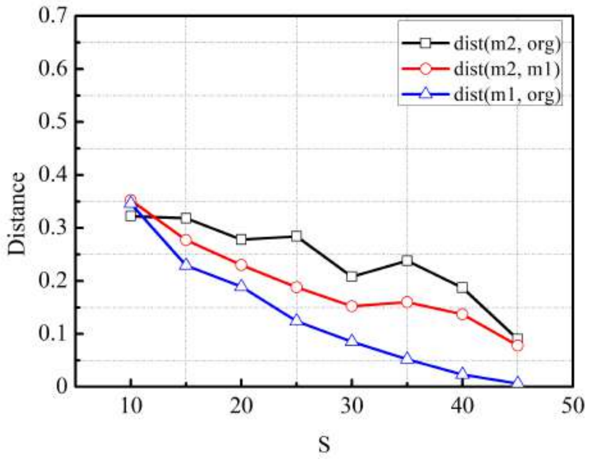


(b) The distance with different s.

Figure 4.9: Experiment results for single GPC with settings in Section 4.5.1.



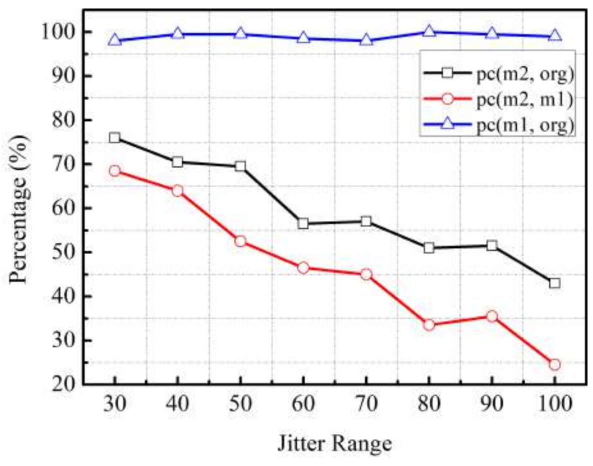
(a) The percentage with different s.



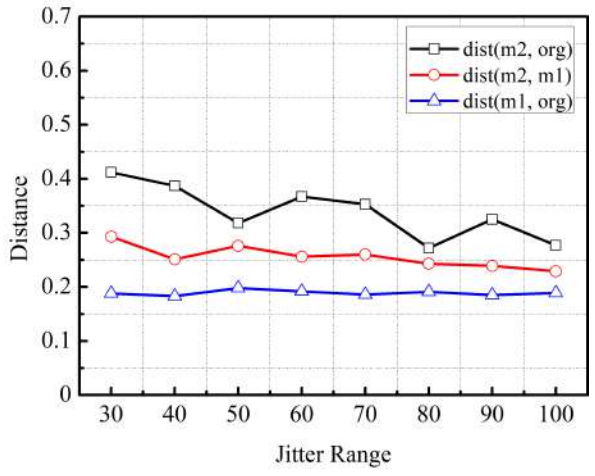
(b) The distance with different s.

Figure 4.10: Experiment results for single GPC with settings in Section 4.5.2.

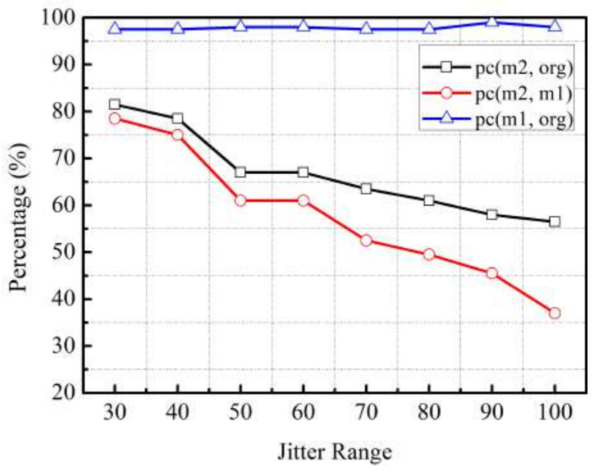
with $3 \times 3, 4 \times 4, 5 \times 5$ GPC networks respectively. In detail, Figure 4.12-(a) shares the same settings as the part of GPC network in Section 4.5.1. Under this setting, the improvement of method 1 over original GPC is more significant with larger scales of GPC network, while there is almost no improvement of method 2 over original GPC. Figure 4.12-(b) to (d) share the same settings as the part of GPC network in Section 4.5.2. When $s = 20$, method 1 generates tighter delay bound than method 2, while method 2 has better performance when s gradually increases to 40.



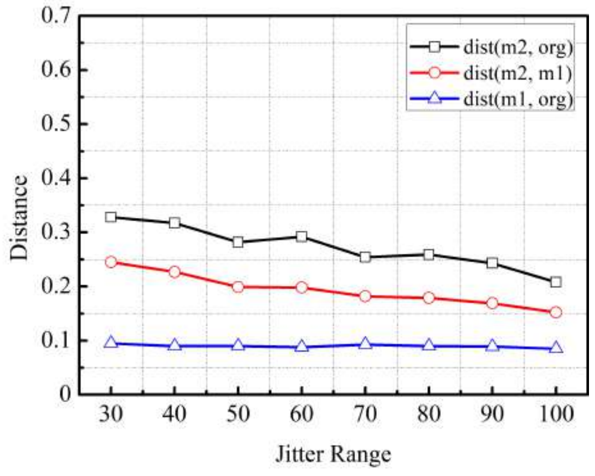
(a) The percentage, $s = 20$.



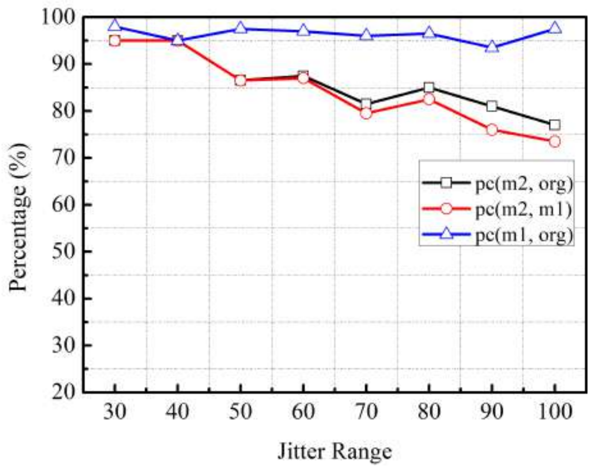
(b) The distance, $s = 20$.



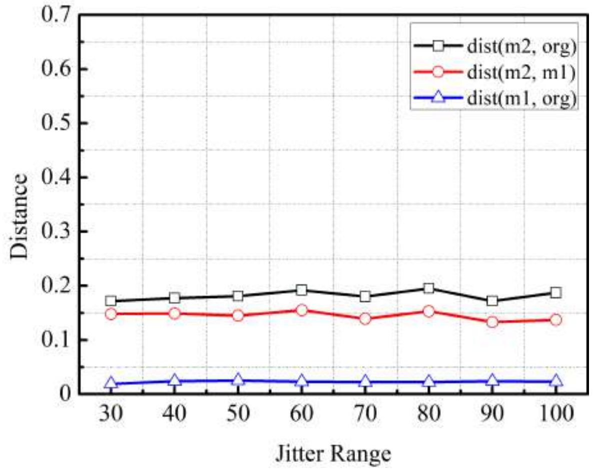
(c) The percentage, $s = 30$.



(d) The distance, $s = 30$.

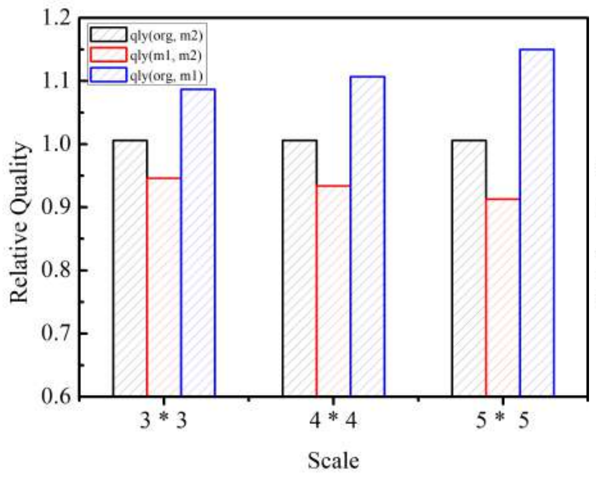


(e) The percentage, $s = 40$.

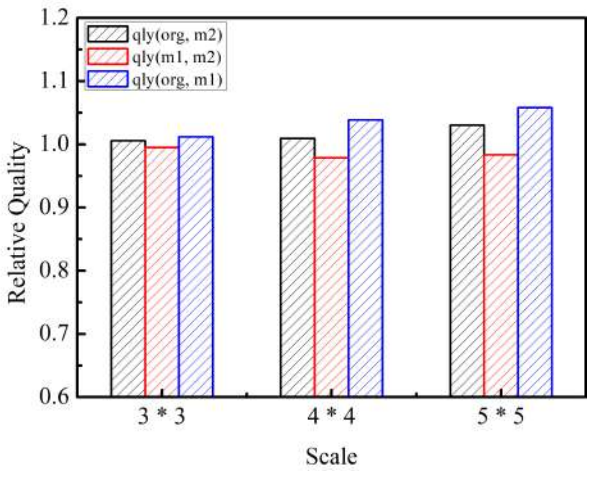


(f) The distance, $s = 40$.

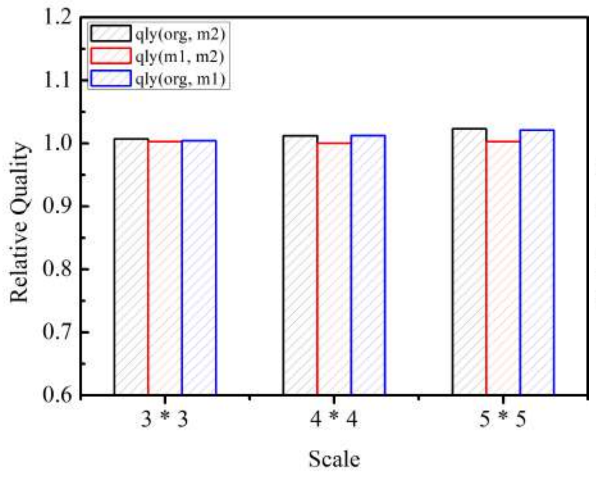
Figure 4.11: Experiment results for single GPC with settings in Section 4.5.2 with different j.



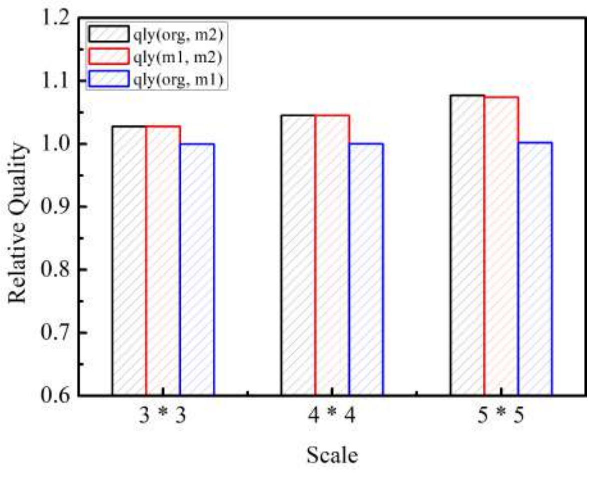
(a) Settings in Section 4.5.1.



(b) Settings in Section 4.5.2, $s = 20$.



(c) Settings in Section 4.5.2, $s = 30$.



(d) Settings in Section 4.5.2, $s = 40$.

Figure 4.12: Experiment results for GPC network.

CHAPTER 5

IMPROVING THE ANALYSIS OF AND CONNECTOR

5.1 Introduction

One of the fundamental abstract components in RTC is AND connector [82], which models synchronization of events from two streams. AND is widely used in many application domains such as sensor networks and IoT, where the states of different parts of the system with the same time stamp should be fused to derive useful information about the system.

This thesis makes a revisit to AND in RTC. It presents new results to fix problems in existing RTC theory, improves the analysis precision and makes it more general. The improvements of new results in AND analysis can generate tighter output curves and delay/backlog bounds in efficient time, which greatly optimize the analysis of a various range of real-time systems with synchronization modules. More specifically, this thesis makes contributions in the following two aspects:

- Identify and fix a problem in the existing analysis method for AND in [82] that may lead to negative values in the lower output curves. New lower output curves are present to fix the problem.

- Generalize AND to support synchronization of more than two input event streams. The original AND only has two input ports. A straightforward way for the generalization is to model a multi-input AND as several dual-input AND connectors cascaded together. However, this straightforward generalization is both imprecise and inefficient. This thesis presents a more elegant way to generalize AND to multiple inputs, which outperforms the straightforward generalization approach in terms of both precision and efficiency.

Finally, experiments are conducted to evaluate our new results in different aspects, with randomly generated system models under different configurations. Experiment results show significant improvement of our new methods in analysis precision and efficiency.

5.2 Revisiting AND

This thesis first identifies the problem in existing analysis methods of AND, and then presents solutions to fix the problem.

5.2.1 Problem of existing AND analysis

Recall the original lower output curve in [82]:

$$\alpha^l_{1,2} = \max(\min(\alpha^l_1 \overline{\oslash} \alpha^u_2 + B_1 - B_2, \alpha^l_2), \min(\alpha^l_2 \overline{\oslash} \alpha^u_1 + B_2 - B_1, \alpha^l_1))$$

We use the following example to illustrate its problem. Suppose we have two strictly periodic event streams with periods $P_1 = 5$ and $P_2 = 4$, and initial buffers $B_1 = B_2 = 0$. The input curves are shown in Figure 5.1. Then their upper and lower arriving curves are

$$\alpha^u_1(\Delta) = \lceil \Delta/5 \rceil, \;\; \alpha^l_1(\Delta) = \lfloor \Delta/5 \rfloor,$$

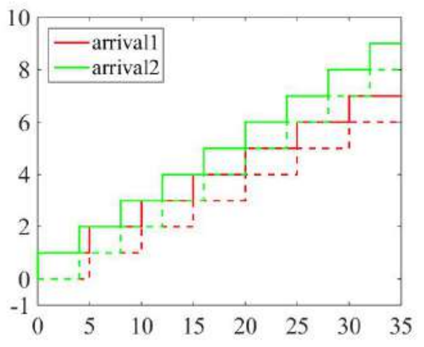
$$\alpha^u_2(\Delta) = \lceil \Delta/4 \rceil, \;\; \alpha^l_2(\Delta) = \lfloor \Delta/4 \rfloor.$$

Let $\Delta = 1$, then following the definition of $\overline{\oslash}$ we have

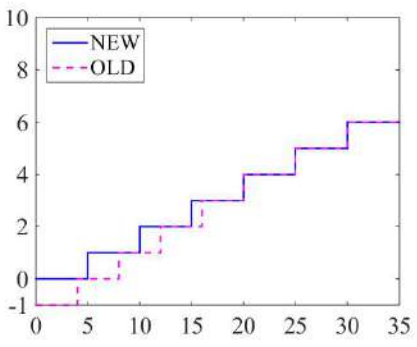
$$\alpha^l_{1,2}(1) = \max\left(\min\left(X_1, 0\right), \min\left(X_2, 0\right)\right),$$

where

$$X_1 = \inf_{\lambda \geq 0}\{\lfloor (1+\lambda)/5 \rfloor - \lceil \lambda/4 \rceil\} \leq \lfloor (1+0.1)/5 \rfloor - \lceil 0.1/4 \rceil = -1,$$
$$X_2 = \inf_{\lambda \geq 0}\{\lfloor (1+\lambda)/4 \rfloor - \lceil \lambda/5 \rceil\} \leq \lfloor (1+0.1)/4 \rfloor - \lceil 0.1/5 \rceil = -1.$$



(a) Input arrival curves.



