The Dissertation Committee for Weirong Wang
certifies that this is the approved version of the following dissertation:

Integration of Hard Real-Time Schedulers

Committee:

Aloysius K. Mok, Supervisor

James C. Browne

Deji Chen

Mohamed G. Gouda

C. Greg Plaxton

Integration of Hard Real-Time Schedulers

by

Weirong Wang, BS, MA

Dissertation

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Doctor of Philosophy

The University of Texas at Austin

December 2004

To my parents: Liu, Aifang and Wang, Shengchuan

Acknowledgments

This dissertation is about resource scheduling. Scheduling algorithms, no matter how powerful they are, can not handle any workload correctly without a set of conditions guaranteed by the resources. The same is true with an academic endeavor. I would like to provide an incomplete account of the privileges and favors I got from other people here.

Thanks to my advisor, Professor Aloysius Mok, for your education, support and inspiration. You are a genial mentor and a great inspirator. I will leave your research group with a great appreciation of being helped. Thank you for giving me the privilege of having a heavy burden of expectations when graduating. I will try my best to meet these expectations in the years to come.

Thanks to my committee, Professor James Browne, Dr. Deji Chen, Professor Mohamed Gouda and Professor Greg Plaxton. Your ideas, especially those discussed in the proposal meeting, make a significant and positive influence to this research.

Thanks to Professor Plaxton. Your algorithm course in Fall 1997 has direct and deep impact on my research. I learned the network-flow problem and its solutions in that course. My solution to the unit-size Window-Constrained (WC) problem is based on network-flow construction. Although the topic of WC is not included in this dissertation, my first academic publication was on it. The round-and-compensate approach, which is included in this dissertation, is inspired by some techniques used in the network-flow analysis too.

Thanks for all teachers helped me along the way. Special thanks to the following teachers: the vice principals of No.123 Middle School of Beijing, Chen, Ying and Bi, Jieguang, who sent me to my first course of computer programming in the summer of 1984; Zhang, Jingcheng, my primary middle school teacher, who believed that I would make a difference and brought me to believe in that; Lu, Chuanjiang, my physical education teacher in middle school, whose intensive training shaped both my muscles and mind; the faculty in the Department of Computer Science in Beijing University of Technology, who delivered a reasonably good undergraduate education in Computer Science and Engineering; the faculty in University of Texas at Austin, who largely formated the intellectual context of this research.

Thanks to my wife, Ye, Hong. You have been sharing the heavy burden of my ambition. You encouraged me when I got frustrated, and you shared my happiness when progresses were made. You helped me reviewing my research papers. Most importantly, you took housework as much as you can, allowing me some more time to work on my research. Not every wife and working mother in this world has the patience of living a graduate student's life for years, particularly when "earlier" alternatives are quite available. I have been feeling deeply indebted.

To my daughter Rona and my son Kyler, you are my sunshine, and I have one piece of advice for you here. Your father hadn't finished schooling when the older of you started it. This is not the fun part of graduate study. Finish your dissertation earlier in your life if you ever want to do it.

Thanks to my mother, Liu, Aifang. You passed me some of your ambition, diligence, and the unreasonable self-confidence, which is a necessity for sailing with only 70% chance for reaching the destiny. In 1991, I visited the factory mass producing the air cleaning machines you designed, and I felt proud to be your son. You helped thousands and thousands of people to breathe cleaner air in harsh working environments. They will probably live healthier and longer. I call your work an accomplishment. I can just wish that I could also leave such a positive impact to the world. You will always be an inspiration of mine.

Thanks to my father, Wang, Shengchuan. You brought me to enjoy the pleasure of intelligence. I cherish the winter day when we investigated the pieces of ice together on "the Little Ditch", and the night that you woke me up 2 am to observe a moon eclipse. Three decades later, I am still not ready to abandon the intoxication of curiosity and exploration, which is also a necessity of staying in graduate program while big money seemed to be just out there. My graduate study actually started informally when you give me some math and physics problems to solve and let me take time to find my own way out. This is an advantage I had over the text-book and exam oriented school education which dominates in my childhood. Professor Mok is my 1^{st} graduate advisor, and you are the 0^{th} of mine. I will forward this family tradition to Rona and Kyler.

To both of my parents: I am sure that you would have done your Ph.D degrees and produced some excellent results if you had had my opportunities. You've done as good as you can under your social and historical context. Let me dedicate this dissertation to you. Remember the picture printed on our 1974 calendar? The peak of Zhu-Mu-Lang-Ma, peaceful, clean, cool, and high. Let us always keep that picture in our hearts.

WEIRONG WANG

The University of Texas at Austin
December 2004

Integration of Hard Real-Time Schedulers

Publication No. ____________________

Weirong Wang, Ph.D.

The University of Texas at Austin, 2004

Supervisor: Aloysius K. Mok

Over the last few decades, numerous research results have been obtained on scheduling specific real-time workloads to run on dedicated resources. In the last few years, research in scheduler composition on shared resources has attracted increasing attention for the following reasons. The capacities of resources in real-time embedded systems, such as processors, communications channels, have been growing rapidly. These hardware advances create possibilities for more complex and integrated functionalities that share the same resources. Heterogeneous workloads are now allocated to shared resources in contemporary designs. The complexity of the scheduler is accordingly increased. Approaches in scheduler composition have been proposed as a divide-and-conquer strategy to deal with the complexity of scheduler design for these integrated systems.

Most of the scheduler composition approaches that have been proposed can be treated within a framework of two-layers: coordinator and components. This dissertation covers our contributions in these two layers, namely, Class-based Component Composition (CCC) approach in the layer of coordinating mechanisms and pre-scheduling in the layer of component construction.

We propose CCC for composing independent components in an open environment. CCC uses a workload classification scheme to guarantee that the supply of shared resource always meets the hard-real-time constraints for on-budget workloads. It also aims to achieve a balance over multiple design objectives including composition overhead, overload handling and accommodating the range of real-time applications.

A pre-schedule is a static schedule that does not require constant and completely predictable rate of resource supply. We present a sound, complete, and PTIME basic pre-scheduler based on Linear Programming (LP). Since infinitely small slices of time are not implementable in time-domain multiplexing for resources with non-negligible context switch overheads, it is desirable to define and solve the pre-scheduling problem on the domain of integers. We construct a rational-to-integral pre-schedule transformer based on a novel technique which we call "round-and-compensate". This transformer is sound, complete and runs in PTIME. We also present an extension of the basic pre-scheduler for solving precedence constraints, and show two examples on how to do resource supply analysis in our framework.

Contents

List of Tables

List of Figures

Chapter 1

An Introduction to Real-Time Scheduler Composition

1.1 Background

In early hard real-time systems, the capacities of resources, such as the execution rates of processors and bandwidths of communication channels, were usually quite limited. Therefore a resource was often used by one or at most a few functions, and the computational complexity of resource scheduling was not a priority issue. The primary research goal of real-time scheduling was to maximize the utilization of resources. The workload is usually modeled as a set of tasks or jobs, and they are scheduled by a monolithic scheduler.

The resource capacity in computer-based systems has improved greatly and the price of resources has been dropping ever since the early days.. The improvement in capacity/price ratios presents opportunities in two directions. Horizontally, more functions in a system can now be controlled by computer based device. Take the electronic controls in an automobile as an example. When micro controllers were slow and expensive, they were applied only to the critical subsystems, such as engine control; when micro controllers have become powerful and cheap, they can be used for controlling multiple components of the power train, and even for auxiliary subsystems such as mirrors and doors. The control over subsystems can be integrated to improve system performance and functionality. For instance, the control over all major components of the power train can be integrated in order to promote handling performance and gas efficiency.

New challenges in resource scheduling have emerged as real-time systems become more complex. First, the size of a typical system increases as the number of features to be implemented increases; therefore the computational complexity of scheduling increases. Second, the workloads have become more heterogeneous; i.e., each workload for implementing certain function(s) may present a different set of temporal assumptions and requirements to be met. Third, in "open" systems, new workloads might need to be admitted online. Scheduling decisions must be made upon the available information about the workload. However, the information might not be completely known at design time, or even at online admission time.

A monolithic scheduler may not be capable of managing a large set of heterogeneous and partially unpredictable workloads. Once again, the wisdom of divide-and-conquer can be applied to solve a complex prolbem. In this dissertation, the technique of divide-and-conquer takes the form of "scheduler composition".

1.2 Coordinator/Component Framework for Scheduler Composition

Compositional scheduling schemes have been proposed in the real-time research community in recent years [4, 20, 23, 17, 25]. All of these composition approaches follow a coordinator/component framework. There are two layers in this framework. At the top layer, there might be a "coordinator" and some communication and regulatory mechanisms. At the bottom layer, there are a number of "components". Each component may have a workload and its internal scheduling mechanism. The coordinator collects information from the components and resolves the resource competition between them; each component makes a local decision on how to make use of a resource when the resource is assigned to it. In this dissertation, we shall assume that the coordinator/component framework is applied.

1.3 Objectives of Scheduler Composition

We consider the following objectives to be fundamental for scheduler composition: wide applicability, good segregation, and low overheads. We now explain them one by one.

A rich legacy of workload models and schedulers for real-time systems have been accumulated in the past a few decades. This legacy shall be reused in the design of components when possible. Therefore, a successful general composition scheme shall have strong *applicability*: typical combinations of workload models and schedulers in real-time systems can be applied in components without major modification.

The purpose of composition is to divide-and-conquer system design complexity. Therefore it is desirable that an approach can facilitate the *segregation* between components and between the coordinator and the components; i.e., the design of a component should be independent to the design of other components and the design of coordinator.

The following three sources of *composition overheads* are commonly considered: (1) Coordinator overheads; (2) Communication and regulation between coordinator and components; (3) Utilization inflation caused by composition.

There might be trade-offs between the optimization objectives. For instance, if a composition can handle a vast variety of heterogeneous applications without a large utilization inflation, then the composition approach tends to be fine-grained, and the communication between the coordinator and components tends to be heavy, so the coordinator and communication overheads tend to be higher.

1.4 A Synopsis

There are two layers of a coordinator/components scheduler composition: (1) coordinatiion mechanisms; (2) component construction. In this dissertation, we shall make contributions on both layers, namely, Class-based Component Composition (CCC) in the layer of coordination mechanisms and pre-scheduling in the layer of component construction.

1.4.1 Class-based Component Composition

We propose the Class-based Component Composition (CCC) for composing independent components in an open environment. CCC applies a workload classification scheme. A component may send a class-based budget request to the coordinator; and the coordinator, upon admission of the component, guarantees that the supply of shared resource always meets the hard-real-time constraints for on-budget workloads. The CCC solution aims to achieve a balance over multiple design objectives in component composition including the width of applicability, segregation, composition overheads, and overload handling.

1.4.2 Pre-Scheduling

Static schedulers have been well accepted in real-time scheduling because of its predictability and simplicity in on-line execution. Traditional *static schedule* generation techniques are usually based on the assumption of constant rate of resource supply that is assumed to be known at design time. Under resource composition schemes, however, this assumption may *not* be valid for a component. A *pre-schedule* is a static schedule without assuming constant and completely predictable rate of resource supply. Instead, the concepts of supply function and supply contract are used to define the actual online resource supply rate and the constraints on this rate. Based on a component interface of supply contract and supply function, the pre-scheduling problem will be defined in a generalized framework, and a sound, complete and PTIME Linear Programming (LP) based pre-schedule generator will be given.

We shall show that one generally cannot produce a one-size-fits-all pre-schedule for a given time-driven workload under different supply contracts. In other words, given a fixed time-driven workload $\mathbf{J}$, it is necessary to produce different pre-schedules of it to fit for different supply contracts.

Since infinitely small time slices are not implementable for resources with context switch overhead, it is desirable to define and solve the pre-scheduling problem on the domain of integers so that context switching can occur only at boundaries of time quantums. However, Integral LP (ILP) is NP-hard in the strong sense in general, so the ILP approach is not applicable and better techniques are needed. This challenge is answered by a sound, complete and PTIME rational-to-integral pre-schedule transformer based on a novel technique which we call "round-and-compensate".

The process of supply contract generation is called "resource supply analysis". There are often two major sources of complexities in a coordinator/component based scheduler composition: the component complexity and the integration complexity. For a pre-scheduled component, the pre-scheduler deals with the component complexity, and the resource supply analysis deals with the integration complexity. Since resource supply analysis depends on knowledge beyond the pre-scheduled components, there is no uniform approach for it. We shall show how to perform the resource supply analysis by two case studies.

We programmed a basic LP-based pre-scheduler and ran the pre-scheduler over randomly generated workloads. Our experiments demonstrate the following results. (1) When system utilization rate is not extremely low, the success rate of LP-based pre-scheduler is significantly higher than that of naive pre-scheduler. (2) Pre-scheduling problems of practical sizes can be solved. In the experiments, problems with hundreds of jobs can be solved within a couple of hours (minutes in many cases), even on a machine with a slow CPU, a limited memory and a non-commercial LP-solver.

Beyond the basic pre-scheduling problem and integral pre-scheduling problem, there is a spectrum of pre-scheduling problems over different types of constraints, such as precedences and mutual exclusions. As a result of the research in this dissertation, we pretty much understand the computational complexities of these pre-scheduling problems.

1.4.3 Dissertation Organization

In the remainder of this dissertation, we first describe CCC in Chapter 2. Then Chapter 3 to Chapter 7 are dedicated to pre-scheduling. Chapter 3 defines the basic pre-scheduling problem and describes an LP-based solution. Chapter 4 describes how to translate a pre-schedule from the domain of rational numbers to the domain of integers. Chapter 5 provides examples on resource supply analysis. Chapter 6 presents experimental results. Chapter 7 further extends the basic pre-scheduling problem to cover more types of real-time constraints. Finally, Chapter 8 summarizes our research results and presents ideas for future work.

Chapter 2

A Class-Based Component Composition

This chapter describes Class-based Component Composition in details as follows. Section 2.1 provides the background, rationale and top layer description of CCC. Section 2.2 lists the assumptions and definitions needed in the design of CCC. Section 2.3 defines and analyzes the coordinator including the admission control module, the regulators, and the system scheduler. Section 2.4 shows how to construct components for three typical combinations of workloads and component schedulers. Section 2.5 puts all together by an example. Section 2.6 is about related work. Section 2.7 summarizes this chapter.

2.1 Introduction

Deadline, priority and share are three fundamental concepts in real-time scheduling, and composition approach have been proposed based on each one of them. In a deadline-based composition, a component provides deadline information to the coordinator. If its workload does not have natural deadline information, some pseudo deadline information will be produced, either by the component itself or by the coordinator. Then resource competition between components is solved by the coordinator according to the deadlines. Priority-based and share-based compositions are similar, except that either priorities or shares take the role of deadlines. When applications on a system are heterogeneous, the translation effort between deadlines, priorities and shares is non-trivial. CCC is based on the follow idea. Instead of translating between deadlines, priorities and shares, we may unify these concepts to "class". A class is a priority with a designated period, which is the guaranteed relative deadline and the aggregate shares that can be allocated to the class. Deadline-based, priority-based and share-based components can easily translate their resource requests to a uniformed, class-based "common ground", on which the composition is conducted.

The framework of CCC is shown in Figure 2.1. There is a system coordinator which consists of an admission-control module, a system scheduler and a number of regulators. Although only one component is shown in Figure 2.1, there may exist multiple components in a system. A component consists of a pre-admission module, a request generator, a component scheduler and a workload. There is one regulator between each admitted component and the system scheduler.

The general scenario of CCC is as follows. The system designer defines a list of *classes* which is indexed from high to low by the sequence of natural numbers from 0 to $K-1$, where K is the number of classes. The system designer defines a period $k.P$ [1] for each class k. The periods of classes from high to low form a monotonically increasing chain, with a higher class having a shorter period. When a component C is ready to run, its pre-admission module produces an *admission contract* and sends it to the coordinator. A contract is a list of bandwidth reservation requests defined as $\{b_0, .., b_{K-1}\}$, The aggregate execution time of all the requests in class k

[1] We shall adopt as a convention in this dissertation the notation $X.a$ which denotes the attribute a of entity X.

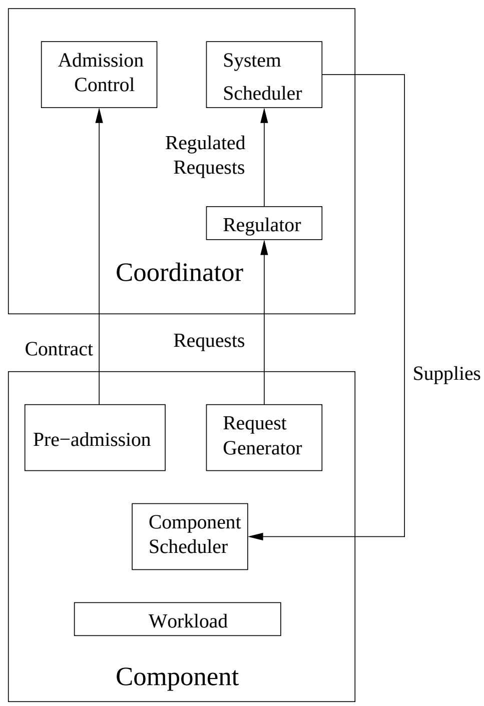

Figure 2.1: Framework of CCC

or higher from C shall not exceed b_k within every time interval of length $k.P$. The admission control module in the coordinator, upon receiving the supply contract from C, admits C if and only if the aggregate bandwidth reservation to each class k from all admitted components remains less than or equal to $k.P$. If C is admitted, bandwidth reservations is made for it according to its contract, and a regulator is established for it. The request generator of C produces a stream of requests according to the actual workload of the component, and sends them to the regulator. The regulator restricts the stream of requests according to the supply contract, and passes them over to the system scheduler. The system scheduler receives the regulated streams of requests from the regulators of all admitted components, and provides a stream of supplies to each admitted component. Upon receiving a supply, the component scheduler schedules the workload. When C terminates, it sends a termination message to the coordinator, and the coordinator deletes the regulator to C, and releases the bandwidths reserved for C.

CCC also provides overrun protection. A component *overruns* if its actual workload exceeds its contract. The first goal of overload handling of CCC is to guarantee the service to other non-overloaded components. However, when possible, CCC also makes the best effort to help the components in overrun with extra resource supply by two mechanisms: residual bandwidth utilization and class downgrading.

2.2 Assumptions

We make the following assumptions in the design of CCC. First, we assume that there is a *resource*, which is an object to be allocated to workload. It could be a CPU, a bus, or a packet switch, etc. In this dissertation, we shall consider the case of a single resource which can be shared by applications, and preemption is allowed. We assume that context switching takes zero time; this assumption can be removed in practice by adding the appropriate overhead to the execution time of the components. Further, we make three other fundamental assumptions: component independence, unit-size time allocation and open environment. Dependencies between jobs or tasks may exist within each component, but they may not exist across different components. *Time* is defined on the domain of non-negative integers. Each non-negative integer represents a *time unit*. The resource is allocated to a component for a time unit as a whole, and context switching may happen between any pair of adjacent time units, but not within a time unit. An *time interval* is a set of consecutive time units. A time interval might be represented by an open-ended interval as (x, y), so that the time interval does *not* include time unit x or y, but it includes all time units between them; a time interval might also be an interval of closed ends as $[x, y]$, which means time units x and y are included. A component may start or terminate at any time unit, and online admission control service is mandatory.

2.3 Coordinator

2.3.1 Admission Control

The admission control is defined in Algorithm 1. For each class k, the coordinator maintains a *residual bandwidth* $k.R$, which is the bandwidth unclaimed by any component.

During system initialization, $k.R$ for each class k is initialized to $k.P$, which is the period of the class. When a component C applies for admission, it provides a contract $\{b_0,..,b_k,..b_{K-1}\}$, where K is the number of classes, and b_k is the bandwidth required for class k. Component C is admitted if and only if $k.R$ is greater than or equal to b_k for every class k. If component C is admitted, then a regulator and some regulator queues (one for each class) are established for it, and the residual bandwidth $k.R$ for each class k will be decreased by b_k. The initialization of regulators is defined later in Algorithm 2. When component C terminates, it sends a termination notice to the coordinator. Upon receiving the notice, the coordinator deletes the regulator and its regulator queues, and reclaims the bandwidths reserved for C by increasing $k.R$ for each class k by the value of b_k.

Algorithm 1: Admission Control

```
Upon system initialization:
    foreach 0 ≤ k ≤ K − 1
        k.R := k.P;

Upon receiving a contract {b_k | 0 ≤ k ≤ K−1} from component C:
    if ∃ class k, such that b_k > k.R
        reject component C;
    else
        foreach 0 ≤ k ≤ K − 1
            k.R := k.R − b_k;
        admit component C by Algorithm 2;

Upon receiving termination notice from component C:
    delete the regulator for C;
    delete the regulator queues for C;
    foreach 0 ≤ k ≤ K − 1
        k.R := k.R + b_k;
```

2.3.2 Post-Admission Work-flow

Post-admission modules of the coordinator and the work-flow of these modules is shown in Figure 2.2. The component request generator may send requests to the regulator queues, and the requests are regulated and forwarded to the system queues by the regulator. The system scheduler selects a request from the system queues and grants the resource to the component corresponding to the request. The regulator queues are open-ended in Figure 2.2, indicating that the lengths of these queues

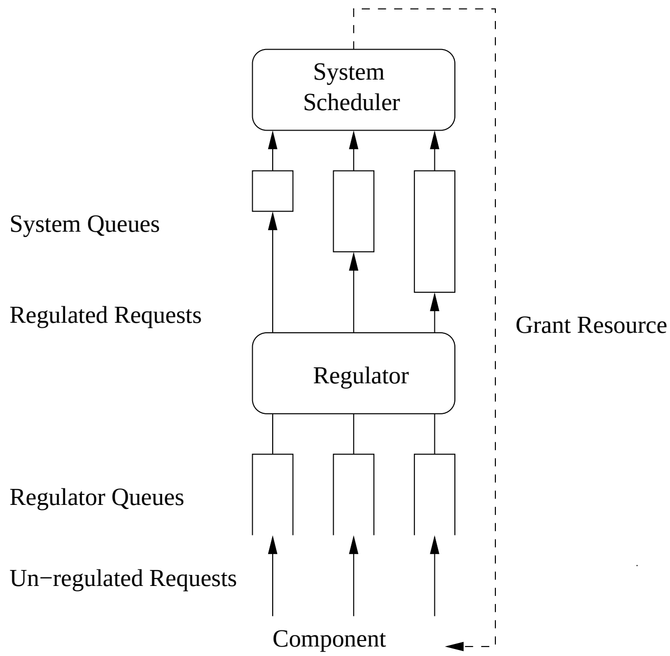

Figure 2.2: Post-Admission Work-flow of Coordinator

are unbounded. On the other hand, the system queues are close-ended, indicating that the lengths of them are bounded. The details are described in the following subsections.

2.3.3 Queues

We define four queuing methods, namely $push\_back$, $push\_front$, $peek$ and $deque$.

Methods $push\_back$ and $push\_front$ add an element to the back and the front of the queue respectively. Both methods $peek$ and $deque$ return the value of the front element of the queue; however, $deque$ removes the front element from the queue while $peek$ does not. For each class k and each admitted component C and its regulator G, there is a *regulator queue* $G.Q_k$, to whom only component C and its regulator G may have access. An element in a regulator queue is defined by a single entity: the requested execution time w. A regulator G maintains an internal *budget replenishment queue* $G.RQ_k$ for each class k, and only G has access to it. An element in a budget replenishment queue is a tuple (t, w), indicating that the budget will be replenished at time t for an amount equal to the value of w. There is a system queue SQ_k for each class k. Only regulators and system scheduler may have access to the *system queues.* Each element in a system queue is a tuple (C, w) which denotes the execution time (w) of the request and which component (C) sends the request.

2.3.4 Regulator

Before we define the algorithms of regulator, we first give the rationale for our design. Consider a time interval of length $k.P$. If the aggregate execution time of all requests of class k or higher from a component C exceeds b_k, then C is *overloaded.* If unchecked, C may obtain more than its negotiated share of the resource and the guarantees to other admitted non-overloaded components might be broken. The primary function of regulators is to keep the guarantees to the non-overloaded admitted components. Meanwhile, we use two best-effort mechanisms to handle the requests from the overloaded components. The first one makes use of the residual bandwidth by a *residual regulator* G_R, and overloaded requests may be forwarded via G_R. The second mechanism is class downgrading: a request from an overloaded component may be forwarded via a class lower than is required for the component.

There are a number of data structures of a regulator. For every class k, there is a *budget* B_k, a *budget limit* L_k, a regulator queue Q_k and a budget replenishment queue RQ_k.

A regulator G for component C is initialized by Algorithm 2. For each class k, the budget B_k is initialized to b_k, which is the bandwidth request in the contract of C. The replenishment queues of the regulator and regulator queues are initialized to empty queues. Since the residual bandwidths are changed upon the admission or termination of a component, the special regulator G_R for the residual bandwidths need to be initialized also.

Algorithm 2: The Initialization of Regulator

(1) Upon the admission of component C, establish regulator G with contract $\{b_k | 0 \leq k \leq K-1\}$:

(2) **foreach** $0 \leq k \leq K-1$

(3) $G.B_k := b_k$;

(4) $G.RQ_k := \emptyset$;

(5) $G.Q_k := \emptyset$;

(6) Upon the admission or termination of component C, initialize regulator G_R with residual bandwidths:

(7) **foreach** $0 \leq k \leq K-1$

(8) $G_R.B_k := k.R$;

(9) $G_R.RQ_k := \emptyset$;

At the beginning of any time unit t, regulators replenish their budget first as defined by Algorithm 3. For a regulator G, if its replenish queue RQ_k is non-empty, and the first element in the queue is (t, w), then budget B_k is increased by w. Then, budget limit L_k for every class k is computed, which is the minimal budget over all classes lower than or equal to k.

Algorithm 3: Budget Replenishment

Upon the beginning of a time unit t:

  foreach regulator G including G_R

    foreach $0 \leq k \leq K-1$

      if $G.RQ_k \neq \emptyset$

        $(t', w) := G.RQ_k.peek();$

        while $G.RQ_k \neq \emptyset$ and $t = t'$

          $G.RQ_k.deque();$

          $G.B_k := G.B_k + w;$

          if $G.RQ_k \neq \emptyset$

            $(t', w) := G.RQ_k.peek();$

    foreach $0 \leq k \leq K-1$

      $G.L_k := min(\{G.B_x | k \leq x \leq K-1\});$

Function Fwd (Algorithm 4) defines the process of forwarding a request by a regulator. A regulator G forwards a request of class k, weight w, and component C as follows. Value w', which is the portion of weight within the budget limit of class k (represented by $G.L_k$, is enqueued at the end of system queue of class k (SQ_k). For each class x such that $x \geq k$, budget of class x (B_x) is reduced by w', and a replenishment notice is pushed to the end of the replenishment queue RQ_x. Budget limit ($G.L$) for each class is also adjusted accordingly.

Algorithm 4: Function $Fwd(G, k, w, C)$

$w' := min(w, G.L_k)$;

$SQ_k.push\_back(C, w')$;

foreach x such that $k \leq x \leq K-1$

    $G.B_x := G.B_x - w'$;

    $G.RQ_x.push\_back(t + x.P, w')$;

foreach $0 \leq i \leq K-1$

    $G.L_i := min(\{G.B_x | i \leq x \leq K-1\})$;

return(w');

Algorithm 5 stipulates that request in a regulator queue may be handled by one of the three cases. In the first case, in-budget execution time of a request of class k is forwarded to the system queue of class k on time by consuming the budgets of its own regulator G. In the second case, over-budget execution time of a request of class k is forwarded to the system queue of either class k or a downgraded class (lower than k) by consuming the budget of either G or G_R, which is the residual regulator, whichever can forward the request by a higher class. In the third case, if the budget limit is zero for every class in G and G_R, the request stays in the regulator queue and waits to be forwarded at a later time unit when budget becomes available again.