(b) Lower output arrival curves.

Figure 5.1: An example illustrating the negative value problem in original output curves of AND.

So both X_1 and X_2 are negative, and consequently $\alpha^l_{1,2}(1)$ is also negative (the resulting output curve is shown as the dash line in Figure 5.1). This violates the basic assumption for all the computation rules in RTC that all curves are non-negative.

By having a closer look into the above example, we will see that these negative values are actually a *precision* problem rather than a *correctness* problem. α^l is a *lower* bound for the number output events, and a negative number is indeed a correct lower bound. Therefore, for a single AND connector, the original lower output curve is still correct, but just too imprecise (even more imprecise than the naive lower bound 0 in some cases). However, when the AND connectors are put into a RTC network, the effect of these negative curves will propagate to other components, and eventually may cause inconsistency to the operational semantics of the system model, as all the computation rules in RTC are based on the implicit assumptions that all curves are positive.

5.2.2 Solution

The problem mentioned above can be easily fixed by changing all the negative values to 0. However, this quick fix only superficially solves the negative value problem, but does not really address the real source of pessimism behind the problematic original lower output curve.

In the following, this thesis presents a new lower output curve for AND. Our new result is tighter than the original one and systematically solves the negative value problem.

Let $R_i[s,t)$ denote the total number of events arrived in time interval $[s,t)$, and use $R_i(t)$ to represent $R_i[0,t)$ for short. Moreover, use $[x]^+$ to denote $\max(x,0)$ for simplicity.

We first quote a known result from [82] (Lemma 1):

Lemma 5.2.1. *$R_i(t) - R_j(s)$ is upper and lower bounded by:*

$$\alpha_i^l \overline{\oslash} \alpha_j^u(t-s) \leq R_i(t) - R_j(s) \leq \alpha_i^u \oslash \alpha_j^l(t-s).$$

Theorem 5.2.1. *The output event stream of an AND connector with two input event streams characterized by arrival curves α_1 and α_2 is lower bounded by the curve:*

$$\alpha_{1,2}^l = \min(\max(\alpha_1^l \overline{\oslash} \alpha_2^u + B_1 - B_2, \alpha_1^l), \max(\alpha_2^l \overline{\oslash} \alpha_1^u + B_2 - B_1, \alpha_2^l)). \tag{5.1}$$

Proof. The backlogs of the two streams at time t are

$$b_1(t) = [R_1(t) + B_1 - (R_2(t) + B_2)]^+,$$

$$b_2(t) = [R_2(t) + B_2 - (R_1(t) + B_1)]^+.$$

Let $R_{1,2}[s,t)$ denote the number of output events in time interval $[s,t)$, which is the minimum between the events of the two streams in this interval:

$$\begin{aligned}R_{1,2}[s,t) &= \min(R_1[s,t) + b_1(s), R_2[s,t) + b_2(s))\\ &= \min(R_1[s,t) + [R_1(s) - R_2(s) + B_1 - B_2]^+,\\ &\qquad R_2[s,t) + [R_2(s) - R_1(s) + B_2 - B_1]^+)\\ &= \min(\max\{R_1[s,t), R_1(t) - R_2(s) + B_1 - B_2\},\end{aligned}$$

$$\max\{R_2[s,t), R_2(t) - R_1(s) + B_2 - B_1\}).$$

and by applying Lemma 5.2.1 we finally get

$$R_{1,2}[s,t) \geq \min(\max(\alpha_1^l \overline{\oslash} \alpha_2^u + B_1 - B_2, \alpha_1^l), \max(\alpha_2^l \overline{\oslash} \alpha_1^u + B_2 - B_1, \alpha_2^l)),$$

by which the theorem is proved. □

The solid line in Figure 4.3-(b) is the lower output curve generated by our new method. We can see that it does not only solve the negative value problem, but also in general yields more precise than the original curve even if the negative values are changed to zero.

5.3 Generalizing AND

Many realistic systems need to synchronize events from *more than two* streams. In the following we generalize the original dual-input AND connector to the multi-input setting. The semantics of an m-input AND connector can be defined as $R'(t) = \min_{i=1}^{m}(R_i(t) + B_i)$, where $R_i(t)$ represents the accumulated number of events arrived at the i^{th} input port.

5.3.1 The cascaded approach

A straightforward approach to analyze AND connectors with multiple inputs is to model a multi-input AND connector as several cascaded dual-input AND connectors. Figure 5.2 shows the cascaded modeling of a multi-input AND connector with four input streams.

We use

$$(\alpha_{i,j}, B_{i,j}) = (\alpha_i, B_i) \oplus (\alpha_j, B_j)$$

to represent the computation of output curves for an AND connector with input curves α_i and α_j (using (3.9) to compute the upper curves and using our new method (5.1) to compute

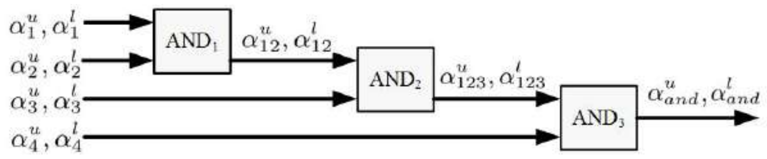


Figure 5.2: Modeling a multi-input AND connector as cascaded dual-input AND connectors.

the lower curves) as well as the initial buffer level $B_{i,j}$ for this event stream that is useful when it is further combined with other streams:

$$B_{i,j} = \min(B_i, B_j).$$

In general, for n-input streams cascaded by dual-input AND connectors in a particular order $\pi = \{\alpha_1, \alpha_2, \cdots, \alpha_n\}$ the final output curve is computed by

$$(\alpha_{and,\pi}, B_{and}) = (\alpha_1, B_1) \oplus \cdots \oplus (\alpha_n, B_n). \tag{5.2}$$

The result of this approach is sensitive to the order to cascade the inputs, and it is generally unknown which order gives the best result. On the other hand, the output bounds obtained with any of the pairing orders are valid. Therefore, we can join the results with all the possible orders to get tighter bounds:

Theorem 5.3.1. *For multi-input AND connector, let* Π *be the full permutation of* $\{\alpha_1, \cdots, \alpha_n\}$*, i.e., the set of all possible cascading orders of the input streams, and* $\alpha^u_{and,\pi}$ *and* $\alpha^l_{and,\pi}$ *are the upper and lower output curves for a particular cascading order* $\pi \in \Pi$*, then output curves of the multi-input AND connector is upper and lower bounded by:*

$$\alpha^u_{and} = \min_{\pi\in\Pi}\{\alpha^u_{and,\pi}\}, \quad \alpha^l_{and} = \max_{\pi\in\Pi}\{\alpha^l_{and,\pi}\}.$$

The proof the theorem is straightforward and thus omitted.

The computation of the maximum delay and backlog of each stream also depends on the cascading order. For example, in Figure 5.2, the maximal delay (backlog) of events in stream α_1 should be counted as the sum of the delay (backlog) incurred at AND$_1$, AND$_2$ and

AND$_3$, while for an event in stream α_4 only the delay (backlog) at AND$_3$ is counted. Obviously, to compute a tight delay (backlog) bound for a stream α_i, we should use a cascading order in which α_i only connects to the last dual-input AND connector:

Theorem 5.3.2. *A multi-input AND connector has n inputs characterized by $\{\alpha_1, \alpha_2, \cdots, \alpha_n\}$. The maximal delay and backlog of events in a stream α_i is upper bounded by*

$$d_i \leq H\left(\alpha_i^u + B_i, \alpha^{l*} + \min_{i \neq j}\{B_j\}\right),$$

$$b_i \leq \max\left(0, V\left(\alpha_i^u + B_i, \alpha^{l*} + \min_{i \neq j}\{B_j\}\right)\right),$$

where α^{l*} is the output curve for joining the other $n-1$ input streams using Theorem 5.3.1.

The proof the theorem is straightforward and thus omitted.

5.3.2 The holistic approach

After introducing the straightforward approach (which does not have good performance for multi-input AND as shown in experiment section), a new approach is present to compute the output curve and delay/backlog bounds for multi-input AND, which is both more precise and more efficient. This new approach is called the *holistic* approach, as it computes the desired results with all the inputs curves at the same time (rather than computing them step by step with two input curves at each step in the cascaded approach).

Theorem 5.3.3. *Given an AND connector with n inputs, which are characterized by the input upper and lower curves (α_1^u, α_1^l), (α_2^u, α_2^l), $\cdots$, (α_n^u, α_n^l), and the initial buffer levels $B_1, \cdots, B_n$, the output arrival curves are computed by:*

$$\alpha_{and}^u = \max_{all\ k}\left\{\min\left(\min_{i \neq k}\left\{\alpha_i^u \oslash \alpha_k^l + B_i - B_k\right\}, \alpha_k^u\right)\right\}, \tag{5.3}$$

$$\alpha_{and}^l = \min_{all\ k}\left\{\max\left(\max_{i \neq k}\left\{\alpha_k^l \overline{\oslash} \alpha_i^u + B_k - B_i\right\}, \alpha_k^l\right)\right\}. \tag{5.4}$$

The maximal delay and backlog of input i are bounded by

$$d_i = H\left(\alpha_i^u + B_i, \min_{j \neq i}\left\{\alpha_j^l + B_j\right\}\right), \tag{5.5}$$

$$b_i = \max\left(V\left(\alpha_i^u + B_i, \min_{j \neq i}\left\{\alpha_j^l + B_j\right\}\right), 0\right). \tag{5.6}$$

Proof. We first prove (5.5). Let $R_i(t)$ denote the accumulated amount of arrived events of input i from time 0 to t. The total amount of available events of input i until time t is $R_i(t) + B_i$. Then $\min_{i \neq j}\{R_j(t) + B_j\}$ is the minimum of the available events among all the other inputs. So the maximal delay of the event backlogged at input i at time t is

$$\begin{aligned}
d_i(t) =& \inf\{\tau \geq 0 : R_i(t) + B_i \leq \min_{i \neq j}\{R_j(t+\tau) + B_j\}\}\\
\leq& \sup_{\lambda \geq 0}\{\inf\{\tau \geq 0 : R_i(\lambda) + B_i \leq \min_{i \neq j}\{R_j(\lambda+\tau) + B_j\}\}\\
\leq& \sup_{\lambda \geq 0}\{\inf\{\tau \geq 0 : \alpha_i^u(\lambda) + B_i \leq \min_{i \neq j}\{\alpha_j^l(\lambda+\tau) + B_j\}\}\\
=& H(\alpha_i^u + B_i, \min_{i \neq j}\{\alpha_j^l + B_j\}).
\end{aligned}$$

Now we prove (5.6). The total amount of output events by time t is $R'(t) = \min_{\text{all } j}\{R_j(t)+B_j\}$, so the buffer of input i at time t is

$$b_i(t) = [R_i(t) + B_i - \{\min_{i \neq j}\{R_j(t) + B_j\}\}]^+.$$

Therefore, we have

$$b_i(t) = [\max_{i \neq j}\{R_i(t) - R_j(t) + B_i - B_j\}]^+. \tag{5.7}$$

By applying Lemma 5.2.1 to this, we get

$$\begin{aligned}
b_i(t) &\leq [\max_{i \neq j}\{\alpha_i^u \oslash \alpha_j^l(0) + B_i - B_j\}]^+\\
&= [\max_{i \neq j}\{(\alpha_i^u + B_i) \oslash (\alpha_j^l + B_j)(0)\}]^+
\end{aligned}$$

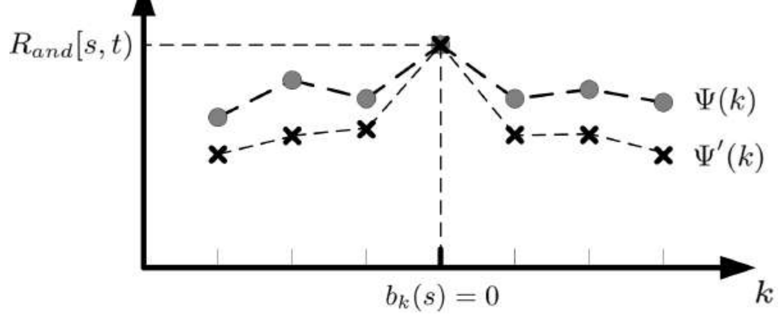


Figure 5.3: The relation between $\Psi(k)$ and $\Psi'(k)$.

$$= [\max_{i\neq j}\{\sup_{\lambda\geq 0}\{\alpha_i^u + B_i - (\alpha_j^l + B_j)\}\}]^+$$

$$= [\sup_{\lambda\geq 0}\{\max_{i\neq j}\{\alpha_i^u + B_i - (\alpha_j^l + B_j)\}\}]^+$$

$$= [V(\alpha_i^u + B_i, \min_{i\neq j}\{\alpha_j^l + B_j\})]^+.$$

In the following we prove (5.3). $R_{and}[s,t)$ denotes the amount of output combined event generated in time interval $[s,t)$, which equals the minimum among all the inputs:

$$R_{and}[s,t) = \min_{\text{all } i}\{R_i[s,t) + b_i(s)\}, \tag{5.8}$$

where $b_i(s)$ is the buffer level of input i at time s.

In the following we prove

$$\min_{\text{all } i}\{R_i[s,t) + b_i(s)\} = \max_{\text{all } k}\{\Psi(k)\}, \tag{5.9}$$

where

$$\Psi(k) = \min\left(\min_{i\neq k}\{R_i[s,t) + b_i(s)\}, R_k[s,t)\right).$$

By the definition of $\Psi(k)$ we know that for any $k \in [1,n]$

$$\Psi(k) \leq \min_{\text{all } i}\{R_i[s,t) + b_i(s)\}. \tag{5.10}$$

On the other hand, at least one of $b_1(s)$, $b_2(s)$, $\cdots$, $b_n(s)$ must be 0. Without loss of generality, let $b_x(s) = 0$, then by the definition of Ψ we have

$$\Psi(x) = \min_{\text{all } i}\{R_i[s,t) + b_i(s)\}. \tag{5.11}$$

In summary, the LHS of (5.9) is no smaller than $\Psi(k)$ for all $k \in [1, n]$, and there exists at least one $\Psi(x)$ that equals to the LHS (5.9), by which (5.9) is proved. Combining (5.8) and (5.9) yields

$$R_{and}[s,t) = \max_{\text{all } k}\{\Psi(k)\}. \tag{5.12}$$

In the following we will derive an upper bound for $\max\limits_{\text{all } k}\{\Psi(k)\}$. Note that we will derive an upper bound for the entire $\max\limits_{\text{all } k}\{\Psi(k)\}$, rather than upper bounding $\Psi(k)$ for each k and then getting their maximum.

By applying (5.7) to $\Psi(k)$, we have

$$\Psi(k) = \min\left(\min_{i\neq k}\{R_i[s,t) + [R_i(s) + B_i - \sigma]^+\}, R_k[s,t)\right),$$

where

$$\sigma = \min_{i\neq j}\{R_j(s) + B_j\}.$$

We define another function $\Psi'(k)$ respect to k as follows:

$$\Psi'(k) = \min\left(\min_{i\neq k}\{R_i[s,t) + [R_i(s) + B_i - \sigma']^+\}, R_k[s,t)\right),$$

where $\sigma' = R_k(s) + B_k$.

Now we discuss the relation between $\Psi'(k)$ and $\Psi(k)$. First, since $\sigma \leq \sigma'$, we know the general relation between $\Psi'(k)$ and $\Psi(k)$:

$$\Psi'(k) \leq \Psi(k).$$

Then we focus on the relation between $\Psi'(k)$ and $\Psi(k)$ with a particular k satisfying $b_k(s) = 0$ which means the available events during [0,t) for k is no more than that for all other inputs. In this case, $R_k(s) + B_k$ must be no larger than $R_i(s) + B_i$ for any i , which implies $\sigma = \sigma'$. Therefore, we know $b_k(s) = 0$ implies $\Psi(k) = \Psi'(k)$.

Moreover, by the definition of $\Psi(k)$, we know $\Psi(k)$ reaches its maximal value with k if $b_k(s) = 0$, i.e.,

$$\Psi(k) = \min_{\text{all } i}\{R_i[s,t) + b_i(s)\} = \max_{\text{all } i}\{\Psi(i)\}. \tag{5.13}$$

Putting the above discussions together, the relation between $\Psi'(k)$ and $\Psi(k)$ can be summarized as follows:

In general $\Psi'(k) \leq \Psi(k)$, while both of them reach the same maximal value with a particular k satisfying $b_k(s) = 0$. Moreover, there must exist $b_k(s) = 0$ among $\{b_1(s), b_2(s), \cdots, b_n(s)\}$ since at any time point at least one of the stream buffers must be empty. These relations are illustrated in Figure 5.3. Therefore, we can conclude that

$$\max_{\text{all } k}\{\Psi(k)\} = \max_{\text{all } k}\{\Psi'(k)\}. \tag{5.14}$$

In the following, we compute an upper bound for $\max_{\text{all } k}\{\Psi'(k)\}$:

$$\max_{\text{all } k}\{\Psi'(k)\} = \max_{\text{all } k}\{\min(\min_{i \neq k}\{R_i[s,t) + [R_i(s) + B_i - \sigma']^+\}, R_k[s,t))\}.$$

$\Psi'(k)$ reaches the maximal value with k satisfying $b_k(s) = 0$. If $b_k(s) = 0$, we know $R_i(s) + B_i - R_k(s) - B_k \geq 0$, i.e., $R_i(s) + B_i - \sigma' \geq 0$, so the above equation is rewritten as

$$\max_{\text{all } k}\{\Psi'(k)\} = \max_{\text{all } k}\{\min(\min_{i \neq k}\{R_i(t) - R_k(s) + B_i - B_k\}, R_k[s,t))\}.$$

and finally by (5.12), (5.14) and Lemma 5.2.1 we have

$$R_{and}[s,t) \leq \max_{\text{all } k}\{\min(\min_{i\neq k}\{\alpha_i^u \oslash \alpha_k^l + B_i - B_k\}, \alpha_k^u)\}(t-s).$$

By now we have proved (5.3) for the upper output curve.

In the following we prove (5.4) for the lower output curve.

The amount of output events in $[s,t)$ is the minimum amount of events among all streams in this time interval:

$$\begin{aligned}
&R_{and}[s,t) = \min_{\text{all } k}\{R_k[s,t) + b_k(s)\}\\
&= \min_{\text{all } k}\left\{R_k[s,t) + [\max_{i\neq k}\{R_k(s) - R_i(s) + B_k - B_i)\}]^+\right\} /\!/ by\,(5.7)\\
&= \min_{\text{all } k}\left\{\max(R_k[s,t), \max_{i\neq k}\{R_k(t) - R_i(s) + B_k - B_i\})\right\}.
\end{aligned}$$

By $R_k[s,t) \geq \alpha_k^l(t-s)$ and Lemma 5.2.1, we get

$$R_{and}[s,t) \geq \min_{\text{all } k}\left\{\max\left(\max_{i\neq k}\{\alpha_k^l \overline{\oslash} \alpha_i^u + B_k - B_i\}, \alpha_k^l\right)\right\}.$$

□

5.4 Experimental evaluation

New theoretical results are implemented in RTC Toolbox [1] and experiments are conducted to evaluate their performance. Experiments are conducted on a computer with a 2.50GHZ Intel Core i7 processor and 4.00 GB RAM.

5.4.1 Evaluation for dual-input AND

Next we evaluate the precision improvement of our new lower output curves for dual-input AND connectors in (5.1). We compare two methods to compute the lower output curves:

- AND-Naive: The lower output curve obtained by simply changing the negative values in (3.10) to 0.

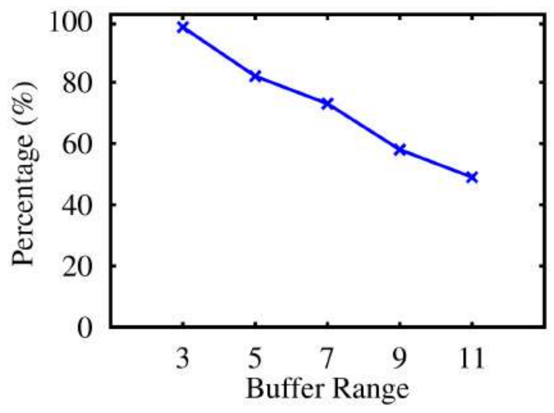
- AND-New: Our new lower output curve in (5.1).

In all the experiments with AND (including the multi-input AND in the next subsection), we use a revised version of the PJD event model to generate input curves. For AND, if the long-term slope of the input curves are different, the events backlogged in the buffer of the one with lower slope will increase infinitely. Therefore, AND is typically used to join curves with the same long-term slope. However, in the original PJD model [82], the long-term slope of a curve only depends on the parameter p, which represents the period of the events. Therefore, with the original PJD model, all the input curves to AND must have the same period, which not only represents a very special case but may lead to biased comparison results. In order to cover more general cases and make our results more convincing, we extend the PJD model from three parameters (p, j, d) to four parameters (p, j, d, r):

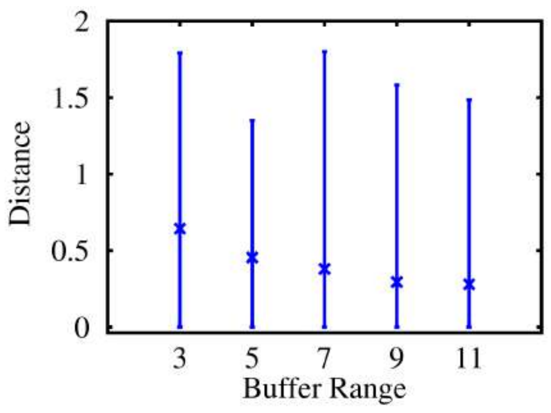
$$\alpha^u(\Delta) = \min\left(\left\lceil \frac{(\Delta + j)r}{p} \right\rceil, \left\lceil \frac{\Delta}{d} \right\rceil\right), \ \ \alpha^l(\Delta) = \left\lfloor \frac{(\Delta - j)r}{p} \right\rfloor$$

With the extended model, the long-term slope of the curves depends on the value of $\frac{r}{p}$. Therefore, we can generate curves with the same long-term slope but different periods.

The input curves are randomly generated by selecting the p, j and d values in the same ranges as the experiments in Figure 4.5-(a) and (b). For the two input curves of an AND, their r values are derived so that the long-term slopes of the two curves are the same. The initial buffer level of each input stream is randomly chosen in the range $[1, x]$, where x corresponds to the x-axis in Figure 5.4. Figure 5.4-(a) reports the percentage of experiments in which AND-New is more precise than AND-Naive, and Figure 5.4-(b) reports the distance between the lower output curves obtained by AND-New and AND-Naive. The results show that the precision improvement of our new results is more significant when the initial buffer level is smaller.

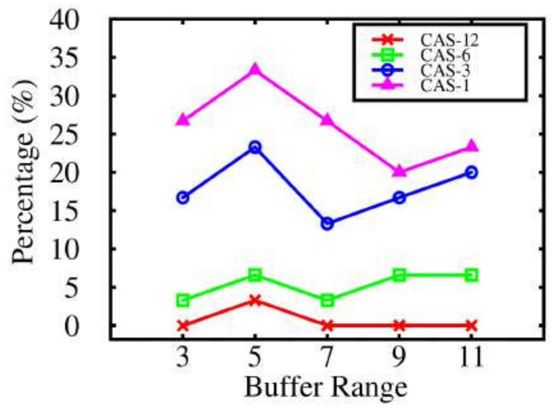


(a) Percentage.

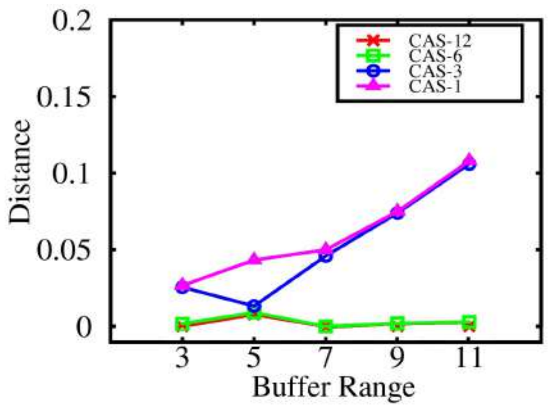


(b) Distance.

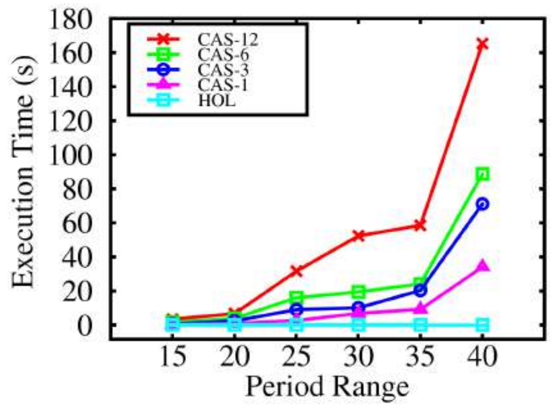
Figure 5.4: Experiment results for dual-input AND.



(a) Percentage.

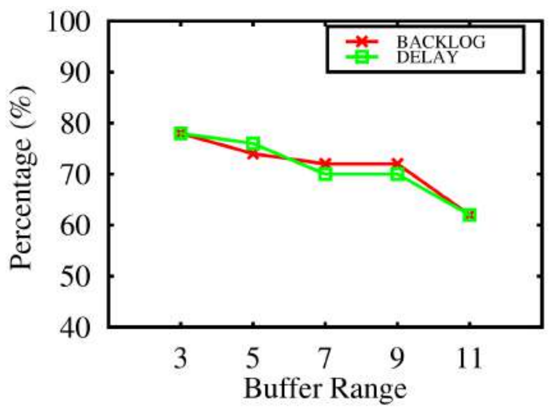


(b) Distance.

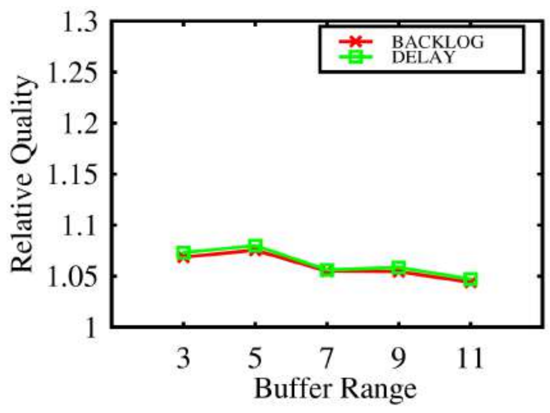


(c) Execution Time.

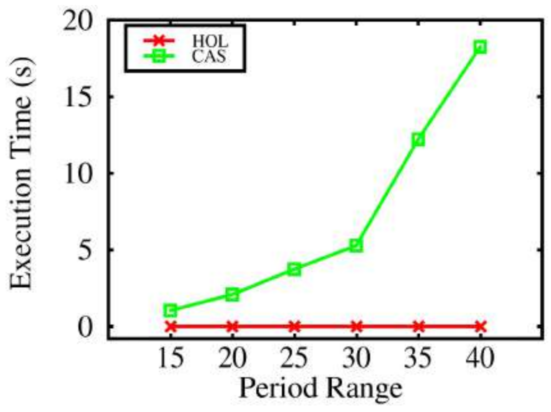
Figure 5.5: Experiment results for multi-input AND (output curves).



(a) Percentage.



(b) Relative Quality.



(c) Execution Time.

Figure 5.6: Experiment results for multi-input AND (delay and backlog).

5.4.2 Evaluation for multi-input AND

Next we evaluate the generalization of AND to multiple inputs. We first compare the output curves obtained by the cascaded approach and the holistic approach:

- CAS-x: The cascaded approach to compute the output curves in (5.2). Recall that the cascaded approach is sensitive to the order to apply dual-input AND to the event streams. The result with any order provides valid upper and lower output curves, while joining the results with different orders in general may improve the precision. CAS-x represents the final results obtained by joining results with x randomly selected orders.
- HOL: The holistic approach to compute the output curves using (5.3) and (5.4).

Figure 5.5 shows experiment results with AND with four input streams. There are in total $4!/2 = 12$ different cascading orders for a four-input AND. Figure 5.5-(a) reports the percentage of experiments that HOL is more precise than CAS-x, and Figure 5.5-(b) reports the distance between the output curves obtained by CAS-x and HOL, with x being 1, 3, 6 and 12, respectively. The input curves are generated in the same way as in Section 5.4.1.

We also compare the time consumed by the analysis of each four-input AND by different methods in Figure 5.5-(c). The input curves are generated in a similar way with above, and the only difference is that we change the range for selecting the period p: the lower bound is 10, while the upper bound is $15, 20, \cdots, 40$ (the values on the x-axis). The initial buffer level is in the range $[1, 5]$. The experiment results show that the HOL method is consistently efficient: it on average takes less than 0.1 second, and rarely exceeds 1 second. However, the efficiency of the cascaded approach is much lower. Even CAS-1 (only one cascading order is analyzed) is much slower than HOL. The time consumption of CAS-x increases exponentially as the range of periods increases.

The low efficiency of the cascaded approach is because of the period explosion problem [43]. Generally, the computation time and memory requirement of an operation between two curves are proportional to the number of segments contained by the curves. The number of segments of a curve representing a PJD event model is generally proportional to its period. The period of the output curve of a dual-input AND is the hyperperiod of the two input curves. In the cascaded approach, the period of the event stream increases exponentially as it travels through the dual-input AND, which leads to the low efficiency of the cascaded approach. When periods of the input curves are selected from a wider range, the resulting hyperperiod of them is larger, and thus the efficiency of the cascaded approach is worse. By contrast, the holistic approach in (5.3) and (5.4) only performs corresponding operations on each pair of the input curves, which avoids the above period explosion problem.

To summarize, when calculating output curves, the HOL method outperforms the cascaded approach with significant advantages with regard to analysis efficiency, and it can generate more precise results in some cases.