Algorithm 5: Forwarding Requests

Upon time unit t:

  foreach regulator G (excluding G_R)

    while $\exists G.Q_x \neq \emptyset$ and (either $\exists G.L_y > 0$ or $\exists G_R.L_y > 0$)

      find k, j and j_R, which are the highest classes satisfying $G.Q_k \neq \emptyset$, $G.L_j > 0$, and $G_R.L_{j_R} > 0$;

      $l := max(j, k)$;

      $l_R := max(j_R, k)$;

      $w := G.Q_k.deque()$;

      if $l \geq l_R$

        $w' := Fwd(G, l, w, C)$;

      else

        $w' := Fwd(G_R, l_R, w, C)$;

      if $w > w'$

        $G.Q_k.push\_front(w - w')$;

2.3.5 System Scheduler

Algorithm 6 defines the system scheduler. At each time unit, the scheduler finds the one with the highest class among all non-empty system queues, and grants the resource to the component defined by the first request of it.

Algorithm 6: System Scheduler

(1) Upon system initialization:

(2)   **foreach** $0 \leq k \leq K-1$

(3) $SQ_k := \emptyset;$

(4)

(5) Upon time unit t:

(6)   Find the highest class h such that $SQ_h \neq \emptyset$;

(7) $(C, w) := SQ_h.deque();$

(8)   **if** $w > 1$

(9) $SQ_h.push\_front(C, w-1);$

(10) $Grant(C);$

2.3.6 Analysis

The response time of a request consists of the queuing delays in a regulator queue and a system queue. The regulator queuing delay is the number of time units that has elapsed between the time at which the request is pushed into a regulator queue by the component request generator and the time at which it is forwarded into a system queue by a regulator. Lemma 2.1 proves that the regulator queuing delay is zero for any request from a non-overloaded component. A request in a system queue is *completely served* when the aggregate time units granted to it is equal to its weight. When a request is completely served, it is dequeued at line 7 and not pushed to the front of the queue at line 9 of Algorithm 6. The system queuing delay of a request is the number of time units that has elapsed between the time at which a request is forwarded into a system queue and the time at which it is completely satisfied. Lemma 2.4 proves that the system queuing delay of a request of class k is bounded by $k.P$, which is the class period. Therefore, the coordinator of CCC provides a class-based responsiveness guarantee (Theorem 2.1).

Lemma 2.1 *The regulator queuing delay of a request of class k from a non-overloaded component is upper-bounded by zero, and the request is forwarded to the system queue of class k.*

Proof: Consider a non-overloaded component C and its regulator G. Assume the contrary, i.e., at time unit t, the following situation happens for the first time during execution: a request w is pushed into Q_k, and either the request must be forwarded to a system queue of a class lower than k, or it must wait to be forwarded at a later time unit. Either way, there must exist a class k' such that $k' \geq k$, such that $B_{k'}|_t \leq w$, where $B_{k'}|_t$ is the budget of class k' after budget replenishment at time t. Let time t' be $max(0, t - k'.P + 1)$, and let $B_{k'}|_{t'}$ be the budget of class k' *before* budget replenishment at time t', and let $Rpl_{k'}([t', t])$ be the total replenishment to the budget of class k' between time $[t', t]$. According to Algorithm 2, 3 and 5, $B_{k'}|_{t'} + Rpl_{k'}([t', t]) = b_{k'}$, where $b_{k'}$ is the bandwidth reserved for class k' for C. Because C is not overloaded, the aggregate execution time of all requests arrived between $[t', t]$ (including the request w) is less than or equal to $b_{k'}$. All requests of C arrived before time t' must have been forwarded to system queues before time t' because we assume that t is the first time unit a non-zero time delay in a regulator queue occurs. Therefore, there must be sufficient budget for request w, and there is a contradiction. ■

Lemma 2.2 *The aggregate execution time of all requests forwarded into the system queues with class k or higher during any time interval of length $k.P$ is less than or equal to $k.P$.*

Proof: According to Algorithm 2, 3 and 5, given any time interval of length $k.P$ and any component C and its regulator G, the aggregate execution time of all requests that G forwarded to system queues of class k or higher does not exceed $C.b_k$ which is the bandwidth reserved for C at class k. According to Algorithm 1, for any class k, $\sum C.b_k \leq k.P$. Therefore the lemma is true. ■

Time t is called *class k idle* if and only if at the beginning of time unit t, all system queues of class k or higher are empty before the execution of Algorithm 3, 5 and 6.

Lemma 2.3 *The length of the time interval between any pair of consecutive class k idle time units is upper-bounded by $k.P$.*

Proof: Proof by induction. Base case: time 0 is class k idle. Induction case: Assuming that the n^{th} class k idle time is t, we need to prove that the $(n+1)^{th}$ class k idle time is between $(t, t+k.P]$.

According to Lemma 2.2, the aggregate execution times of all requests forwarded to system queues of class k or higher between $[t, t+k.P)$ is less than or equal to $k.P$. If there is a class k idle time between $(t, t+k.P)$, the induction step holds; otherwise, every time unit in $[t, t+k.P)$ is granted to a request of class k or higher, and then time $t+k.P$ must be a class k idle time. ■

Lemma 2.4 *The system queuing delay of a request forwarded into the system queue of class k is upper-bounded by $k.P$.*

Proof: A request forwarded to a system queue of class k or higher at time t must be completely satisfied before a class k idle time right next to t. Therefore, this lemma follows Lemma 2.3. ■

Theorem 2.1 *The response time of a request of class k from an non-overloaded component is upper-bounded by $k.P$.*

Proof: According to the design of CCC, the response time of a request consists of queuing delays in a regulator queue and a system queue. The theorem follows Lemma 2.1 and Lemma 2.4. ■

Now we turn to the discussion of the computational complexities of the coordinator. The execution of admission control can be delayed until the system has sufficient resources in CPU time and memory space. However, the execution of the post-admission modules must be completed per time unit within strict upper-bounds of resources for all the admitted components. Therefore, we focus on the complexity analysis of the post-admission modules.

Time complexity is defined by the execution time of schedulers per time unit. The time complexity of a regulator is linear to the number of queue operations it executes per time unit. If the component is not overloaded, the number of queue operations is $O(N)$, where N is the maximal number of requests sent to the regulator per time unit. If the component is overloaded, requests might wait in the regulator queues for more budget. Therefore, requests sent in multiple time units may be accumulated into one time unit for processing, so the number of queue operations may exceed $O(N)$ in a time unit. In practice, we may set a limit on the number of requests processed per time unit to bound the execution time of each regulator. The time complexity of the system scheduler is upper bounded by a constant ($O(1)$).

Space complexity is given by the memory space occupied by the queues. Since the size of each element in a queue is $O(1)$, the space complexity of the queues is bounded by the aggregate length (number of elements) of queues. The aggregate weight of all replenishment queues of all the components is bounded by $\sum_{0 \leq k \leq K-1} k.P$. The weight of each element is at least 1. Therefore the aggregate length of replenishment queues is bounded by $O(\sum_{0 \leq k \leq K-1} k.P)$. According to Lemma 2.3, the aggregate execution time of all requests in all system queues is bounded by $O((K-1).P)$. Since the execution time of each request is at least 1, The aggregate length of all system queues is bounded by $O((K-1).P)$. Notice that CCC does not set any limit on the number or the aggregate execution time of requests that could be sent by a component per time unit. Therefore, the lengths of regulator queues of an overloaded component may be infinite. This problem can be solved in practice by for instance, discarding some requests once the length of a regulator queue reaches a limit.

2.4 Components

CCC is a generic composition scheme. Although the coordinator of CCC is class-based, the original applications do not need to be so because a component is established for each application and takes charge of the "translation". The design of a component is application-specific, and it is impossible for us to cover the component design for all possible applications. Instead, we define three types of components, each with a unique combination of workload model and application scheduler. The workload models we cover are periodic and sporadic tasks, and the schedulers we cover are EDF (Earliest Deadline First), FP (Fixed Priority), and static scheduler, since they are all commonly used in real-time research and practice.

2.4.1 Workload Models and Component Schedulers

First, let us review the workload models. A *job* is defined by a triple of (r, d, c), which means that an *execution time* of c is required to satisfy this job between its *ready time* r and *deadline* d. As defined in [18], a *periodic task* is an infinite stream of jobs. A periodic task T is defined by a triple (p, d, c), where the attributes define the period, relative deadline and execution time of the task respectively.

The first job of a periodic task is ready at time 0, and subsequent jobs are ready at exactly p time units apart. The j^{th} (starting from 0) job of a periodic task T is defined by the tuple $(j \cdot T.p,\ j \cdot T.p + T.d,\ T.c)$. A *sporadic task* is a stream of zero to infinite number of jobs, depending on the number of occurrences of the task in a computation. The ready time of a job of a sporadic task is also called its *arrival time*. The arrival time of a sporadic job is unknown *a priori*. An arrival function $A(J)$ represents the arrival times of a job J of a sporadic task in a computation. A sporadic task is defined by a triple (p, d, c), where the attributes are respectively the *minimal arrival interval*, relative deadline and execution time of the task. A job J of sporadic task T is defined as $(A(J),\ A(J)+T.d,\ T.c)$. A *valid* arrival function must satisfy the minimal arrival interval constraints: for any two consecutive jobs J_i and J_{i+1} of a sporadic task T, the following must be true: $A(J_{i+1})-A(J_i) \geq T.p$. For convenience, we shall call a job of a periodic task a *periodic job*, and a job of a sporadic task a *sporadic job*.

Next we review component schedulers. Either Earliest Deadline First (EDF) scheduler or Fixed Priority (FP) scheduler can schedule periodic tasks, sporadic tasks, or a combination of both types of tasks. EDF scheduler always schedules a job with the earliest deadline among all the jobs that are ready and not completely satisfied. FP scheduler works as follows. There are F priorities from 0 to $F-1$, where priority 0 is the highest. A FP scheduler assigns a *fixed priority* $f(T)$ to each task T, and the scheduler always schedules a job with the highest priority among all jobs that are ready and not completely satisfied.

The static scheduler is designed primarily for periodic tasks. A *static schedule* is defined by a hyper period P and a list of cyclic executives $\mathbf{E}$. An executive E in $\mathbf{E}$ is defined by a tuple $(J_{i,j}, r, d, c)$, with the meaning that the jth job of task i in a hyper period is to be scheduled for a length of time c between ready time r and deadline d determined as offsets from the beginning of each hyper period. The r values of all the executives in the list are monotonically non-decreasing, and so are the d values of all executives in the list. During execution, the static scheduler follows the list of cyclic executives within every hyper period, and starts over again from the first executive at the beginning of every hyper period.

2.4.2 EDF Component

In this subsection, we shall assume that the workload of an application is specified as a set of sporadic or periodic tasks, and the application scheduler is EDF. We show how to construct an EDF component for such an application.

The pre-admission module is defined in Algorithm 7. First, a mapping function M is computed. Each task T is mapped to the lowest class that satisfies the following constraint: the class period is less than or equal to the relative deadline of task T. Then a contract is produced. For each class k, its bandwidth reservation requirement b_k in a contract is computed as the maximal aggregate execution time of all jobs of class k or higher that may possibly arrive within any time interval of $k.P$. Finally the contract is sent to the coordinator.

Algorithm 7: Pre-Admission Module of EDF Component

foreach Task T
    $M(T) := max\{k|0 \leq k \leq K-1 \text{ and } k.P \leq T.d\}$;
foreach $0 \leq k \leq K-1$
    $b_k := 0$;
    foreach task T that satisfies $M(T) \leq k$
        $b_k := b_k + \lceil \frac{k.P}{T.p} \rceil \cdot T.c$;
$Send\_To\_Coordinator(\{b_k|0 \leq k \leq K-1\})$;

Request generator is defined as follows. Upon the arrival of a job of a task T, it sends a request of value $T.c$ to the regulator queue of class $M(T)$ of the corresponding regulator G: $G.Q_{M(T)}.push\_back(T.c)$.

2.4.3 FP Component

In this subsection, we assume that the application workload is still specified as a set of sporadic or periodic tasks, but the application scheduler is FP. We show how to construct an FP component.

The pre-admission module is defined by Algorithm 8. First, the mapping function M from a priority to a class is defined as follows. For each priority f,

$M(f)$ is the lowest class (i.e., with highest class index) that satisfies the following constraints: (1) For every task T with priority f, $M(f).P \leq T.d$; (2) For any priority x such that $x < f$, class $M(x) \leq M(f)$. Then a contract is produced as follows: For each class k, the bandwidth reservation requirement b_k is the aggregate execution time of jobs with priorities mapped to class k or higher that may arrive within any time interval with a length of $k.P$. Finally the contract is sent to the coordinator.

Algorithm 8: Pre-Admission Module of FP Component

```
foreach  fixed priority x
    M(x) := K − 1;
foreach task T
    find the lowest (maximal) class k that satisfies k.P ≤ T.d;
    foreach priority x such that x ≤ f(T)
        M(x) := min(M(x), k);
foreach 0 ≤ k ≤ K − 1
    b_k := 0;
    foreach task T that satisfies M(f(T)) ≤ k
        b_k := b_k + ⌈k.P / T.p⌉ · T.c ;
Send_To_Coordinator({b_k | 0 ≤ k ≤ K − 1});
```

The request generator is defined as follows. Upon the arrival of a job of a task T, a request of value $T.c$ is sent to the regulator queue of class $M(f(T))$: $G.Q_{M(f(T))}.push\_back(T.c)$.

2.4.4 Statically Scheduled Component

In this subsection, we assume that the application workload is specified by periodic tasks only, and the application is statically scheduled. We show how to construct such a component.