Finally, we compare the precision and computation efficiency of the delay (backlog) bounds using the cascaded method (Theorem 5.3.2) and holistic method (Theorem 5.3.3) for four-input AND. The input curves are generated in the same way as the corresponding experiments in above. Figure 5.6-(a) and (b) report the percentage of experiments where the holistic approach gives more precise results and ratio between the delay (backlog) bounds by the cascaded approach and by the holistic approach. Figure 5.6-(c) gives the time consumption of the two approaches, where the holistic approach on average takes less than 0.1 second, while the time consumption of the cascaded approach is much longer and increases exponentially as the period range is larger. Note that to compute the delay and backlog bounds of an event stream in a four-input AND, we only need to compute the output curve joining the remaining three streams, so the time consumption is lower than the experiments in Figure 5.5-(c) which join all the four streams. In summary, to generalize AND to support multiple inputs, our new holistic approach is not only more precise but also significantly more efficient than the naive approach by cascading dual-input AND.

CHAPTER 6

PAY-BURST-ONLY-ONCE IN RTC

6.1 Introduction

Pay-Burst-Only-Once is a fundamental and powerful property in NC, which states that the delay bound obtained by considering the overall service curve of nodes a flow traverses is better (tighter) than that by summing up the delay bound of each individual traversed node. The Pay-Burst-Only-Once property relies on the concatenation property in NC, which proves that concatenating the service curve of a sequence of nodes generates the overall service curve for the whole system composed of these nodes. The Pay-Burst-Only-Once property is proved based on the concatenation property.

Since RTC inherits many concepts and properties from NC, people naturally expect the Pay-Burst-Only-Once property to also hold in RTC. Much work has already assumed it to be true and applies it for performance analysis. However, the Pay-Burst-Only-Once property has never been proved in RTC. Even worse, some existing results seem to be against the concatenation property in RTC [23] (recall that in NC the Pay-Burst-Only-Once property is proved based on the concatenation property). Therefore, it leaves an important open problem to find out whether the Pay-Burst-Only-Once property holds in RTC.

Our work gives the answer to the problem mentioned above. We prove that the Pay-Burst-Only-Once property indeed holds in RTC. The basic idea is that we first find some numerical relations between curves in RTC and NC and then prove the property in RTC indirectly by using the Pay-Burst-Only-Once property in NC.

Moreover, since the concatenation property does not hold in RTC, the service curve for the whole system can not be derived, and then the output arrival curves of the whole system seemingly can only be derived by calculating the output arrival curves one node after another. However, this process is so tedious, especially with large-scale systems. As a consequence, it raises a question whether we can find a once-for-all derivation for output arrival curves without knowing the exact overall service curve.

This thesis solves the output arrival curve problem: figure out how to compute the output arrival curves of a concatenated system with a once-for-all method. The basic idea is similar to the proof of the Pay-Burst-Only-Once property, and the difficulty lies in the calculation of lower output arrival curves in NC, which as far as known have never been defined and calculated before.

	Output arrival curve (a single node)		Output arrival curve with concatenation
RTC	upper output arrival curve	lower output arrival curve	
Status	derived in [82]		no results
			derived in our work
NC	output arrival curve	lower output arrival curve	
Status	derived in [21]	derived in our work	same as for a single node

Table 6.1: The comparison of our contribution and existing work (curves).

	Concatenation	Pay-Burst-Only-Once
RTC	about lower service curve	
Status	proved to ***not*** hold in [23]	used but not proved
		proved to hold in our work
NC	about maximum/minimum service curve	
Status	proved to hold in [21]	proved to hold in [21]

Table 6.2: The comparison of our contribution and existing work (properties).

The comparison between our contribution and existing work is summarized in Table 6.1 and 6.2, where our work is marked in blue.

6.2 Comparing curves in NC and RTC

Before stepping into the analysis of the Pay-Burst-Only-Once property, this thesis first explores the relations among curves in NC and RTC, which are the basis of further analysis and summarized in Table 6.3.

Curve A	Curve B	Relation
arrival curve in NC	upper arrival curve in RTC	$A \iff B$
lower arrival curve in NC	lower arrival curve in RTC	$A \iff B$
strict service curve in NC	minimum service curve in NC	$A \implies B$
lower service curve in RTC	strict service curve in NC	$A \implies B$
upper service curve in RTC	maximum service curve in NC	$A \implies B$

Table 6.3: The summary of relations among curves in RTC and NC

6.2.1 Comparison of arrival curves

Based on the definitions in Section 3.2.1, the upper arrival curve in RTC is equivalent to the arrival curve in NC. As a counterpart, the definition of lower arrival curve in NC is adopted from [72], which lower bounds the number of arrived bits in NC and is equivalent to lower arrival curve in RTC by definition.

Definition 6.2.1 (Lower Arrival Curve in NC)**.** *A lower arrival curve α^L describes the minimum number of arrived bits on a flow with input function $R(t)$ in any time interval iff for all*

$0 \leq s \leq t$*, there holds*

$$R(t) - R(s) \geq \alpha^L(t-s),$$

which is equivalent to say:

$$R(t) \geq R\bar{\otimes}\alpha^L,$$

.

6.2.2 Comparison of service curves

In both NC and RTC, the amount of available resource is described with (different types of) service curves. However, what the definitions in these two frameworks emphasize is different. In NC, service curve expresses more about how much resource is used by some input, that is, how much service is provided, while in RTC, it addresses the capability of the system, no matter whether there is an input or how much traffic the input brings. That is why the definitions of service curves in NC always involve an input. In this section we first explore the relations between strict service curve and minimum service curve in NC, and then the relations between lower/upper service curve in RTC and strict/maximum service curve in NC[1].

Theorem 6.2.1. *If a node offers* β_{strict} *as a strict service curve to a flow with input function* $R(t)$ *and output function* $R^*(t)$*, and then it also offers* β_{strict} *as a minimum service curve to the flow.*

Proof. For any time $t \geq 0$, consider the output $R^*(t)$. There are two cases: t is either in a backlogged period or not.

[1]Note that in the following analysis, we assume that the start time of the system is 0, which is consistent with the definitions of arrival and service curves in both RTC and NC.

If t is in a backlogged period, let s_0 denote the start time of the backlogged period, implying all arrivals that have arrived before time s_0 have been served by s_0 and hence $R^*(s_0) = R(s_0)$. We then have:

$$\begin{aligned} R^*(t) &= R^*(s_0) + R^*(t) - R^*(s_0) \\ &= R(s_0) + R^*(t) - R^*(s_0) \\ &\geq R(s_0) + \beta_{strict}(t - s_0) \geq (R \otimes \beta_{strict})(t) \end{aligned}$$

If t is not in any backlogged period[2], let t_0 denote the finish time of the latest backlogged period before t. Then $R(t_0) = R(t)$ and $R^*(t_0) = R^*(t)$, since otherwise we would not have had t_0 being the finish time of that backlogged period. In addition, the definition of t_0 also means that all arrivals that have arrived before time t_0 have been served by t_0, and hence $R^*(t_0) = R(t_0)$. Combining these, we consequently have $R^*(t) = R(t)$. Finally in this case, together with $\beta_{strict} = 0$, we obtain

$$R \otimes \beta_{strict} = \inf_{0 \leq s \leq t} \{R(t) - R(s) + \beta_{strict}(s)\} \leq R(t) = R^*(t)$$

Then the theorem is proved. $\square$

Corollary 2. *Let β_{min}^{sup} denote the supremum among all minimum service curves offered by a node in NC, then $\beta_{strict} \leq \beta_{min}^{sup}$.*

Next we show the relation between lower service curve in RTC and strict service curve in NC.

Theorem 6.2.2. *If a node offers a lower service curve β^l to a flow with input function $R(t)$ and output function $R^*(t)$, then it offers strict service curve β^l, also minimum service curve β^l to the flow.*

[2]This case is not considered in [23].

Proof. Consider any backlogged period from s to t. In this period, the node will be busy serving, implying no capacity remaining in the period after serving the input. Hence, $R^*(0,t] - R^*(0,s] = C(0,t] - C(0,s]$. Since the node offers lower service curve, i.e. $C(0,t] - C(0,s] \geq \beta^l(t-s)$, we hence have

$$R^*(t) - R^*(s) = R^*(0,t] - R^*(0,s] \geq \beta^l$$

This proves that the node offers a strict service curve β^l. In addition, since strict service curve implies minimum service curve in NC, β^l is a minimum service curve to the flow. □

Corollary 3. *Let β^{sup}_{strict} denote the supremum among all strict service curves offered by a node in NC, then $\beta^l \leq \beta^{sup}_{strict}$.*

Theorem 6.2.3. *If a node offers an upper service curve β^u to a flow with input function $R(t)$ and output function $R^*(t)$, then it offers a maximum service curve β^u to the flow.*

Proof. For any time $t \geq 0$, consider the output $R^*(0,t]$. For any $0 \leq s \leq t$, by the definition of upper service curve together with the fact $R^*(0,s] \leq R(0,s]$, there holds: $\forall 0 \leq s \leq t$,

$$\begin{aligned} R^*(t) = R^*(0,t] &= R^*(0,s] + R^*(0,t] - R^*(0,s] \\ &\leq R(0,s] + R^*(0,t] - R^*(0,s] \\ &\leq R(s) + \beta^u(t-s) \end{aligned}$$

Hence

$$R^*(t) \leq \inf_{0 \leq s \leq t} \{R(s) + \beta^u(t-s)\} = R \otimes \beta^u(t)$$

□

Corollary 4. *Let β^{inf}_{max} denote the infimum among all maximum service curves offered by a node in NC, then $\beta^u \geq \beta^{inf}_{max}$.*

6.3 Pay-Burst-Only-Once in RTC

The Pay-Burst-Only-Once property has been assumed to be true in RTC and much work has already applied it for deriving tighter delay bound. However, none of these work directly proves the correctness of applying the Pay-Burst-Only-Once property in RTC, but simply refers to the proof in NC. This direct mapping seems not so convincing since the concatenation property, based on which the Pay-Burst-Only-Once property is proved in NC, does not hold in RTC.

Theorem 6.3.1. *If β_1^l, β_2^l are lower service curves offered by two nodes S_1, S_2 respectively, $\beta_1^l \otimes \beta_2^l$ is not necessarily a lower service curve offered by the concatenation of these two nodes.*

Proof. Based on Theorem 6.2.2, if a node S_i offers β_i^l as a lower service curve, then it offers β_i^l as a strict service curve. Since it has been proved in [23] that the concatenation of strict service curves does not necessarily generate a strict service curve, the concatenation of lower service curve is not necessarily a lower service curve, so the concatenation property does not hold for lower service curve in RTC. Then the theorem is proved. $\square$

However, although the concatenation property does not hold in RTC, the Pay-Burst-Only-Once property can be proved in RTC. Next we first explain the basic idea of the proof and then give the formal deduction.

According to Section 3.2.1, the delay bound a flow experiences when it traverses a concatenated system in RTC is calculated with a similar method as in NC and the difference lies in the service curve involved: in RTC lower service curve is adopted while in NC it is minimum service curve. To prove the Pay-Burst-Only-Once property in RTC, we first calculate the delay bound in RTC and NC respectively. Then by utilizing the numerical relations among curves explored in Section 6.2, we compare the derived delay bounds and then prove the property.

Theorem 6.3.2. *The Pay-Burst-Only-Once property holds in RTC.*

Proof. The proof of the theorem involves two aspects:

(1) Safety. The delay bound derived with convolved lower service curve is no smaller than maximum delay experienced by the flow.

(2) Tightness. The delay bound derived with convolved lower service curve is no larger than summing up the delay bound at each node.

Assume that a flow with upper arrival curve α^u in RTC, arrival curve α^U in NC traverses a sequence of m nodes, each offering a lower service curve $\beta_i^l, i = 1, 2, ...m$, the supremum of strict service curves of node S_i is denoted by $\beta_{strict,i}^{sup}$, and supremum of minimum service curves of node S_i is denoted by $\beta_{min,i}^{sup}$.The upper service curve offered by each node is denoted by β_i^u, and the infimum of maximum service curves is denoted by $\beta_{max,i}^{inf}$.

We first prove (1). According to Section 3.2.1, the maximum delay experienced d_{max} satisfies

$$d_{max} \leq H(\alpha^U, \beta_{min,1}^{sup} \otimes \beta_{min,2}^{sup} ... \otimes \beta_{min,m}^{sup})$$

Since arrival curve in NC is equivalent to upper arrival curve in RTC, we have $\alpha^u = \alpha^U$.

According to Section 6.2, $\beta_i^l \leq \beta_{strict,i}^{sup} \leq \beta_{min,i}^{sup}$, then we have

$$\beta_1^l \otimes \beta_2^l ... \otimes \beta_m^l \leq \beta_{strict,1}^{sup} \otimes \beta_{strict,2}^{sup} ... \otimes \beta_{strict,m}^{sup} \leq \beta_{min,1}^{sup} \otimes \beta_{min,2}^{sup} ... \otimes \beta_{min,m}^{sup}$$

Then

$$H(\alpha^u, \beta_1^l \otimes \beta_2^l ... \otimes \beta_m^l) \geq H(\alpha^U, \beta_{min,1}^{sup} \otimes \beta_{min,2}^{sup} ... \otimes \beta_{min,m}^{sup})$$

So

$$d_{max} \leq H(\alpha^u, \beta_1^l \otimes \beta_2^l ... \otimes \beta_m^l)$$

showing that the delay bound calculated with concatenated lower service curve in RTC is safe.

Next we prove (2). Since $\beta_i^l \leq \beta_{strict,i}^{sup} \leq \beta_{min,i}^{sup}$, there must exist a minimum service curve $\beta_{min,i} = \beta_i^l$ for each node S_i. Since $\beta_i^u \geq \beta_{max,i}^{inf}$, there must exist a maximum service curve $\beta_{max,i} = \beta_i^u$ for each node S_i. According to the Pay-Burst-Only-Once property in NC, we have

$$H(\alpha^U, \beta_{min,1} \otimes \beta_{min,2} ... \otimes \beta_{min,m}) \leq H(\alpha_0'^U, \beta_{min,1}) + H(\alpha_1'^U, \beta_{min,2}) + ... + H(\alpha_{m-1}'^U, \beta_{min,m})$$

Then by replacing $\beta_{min,i}$ with β_i^l, replacing $\beta_{max,i}$ with β_i^u, and replacing α^U with α^u, we have

$$H(\alpha^u, \beta_1^l \otimes \beta_2^l ... \otimes \beta_m^l) \leq H(\alpha_0'^u, \beta_1^l) + H(\alpha_1'^u, \beta_2^l) + ... + H(\alpha_{m-1}'^u, \beta_m^l)$$

Then (2) is proved[3]. □

6.4 Computation of output arrival curves

After verifying the correctness of the Pay-Burst-Only-Once property in RTC, we move on to compute the output arrival curves when concatenating the nodes a flow traverses. Similar as the proof of Theorem 6.3.2, this thesis first explores how to calculate output arrival curves in NC, then prove that the corresponding results are applicable in RTC.

Since the derivation of (upper) output arrival curve is known in NC, we focus on how to calculate the lower output arrival curve in NC.