The pre-admission module is given in Algorithm 9. First, a mapping function M from the executives to classes is produced as follows. For each executive E in the list of executives $\mathbf{E}$, $M(E)$ is the lowest class k that satisfies $k.P \leq (E.d - E.r)$. Then a contract is computed as follows. For every class k, the bandwidth reservation requirement b_k is computed as the maximal aggregate execution times of all executives of class k or higher that arrived within any time interval of length $k.P$. Finally the contract is sent to the coordinator.

Algorithm 9: Pre-Admission Module of Statically Scheduled Component

(1) **foreach** executive E in $\mathbf{E}$

(2) $M(E) := min\{k|k.P \leq (E.d - E.r)\}$;

(3) **foreach** $0 \leq k \leq K - 1$

(4) **foreach** E in $\mathbf{E}$ that satisfies $M(E) \leq k$

(5) construct a set of executives Φ_E, such that an executive X is in Φ_E if and only if $M(X) \leq k$ and $E.r \leq X.r \leq E.r + k.P$;

(6) let $W(\Phi_E)$ be the aggregate execution time of all executives in Φ_E;

(7) $b_k := max(\{W(\Phi_E)|E \in \mathbf{E} \text{ and } M(E) \leq k\})$;

(8) $Send\_To\_Coordinator(\{b_k|0 \leq k \leq K - 1\})$;

The request generator is defined as follows. Upon the ready time of an executive E in a hyper period, a request of value $E.c$ is sent to the regulator queue of class $M(E)$: $G.Q_{M(E)}.push\_back(E.c)$.

2.4.5 Analysis

A specification of an application usually defines by conditions and requirements. The workload must comply with the conditions. For instance, the minimal arrival intervals between consecutive sporadic jobs are conditions. The requirements are the constraints required by the application but implemented by the schedulers. For instance, the deadlines are requirements. A scheduling system is *correct* for an application if the requirements are guaranteed under the conditions.

The correctness of scheduling a component is implemented in CCC by the following three guarantees:

- Guarantee (1): the stream of requests sent to the coordinator shall satisfy the contract.

- Guarantee (2): the class-based responsiveness guarantee of the coordinator.

- Guarantee (3): the component schedule satisfies the application requirements.

Guarantee (1) is implemented by the pre-admission modules. When a contract is produced, the pre-admission algorithms guarantee that the bandwidth reservation b_k for each class k in the contract is sufficient to hold the maximal aggregate execution time of class k or higher that may arrive within any time interval of length $k.P$.

If Guarantee (1) holds, Guarantee (2) is provided by the coordinator, which is proved in Theorem 2.1.

We show how Guarantee (3) is expressible in terms of three requirements. The first one is the requirement of *valid scope*: each job shall be scheduled between its ready time and deadline. This requirement applies to EDF, FP and statically scheduled components. The guarantee on this requirement is made jointly by the pre-admission module, the request generator and the component scheduler of each component. The pre-admission modules map each task or executive to a class whose period is shorter than or equal to the relative deadline of either the task or the executive, and the request generator sends a request to the class upon the arrival or ready time of either a job or an executive. Since Guarantee (2) is provided by the coordinator, the property of valid scope is guaranteed by the EDF, FP and statically scheduled components. The second requirement applies to the FP component only. It is the requirement of *priority-based non-preemptive allocation*, which means that a job with a higher priority must not be preempted by a job with a lower or equal priority. The third requirement applies to the statically scheduled component only. There is the requirement of *fixed total order* in execution: if an executive E_x is before another executive E_y in the list, then executive E_x will always be scheduled before executive E_y in every hyper period. The priority-based non-preemptiveness in a FP component and fixed total order in a CE component are guaranteed, respectively, by their component schedulers.

2.5 Example

We illustrate how CCC works by an example. Assume that there are seven classes, and the class periods are given by $1, 5, 10, 20, 50, 100, 1000$. Also assume that there are four components defined as follows.

- Component C_0: The workload consists of one sporadic task and two periodic tasks, and the component scheduler is EDF. The sporadic task $T_{0,0}$ is defined as $(\infty, 1, 1)$, where the execution time and relative deadline are both 1, and the minimum arrival interval is infinite; i.e., this task occurs only once in every computation, but immediate attention is required upon job arrival. The periodic tasks $T_{0,1}$ and $T_{0,2}$ are defined as $(80, 8, 1)$ and $(100, 10, 1)$.

- Component C_1: The workload consists of two sporadic tasks, and the component scheduler is FP. Tasks $T_{1,0}$ and $T_{1,1}$ are defined as $(30, 10, 2)$ and $(30, 20, 1)$. The priorities of $T_{1,0}$ and $T_{1,1}$ are 0 (higher) and 1 (lower).

- Component C_2 is statically scheduled. The hyper period is 100, and the cyclic list of executives is defined as $\mathbf{E} = \{E_0, E_1, E_2\}$. We ignore the corresponding job id of each executive here because it does not influence the composition. Therefore each executive is defined by a triple of attributes representing the ready time, deadline and execution time, as follows: $E_0 : (0, 10, 2)$, $E_1 : (0, 100, 50)$, $E_2 : (70, 100, 5)$.

- Component C_3 is a bandwidth-intensive application which needs 40 percent of the resource on average.

The mapping functions and contracts of C_0, C_1 and C_2 are defined according to Algorithm 7, 8, and 9. The mapping function and contract of C_3 is *ad hoc*.

- C_0: Mapping function: $M(T_{0,0}) = 0$, $M(T_{0,1}) = 1$, $M(T_{0,2}) = 2$.

  Contract: $\{1, 2, 3, 3, 3, 4, 24\}$.

- C_1: Mapping function: $M(0) = 2$; $M(1) = 3$.

  Contract: $\{0, 0, 2, 3, 6, 12, 102\}$.

- C_2: Mapping function: $M(E_0) = 2$, $M(E_1) = 5$, $M(E_2) = 3$.

  Contract: $\{0, 0, 2, 5, 7, 57, 570\}$.

- C_3: Mapping function: All requests are mapped to Class 6.

  Contract: $\{0, 0, 0, 0, 0, 0, 400\}$.

Now we illustrate the admission control given by Algorithm 1. Assume that all components apply for admission at time 0, and the admission decisions are made in the index order of components. Table 2.1 shows the changes in residual bandwidth. Components C_0, C_1 and C_2 are admitted because there are sufficient residual bandwidths for them on all classes. Component C_3 is rejected because it requires a bandwidth of 400 on class 6 which is greater than the residual bandwidth (which is 304) of the class by the time its admission is processed.

Table 2.1: Residual Bandwidths During Admission Process

	$0.R$	$1.R$	$2.R$	$3.R$	$4.R$	$5.R$	$6.R$
after initialization	1	5	10	20	50	100	1000
after C_0 is admitted	0	3	7	17	47	96	976
after C_1 is admitted	0	3	5	14	41	84	874
after C_2 is admitted	0	3	3	9	34	27	304

In the remainder of this section, we use *snapshots* to illustrate the post-admission execution. A snapshot refers to the values of budgets and queues at certain time. At time 0, after components C_0, C_1 and C_2 are admitted, regulators G_0, G_1 and G_2 are established, and budgets and regulator queues are initialized, as defined by Algorithm 2. The request generators produce and send requests into the regulator queues. Table 2.2 is the snapshot taken after these executions. We assume that the first jobs of sporadic tasks $T_{1,0}$ and $T_{1,1}$ arrive at time 0.

Table 2.2: Budget Initialization and Adding Requests to Regulator Queues

class	G_0		G_1		G_2		G_R	SQ_k
k	B_k	Q_k	B_k	Q_k	B_k	Q_k	B_k	
0	1		0		0		0	
1	2	{ 1 }	0		0		3	
2	3	{ 1 }	2	{2}	2	{2}	3	
3	3		3	{1}	5		9	
4	3		6		7		34	
5	4		12		57	{50}	27	
6	24		102		570		304	

At this time, none of the component is overloaded. Therefore, there is sufficient budget to forward all requests in components queues to system queues. Table 2.3 shows the snapshot after the execution of the regulators (given by Algorithm 3 and 4) but before the execution of the system scheduler.

The highest class with a non-empty system queue is class 1. Therefore, the system scheduler as given by Algorithm 6 dequeues the first and only request from SQ_1, and grants time 0 to component C_0. The snapshot after the execution of the

Table 2.3: Executions of The Regulators under Non-Overloading Condition

class	G_0		G_1		G_2		G_R	SQ_k
k	B_k	Q_k	B_k	Q_k	B_k	Q_k	B_k	
0	1		0		0		0	
1	1		0		0		3	$\{(\mathbf{C_0}, \mathbf{1})\}$
2	1		0		0		3	$\{(C_2, 2),$ $(C_1, 2),$ $(C_0, 1)\}$
3	1		0		3		9	$\{(C_1, 1)\}$
4	1		3		5		34	
5	2		9		5		27	$\{(C_2, 50)\}$
6	22		99		518		304	

system scheduler is shown in Table 2.4.

Table 2.4: Execution of The System Scheduler

class	G_0		G_1		G_2		G_R	SQ_k
k	B_k	Q_k	B_k	Q_k	B_k	Q_k	B_k	
0	1		0		0		0	
1	1		0		0		3	
2	1		0		0		3	$\{(C_2, 2),$ $(C_1, 2),$ $(C_0, 1)\}$
3	1		0		3		9	$\{(C_1, 1)\}$
4	1		3		5		34	
5	2		9		5		27	$\{(C_2, 50)\}$
6	22		99		518		304	

In order to illustrate the overload handling mechanism of residual bandwidth utilization defined in Algorithm 5, assume that the second jobs of $T_{1,0}$ and $T_{1,1}$ both arrive at time 1. These arrivals violate their task specification and overload C_1. However, CCC can accommodate the overloaded requests with its residual bandwidths under this situation. Table 2.5 is the snapshot after the execution of Algorithm 3 and 5 but before the execution of Algorithm 6 at time 1. Notice that the budgets of G_R are decreased, and new requests are forwarded into the system queues.

Table 2.5: Forwarding Overloaded Requests Via Residual Bandwidths

class	G_0		G_1		G_2		G_R	SQ_k
k	B_k	Q_k	B_k	Q_k	B_k	Q_k	B_k	
0	1		0		0		0	
1	1		0		0		3	
2	1		0		0		**1**	$\{(\mathbf{C_1, 2}),$ $(C_2, 2),$ $(C_1, 2),$ $(C_0, 1)\}$
3	1		0		3		**6**	$\{(\mathbf{C_1, 1}),$ $(C_1, 1)\}$
4	1		3		5		**31**	
5	2		9		5		**24**	$\{(C_2, 50)\}$
6	22		99		518		**301**	

In order to illustrate the overload handling mechanism of class downgrading as given in Algorithm 5, we assume that the third job of $T_{1,0}$ arrives at time 2. This time, the residual regulator does not have sufficient budget at class 2 for forwarding the overloaded request. Therefore, part of the request is downgraded to class 3 and forwarded to system queue via G_R, as shown in Table 2.6. Notice the newly forwarded element to the system queue of class 3.

Finally, we demonstrate the budget replenishment mechanism in Algorithm 3. At time 5, the budget consumed at time 0 on class 1 in C_0 is replenished. Suppose no new job arrives between time 2 and time 5. Then the snapshot after the execution of the coordinator at time 5 is as shown in Table 2.7. Notice the increase of budget B_1 of regulator G_0.

2.6 Related Work

A sizeable literature has been accumulated on component composition and we can only briefly review a part of it here. A major paper is by Deng and Liu who

Table 2.6: Forwarding An Overloaded Request Via A Downgraded Class

class	G_0		G_1		G_2		G_R	SQ_k
k	B_k	Q_k	B_k	Q_k	B_k	Q_k	B_k	
0	1		0		0		0	
1	1		0		0		3	
2	1		0		0		**0**	$\{(\mathbf{C_1}, \mathbf{1}), (C_1, 2), (C_2, 2), (C_1, 2)\}$
3	1		0		3		**4**	$\{(\mathbf{C_1}, \mathbf{1}), (C_1, 1), (C_1, 1)\}$
4	1		3		5		**29**	
5	2		9		5		**22**	$\{(C_2, 50)\}$
6	22		99		518		**299**	

proposed the open system environment model where application components may be admitted online and the scheduling of the component schedulers is performed by a kernel scheduler [4]. Mok and Feng exploited the idea of temporal partitioning [20], by which individual applications and schedulers work as if each one of them owns a dedicated "real-time virtual resource". Lipari et. al. proposed an EDF-based framework for composition [17]. Regehr and Stankovic investigated hierarchical schedulers [23].

POSIX.4 [10] defines two fixed-priority-based schedulers: $SCHD\_FIFO$ and $SCHD\_RR$. For both of them, there may exist multiple fixed priorities, and multiple tasks may be assigned to each priority. The tasks with the same priority are scheduled with First-In-First-Out by $SCHD\_FIFO$, or with Round Robin by $SCHD\_RR$. However, POSIX.4 does not prescribe any priority assignment algorithm, nor can it provide any real-time guarantee. Cayssials et. al. investigated the problem of assigning real-time tasks to a fixed but limited number of priorities [3]. They assume that all tasks to be scheduled are known off-line, therefore sophisticated off-line algorithms can be applied to obtain optimal solution. However,

Table 2.7: Budget Replenishment

class	G_0		G_1		G_2		G_R	SQ_k
k	B_k	Q_k	B_k	Q_k	B_k	Q_k	B_k	
0	1		0		0		0	
1	**2**		0		0		3	
2	1		0		0		0	$\{(C_1, 1), (C_1, 2)\}$
3	1		0		3		4	$\{(C_1, 1), (C_1, 1), (C_1, 1)\}$
4	1		3		5		29	
5	2		9		5		22	$\{(C_2, 50)\}$
6	22		99		518		299	

their approach cannot be applied to an open environment where the components are heterogeneous and dynamic. Our CCC scheme makes use of the concept of class instead of priority. The difference between them is that a class has an inherent responsiveness guarantee, which is defined by its period. For this reason, hard real-time guarantees could be made by CCC in an open environment with low overhead.