[3]Here we assume $\alpha_i'^U = (\alpha_{i-1}'^U \otimes \beta_{max,i}) \oslash \beta_{min,i}$ and $\alpha_i'^u = (\alpha_{i-1}'^u \otimes \beta_i^u) \oslash \beta_i^l$

Theorem 6.4.1. *Consider a node offers a minimum service curve β_{min} and maximum service curve β_{max} to the input flow $R(t)$ with arrival curve α^U and lower arrival curve α^L in NC. Then output flow has a lower output arrival curve α'^L with*

$$\alpha'^L = [(\alpha^L \otimes \beta_{min}) \wedge (\beta_{min} \bar{\oslash} \alpha^U)]^+.$$

Proof. Since the output in any time interval can not be larger than the input, we have $R^*(t) \leq R(t)$.

Based on the definitions of minimum service curve and maximum service curve, for any time t we have $(R \otimes \beta_{min})(t) \leq R^*(t) \leq (R \otimes \beta_{max})(t)$.

Based on the definitions of upper and lower arrival curve, we have

$$(R \bar{\otimes} \alpha^L)(t) \leq R(t) \leq (R \otimes \alpha^U)(t).$$

Then

$$\begin{aligned}
R^*(t-s) &= R^*(t) - R^*(s) \\
&\geq (R \otimes \beta_{min})(t) - (R \otimes \beta_{max})(s) \\
&= \inf_{0\leq u\leq t}[R(t-u) + \beta_{min}(u)] - \inf_{0\leq v\leq s}[R(s-v) + \beta_{max}(v)] \\
&= \inf_{0\leq u\leq t} \sup_{0\leq v\leq s} [R(t-u) - R(s-v) + \beta_{min}(u) - \beta_{max}(v)] \\
&= \sup_{0\leq v\leq s} \inf_{0\leq u\leq t}[R(t-u) - R(s-v) + \beta_{min}(u) - \beta_{max}(v)] \\
&= \sup_{0\leq v\leq s} [\inf_{0\leq u\leq t-s+v}[R(t-u) - R(s-v) + \beta_{min}(u) - \beta_{max}(v)] \\
&\wedge \inf_{t-s+v\leq u\leq t}[R(t-u) - R(s-v) + \beta_{min}(u) - \beta_{max}(v)]] \\
&= \sup_{0\leq v\leq s} \inf_{0\leq u\leq t-s+v}[R(t-u) - R(s-v) + \beta_{min}(u) - \beta_{max}(v)]
\end{aligned}$$

$$\wedge \sup_{0\leq v\leq s} \inf_{t-s+v\leq u\leq t}[R(t-u)-R(s-v)+\beta_{min}(u)-\beta_{max}(v)] \tag{6.1}$$

$$\geq \sup_{0\leq v\leq s} \inf_{0\leq u\leq t-s+v}[\alpha^L(t-u-s+v)+\beta_{min}(u)-\beta_{max}(v)]$$

$$\wedge \inf_{t-s\leq u\leq t}[R(t-u)-R(s)+\beta_{min}(u)] \tag{6.2}$$

$$= \sup_{0\leq v\leq s} \inf_{0\leq u\leq t-s+v}[\alpha^L(t-u-s+v)+\beta_{min}(u)-\beta_{max}(v)]$$

$$\wedge \inf_{t-s\leq u\leq t}[\beta_{min}(u)-(R(s)-R(t-u))]$$

$$= \sup_{0\leq v\leq s}[(\alpha^L\otimes\beta_{min})(t-s+v)-\beta_{max}(v)]$$

$$\wedge \inf_{0\leq u'\leq s}[R(u')-R(s)+\beta(t-u')] \tag{6.3}$$

$$\geq (\alpha^L\otimes\beta_{min})(t-s)\wedge \inf_{0\leq u'\leq s}[\beta_{min}(t-u')-\alpha^U(s-u')] \tag{6.4}$$

$$= (\alpha^L\otimes\beta_{min})(t-s)\wedge \inf_{0\leq u'\leq s}[\beta_{min}(t-s+s-u')-\alpha^U(s-u')]$$

$$= (\alpha^L\otimes\beta_{min})(t-s)\wedge \inf_{0\leq u''\leq s}[\beta_{min}(t-s+u'')-\alpha^U(u'')]$$

$$\geq (\alpha^L\otimes\beta_{min})(t-s)\wedge \inf_{0\leq u''}[\beta_{min}(t-s+u'')-\alpha^U(u'')] \tag{6.5}$$

$$= (\alpha^L\otimes\beta_{min})\wedge(\beta_{min}\bar{\oslash}\alpha^U) \tag{6.6}$$

where $\wedge$ gets the minimum of the two parts it connects; in obtaining (6.2) we take $v=0$ of the second part of (6.1); (6.4) is to take $v=0$ in the first item of (6.3) and apply upper arrival curve to $R(s)-R(u')$ in the second half of (6.3); (6.5) is because of taking larger range for infimum operation. □

Based on the definition of lower arrival curve, for any interval Δ, $\alpha'^L(\Delta)\geq 0$, then we have $\alpha'^L = [(\alpha^L\otimes\beta_{min})\wedge(\beta_{min}\bar{\oslash}\alpha^U)]^+$. Then we know how to calculate the output arrival curves after concatenation in RTC.

Corollary 5. *Assume an event stream with arrival curve α^u,α^l traverses m GPC components with service curves $(\beta_1^u,\beta_1^l),(\beta_2^u,\beta_2^l),...,(\beta_m^u,\beta_m^l)$, then the output arrival curve at*

$m - th$ *component is upper and lower bounded by:*

$$\alpha'^u = (\alpha^u \otimes \beta^u) \oslash \beta^l$$

$$\alpha'^l = [(\alpha^l \otimes \beta^l) \wedge (\beta^l \bar{\oslash} \alpha^u)]^+$$

where $\beta^u = \beta_1^u \otimes \beta_2^u ... \otimes \beta_m^u$, $\beta^l = \beta_1^l \otimes \beta_2^l ... \otimes \beta_m^l$.

Proof. The proof is similar to that of Theorem 6.3.2 with the combination of Theorem 6.2.1, 6.2.2 and 6.2.3. □

CHAPTER 7

IMPROVING GLOBAL EDF USING SHAPERS

7.1 Introduction

It is widely accepted that future real-time embedded systems will be deployed on multiprocessors, to meet their rapidly increasing requirements of both high computational capacity and low power consumption. One of the crucial requirements that must be satisfied by a multiprocessor real-time system is that it can provide bounded delay, which describes the duration from each task activation to its execution completion. Deriving the delay bound of each task in a system during the design stage is important for both hard and soft real-time systems, since delay bound can not only be used to test the schedulability of hard real-time systems, but perform as an indicator for system performance of soft real-time systems. This thesis considers the analysis of delay bound under GEDF for multiprocessor systems.

Global Earlist Deadline First (GEDF) scheduling is widely used and studied since it incurs less preemptions and migrations compared with optimal scheduling and the maximum delay is bounded when it is used to schedule tasks in soft real-time systems. Most of existing work for calculating delay bound on multiprocessors under GEDF analyzes sporadic tasks [35, 38, 39, 54], where any two consecutive jobs of task τ_i are assumed to be released with minimum separation time T_i. However in real applications burst exists, where a relatively large number of jobs are released over a short interval. Comparing sporadic and bursty inputs, the minimum separation time between two consecutive jobs in different intervals keeps more stable for the former, and it changes a lot for the latter. As a result, the sporadic task model denoting sporadic inputs is not a good choice for bursty inputs. When setting the period of a sporadic task as the minimum separation time during bursty interval in modeled bursty inputs, the over-estimated workload leads to pessimistic results, and setting that as the minimum separation time during more smooth intervals generates a wrong model due to under-estimated workload. Further, when jobs arrive simultaneously, the minimum separation time can not be specified. In this case, the period does not exist and bursty inputs can not be modeled as sporadic tasks.

As a result, most existing analysis techniques considering sporadic tasks can not directly be used to analyze systems with bursty inputs. What's more, it is difficult to extend the analysis techniques to bursty inputs since the complex job sequences make it hard to specify the worst case of execution. Compared with sporadic tasks, the work analyzing bursty tasks is much less. And for the problem under our setting, there is only one [55]. However, the delay bound derived by [55] is very large compared with the real requests of each input task. In summary, none of existing work provides a satisfactory solution for the analysis of bursty inputs .

This thesis proposes a new approach to calculate delay bound for a multiprocessor system with bursty tasks under GEDF. In detail, bursty inputs are modeled as bursty tasks and shapers are deployed for input bursty tasks. Once job sequences generated by bursty tasks enter the system, they first go through corresponding shapers, after which the behaviors of these jobs conform to sporadic patterns. Then these output jobs of shapers go into the scheduler and complete execution. With shapers, existing analysis techniques analyzing sporadic tasks can be applied (in this work the techniques in [54] are adopted), and the delay bound of each task is calculated. To further improve our approach, this thesis designs a heuristic algorithm which can increase the number of tasks meeting their deadlines in a task set by adjusting settings of shapers. Experiments show that the proposed algorithm can improve the acceptance ratio, and the derived delay bound is much smaller compared with the state-of-art.

7.2 Overview

Basic idea. In existing work about calculating the delay bound for tasks scheduled on multiprocessors under global EDF, jobs are assumed to directly enter the ready queue of the scheduler as soon as being released. Under this assumption, the more complex the released job sequence is, the harder it is to accurately analyze the worst-case amount of interference suffered by the analyzed task. As a result, it is difficult to analyze the delay bound of each task.

The difficulty brought by complex job sequences to scheduling analysis can be sidestepped when deriving the delay bound, which is implemented in our work by the adoption of shapers. We assume that jobs first go through shapers then enter the ready queue of scheduler, rather than directly being ready for scheduling after being released as in existing work. Using the traffic control function of shapers, complex job sequences are shaped to conform to sporadic patterns, where the separation time between two consecutive jobs is no smaller than a constant, and then existing analysis techniques targeting sporadic tasks are applied.

A system with shapers behaves as follows: after a job is released by a task, it first goes through a shaper. The shaper checks the separation between release time of current job and the output time of its predecessor job. If it is no shorter than the supposed constant specified by the shaping function, the shaper outputs the job instantaneously and then the job enters the ready queue of the scheduler. Otherwise, the shaper buffers the job and outputs it as soon as the minimum separation time is satisfied. In total, the delay a job experiences is from its release to leaving the shaper to completing execution at the scheduler, rather than the single part at the scheduler. And the delay bound of a task is the maximum delay among that of all its jobs.

System architecture. A system with shapers eliminating burst in inputs is shown in Figure 7.1. For each bursty task τ_i, a greedy shaper S_i is deployed to shape the job sequence released by it. Each shaper has a buffer, which is used to store jobs that can not enter the ready queue of the scheduler due to violation of minimum separation time constraint. We assume that the buffer size of each shaper equals 0 when the first job is released and is large enough so that there is no job loss. Then the input of the scheduler is transformed from bursty tasks (when without shapers) to sporadic tasks and the delay bound becomes analyzable.

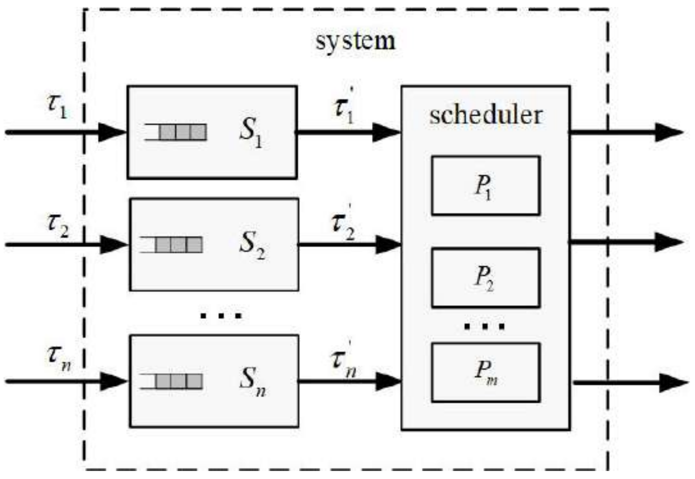


Figure 7.1: The system architecture with shapers.

7.3 Analyzing the system with shapers

This section introduces the approach to derive the delay bound for a multiprocessor real-time system with bursty inputs under global EDF by the adoption of shapers. We first model the scheduler as an abstract component. Then the system with shapers can be modeled as sequential components where the outputs of shapers constitute one of the inputs of the scheduler component. After that, we show how to compute the delay bound and adjust shapers to increase the number of schedulable tasks in a task set.

7.3.1 Modeling the scheduler

This subsection first models the input job sequence of the scheduler, then models EDF scheduler as an abstract component, and specifies how to calculate its inputs and outputs. Note that original input tasks and shapers are assumed to be given.

In a system with shapers, the burst in original input job sequences is eliminated and the inputs to the scheduler conform to a sporadic pattern. Based on this, we define *virtual* *sporadic task*:

Definition 7.3.1 (Virtual sporadic task)**.** *Suppose that the job sequence released by task τ_i passes a greedy shaper S_i with shaping function $\theta_i(\Delta) = \lceil\frac{\Delta}{T_s^i}\rceil$. Then the output job sequence of S_i has minimum separation time T_s^i, and is modeled by an implicit-deadline sporadic task τ_i'. τ_i' is called the virtual sporadic task of τ_i, whose period equals T_s^i and WCET equals that of τ_i.* [1]

Next we model an EDF scheduler as an EDF component.

Definition 7.3.2 (EDF component)**.** *A scheduler that schedules a task set $\tau = \{\tau_1, \tau_2, ..., \tau_n\}$ under global EDF is modeled as an EDF component, which is characterized by $(\overrightarrow{C}, \overrightarrow{\alpha^U}, \overrightarrow{\beta}, \overrightarrow{\alpha'^U})$, where $\overrightarrow{C} = \{C_1, C_2, ..., C_n\}$ is the WCET of each task, $\overrightarrow{\alpha^U} = \{\alpha_1^U, \alpha_2^U, ..., \alpha_n^U\}$ is the arrival curve, $\overrightarrow{\beta} = \{\beta_{min,1}, \beta_{min,2}, ..., \beta_{min,n}\}$ denotes the service curve provided for each task, and $\overrightarrow{\alpha'^U} = \{\alpha_1'^U, \alpha_2'^U, ..., \alpha_n'^U\}$ is the arrival function of processed jobs.*

In the framework shown in Figure 7.1, the inputs to EDF scheduler are virtual sporadic tasks. So for an EDF component, each element in $\overrightarrow{C}$ equals the WCET of corresponding input bursty task. Next we will show how to derive $\overrightarrow{\alpha^U}$, $\overrightarrow{\beta_{min}}$, $\overrightarrow{\alpha'^U}$ for an EDF component.

1) Deriving $\overrightarrow{\alpha^U}$

Suppose the periods of virtual sporadic tasks which are inputs to an EDF scheduler are: $\overrightarrow{T_s} = \{T_s^1, T_s^2, ..., T_s^n\}$, then the arrival function of each input task of EDF component is denoted by:

$$\alpha_i^U(\Delta) = \lceil\frac{\Delta}{T_s^i}\rceil$$

[1]Note that virtual task τ_i' does not exist in real systems and is defined only for analysis. Also, the deadline of a virtual sporadic task is more used as an indicator for scheduling priority than a constraint for completion time.