Many hard and/or soft real-time scheduling approaches depend on budget control to maintain a fair share among either tasks or components. Total Bandwidth Server [26] is one of these approaches. Budget control is critical in CCC for keeping the responsiveness guarantees to the non-overloaded components. Because CCC is class-based, it adopts a straightforward budget replenishment strategy – every consumed budget of a class is replenished after the period of the class.

2.7 Summary

CCC provides a balanced solution for meeting multiple design objectives in scheduler composition. The definition of CCC starts with the goal of wide applicability. It unifies some most popular approaches for workload modeling and scheduling for real-time systems. If the workload of a component is based on deadline, priority or shares, the translation to the class-based "common ground" is straight forward.

The segregation between a component and other parts of the system is provided by CCC: The coordinator provides class-based guarantees for all admitted components, and the component meets its own specific timeliness requirements based on the class-based guarantees it acquires in its admission contract.

CCC has following features on composition overheads. First, the online average overhead on each component is low. Second, the scheduling overhead of a component can be computed at pre-admission time, therefore it is predictable. Third, the overhead is scalable: the overhead on each component will not increase with the total number of components.

However, the utilization inflation depends on how a coodinator and components are are designed: how many classes are defined and what are the periods of them, how the component workload and scheduler are defined, and how to map component workload to classes, etc.

Chapter 3

The Basic Pre-Scheduling Problem and A LP-based Solution

This chapter establishes a basic pre-scheduling framework and problem, and focuses on the description and analysis of the basic Linear-Programming (LP) based pre-scheduler. Section 3.1 provides the background, rationale of the basic pre-scheduling problem and top layer description of our solution. Section 3.2 formally defines the basic pre-scheduling problem. Section 3.3 describes the LP-based pre-scheduler. Section 3.4 analyzes the pre-scheduler. Section 3.5 shows the non-existence of universally valid pre-schedule in general. Section 3.6 addresses relation work. Section 3.7 summarizes the merits of the LP-based pre-scheduler.

3.1 Introduction

Pre-scheduling extends a classic hard real-time scheduling approach, namely static scheduling, to the context of scheduler composition.

Static schedule is well accepted for time-driven workloads for its predictability and its simplicity in online execution. Given a time-driven workload, a static schedule, which is a list of "executives" [1], is generated at design time. Each executive defines that the resource shall be allocated to a specific job for a length of time within a pair of ready time and deadline. A static schedule covers the length of a "hyper-period". During online execution, the time line is divided into an infinite number of consecutive hyper intervals, each of the length of a hyper-period, and the static schedule is repeated within each hyper interval. A variety of timing constraints can be effectively solved at design time [6, 22, 27]. Moreover, online monitoring and exception handling mechanisms can be readily devised to catch timing abnormalities such as unexpectedly long execution times [1]. The online overhead is $O(1)$ and can usually be bounded by a small constant.

In recent years, there is a trend in utilizing static scheduling under compositional schemes in industry, for instance, TTCAN [11]. The rational is as follows. In some control systems, such as automotives, time-driven workload and event-driven workload co-exist. The time-driven workload may still be statically scheduled to obtain the advantages of predictability and online execution simplicity; however, event-driven workload usually needs to be scheduled dynamically. Therefore, a composition scheme is needed; a critical assumption for traditional static scheduling needs to be relaxed, which we will explain next.

In many previous work in static schedule generation, e.g, [1, 6, 16, 21, 22, 27], the following assumption is often implicitly made by the authors: the resource supply rate is a constant known at design time. This assumption is appropriate for many traditional embedded systems, where the controllers are non-super-scalar and non-pipelined, and they run at a fixed frequency, and the programs are locked in one layer of memory (no cache). In the remainder of this dissertation, we call this assumption as *constant supply rate assumption*. However, the supply rate to a component under a compositional scheme might be neither constant nor known at design time, since the supply rate to a component is a result of resource competition among all components. Therefore, the assumption on supply rate needs to be weakened.

In order to distinguish from the traditional concept of static schedule, we introduce the term "pre-schedule", which specifically refers to a static schedule without assuming constant and completely predictable resource supply rate. The pre-schedule generation problem is also called the "pre-scheduling problem", and a pre-schedule generator is called a "pre-scheduler".

A generalized pre-scheduling framework, as shown in Figure 3.1, is proposed in this chapter. We assume there is a time-driven workload in a "subject" component. There is a *supply function* and a *supply contract* between the subject component and the coordinator. The supply function defines when the resource is assigned to the subject component, and it is usually computed online by a composition mechanism. The supply contract defines supply constraints that must be satisfied by the supply function, and it is computed off-line according to *a priori* knowledge on the subject component and the competing components, together with their scheduling and composition mechanisms. The pre-scheduler produces a pre-schedule for the subject component according to the supply contract, and the online scheduler within the subject component produces a schedule according to its pre-schedule and supply function.

There are two major steps in the basic pre-scheduler. The first step construct a partially defined pre-schedule $\mathbf{F}$ according to the subject workload. $\mathbf{F}$ is a sequence of executives; however, the execution time of each executive remain un-defined. Then the second step solves the execution times using Linear-Programming solver. This pre-scheduler is also called the *LP-based pre-scheduler*.

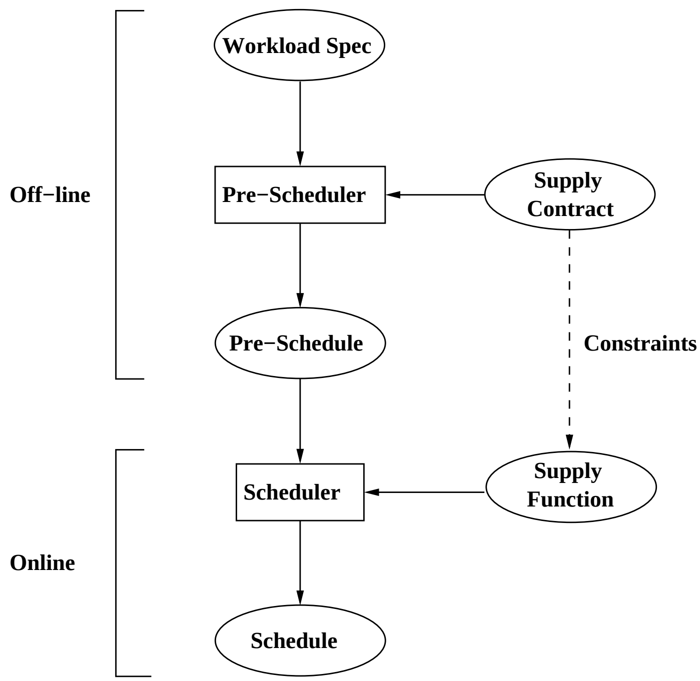

Figure 3.1: Framework of Pre-Scheduling

3.2 Assumptions and Definitions

The online execution time line is divided into an infinite number of *hyper intervals*, each with a constant length of P called *hyper period*. For every natural number (non-negative integer) n, the time interval $(n \cdot P, (n+1) \cdot P)$ is the n^{th} hyper interval.

A subject workload is modeled as a set of jobs $\mathbf{J}$. Each job J in $\mathbf{J}$ is defined by a tuple of (r, d, c), standing for ready time, deadline, and execution time.

For any job J, the time interval between its ready time and deadline, represented as $(J.r, J.d)$, is called the *valid scope* of the job. There is exactly one instance of each job that becomes ready (or arrives) in each hyper interval. The instance of a job J that becomes ready within the n^{th} hyper interval is called the n^{th} instance of job J, and it must be scheduled within time interval $(n \cdot P + J.r, n \cdot P + J.d)$.

The following constraints must be satisfied by the definition of each job J: (1) $J.d - J.r \leq P$; (2) $0 \leq J.r < P$; (3) $J.c > 0$; (4) $0 < J.d \leq P$, which means a job in subject workload does not straddle hyper periods. We showed in [32] that the pre-scheduling problem can still be solved by the LP-based pre-scheduler even if constraint (4) does not hold; However, we make this assumption here to simplify the discussion on the basic pre-scheduling problem. Also notice that a periodic task as defined in Subsection 2.4.1 and [18] might be represented as multiple jobs in this workload model.

A *time interval* is defined by a tuple of (b, e), which starts at time b and ends at time e. We define the relative positions between two time intervals as follows. Let X and Y be two time intervals. X is *before* Y and Y is *after* X if and only if at least one of the following conditions is true: (1) $X.b < Y.b$ and $X.e \leq Y.e$; (2) $X.b \leq Y.b$ and $X.e < Y.e$. X *contains* Y or Y *is contained by* X if and only if $X.b < Y.b$ and $Y.e < X.e$. X is *parallel to* Y if and only if $X.b = Y.b$ and $X.e = Y.e$. The relative positions of jobs are defined according to the relative positions of their valid scopes. For instance, job X is before job Y if and only if $(X.r, X.d)$ is before $(Y.r, Y.d)$. In Figure 3.2, for instance, job C is before jobs D and E, and job C contains jobs A and B.

We assume that $\mathbf{J}$ is in order by the following rule: Let J_x and J_y be arbitrary jobs in $\mathbf{J}$, where x and y are indexes; If either J_x is before J_y or J_x is contained by J_y, $x < y$.

Example 1 *A subject workload* $\mathbf{J}$ *is defined as follows. Hyper period* P *is 45. Each job is identified by a name and defined by a triple of ready-time, deadline, and execution-time.*

$$\mathbf{J} = [A : (1, 9, 1), B : (16, 24, 1), C : (0, 40, 8),$$
$$D : (14, 40, 4), E : (0, 45, 3)]$$

J in Example 1 is illustrated in Figure 3.2. A pair of short vertical lines are positioned at the ready time and deadline of each job, and they are connected by a horizontal line, showing the length of the valid scope. The length of the box inside the scope of a job indicates the execution time of the job. Long dashed vertical lines define the scope of a hyper interval. ■

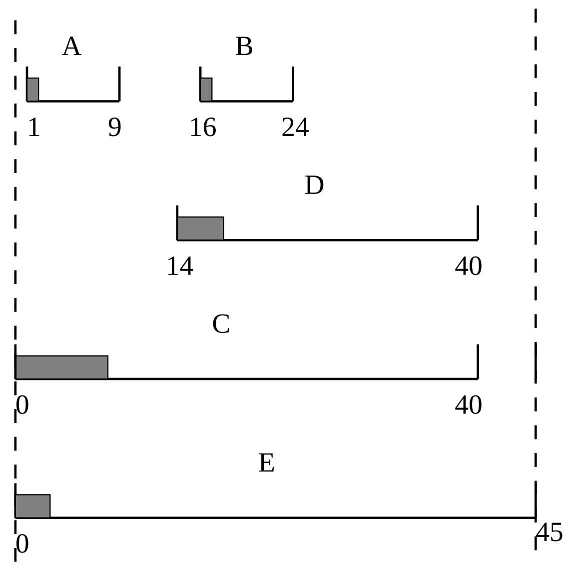

Figure 3.2: A Subject Workload **J**

An *executive* E is defined by a 4-tuple of (J, r, d, c), standing for corresponding job, ready time, deadline and execution time. The n^{th} instance of job J must be scheduled by an aggregate length of c between time interval $(n \cdot P + r, n \cdot P + d)$. Time interval (r, d) is the *valid scope* of E. A *pre-schedule* **E** is a list of executives, and the order of the executives in the list defines their scheduling order. There exists one or multiple executives in **E** for each job in **J**.

A *supply function* $U(t)$ defines the resource supply to a pre-scheduling space. If at time t, the resource is assigned to the pre-schedule space, $U(t) = 1$; otherwise,

$U(t) = 0$.

A *schedule* S in a pre-scheduling space is a function from the domain of time to $\mathbf{J}$. At any time t, if the resource is scheduled to job J in $\mathbf{J}$, $S(t) = J$; if the resource is not scheduled to any job J in $\mathbf{J}$, $S(t) = \perp$. For the purpose of defining the basic pre-scheduling problem, we consider a schedule S is *valid* if and only if it satisfies the following constraints. (1) Scope constraints: if $S(t) = J$, then $n \cdot P + J.r \leq t \leq n \cdot P + J.d$. (2) Demand constraints: For any job J, the aggregate time that scheduled to it between $(n \cdot P + J.r, n \cdot P + J.d)$ is equal to $J.c$. (3) Supply constraints: At any time t, if the resource is not supplied to the pre-scheduling space, then no job in $\mathbf{J}$ is scheduled; i.e., if $U(t) = 0$, $S(t) = \perp$.

The *online scheduler* of a pre-scheduled component is defined as follows. Let E^{cur} represent the current executive in pre-schedule $\mathbf{E}$. At the start of every n^{th} hyper interval, where n is a natural number, let E^{cur} be the first executive in $\mathbf{E}$. At time t, if the resource is granted to this pre-scheduling space, i.e., $U(t) = 1$, and $E^{cur}.r + n \cdot P \leq t \leq E^{cur}.d + n \cdot P$, assign the resource to the job corresponding to E^{cur}, i.e., $S(t) = E^{cur}.J$; otherwise, $S(t) = \perp$. When the length of time scheduled via E^{cur} is accumulated to $E^{cur}.c$, the E^{cur} is completed. Let the next executive be E^{cur}.