Then we have

$$\overrightarrow{\alpha^U} = \{\lceil\frac{\Delta}{T_s^1}\rceil, \lceil\frac{\Delta}{T_s^2}\rceil, ..., \lceil\frac{\Delta}{T_s^i}\rceil\}$$

2) Deriving $\overrightarrow{\beta_{min}}$ and $\overrightarrow{\alpha'^U}$

Suppose the delay bound experienced by each input task at EDF scheduler is known as $DLY_{schd} = \{DLY_{schd,1}, DLY_{schd,2}, ..., DLY_{schd,n}\}$, then $\overrightarrow{\beta_{min}}$ and $\overrightarrow{\alpha'^U}$ can be derived. Next we will first introduce the derivation of delay bound of each task at EDF scheduler part, and then show how to derive $\overrightarrow{\beta_{min}}$ and $\overrightarrow{\alpha'^U}$ with DLY_{schd}.

Since each input task of EDF scheduler is a virtual sporadic task in our system, the delay bound experienced can be calculated by existing analysis techniques analyzing sporadic tasks scheduled under global EDF. We first model the analysis process of deriving the delay bound at the scheduler:

Definition 7.3.3 (Calculation of delay bound). *Assume that the input of a global EDF scheduler is a task set of n sporadic tasks. The task set is characterized by $\overrightarrow{C^*} = \{C_1^*, C_2^*, ..., C_n^*\}$, $\overrightarrow{T^*} = \{T_1^*, T_2^*, ..., T_n^*\}$, and $\overrightarrow{D^*} = \{D_1^*, D_2^*, ..., D_n^*\}$, which denote each sporadic task's WCET, period and relative deadline respectively. Suppose the analysis technique ξ is adopted to derive the delay bound $DLY_{schd} = \{DLY_{schd,1}, DLY_{schd,2}, ..., DLY_{schd,n}\}$ at the scheduler for each task in the task set, then the analysis process can be modeled as a function $Caldelay$ with*

$$DLY_{schd} = Caldelay(\xi, \overrightarrow{C^*}, \overrightarrow{T^*}, \overrightarrow{D^*})$$

In our work, we adopt analysis techniques in [54] (denoted by ξ_G) and the delay bound at the scheduler can be calculated as

$$DLY_{schd} = Caldelay(\xi_G, \overrightarrow{C}, \overrightarrow{T_s}, \overrightarrow{T_s})$$

, where $\overrightarrow{C}$ equals the WCET and $\overrightarrow{T_s} = \{T_s^1, T_s^2, ..., T_s^n\}$ equals the period of n virtual sporadic tasks.

Then $\overrightarrow{\beta_{min}}$ and $\overrightarrow{\alpha'^U}$ can be derived based on DLY_{schd}.

The service curve $\beta_{min,i}$ provided for each virtual task τ_i' can be derived based on DLY_{schd} and Corollary 1:

$$\beta_{min,i}(\Delta) = \delta_{DLY_{schd,i}}(\Delta) \begin{cases} 0, & 0 \le \Delta \le DLY_{schd,i} \\ +\infty, & \Delta > DLY_{schd,i} \end{cases}$$

Each output arrival function is calculated as $\alpha_i'^U(\Delta) = \alpha_i^U(\Delta - DLY_{schd,i})$, so we have

$$\overrightarrow{\alpha'^U} = \{\alpha_1^U(\Delta - DLY_{schd,1}), \alpha_2^U(\Delta - DLY_{schd,2}), ..., \alpha_i^U(\Delta - DLY_{schd,i})\}$$

7.3.2 Calculating the delay bound

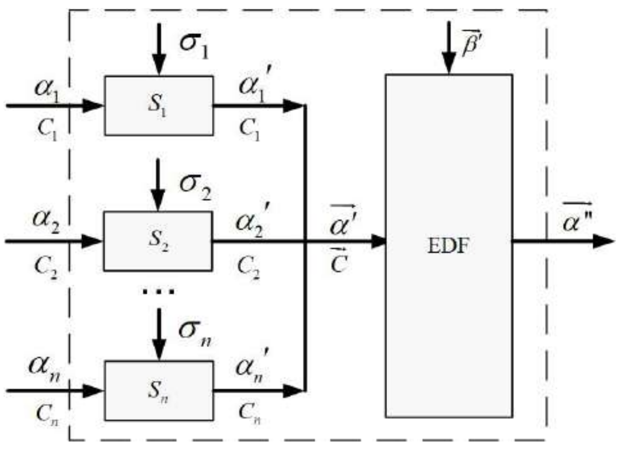


Figure 7.2: Modeling the system in Figure 7.1 into a set of abstract components.

After modeling the EDF scheduler as EDF component, the system in Figure 7.1 is modeled into a network composed of n greedy shapers and one EDF component in Figure 7.2. Based on formula (3.13) and (3.12), to derive the delay bound of each task traversing the system with shapers, the service function provided by each shaper component and EDF component must be known. In this subsection, we assume that $\sigma_i(\Delta)$ is given, based on which we can derive the period of each virtual sporadic task, the delay bound at the scheduler DLY_{schd}, and $\overrightarrow{\beta_{min}}$. Next we explore the remaining unknown parameter, that is, the service function provided by each shaper.

Lemma 7.3.1. *Assume a job sequence generated by task τ_i with WCET C_i passes a shaper with shaping function σ_i, then the shaper provides to the task a service curve β_{min}^s denoted by $\beta_{min}^s = C_i * \sigma_i$.*

Proof. The lemma can be easily proved with Corollary 1.5.1 in [21]. □

Now we can calculate the delay bound DLY_i for each task τ_i.

Lemma 7.3.2. *The delay bound of each task τ_i is calculated as $DLY_i = DLY_{s,i} + DLY_{schd,i}$.*

Proof. Based on the concatenation property, the delay bound is computed as

$$DLY_i = Del(C_i, \alpha_i^U, (C_i * \sigma_i(\Delta)) \otimes \beta_{min,i})$$

, where $\beta_{min,i} = \delta_{DLY_{schd,i}}$.

Based on the property of $\beta_{min,i}$,

$$(C_i * \sigma_i(\Delta)) \otimes \beta_{min,i} = C_i * \sigma_i(\Delta - DLY_{schd,i})$$

Then we have

$$\begin{aligned} DLY_i &= Del(C_i, \alpha_i^U, C_i * \sigma_i(\Delta - DLY_{schd,i})) \\ &= Del(C_i, \alpha_i^U, C_i * \sigma_i(\Delta)) + DLY_{schd,i} \\ &= Del(1, \alpha_i^U, \sigma_i(\Delta)) + DLY_{schd,i} \\ &= DLY_{s,i} + DLY_{schd,i} \end{aligned}$$

Then the lemma is proved. □

From Lemma 7.3.2, the delay bound of each task equals the sum of that at the shaper and at the scheduler. Also, given fixed inputs, processors and analysis techniques for sporadic tasks, the shaping function of each shaper uniquely decides the delay bound of each task.

7.3.3 The design of shapers

Different from above two subsections where shapers are known, in this subsection they are not given and become the target of our analysis. We will discuss how to implement a shaper with sporadic output and specify settings of shapers to increase the number of schedulable tasks.

1) The implementation of greedy shaper

At the first sight, a token bucket seems to be one choice for generating sporadic outputs. However, the following example shows that the output job sequence of a token bucket can not satisfy the minimum separation time constraint in all cases.

An example. Suppose that a token bucket with token generation rate $1/T_s$ is used to generate a job sequence with minimum separation time T_s. Consider that two jobs arrive at the same time at $a(1)$ and the bucket is full with one token generated at $e(1)$ with $e(1) \leq a(1)$. At $a(1)$, one of the two jobs can pass the shaper and the other is buffered. Then at $e(1) + T_s$, a new token is generated and the second job can be emitted. However, the separation time between it and the first job is $e(1) + T_s - a(1) \leq T_s$. The example shows that only when $e(1) = a(1)$, the minimum separation time constraint can be guaranteed by the token bucket. However, $e(1) = a(1)$ is only one special case and the behavior of token bucket when $e(1) = a(1)$ can not represent general cases.

Based on this observation, we define an interval-guaranteeing shaper whose output always satisfies the minimum separation time constraint.

Definition 7.3.4 (Interval-guaranteeing shaper [2]). *[30] An interval-guaranteeing shaper outputs jobs at $e(i)$ satisfying*

$$e(1) = a(1); e(i) = max\{a(i), e(i-1) + T_s\}$$

[2]An interval-guaranteeing shaper is a type of greedy shaper.

where $a(i)$ denotes the arrival time of job i to the shaper, and T_s is the minimum separation time in the output job sequence.

Lemma 7.3.3. *The shaping function $\sigma(\Delta)$ of an interval-guaranteeing shaper satisfies $\sigma(\Delta) = \lceil \frac{\Delta}{T_s} \rceil$, where T_s is the minimum separation time in its output.*

Proof. Can be obtained based on the behavior of an interval-guaranteeing shaper. □

2) Specify the setting of shapers

Now we have already designed a shaper with shaping function $\sigma(\Delta) = \lceil \frac{\Delta}{T_s} \rceil$ to implement minimum separation time T_s in its output job sequence. From last two subsections, for fixed input tasks executed on a multiprocessor platform, the periods of shapers uniquely decide the delay bound. Next we focus on how to adjust the value of period T_s^i of each shaper S_i to make more tasks schedulable.

To guarantee the system is not overloaded and jobs buffered at the shaper are limited, the following two conditions must be satisfied:

$$\sum_{i=1}^{n} \frac{C_i}{T_s^i} \leq m \tag{7.1}$$

$$0 \leq T_s^i \leq \frac{1}{\eta_i} \tag{7.2}$$

A simple choice to set $T_s^i = \frac{1}{\eta_i}$ with η_i defined in Section 3.2.1. However, this can not guarantee the task meets its deadline. Actually, the shaper that makes a task meet its deadline can not be derived with simple observation and the reason is two-fold. First, the total delay bound is the sum of that at the shaper and that at the scheduler. Although larger period leads to lower utilizations and intuitively smaller delay bound in the part of EDF scheduler, it causes larger delay bound at the shaper. So it is hard to decide whether to increase or decrease the shaper's period without real computation. Second, changing the period of shaper corresponding to one virtual sporadic task will influence the execution of other virtual sporadic tasks. So the choice of period can not be independent.

To make more tasks schedulable, it is necessary to enumerate all possible values of T_s^i satisfying formula (7.1) and (7.2). The process can be time-consuming and we propose a simple heuristic to increase the number of schedulable tasks.

Algorithm 7.3.1 Decide the period of each shaper

```
1: for each shaper $S_i$, $T_s^i = 1/\eta_i$
2: calculate $DLY_i = DLY_{s,i} + DLY_{sched,i}$
3: calculate *sched* and *unsched*
4: $num_{sched} = size(sched)$
5: sort *unsched* with descending $DLY - D$
6: sort *sched* with descending $D - DLY$
7: $num = size(\tau)$
8: **while** $unsched(\tau)$ and $num_{sched} < num$ **do**
9:     $num_{sched} = size(sched)$
10:    **while** notempty(*unsched*) **do**
11:        current $\longleftarrow$ pop the first from *unsched*
12:        **while** $unsched(\tau)$ and $candecrease(T_s^{current})$ **do**
13:            $T_s^{current} = T_s^{current} - 1$
14:        **end while**
15:    **end while**
16:    calculate *sched* and *unsched*
17:    **while** notempty(*sched*) **do**
18:        current $\longleftarrow$ pop the first from *sched*
19:        **while** $unsched(\tau)$ and $canincrease(T_s^{current})$ **do**
20:            $T_s^{current} = T_s^{current} + 1$
21:        **end while**
22:    **end while**
23:    calculate *sched* and *unsched*
24:    $num = size(sched)$
25: **end while**
```

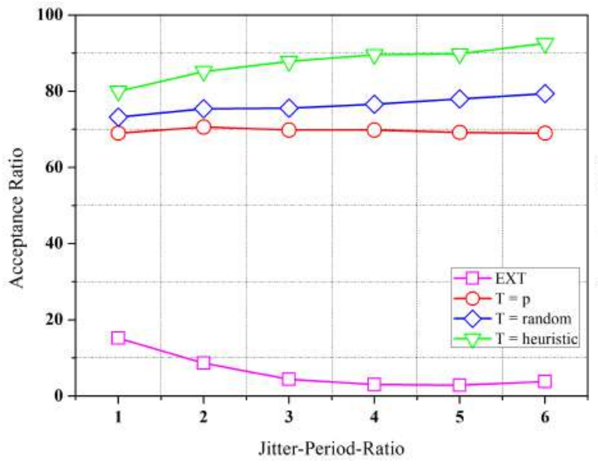
In Algorithm 7.3.1, τ is the input task set. A shaper is put into set *sched* if its corresponding bursty task meets its deadline, else is put into *unsched*. We first decrease the period of each shaper in *unsched* on the condition that the number of shapers in *sched* does not decrease and the corresponding bursty task's delay bound is smaller than before. After adjusting the periods of all shapers in *unsched*, we increase the period of each shaper in *sched* on the condition that its corresponding bursty task does not miss its deadline. For each iteration, we check whether the number of shapers in *sched* increases. If not, the iteration stops.

An example. We consider a task set composed of 5 tasks $\tau = \{\tau_1, \tau_2, ..., \tau_5\}$. The arrival function α_i of each task τ_i is characterized by $\alpha_i^U(\Delta) = \lceil \frac{\Delta + j_i}{p_i} \rceil$. The parameters and calculated delay bounds are shown in Table 7.1. DLY is the calculated delay bound when setting $T_s^i = p_i$. T' is randomly generated period for shapers on the condition that formula (7.1) and (7.2) are satisfied, and DLY' is corresponding delay bound. T'' is period derived with Algorithm 7.3.1, and DLY'' is the corresponding delay bound.

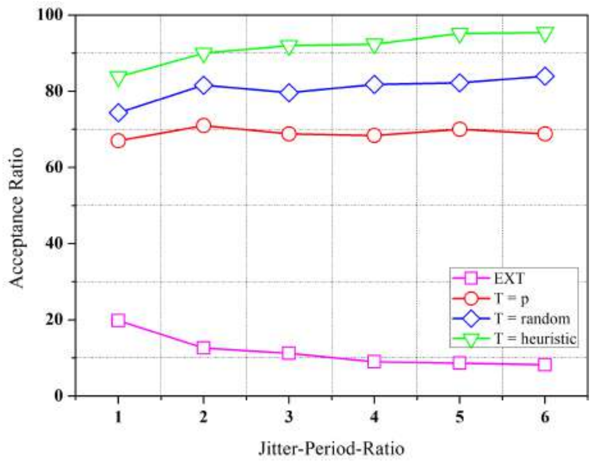
From the table, in the first two settings of shapers, some task will miss its deadline. After adjusting the period with the heuristic, all tasks meet their deadlines and the original task set becomes schedulable.