Example 2 *Workload* $\mathbf{J}$ *is defined in Example 1. Show a pre-schedule* $\mathbf{E}$ *and its corresponding schedules under different supply functions.*

$$\begin{aligned}\mathbf{E} &= [(C,0,9,1),(A,1,9,1),(C,1,24,7),(E,1,24,1),(D,14,24,2),(B,16,24,1),\\ &\quad (D,16,40,2),(E,16,45,2)]\end{aligned}$$

$\mathbf{E}$ is illustrated in the upper part of Figure 3.3. A pair of short vertical lines define the valid scope of each executive, and the length of the blank box within the valid scope represents the execution time. Also, two supply functions and two corresponding schedules are illustrated in the lower part of Figure 3.3. The black boxes in the row of supply functions indicate the time intervals in which the resource is *not* supplied to the pre-scheduled component. Each schedule is shown as a sequence of grey boxes. Two different valid schedules are generated according to two different valid supply functions, but the order of executives defined by the pre-schedule is always followed, and each executive must always be scheduled to the length of its execution time and within its valid scope. ■

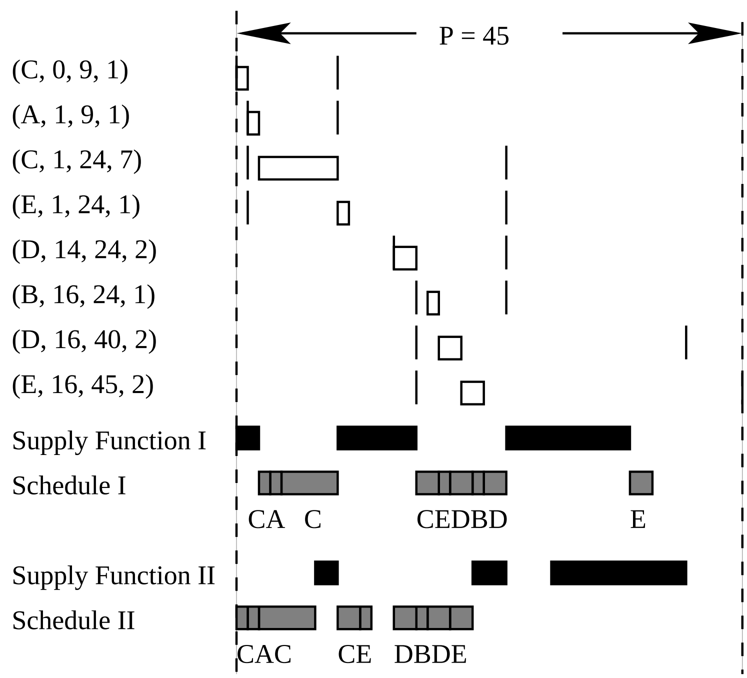

Figure 3.3: Pre-schedule and Online Schedule Generation

Since the resource supply rate is variable and it is not completely predictable, the supply function is unknown at design time. However, a *supply contract* can be computed at design time according to *a priori* knowledge of workloads and their scheduling and composition schemes. Given a time interval I, supply contract $B(I)$ is the aggregate execution time guaranteed to the subject component within I by the supply function.

We assume the following properties of supply contract: *localization*, *recursiveness* and *regularity.* Localization is rooted from the following observation: in many applications, the resource competition over large time scale can be approximated as a rate-based resource sharing, which is not sensitive to how a workload is pre-scheduled. We assume that hyper period P is large enough such that the supply constraints over time intervals longer than P need not to be considered in pre-scheduling. Recursiveness means that the supply contract repeats itself by hyper period: $B(I) = B(I.b + P, I.e + P)$. For instance, if competing workloads have periods, and hyper period P is a common multiple of these workload periods, recursiveness holds. Regularity means the following: Given any pair of time intervals X and Y such that $X.b \leq Y.b$ and $Y.e \leq X.e$, $B(Y) \leq B(X)$.

A pre-schedule $\mathbf{E}$ is *valid* if and only if the following sets of constraints are all satisfied. (1) Non-negative constraints: For any executive E in $\mathbf{E}$, the execution time $E.c \geq 0$. (2) Scope constraints: The valid scope of any executive is within the valid scope of its corresponding job; i.e., let E be an executive of job J, $J.r \leq E.r \leq E.d \leq J.d$. (3) Demand constraints: For every job J in $\mathbf{J}$, the aggregate execution time of its executive(s) is equal to the execution time of J. (4) Supply constraints: An executive E is *within* time interval I if and only if one of the following cases is true: (a) $I.b \leq E.r$ and $E.d \leq I.e$, or (b) $I.b \leq E.r + P$ and $E.d + P \leq I.e$; for every time interval I such that $0 \leq I.b < P$ and $I.e - I.b \leq P$, the aggregate execution time of all executives within I is upper bounded by $B(I)$. Later in Chapter 7, we consider other types of constraints.

3.3 LP-Based Basic Pre-Scheduler

The pre-scheduler is defined by two steps. Step One creates a partially defined pre-schedule $\mathbf{F}$, which does not define the execution times of executives. Step Two solves the execution times and produces a fully defined and valid pre-schedule $\mathbf{E}$.

3.3.1 Step One: Generate F

This step creates a list of partial executives $\mathbf{F}$. The corresponding job and valid scope are defined in each of these partial executives, but the execution time is not. This step consists of several sub-steps.

In the first sub-step, $\mathbf{F}$ is initiated as follows: One partially defined executive $(J, J.r, J.d)$ is created in $\mathbf{F}$ for each job J in $\mathbf{J}$.

The second sub-step transforms $\mathbf{F}$ into a set of *simple* executives. An executive F_x is simple if and only if for any executive F_y in $\mathbf{F}$, valid scope of F_x does not contain the valid scope of F_y. In this sub-step, the following transformation is iteratively applied until the condition is no longer true: If there exists a pair of executives F_x and F_y in $\mathbf{F}$ and $(F_x.r,\ F_x.d)$ contains $(F_y.r,\ F_y.d)$, then replace F_x by two executives — $(F_x.J,\ F_x.r,\ F_y.d)$ and $(F_x.J,\ F_y.r,\ F_x.d)$.

The third sub-step sorts $\mathbf{F}$ such that the following condition is true thereafter: For arbitrary pairs of executives F_x and F_y in $\mathbf{F}$, where x and y are indexes of $\mathbf{F}$, $x < y$ if and only if either (1) $(F_x.r,\ F_x.d)$ is before $(F_y.r,\ F_y.d)$ or (2) $(F_x.r,\ F_x.d)$ is parallel to $(F_y.r,\ F_y.d)$, $F_x.J = J_u$ and $F_y.J = J_v$, where u and v are indexes of $\mathbf{J}$ and $u < v$. Notice that $(F_x.r,\ F_x.d)$ can not contain or be contained by $(F_y.r,\ F_y.d)$, since all executives in $\mathbf{F}$ are simple at this point. Text-book algorithms are applicable for the sorting.

The fourth sub-step augments a variable to each partial executive F in $\mathbf{F}$. Assume that F is defined as (J, r, d), transform it to $(J, r, d, x_{J,k})$, where k is the sequence number for all partial executives of J in $\mathbf{F}$. Variable $x_{J,k}$ represents the unsolved execution time of the k^{th} executive of job J in $\mathbf{F}$.

Example 3 $\mathbf{J}$ *is defined in Example 1. Compute* $\mathbf{F}$.

$$\begin{aligned}\mathbf{F} \ = \ & [(C,0,9,x_{C,0}),(E,0,9,x_{E,0}),(A,1,9,x_{A,0}),(C,1,24,x_{C,1}),(E,1,24,x_{E,1}),\\ & (D,14,24,x_{D,0}),(B,16,24,x_{B,0}),(C,16,40,x_{C,2}),(D,16,40,x_{D,1}),\end{aligned}$$

$(E, 16, 45, x_{E,2})]$

3.3.2 Step Two: Solve the Execution Times of Executives

It turns out that the execution times of executives can be solved as a Linear Programming (LP) problem. We review LP problem first. A LP problem is defined by the following entities:

- a set of n variables: $\mathbf{V} = \{x_i | 0 \leq i < n\}$.
- a set of linear constraints: $\mathbf{L} = \{\sum_{\mathbf{V}} a_{i,j} \cdot x_i = b_j | 0 \leq j < m\}$, where $a_{i,j}$ and b_j are constants.
- an objective function: $o = \sum_{\mathbf{V}} c_i \cdot x_i$, where c_i are constants.

A *solution* to the LP problem is a non-negative value assignment to the variables in $\mathbf{V}$ such that the constraints in $\mathbf{L}$ are satisfied. An *optimal solution* is a solution which minimizes the objective function.

Notice that the following varieties can be made in the definition of LP. First, the existence of objective function is optional, and the objective function can be maximized instead of minimized. Second, an linear constraint can also be defined in the following forms: $\sum_{\mathbf{V}} c_{i,j} \cdot x_i \geq b_j$; $\sum_{\mathbf{V}} c_{i,j} \cdot x_i \leq b_j$. An LP problem with any of these varieties can be easily transformed to an LP problem in the form we defined above.

The execution times of executives are solved under the following three sets of constraints: non-negative constraints, demand constraints, and supply constraints. If solution does not exist, pre-scheduler returns failure.

(1) Non-negative constraints: the execution time of each executive to be non-negative; i.e., $x_{J,k} \geq 0$ for every executive.

(2) Demand constraints: for every job J in $\mathbf{J}$, the aggregate execution time of its executive(s) is equal to the execution time of J; i.e., $\sum_J x_{J,k} = J.c$.

Table 3.1: Supply Contract $B(I)$ on Critical Intervals

I.b	I.e	9	24	40	45	54
0		7	13	18	18	
1		7	13	18	18	
14			7	9	9	18
16			7	9	9	18

(3) Supply constraints on critical intervals: A time interval (b, e) is *critical* if and only if the following conditions are all true: (1) $0 < e - b \leq P$; (2) time b is between $(0, P)$, and there exists a job J_x in $\mathbf{J}$ and $b = J_x.r$; (3) there exists a job J_y in $\mathbf{J}$, such that either $e = J_y.d$ or $e = J_y.d + P$. Supply constraints on critical intervals are defined as follows. Recall that an executive E is *within* I if and only if either (1) $I.b \leq E.r$ and $E.d \leq I.e$ or (2) $I.b \leq E.r + P$ and $E.d + P \leq I.e$.

$$\text{for every critical interval } I, \quad \sum_{E \text{ is within } I} E.x \leq B(I)$$

Example 4 *Show an example of supply constraints.*

A supply contract $B(I)$ [1] on all critical intervals are defined in Table 3.1. in which the start times and end times of critical intervals are shown in the first column and the first row, and $B(I)$ is shown at the cross of row $I.b$ and column $I.e$. ■

Three sets of constraints are all linear. Therefore the execution times can be solved by a Linear Programming(LP) solver.

Example 5 $\mathbf{J}$ *and* $\mathbf{F}$ *are defined in Example 1 and 4 respectively. Compute* $\mathbf{E}$.

Non-negative constraints are defined as follows:

$$x_{A,0}, x_{B,0}, x_{C,0}, x_{C,1}, x_{C,2}, x_{D,0}, x_{D,1}, x_{E,0}, x_{E,1}, x_{E,2} \geq 0$$

[1] Subsection 5.2 of [30] shows how this supply contract is obtained from an example.

Demand constraints are defined as follows:

$$\begin{aligned}
x_{A,0} &= 1 \\
x_{B,0} &= 1 \\
x_{C,0} + x_{C,1} + x_{C,2} &= 8 \\
x_{D,0} + x_{D,1} &= 4 \\
x_{E,0} + x_{E,1} + x_{E,2} &= 3
\end{aligned}$$

There is one supply constraint corresponding to every critical interval. If a supply constraint is satisfied by any solution that satisfies other constraints, the supply constraint is *trivial.* A set of non-trivial supply constraints, which are on critical intervals (0, 9), (0, 24) and (14, 45), are listed below.

$$\begin{aligned}
x_{C,0} + x_{E,0} + x_{A,0} &\leq 7 \\
x_{C,0} + x_{E,0} + x_{A,0} + x_{C,1} + x_{E,1} + x_{D,0} + x_{B,0} &\leq 13 \\
x_{D,0} + x_{B,0} + x_{C,2} + x_{D,1} + x_{E,2} &\leq 9
\end{aligned}$$

A solution to this LP problem is as follows:

$$\begin{aligned}
x_{A,0} &= 1, \\
x_{B,0} &= 1, \\
x_{C,0} &= \frac{1}{2}, \quad x_{C,1} = 7, \quad x_{C,2} = \frac{1}{2}, \\
x_{D,0} &= 2\frac{1}{3}, \quad x_{D,1} = 1\frac{2}{3}, \\
x_{E,0} &= \frac{2}{5}, \quad x_{E,1} = \frac{3}{5}, \quad x_{E,2} = 2
\end{aligned}$$

The pre-schedule corresponding to this solution is defined as follows:

$$\begin{aligned}
\mathbf{E} = {} & [(C,0,9,\frac{1}{2}),(E,0,9,\frac{2}{5}),(A,1,9,1),(C,1,24,7),(E,1,24,\frac{3}{5}),(D,14,24,2\frac{1}{3}), \\
& (B,16,24,1),(C,16,40,\frac{1}{2}),(D,16,40,1\frac{2}{3}),(E,16,45,2)]
\end{aligned}$$

3.4 Soundness, Completeness and Time Complexity

We prove the soundness and completeness of the LP-based pre-scheduler defined in Section 3.3 by Theorem 1 and 2. Then we discuss the time complexity of the pre-scheduler.

Lemma 1 *If supply constraints on critical intervals are satisfied, supply constraints on all intervals are satisfied.*

Proof: Recall that localization of supply contract requires that hyper period P is sufficiently long such that for any time interval longer than P, supply constraint will be satisfied. Let I be a time interval whose length is less than or equal to P. Let $Demand(I)$ be the aggregate execution time of all executives that must be scheduled within I. There are two cases. Case 1: I is located in one hyper interval; i.e., $\lfloor \frac{I.b}{P} \rfloor = \lfloor \frac{I.e}{P} \rfloor$. Define time interval $\overline{I}$ as follows: $I^m.b = I.b \bmod P$ and $I^m.e = I.e \bmod P$. Since the same pre-schedule is followed in every hyper period, $Demand(I) = Demand(I^m)$. By recursiveness of supply contract, $B(I) = B(I^m)$. Let E_b be the first executive in $\mathbf{E}$ satisfying $I^m.b \leq E_b.r$ and E_e be the last executive in $\mathbf{E}$ satisfying $E_e.d \leq I^m.e$. Let time interval I^c be $(E_b.r,\ E_e.d)$, then $Demand(I^m) = Demand(\overline{I}^c)$. I^c is a critical interval, therefore supply contract is satisfied on I^c: $Demand(I^c) \leq B(I^c)$. By regularity of supply contract, $B(I^c) \leq B(I^m)$. Therefore $Demand(I) \leq B(I)$.

Case 2: Time interval I straddles a pair of adjacent hyper intervals; i.e., $\lfloor \frac{I.b}{P} \rfloor + 1 = \lfloor \frac{I.e}{P} \rfloor$. Define time interval I^m as follows: $I^m.b = I.b \bmod P$ and $I^m.e = P + I.e \bmod P$. Still, $Demand(I) = Demand(I^m)$, and $B(I) = B(I^m)$. Let E_b be the first executive in $\mathbf{E}$ satisfying $I^m.b \leq E_b.r$ and E_e be the last executive in $\mathbf{E}$ satisfying $P + E_e.d \leq I^m.e$. Let time interval I^c be $(E_b.r,\ P + E_e.d)$, then $Demand(I^m) = Demand(I^c)$. I^c is a critical interval, then still $Demand(I^c) \leq B(I^c)$. By regularity of supply contract, $B(I^c) \leq B(I^m)$. Therefore $Demand(I) \leq$

$B(I)$. ■

Theorem 1 *A pre-schedule produced by the LP-based pre-scheduler is valid.*

Proof: We need to prove that the sets of constraints of a valid pre-schedule defined in Section 3.2 are all satisfied.

Non-negative constraints and demand constraints are explicitly satisfied by Step Two. Supply constraints on critical intervals are explicitly satisfied in Step Two. According to Lemma 1, all supply constraints are satisfied. In Step One, the valid scope of every executive is created to be within the valid scope of its corresponding job. Therefore scope constraints are satisfied. ■

Theorem 2 *The pre-scheduler produces a pre-schedule if a valid pre-schedule exists.*

Proof: The pre-scheduler produces a pre-schedule if and only if there is a solution for the three sets of constraints defined in Step Two. Let $\mathbf{E}^v$ be a valid pre-schedule, we construct a pre-schedule $\mathbf{E}$ according to the partial pre-schedule $\mathbf{F}$ produced in Step One and $\mathbf{E}^v$, and prove that $\mathbf{E}$ satisfies the three sets of constraints.

Let E^v be an executive of a job J in $\mathbf{E}^v$. According to valid scope constraints in the definition of a valid pre-schedule and the construction of $\mathbf{F}$ in Step One, there must exist a partial executive E of job J in $\mathbf{F}$, such that E^v is always scheduled within $(E.r, E.d)$. We say such an E is *corresponding* to E^v. Since the valid scopes of adjacent executives in $\mathbf{F}$ may overlap, there exists one *or two* corresponding executives for one E^v.

Pre-schedule $\mathbf{E}$ is constructed as follows. (1) Initialization: Let $\mathbf{E}$ be a copy of $\mathbf{F}$, except that for every executive E of in $\mathbf{E}$, $E.c = 0$. (2) For every executive E^v in $\mathbf{E}^v$, add $E^v.c$ to *one* of its corresponding executives in $\mathbf{E}$.

$\mathbf{E}$ satisfies the three sets of constraints. (1) Non-negative constraints are obviously satisfied. (2) Demand constraints: For every job J, let W_J and W_J^v be the aggregate execution time of its executives in $\mathbf{E}$ and $\mathbf{E}^v$ respectively. Because $\mathbf{E}^v$

is a valid pre-schedule, $W_J^v = J.c$. According to the construction of $\mathbf{E}$, $W_J = W_J^v$, therefore $W_J = J.c$. (3) Supply constraints: Let (b, e) be a critical interval. Let $\mathbf{W}$ and $\mathbf{W}^v$ be the set of executives that must scheduled between a critical interval I in $\mathbf{E}$ and $\mathbf{E}^v$ respectively. Since $\mathbf{E}^v$ is valid, $\sum_{E^v \in \mathbf{W}^v} E^v.c \leq B(I)$. For an executive $E \in \mathbf{W}$, for every E^v whose execution time is added to E in the construction, $E^v \in \mathbf{W}^v$. Therefore, $\sum_{E \in \mathbf{W}} E.c \leq \sum_{E^v \in \mathbf{W}^v} E^v.c \leq B(I)$. ■

The time complexity of pre-scheduler is dominated by that of the LP solver. Let n be the number of jobs in $\mathbf{J}$, and $LP(x, y)$ be the complexity of LP with x variables and y constraints. The number of executives is upper bounded by n^2. The number of non-negative constraints and the number of sufficient constraints are both upper bounded by n, and the number of supply constraints is upper bounded by n^2. Therefore, the dominating factor of the pre-scheduler is bounded by $LP(n^2, n^2)$. Linear Programming is polynomial [13]. Algorithms and programs have been developed to solve practical linear programming problems with hundreds of thousands of constraints within reasonable length of time.

3.5 The Non-Existence of Universally Valid Pre-schedule

A pre-schedule is targeted to a specific supply contract, which imposes a set of supply constraints. Given a subject workload to be pre-scheduled, is it possible to produce a one-size-fits-all pre-schedule? To formalize the discussion, we define the concept of *universally valid pre-schedule*. For a given subject workload defined by $\mathbf{J}$, a pre-schedule $\mathbf{E^u}$ is universally valid if and only if one of the following conditions is true for any supply contract B: either (1) $\mathbf{E^u}$ is a valid pre-schedule; or (2) valid pre-schedule does not exist.

If universally valid pre-schedule exists, the following design scenario is complete: First generate a universally valid pre-schedule without any knowledge of competing components, then a feasibility test can be made to decide if a set of components, including the pre-scheduled one, is feasible. However, by Example 6, we will show that universally valid pre-schedule does not commonly exist. Therefore the scenario we surmise above is not complete. Instead, we shall take the following design scenario: First, the system designer shall produce a supply contract via a resource supply analysis, then the pre-scheduler produces a supply contract specific pre-schedule, or report un-pre-schedulability.

Example 6 *A workload to be pre-scheduled is defined as follows:*

$$\mathbf{J} = [A : (56, 75, 9), B : (0, 100, 71)]$$

Hyper period P is 100. Show universally valid pre-schedule does not exist for this workload to be pre-scheduled.

Construct two alternative sets of competing components modeled as sporadic task sets:

$$\mathbf{C} = \{(50, 10, 10)\}; \quad \mathbf{C}' = \{(20, 4, 4)\}$$

In both cases, hyper-period P is a common multiple of periods of competing workload.

Assume that the coordinating algorithm is Constrained Earliest Deadline First (CEDF). CEDF scheduler schedules the current executive in the pre-schedule and the sporadic jobs together by EDF: All arrived and uncompleted sporadic jobs and the current executive of the pre-schedule compete resource by deadline, a sporadic job or the current executive with the earliest deadline wins the resource. It can be implemented as follows. At the beginning of each hyper interval, let the first executive in the pre-schedule be marked as "current". Define $\mathbf{R}$ as the set of sporadic jobs waiting to be scheduled. The set $\mathbf{R}$ is initialized at time 0 as an empty set. When a sporadic job becomes ready, it is added into $\mathbf{R}$; when it is completely scheduled, it is removed from $\mathbf{R}$. At any time t, if the deadline d of the current executive is earlier than the deadline of any job in $\mathbf{R}$, the supply function to the pre-scheduled component $U(t) = 1$, then the current executive is scheduled; otherwise, $U(t) = 0$ and the sporadic job with the earliest deadline in $\mathbf{R}$ is scheduled. When the execution time of the current executive is completely scheduled, mark the next executive in the pre-schedule as "current", and so on.

There exists a valid pre-schedule $\mathbf{E}$ for $\mathbf{J}$ and $\mathbf{C}$, and a valid pre-schedule $\mathbf{E}'$ for $\mathbf{J}$ and $\mathbf{C}'$:

$$\begin{aligned} \mathbf{E} &= [(B,0,75,46),(A,56,75,9),(B,56,100,25)] \\ \mathbf{E}' &= [(B,0,75,48),(A,56,75,9),(B,56,100,23)] \end{aligned}$$

Suppose there is a universally valid pre-schedule $\mathbf{E^U}$. Let x be the aggregate execution time of all executives of B before the last executive of A in $\mathbf{E^U}$; let y be the aggregate execution time of all executives of B after the first executive of A in $\mathbf{E^U}$. A universally valid pre-schedule $\mathbf{E^U}$ must satisfy the following set of contradicting constraints, so it does not exist.

$$\begin{aligned} x+y &\geq 71 && \text{demand constraint for } B \\ x &\leq 46 && \text{supply constraint on } (0,75) \text{ for } \mathbf{C} \\ y &\leq 23 && \text{supply constraint on } (56,100) \text{ for } \mathbf{C}' \end{aligned}$$

3.6 Related Work

Search-based algorithms have been developed for static schedule generation. Peng *et al* proposed a branch and bound search algorithm [21]. Ramamritham proposed a heuristic search algorithm [22]. Fohler proposed a search algorithm based on precedence graph traversing [6]. Tsou proposed a search algorithm, which solves mutual exclusion and distance constraints with sophisticated backtracking techniques [27]. Pre-scheduling technique presented in this paper does not assume constant and predictable resource supply rate, and it is based on LP instead of search.

Fohler and Isovic developed acceptance tests for sporadic and aperiodic tasks competing with a given static schedule under the assumption that the online scheduler is Slot Shifting [7, 12]. This paper investigates the pre-schedule generation problem instead of the acceptance test problem.

Gerber *et al* proposed a parametric scheduling scheme [9]. They assumed that the execution times of tasks may range between upper and lower bounds, and there are relative timing constraints between tasks. The off-line component formulates a "calendar" which stores functions to compute the lower and upper bounds of the start time for each task. The bounds on the start time are computed online, upon which the online dispatcher decides when to start the real-time tasks. The parametric scheduling scheme assumes that the order of the tasks is *given* and is fundamentally different from the pre-scheduling problem we investigate. The techniques applied in pre-scheduling are also quite different from those applied in parametric scheduling scheme.

Erschler *et al* [5] and Yuan *et al* [37] focused on non-preemptive scheduling of periodic tasks. Erschler *et al* introduced the concept of "dominant sequence" which defines the set of possible sequences for non-preemptive schedules. Building upon the work of Erschler*et al*, Yuan*et al* proposed a "decomposition approach". Yuan*et al* defined several relations between jobs, such as "leading" and "containing", and applied them in a rule-based definition of "super sequence" which is equivalent to dominant sequence. The partially defined pre-schedule **F** in our paper is similar to the dominant sequence or the super sequence, and we adopt some of their concepts and terminology as mentioned. However, in view of the NP-hardness of the non-preemptive scheduling problem, those authors relied on approximate search algorithms to find a schedule. Our paper shows that the preemptive version of pre-scheduling problem can be completely solved in polynomial time by the LP-based approach on the domain of rational numbers.

3.7 Summary

This chapter defines a LP-based pre-scheduler with the following properties.

- Generality: The pre-scheduler does not depend on detailed assumptions about competing workloads and composition mechanisms.
- Segregation: The interface of supply function and supply contract segregate a pre-scheduled component and the system. The pre-scheduler depends on supply contract and the specification of workload to be pre-schedule, and the online scheduler of a pre-scheduled component depends on the supply function and the pre-schedule. However, he pre-scheduler and online scheduler do not depend on detailed assumptions about competing workloads and their scheduling and composition mechanisms.
- Soundness: a pre-schedule produced by the pre-scheduler is always valid.
- Completeness: the pre-scheduler produces a pre-schedule if there exists a valid pre-schedule.
- Efficiency: The complexity of online scheduler of a pre-scheduled component is $O(1)$; the off-line pre-scheduler terminates in time polynomial to the number of jobs in the subject workload.

Chapter 4

Pre-Scheduling on The Domain of Integers

Since infinitely small time slices are not implementable for resources with context switch overhead, it is desirable to define and solve the pre-scheduling problem on the domain of integers so that context switching can occur only at boundaries of time quantums. However, Integral LP (ILP) is NP-hard in the strong sense in general, so the ILP approach is not applicable and better techniques are needed. This chapter answers this challenge by giving a sound, complete and PTIME rational-to-integral pre-schedule transformer based on a novel technique which we call “round-and-compensate”. Section 4.1 provides the background, rationale of the integral pre-scheduling problem and top layer description of our solution. Section 4.2 describes our “round-and-compensate” approach for transforming pre-schedules to the domain of integers. Section 4.3 analyzes the transformer. Section 4.4 presents a direct LP approach for generating integral pre-schedules, which is built upon the idea of round-and-compensate. Section 4.5 addresses relation work. Section 4.6 summarizes the transformer and its implication.

4.1 Introduction

Context switches require overheads. For instance, when a CPU is switched between processes, values of registers need to be saved and restored, which consumes computation time. Since context switch overhead must be counted into a schedule, a minimum size must be set for every "slice", which is the time interval in a schedule assigned to a job. For this purpose, the concept of "time unit" is introduced. A time unit has a fixed length; e.g., it could be 10 ms. The resource could be assigned to at most one job in a single time unit (commonly called the quantum) and context switch may only occur between adjacent time units. The size of a time unit can be set to a value great enough such that context switch overhead is upper bounded by a fraction of a time unit. When resource is scheduled by whole time units, the scheduling problem is defined on the domain of integers. Due to the common existence of context switch overheads, the pre-scheduling problem shall also be defined and solved on the domain of integers in order to be practically useful.