	C	p	j	λ	D	T'	T''	DLY	DLY'	DLY''
τ_1	4	24	24	16	36	12	10	62	39	34
τ_2	4	8	8	8	36	7	8	30	29	30
τ_3	12	16	16	16	56	14	16	54	51	54
τ_4	8	12	12	12	52	11	12	42	41	42
τ_5	4	28	28	24	40	22	12	70	59	38

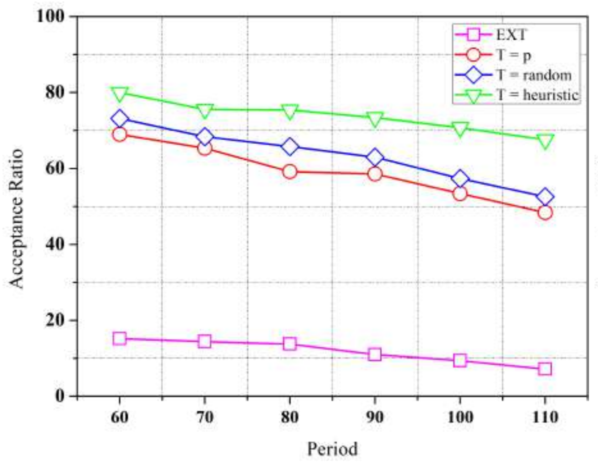
Table 7.1: A task set and its delay bound calculated with different settings of shapers.



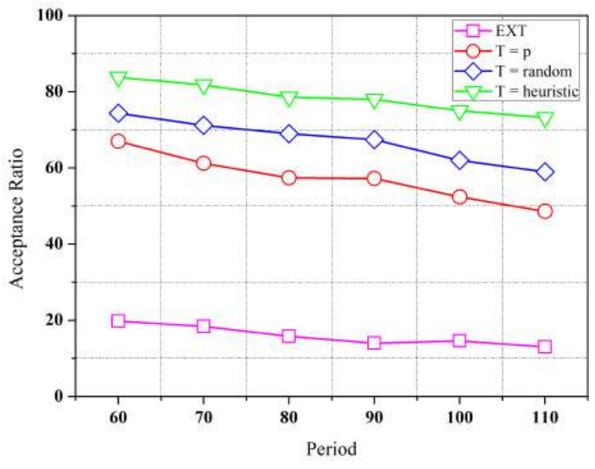
(a) Different jitter/period, $d = 0$.



(b) Different jitter/period, $d \neq 0$.



(c) Different period, $d = 0$.



(d) Different period, $d \neq 0$.

Figure 7.3: The comparison of acceptance ratio

7.4 Experimental evaluation

We implement our proposed approach in RTC Toolbox [1] and conduct experiments to evaluate the performance.

Task generation. We first generate its arrival function based on the PJD workload model in RTC Toolbox [83] characterized by (p, j, d).

The range of p, j, d will be specified in following content and j/p is set to be an integer in all experiment settings. After generating p, j, d, other parameters are randomly chosen in following ranges: $C \in [1, p]$, $\lambda \in [C, p]$, and $D \in [p * \frac{j}{p} + C + p, p * \frac{j}{p} + C + p + 500]$.

For each task set, we generate 5 tasks. Then the total utilization U of the task set is calculated as $U = \sum_{i=1}^{5} \frac{C_i}{p_i}$, and the number of processors m is set as $m = ceil(U)$.

In this section, we compare the performance of 4 approaches:

(1) the approach in [55], denoted by $'EXT'$.

(2) the approach proposed in our work with setting $T_s^i = p_i$, denoted by $'T = p'$.

(3) the approach proposed in our work with randomly generated T_s^i satisfying formula (7.1) and (7.2), denoted by $'T = random'$.

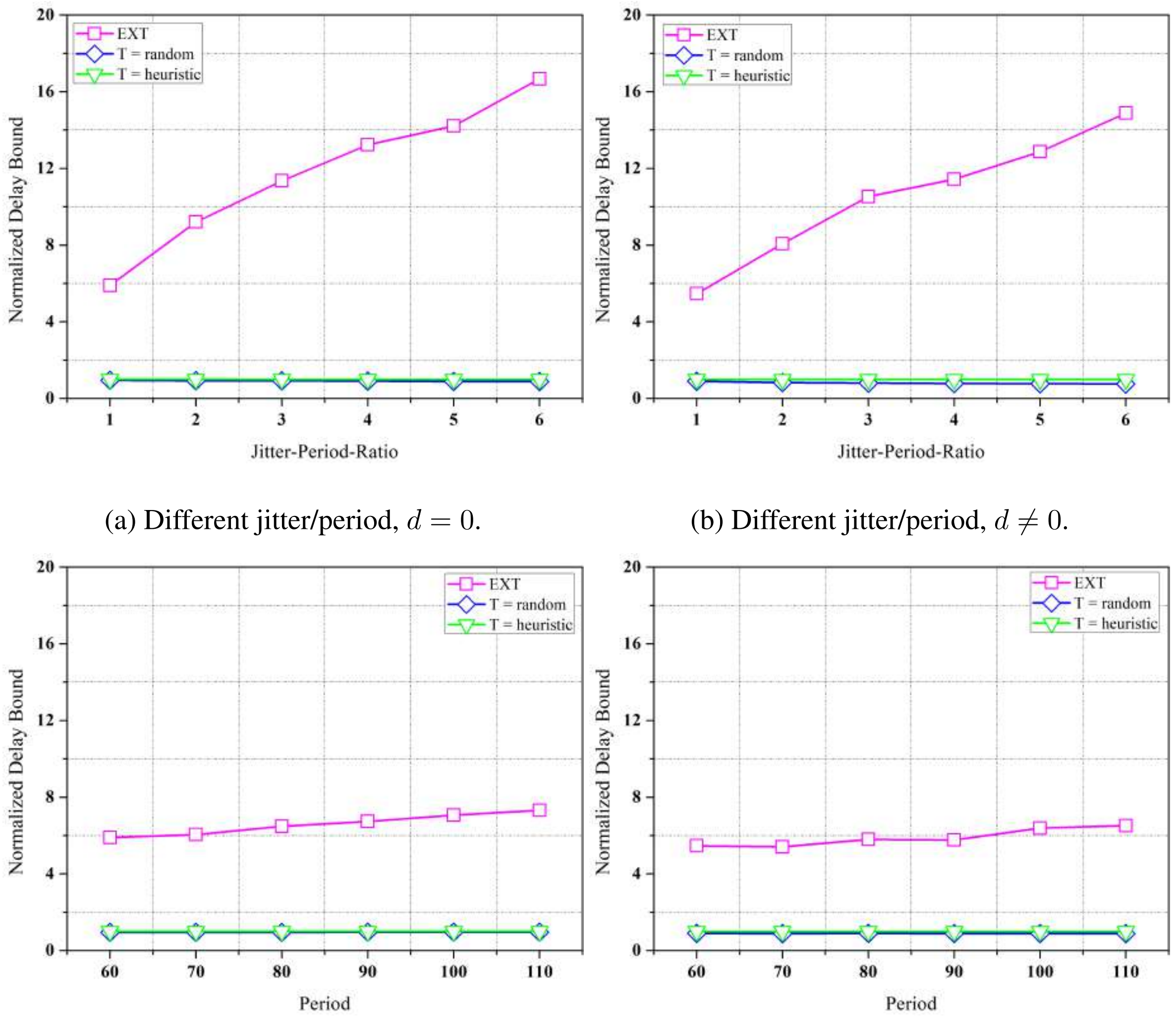
(4) the approach proposed in our work with the period setting derived with Algorithm 7.3.1, denoted by $'T = heuristic'$.

Note that in (2) (3) (4) we adopt the techniques in [54] at EDF scheduler part, T_s^i is the period of shaper corresponding to task τ_i and p_i is defined in the generation of task τ_i.

Experimental results. The above 4 approaches are compared in two aspects: acceptance ratio and normalized delay bound.

1) Acceptance ratio

Figure 7.3-(a) and (b) show the comparison among 4 approaches under different jitter-period-ratio (X-axis). We generate 500 task sets for each different jitter-period-ratio. In Figure 7.3-(a), p is randomly generated in $[1, 60]$ and $d = 0$. $d = 0$ means that jobs can be released simultaneously. Figure 7.3-(b) has the same parameter setting as Figure 7.3-(a) except that d is randomly generated in $[1, p]$, meaning that no two jobs are released at the same time. The Y-axis shows the percentage of schedulable task sets among all task sets generated for each jitter-period-ratio, denotedy by *Acceptance Ratio*. Based on the experiment results, our approach performs better than $'EXT'$ regardless of the jitter-period-ratio and settings of shapers. In detail, with the increasing of jitter-period-ratio, the acceptance ratio of $'EXT'$ decreases, since jobs released simultaneously can potentially lead to unfair allocation of processor resources, causing large incremental in calculated delay bound. What's more, the improvement by the heuristic increases with jitter-period-ratio, since same



(a) Different jitter/period, $d = 0$.

(b) Different jitter/period, $d \neq 0$.

(c) Different period, $d = 0$.

(d) Different period, $d \neq 0$.

Figure 7.4: The comparison of normalized delay bound

decrease of shaper period can potentially lead to more delay reduction at the shaper. The performance under $d \neq 0$ is slightly better than $d = 0$, since when $d \neq 0$ no two jobs are released simultaneously and the delay experienced by each task at the shaper is comparatively smaller.

Figure 7.3-(c) and (d) show the comparison among 4 approaches under different range of p (X-axis). For each value i in X-axis, we generate 500 task sets with $p \in [1, i]$. In Figure 7.3-(c), $j/p = 1$ and $d = 0$. Figure 7.3-(d) has the same parameter setting as 7.3-(c) except that $d \in [1, p]$. It can be seen that $'T = heuristic'$ still has the best performance.

2) Normalized delay bound

Figure 7.4-(a) and (b) show the comparison among 4 approaches under different jitter-period-ratio (X-axis). The parameter settings and corresponding number of task sets are same as Figure 7.3-(a) and (b) respectively. We choose the delay bound calculated with $'T = p'$ as the standard, and the Y-axis shows the ratio between the delay bounds derived with other three different approaches and $'T = p'$, namely *Normalized delay bound*, calculated as $\frac{\sum_{i=1}^{5} DLY_{comp,i}/DLY_{p,i}}{5}$, where $DLY_{p,i}$ is the delay bound of task τ_i calculated with $'T = p'$, and $DLY_{comp,i}$ is that calculated with one of other three approaches. Each result for X-axis value i is the average of the normalized delay bound of task sets with jitter-period-ratio equal to i.

Figure 7.4-(c) and (d) show the comparison among 4 approaches under different range of p (X-axis). The parameter settings and corresponding number of task sets are same as Figure 7.3-(c) and (d) respectively.

Experiments show that the delay bound calculated by our methods is much smaller than that by $'EXT'$ under both different jitter-period-ratio and range of p.

CHAPTER 8

CONCLUSION AND FUTURE WORK

8.1 Conclusion

Timing constraints are of great significance in real-time systems and developing analysis techniques to determine the validity of a real-time system deserves the most concern. RTC is a powerful real-time performance analysis framework for networked systems, providing the most expressive and effective analysis for timing behaviors. This thesis revisits Real-Time Calculus, improves the analysis of fundamental abstract components and proves the correctness of basic properties, thus improving the overall timing analysis of real-time systems under RTC framework. In detail, this thesis improves existing RTC theory in the following aspects:

(1) Improve the analysis of fundamental components, GPC and AND Connector. For GPC, the thesis revises the proof for output arrival curves and proposes two methods to improve the analysis precision. Experiment results show that these two methods can improve the calculation results in original GPC under different cases, and neither dominates the other. For AND, the thesis fixes a fundamental issue in existing analysis and proposes a more precise and efficient approach to analyze the outputs in multi-input AND. The advantages of improvements with regard to precision and efficiency are demonstrated in corresponding experiments.

(2) Prove the correctness of Pay-Burst-Only-Once property in RTC. This work is the theoretical basis for obtaining tighter delay bound in a networked system by concatenating sequential nodes.

(3) Extend NC to multiprocessors and improve global EDF with the integration of real-time scheduling theory. Experiments compare the acceptance ratio and relative quality of delay bound calculated by the new method and the only existing work, and show that the new method can derive much tighter delay bound.

It is worth noting that the improvements above all contribute to the same target: they greatly improve the delay bound analysis in complex real-time systems. Delay bound is one of the most important criteria for timing performance evaluation, and work in this thesis undoubtedly improves the overall analysis of real applications.

8.2 Future work

For future research, we plan to extend analysis techniques in RTC/NC and real-time systems to other time critical network environments.

(1) Timing analysis in Time-Sensitive Networking (TSN): TSN is a set of standards under development by the Time-Sensitive Networking task group of the IEEE 802.1 working group which gets more attention of the industry these years. TSN has superiorities of low jitter, low latency and deterministic transmission, and is applicable to systems with cruel requirements of transmission delay, and especially provides strong support for the development of industrial Internet. The rapid development of industrial Internet has put forward much higher requirements for communication network addressing the transmission of very low latency and high availability. The improvement of manufacturing technology and the refinement of control process all need a network capable of fast real-time communication to support, whereas traditional Ethernet is difficult to meet the needs of various industrial scenarios. TSN will play a significant role in the industrial Internet.

The scheduling in TSN. There is no scheduling mechanism based on service class priorities that can effectively guarantee bounded end-to-end delay of time-sensitive data load. Due to queuing, if a low-priority Ethernet frame is already in the transmission process, it will delay the transmission of other Ethernet frames, even if these frames have the highest priority.

This can happen every time the transmission path is switched or routed. In order to solve this problem, sufficient and necessary conditions for schedulability, throughput bounds, and other classic measures in TSNs need to be explored.

Traffic shaping. Different traffic classes with various priorities coexist on the same network, and each has different requirements to available bandwidth and end-to-end latency. The traditional 'Best-effort' transmission mode has very poor predictability and is difficult to predict when a 'Best-effort' packet is transmitted, which will lead to potential transmission conflicts. We plan to study the mechanisms to manage the bursty traffics (shaper / traffic regulation) with regards to throughput and delay performance.

The cooperation and synchronization. To guarantee the latency requirement of different services, all devices in this network need to have a common time reference and therefore, it is needed to synchronize their clocks among each other. Only through synchronized clocks, it is possible for all network devices to operate in union and execute the required operation at exactly the required point in time. Furthermore, it is necessary to conduct closer cooperation between devices globally in a TSN. For the above reasons, we plan to investigate the time synchronization approach and global management of timing in TSNs.

(2) The scheduling timing analysis in other networks: Another direction focuses on extending the RTC/NC analysis techniques to different networks, e.g., DNNs and SoCs. Nowadays, we see the applications of DNNs in many applications, such as self-driving cars. On one hand, traditional empirical evaluation approaches for the efficiency of DNNs (e.g., inference period) can hardly meet the timing predictability requirements in many hard real-time environments. We will seek to collaborate RTC and the real time analysis techniques with the inference latency analysis of DNNs. On the other hand, there are rarely scheduling techniques involved for improving the efficiency of DNNs since the computation among layers is sequential. However, the interconnection among layers in more recent CNN algorithms, e.g. ResNets and Google Inception, are more complex, and it is necessary to manage the execution sequences by developing real-time scheduling algorithms to meet the latency requirement in hard real-time systems.

REFERENCES

[1] Real-Time Calculus (RTC) Toolbox. http://www.mpa.ethz.ch/Rtctoolbox.

[2] The SymTA/S. http://www.symta.org.

[3] Y. Abdeddaim, A. Kerbaa, and O. Maler. Task graph scheduling using timed automata. In *Proceedings of the 17th International Symposium on Parallel and Distributed Processing (IPDPS)*, 2003.

[4] K. Altisen and M. Moy. Causality problem in real-time calculus. In *Formal Methods in System Design*, 2016.

[5] R. Alur and D. L. Dill. The theory of timed automata. In *Proceedings of the Real-Time: Theory in Practice, REX Workshop*, 1991.

[6] B. Anderson and J. Jonsson. Fixed-priority preemptive multiprocessor scheduling: To partition or not to partition. In *Proceedings of the International Workshop on Real-Time Computing Systems and Applications (RTCSA)*, 2000.

[7] N. C. Audsley, A. Burns, M. F. Richardson, and A. J. Wellings. Hard real-time scheduling: The deadline-monotonic approach. In *IFAC/IFIP Workshop on Real Time Programming (WRTP)*, 1991.

[8] T.P. Baker. Multiprocessor edf and deadline monotonic schedulability analysis. In *Proceedings of 24th IEEE Real-Time Systems Symposium (RTSS)*, 2003.

[9] T.P. Baker. An analysis of edf schedulability on a multiprocessor. *IEEE Transactions on Parallel and Distributed Systems*, 16(8):760–768, 2005.

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

[11] S. K. Baruah, A. K. Mok, and L. E. Rosier. Preemptively scheduling hard-real-time sporadic tasks on one processor. In *Proceedings of 11th Real-Time Systems Symposium (RTSS)*, 1990.

[12] S.K. Baruah. Techniques for multiprocessor global schedulability analysis. In *Proceedings of 28th IEEE International Real-Time Systems Symposium (RTSS)*, 2007.

[13] H. Bauer, J. Scharbarg, and C. Fraboul. Applying and optimizing trajectory approach for performance evaluation of afdx avionics network. In *Proceedings of Emerging Technologies and Factory Automation (ETFA)*, 2009.

[14] V. Berten, P. Courbin, and J. Goossens. Gang fixed priority scheduling of periodic moldable real-time tasks. In *Proceedings of the Junior Researcher Workshop Session of the 19th International Conference on Real-Time and Network Systems (JRWRTNS)*, 2011.

[15] S. Bondorf and J.Schmitt. Improving cross-traffic bounds in feed-forward networks - there is a job for everyone. *Remke A., Haverkort B.R. (eds) Measurement, Modeling and Evaluation of Dependable Computer and Communication Systems, Lecture Notes in Computer Science*, 9629:9–24, 2016.

[16] S. Bondorf, P. Nikolaus, and J. B. Schmitt. Quality and cost of deterministic network calculus: design and evaluation of an accurate and fast analysis. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 1(1):15:1–15:34, 2017.

[17] S. Bondorf, P. Nikolaus, and J. B. Schmitt. Catching corner cases in network calculus ? flow segregation can improve accuracy. In *International Conference on Measurement, Modelling and Evaluation of Computing Systems (MMB 2018)*, 2018.

[18] S. Bondorf and J. B. Schmitt. Boosting sensor network calculus by thoroughly bounding cross-traffic. In *Proceedings of IEEE International Conference on Computer Communications (INFOCOM)*, 2015.

[19] S. Bondorf and J. B. Schmitt. Calculating accurate end-to-end delay bounds - you better know your cross-traffic. In *Proceedings of the 9th EAI International Conference on Performance Evaluation Methodologies and Tools (VALUETOOLS' 15)*, 2015.

[20] V. Bonifaci, A. Marchetti-Spaccamela, S. Stiller, and A. Wiese. Feasibility analysis in the sporadic dag model. In *Proceedings of Euromicro Conference on Real-Time Systems (ECRTS)*, 2013.

[21] J. L. Boudec and P. Thiran. Network calculus - a theory of deterministic queuing systems for the internet. In *LNCS 2050*. Springer Verlag, 2001.

[22] A. Bouillard. Composition of service curves in network calculus. *Proceedings of the 1st International Workshop on Worst-Case Traversal Time (WCTT)*, 2011.

[23] A. Bouillard, L. Jouhet, and E. Thierry. Service curves in network calculus: dos and don'ts. Research Report, 2009.

[24] A. Bouillard, L. Jouhet, and E. Thierry. Comparison of different classes of service curves in network calculus. *IFAC Proceedings Volume*, 43(12):306–311, 2010.

[25] A. Bouillard, L. Jouhet, and E. Thierry. Tight performance bounds in the worst-case analysis of feed-forward networks. In *Proceedings of IEEE International Conference on Computer Communications (INFOCOM)*, 2010.

[26] A. Bouillard, L. Jouhet, and E. Thierry. Tight performance bounds in the worst-case analysis of feed-forward networks. In *Proceedings of IEEE International Conference on Computer Communications (INFOCOM)*, 2010.

[27] A. Bouillard and G. Stea. Exact worst-case delay in fifo-multiplexing feed-forward networks. *IEEE/ACM Transactions on Networking*, 23(5):1387–1400, 2015.

[28] S. Chakraborty, S. Künzli, and L. Thiele. A general framework for analysing system properties in platform-based embedded system designs. In *Proceedings of Design, Automation and Test in Europe Conference and Exhibition (DATE)*, 2003.

[29] S. Chakraborty, L. T. X. Phan, and P. S. Thiagarajan. Event count automata: a state-based model for stream processing systems. In *Proceedings of 26th IEEE International Real-Time Systems Symposium (RTSS)*, 2005.

[30] C. S. Chang. Performance guarantees in communication networks. 2000.

[31] G. Chen, K. Huang, C. Buckl, and A. Knoll. Applying pay-burst-only-once principle for periodic power management in hard real-time pipelined multiprocessor systems. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 20(2):26:1–26:27, 2015.

[32] E.M. Clarke and E.A. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. *Kozen D. (eds) Logics of Programs. Lecture Notes in Computer Science*, 131:52–71, 1981.

[33] R. I. Davis and A. Burns. A survey of hard real-time scheduling for multiprocessor systems. *ACM Computing Surveys (CSUR)*, 43(4):1–35, 2011.

[34] U. Devi and J. Anderson. Tardiness bounds for global edf scheduling on a multiprocessor. In *Proceedings of 26th IEEE International Real-Time Systems Symposium (RTSS)*, 2005.

[35] U. Devi and J. Anderson. Tardiness bounds for global edf scheduling on a multiprocessor. *Real-Time Systems*, 38(2):133–189, 2008.

[36] U. C. Devi. Soft real-time scheduling on multiprocessors. In *Ph.D. Thesis, Chapel Hill*, 2006.

[37] Z. Dong and C. Liu. Analysis techniques for supporting hard real-time sporadic gang task systems. In *Proceedings of 38th IEEE Real-Time Systems Symposium (RTSS)*, 2017.

[38] J.P. Erickson, U. Devi, and S.K. Baruah. Improved tardiness bounds for global edf. In *Proceedings of 22nd Euromicro Conference on Real-Time Systems (ECRTS)*, 2010.

[39] J.P. Erickson, N. Guan, and S.K. Baruah. Tardiness bounds for global edf with deadlines different from periods. In *International Conference On Principles Of Distributed Systems (OPODIS)*, 2010.

[40] E. Fersman, L. Mokrushin, P. Pettersson, and W. Yi. Schedulability analysis of fixed-priority systems using timed automata. *Theoretical Computer Science - Tools and algorithms for the construction and analysis of systems*, 354(2):301–317, 2006.

[41] Fabien Geyer and Georg Carle. Network engineering for real-time networks: comparison of automotive and aeronautic industries approaches. *IEEE Communications Magazine*, 54(2):106–112, 2016.

[42] N. Guan, Y. Tang, Y. Wang, and W. Yi. Delay analysis of structural real-time workload. In *Proceedings of Design, Automation and Test in Europe Conference and Exhibition (DATE)*, 2015.

[43] N. Guan and W. Yi. Finitary real-time calculus: Efficient performance analysis of distributed embedded systems. In *Proceedings of 34th IEEE Real-Time Systems Symposium (RTSS)*, 2013.

[44] A. Hamann, M. Jersak, K. Richter, and R. Ernst. Design space exploration and system optimization with symta/s symbolic timing analysis for systems. In *Proceedings of 25th IEEE International Real-Time Systems Symposium (RTSS)*, 2004.

[45] R. Henia, A. Hamann, M. Jersak, R. Racu, K. Richter, and R. Ernst. System level performance analysis - the symta/s approach. *IEEE proceedings on Computers and Digital Techniques*, 152(2):148–166, 2005.

[46] X. Jiang, N. Guan, X. Long, and W. Yi. Semi-federated scheduling of parallel real-time tasks on multiprocessors. In *Proceedings of 38th IEEE Real-Time Systems Symposium (RTSS)*, 2017.

[47] X. Jiang, X. Long, N. Guan, and H. Wan. Decomposition-based global edf scheduling of parallel real-time tasks. In *Proceedings of 37th IEEE Real-Time Systems Symposium (RTSS)*, 2016.

[48] P. Kumar and L. Thiele. Cool shapers: shaping real-time tasks for improved thermal guarantees. In *Proceedings of the 48th Design Automation Conference (DAC)*, 2011.

[49] K. Lampka, S. Bondorf, and J. B. Schmitt. Achieving efficiency without sacrificing model accuracy: Network calculus on compact domains. In *Proceedings of IEEE 24th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*, 2016.

[50] K. Lampka, S. Bondorf, J. B. Schmitt, N. Guan, and W. Yi. Generalized finitary real-time calculus. In *Proceedings of IEEE International Conference on Computer Communications (INFOCOM)*, 2017.

[51] K. Lampka, S. Perathoner, and L. Thiele. Analytic real-time analysis and timed automata: a hybrid method for analyzing embedded real-time systems. In *Proceedings of the 9th ACM and IEEE International Conference on Embedded software (EMSOFT)*, 2009.

[52] K. Lampka, S. Perathoner, and L. Thiele. Analytic real-time analysis and timed automata: a hybrid methodology for the performance analysis of embedded real-time systems. *Design Automation for Embedded Systems*, 14(3):193–227, 2010.

[53] J. Lehoczky. Fixed priority scheduling of periodic task sets with arbitrary deadlines. In *Proceedings of 11th IEEE Real-Time Systems Symposium (RTSS)*, 1990.

[54] H. Leontyev and J. Anderson. Generalized tardiness bounds for global multiprocessor scheduling. *Real-Time Systems*, 44(1):26–71, 2010.

[55] H. Leontyev, S. Chakraborty, and J. Anderson. Multiprocessor extensions to real-time calculus. *Real-Time Systems*, 47(562):1–49, 2011.

[56] J.Y.-T Leung and M. Merrill. A new algorithm for scheduling periodic, real-time tasks. *Information processing letters*, 11(3):115–118, 1980.

[57] J.Y.-T Leung and J. Whitehead. On the complexity of fixed-priority scheduling of periodic, real-time tasks. *Performance Evaluation*, 2(4):237–250, 1980.

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

[59] Y. Liu and Y. Jiang. Stochastic network calculus. Springer Verlag, 2008.

[60] Z. Manna and A. Pnueli. The temporal logic of reactive and concurrent systems: Specification. Springer verlag, 1991.

[61] A. Melani, M. Bertogna, V. Bonifaci, A. Marchetti-Spaccamela, and G. C. Buttazzo. Real-time scheduling of parallel tasks under a general dag model. In *Technical Report, Washington University in St. Louis*, 2015.

[62] M. Mohaqeqi, J. Abdullah, N. Guan, and W. Yi. Schedulability analysis of synchronous digraph real-time tasks. In *Proceedings of Euromicro Conference on Real-Time Systems (ECRTS)*, 2016.

[63] A. K. Mok. Fundamental design problems of distributed systems for the hard-real-time environment. In *PhD thesis, Cambridge*, 1983.

[64] A. K. Mok and D. Chen. A multiframe model for real-time tasks. *IEEE Transactions on Software Engineering*, 23(10):635–645, 1997.

[65] M. Moy and K. Altisen. Arrival curves for real-time calculus: the causality problem and its solutions. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, 2010.

[66] L. T. X. Phan, S. Chakraborty, P. S. Thiagarajan, and L. Thiele. Composing functional and state-based performance models for analyzing heterogeneous real-time systems. In *Proceedings of 28th IEEE International Real-Time Systems Symposium (RTSS)*, 2007.

[67] L.T.X. Phan and I. Lee. Improving schedulability of fixed-priority real-time systems using shapers. In *Proceedings of 19th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2013.

[68] K. Richter. Compositional scheduling analysis using standard event models. In *Ph.D. Thesis, Technical University Carolo-Wilhelmina of Braunschweig*, 2005.

[69] K. Richter, M. Jersak, and R. Ernst. A formal approach to mpsoc performance verification. *Computer*, 36(4):60–67, 2003.

[70] A. Saifullah, K. Agrawal, C. Lu, and C. Gill. Multi-core real-time scheduling for generalized parallel task models. In *Proceedings of 32th IEEE Real-Time Systems Symposium (RTSS)*, 2011.

[71] A. Saifullah, D. Ferry, K. Agrawal, C. Lu, and C. Gill. Real-time scheduling of parallel tasks under a general dag model. In *Technical Report, Washington University in St Louis*, 2012.

[72] H. Schioler, J. J. Jessen, J. D. Nielsen, and K. G. Larsen. Network calculus for real time analysis of embedded systems with cyclic task dependencies. In *Proceedings of 20th International Conference on Computers and Their Applications (CATA)*, 2005.

[73] J. B. Schmitt, F. A. Zdarsky, and M. Fidler. Delay bounds under arbitrary multiplexing: When network calculus leaves you in the lurch... In *Proceedings of IEEE International Conference on Computer Communications (INFOCOM)*, 2008.

[74] J. B. Schmitt, F. A. Zdarsky, and I. Martinovic. Improving performance bounds in feed-forward networks by paying multiplexing only once. In *Proceedings 14th GI/ITG Conference on Measurement, Modelling and Evaluation of Computer and Communication Systems (MMB)*, 2008.

[75] M. Stigge. Real-time workload models expressiveness vs. analysis efficiency. In *Ph.D. Thesis, Uppsala University*, 2014.

[76] M. Stigge, P. Ekberg, N. Guan, and W. Yi. The digraph real-time task model. In *Proceedings of 17th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2011.

[77] M. Stigge, P. Ekberg, N. Guan, and W. Yi. On the tractability of digraph-based task models. In *Proceedings of Euromicro Conference on Real-Time Systems (ECRTS)*, 2011.

[78] M. Stigge, N. Guan, and W. Yi. Refinement-based exact response-time analysis. In *Proceedings of Euromicro Conference on Real-Time Systems (ECRTS)*, 2014.

[79] M. Stigge and W. Yi. Hardness results for static priority real-time scheduling. In *Proceedings of Euromicro Conference on Real-Time Systems (ECRTS)*, 2012.

[80] M. Stigge and W. Yi. Combinatorial abstraction refinement for feasibility analysis. In *Proceedings of 34th IEEE Real-Time Systems Symposium (RTSS)*, 2013.

[81] L. Thiele, S. Chakraborty, and M. Naedele. Real-time calculus for scheduling hard real-time systems. In *Proceedings of IEEE International Symposium on Circuits and Systems (ISCAS)*, 2000.

[82] E. Wandeler. Modular performance analysis and interface-based design for embedded real-time systems. In *Ph.D. Thesis, Swiss Federal Institute Of Technology Zurich*, 2006.

[83] E. Wandeler, A. Maxiaguine, and L. Thiele. Performance analysis of greedy shapers in real-time systems. In *Proceedings of the conference on Design, Automation and Test in Europe (DATE)*, 2006.