The pre-scheduling problem can be easily defined on the domain of integers: (1) Common workload models, such as periodic tasks and sporadic tasks, can be defined by integers; (2) Common composition algorithms, such as Slot Shifting [12], Earliest Deadline First, and Fixed Priorities, can be applied on the domain of integers; (3) An online scheduler in a pre-scheduled component, such as which is defined in Section 3.2, can also be applied on the domain of integers. However, solving the integral pre-scheduling problem is non-trivial. The LP-based pre-scheduler described in Chapter 3 constructs and solves a Linear Programming (LP) problem. LP is polynomial on the domain of rational numbers [13, 15], but it is NP-Complete in the strong sense on the domain of integers [2, 14]. Therefore, the naive solution of solving the Integral LP (ILP) problem is not effective.

This chapter solves the integral pre-scheduling problem. The framework of this solution is illustrated in Figure 4.1. A LP-based pre-scheduler produces a valid pre-schedule of rational numbers, then a rational-to-integer transformer produces a valid integral pre-schedule.

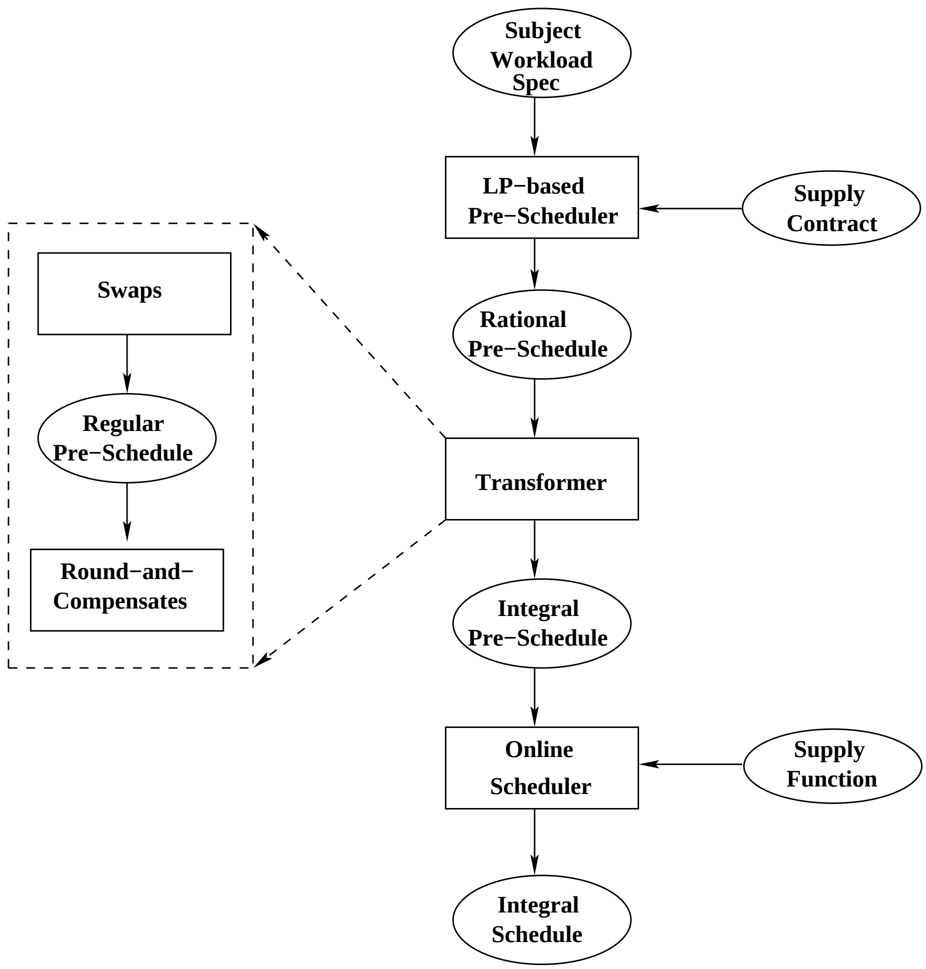

Figure 4.1: Framework of Pre-Scheduling on The Domain of Integers

The rational-to-integral transformer is the highlight of this chapter. Naive rounding has been a common practice in producing approximate results of ILP problems: Given an ILP problem, "relax" it to the domain of rational numbers and obtain a solution there, then "round" the solution back to the domain of integers.

This naive rounding approach is approximate by nature. The transformer in this chapter, however, is based on a sophisticated rounding technique, which we call "round-and-compensate": if the execution time of an executive of job J is rounded off by a value of δ, then the execution time of another executive of job J will be increased by δ. The rational-to-integral transformer is designed as follows. First, the transformer executes a sequence of swaps, which translates a valid pre-schedule into a "regular" form. Then the regular and valid pre-schedule will be iteratively rounded-and-compensated until execution times of all executives are changed to integers. This transformer is *not* approximate; instead, it is sound and complete: if the pre-scheduling problem is defined on the domain of integers, every valid pre-schedule is transformed to a valid integral pre-schedule.

To deepen the theoretical insight over the integral pre-scheduling problem, we also show that the integral pre-scheduling problem can be solved by a direct (non-integral) LP approach, without explicit round-and-compensate.

4.2 Rational-to-Integral Transformer

Assume that a pre-scheduling problem is defined on the domain of integers. The ready time and deadline of each executive is always on the domain of integers in the pre-schedule produced by the basic LP-based pre-scheduler. However, since the LP problem is solved on the domain of rational numbers, the execution times are not guaranteed to be integers. The mission of the rational-to-integral transformer is to transform a valid pre-schedule from the domain of rational numbers to the domain of integers. There are two major steps in the transformer. In the first step, a sequence of swaps transforms a pre-schedule to be "regular"; in the second step, a sequence of round-and-compensate actions transforms the execution times of a regular pre-schedule to integers.

4.2.1 Swaps

To facilitate the definition of swap, we introduce the concept of *overlapping pair.* Assume that there is a pair of jobs J_x and J_y in $\mathbf{J}$. Let E_u be an executive of J_x, and let E_v be an executive of J_y. Without losing generality, assume $x < y$, which implies that one of the following two cases apply: (1) J_x is contained by J_y ; or (2) Either J_x is before or parallel to J_y. Under Case (1), executives E_u and E_v form an overlapping pair if $E_u.r = E_v.r$; Under Case (2), they form an overlapping pair if either $E_u.r = E_v.r$ or $E_u.d = E_v.d$. Let $\mathbf{O}(J_x, J_y)$ be a list of all overlapping pairs of executives of J_x and J_y, which is in the ascending order of the ready times of all executives of J_x in all pairs. $\mathbf{O}(J_x, J_y)$ is also notated as $[\{E_{x_i}, E_{y_i}\}|0 \leq i < n]$, where n is the number of overlapping pairs, i is the index of overlapping pairs, and x_i and y_i are the indexes of executives in $\mathbf{E}$.

$\mathbf{O}(J_x, J_y)$ is *regular* if and only if the following condition is true: There exists a *middle pair* (E_{x_m}, E_{y_m}) in $\mathbf{O}(J_x, J_y)$, such that the following conditions are all true. (1) For any $0 \leq i < m$, $E_{y_i}.c = 0$; (2) For any $m < i < n$, $E_{x_i}.c = 0$. If for every pair of jobs J_x and J_y in $\mathbf{J}$ with $x < y$, $\mathbf{O}(J_x, J_y)$ is regular, then pre-schedule $\mathbf{E}$ is *regular.*

A *swap* between executives of jobs J_x and J_y is notated as $SWAP(J_x, J_y)$, and it modifies the execution times of the executives in $\mathbf{E}$ under the following constraints. X and X' represent the value of an entity before and after $SWAP(J_x, J_y)$ here. (1) Only the execution times of executives in overlapping pairs in $\mathbf{O}(J_x, J_y)$ can be modified. (2) $\mathbf{O}'(J_x, J_y)$ is regular. (3) The aggregate execution time of executives in each overlapping pair in $\mathbf{O}(J_x, J_y)$ remains the same before and after $SWAP(J_x, J_y)$; i.e., for each $0 \leq i < n$, where n is the number of overlapping pairs, $E_{x_i}.c + E_{y_i}.c = E'_{x_i}.c + E'_{y_i}.c$. (4) The aggregate execution time of all executives of J_x remains the same before and after $SWAP(J_x, J_y)$; i.e., $\sum_{0 \leq i < n} E_{x_i}.c = \sum_{0 \leq i < n} E'_{x_i}.c$. (5) The aggregate execution time of all executives of J_y remains the same before and after $SWAP(J_x, J_y)$; i.e., $\sum_{0\leq i<n} E_{y_i}.c = \sum_{0\leq i<n} E'_{y_i}.c$.

Example 7 **J** *and* **E** *are defined in Example 1 and 5. Execute $SWAP(C, D)$.*

Let $\mathbf{O}(C, D)$ be the overlapping pairs before $SWAP(C, D)$; and let $\mathbf{O}'(C, D)$ and $\mathbf{E}'$ be the overlapping pairs and the pre-schedule after it.

$$
\begin{aligned}
\mathbf{O}(C,D) &= [((C,1,24,7),(D,14,24,2\tfrac{1}{3})),((C,16,40,\tfrac{1}{2}),(D,16,40,1\tfrac{2}{3}))] \\
\mathbf{O}'(C,D) &= [((C,1,24,\underline{7\tfrac{1}{2}}),(D,14,24,\underline{1\tfrac{5}{6}})),((C,16,40,\underline{0}),(D,16,40,\underline{2\tfrac{1}{6}}))] \\
\mathbf{E}' &= [(C,0,9,\tfrac{1}{2}),(E,0,9,\tfrac{2}{5}),(A,1,9,1),(C,1,24,\underline{7\tfrac{1}{2}}),(E,1,24,\tfrac{3}{5}), \\
&\quad (D,14,24,\underline{1\tfrac{5}{6}}),(B,16,24,1),(C,16,40,\underline{0}),(D,16,40,\underline{2\tfrac{1}{6}}), \\
&\quad (E,16,45,2)]
\end{aligned}
$$

■

The sequence of swaps is defined by Algorithm 10, in which n is the number of jobs in **J**.

Algorithm 10: The Sequence of Swaps

```
i := 1;
while i ≤ n − 1
    j := 0;
    while j < i
        SWAP(J_j, J_i);
        j := j + 1;
    i := i + 1;
```

Example 8 **J** *and* **E** *are defined in Example 1 and 5. Transform* **E** *according to Algorithm 10.*

Before the execution of Algorithm 10, $\mathbf{O}(C, D)$ and $\mathbf{O}(C, E)$ are not regular. According to Algorithm 10, $SWAP(C, E)$ is executed after $SWAP(C, D)$. After Algorithm 10, $\mathbf{E}'$, as shown below, is regular. The underlined values are modified during $SWAP(C, E)$.

$$\begin{aligned}\mathbf{E}' \quad = \quad & [(C, 0, 9, \underline{\frac{9}{10}}), (E, 0, 9, \underline{0}), (A, 1, 9, 1), (C, 1, 24, \underline{7\frac{1}{10}}), (E, 1, 24, \underline{1}), \\ & (D, 14, 24, 1\frac{5}{6}), (B, 16, 24, 1), (C, 16, 40, 0), (D, 16, 40, 2\frac{1}{6}), (E, 16, 45, 2)]\end{aligned}$$

4.2.2 Round-And-Compensate Transformations

For presentation convenience, we introduce the notations of sublists of $\mathbf{E}$. Let E_b and E_e be executives in pre-schedule $\mathbf{E}$ and $b < e$. $[E_b, E_e]$ represents the sublist of all executives in $\mathbf{E}$ between and *including* E_b and E_e; (E_b, E_e) represents the sublist of those between and *excluding* E_b and E_e; $[E_b, E_e)$ represents the sublist of those between E_b and E_e, *including* E_b but *excluding* E_e; and $(E_b, E_e]$ is symmetric to $[E_b, E_e)$.

A sublist is an *integral scope* if and only if the aggregate execution time of all executives in it is an integer. An integral scope $[E_b, E_e]$ is *simple* if and only if there exists *no* executive $E_{e'} \in [E_b, E_e)$ such that $[E_b, E_{e'}]$ is also an integral scope. A simple integral scope is called a *scope* for short under the context of executive sublist. A *coverage* $\mathbf{C}$ is a list of scopes of $[E_{b_i}, E_{e_i}]$, where i represents the index of scope in $\mathbf{C}$, and b_i (e_i) represents the index in $\mathbf{E}$ of the first (last) executive in the i^{th} scope in $\mathbf{C}$; the concatenation of all scopes in $\mathbf{C}$ is equal to $\mathbf{E}$.

Round-and-compensate transformation is defined as follows.

1. Compute $\mathbf{C}$.

2. Compute δ as follows. For any executive E_x in $\mathbf{E}$, if $E_x.c$ is an integer, $\Delta(E_x) = \infty$. Otherwise, there must exist i where $E_x \in [E_{b_i}, E_{e_i}]$, which is a scope in

C. $\Delta(E_x)$ is computed as follows:

$$\Delta(E_x) = \lceil \sum_{E_y \in [E_{b_i}, E_x]} E_y.c \rceil - \sum_{E_y \in [E_{b_i}, E_x]} E_y.c$$

Let δ be the minimum of $\Delta(E_x)$ for any executive E_x in **E**.

3. For every scope $[E_{b_i}, E_{e_i}]$ in **C**, conduct *execution time move* $E_{b_i} \leftarrow E_{e_i}(\delta)$, which is defined as $E_{b_i}.c := E_{b_i}.c + \delta$ and $E_{e_i}.c := E_{e_i}.c - \delta$.

If there exists any scope in **C** with more than one executive, **C** is rounded-and-compensated such that at least one scope is further split into two or more scopes. Iteratively apply this transformation until every scope has single executive, whose execution time must be an integer. Then concatenate **C** to **E** and eliminate executives with zero execution times.

Example 9 *Pre-schedule* **E** *is computed in Example 8. Transform* **E** *to the domain of integers.*

We list **C** and δ at each iteration of round-and-compensates. The modified values are underlined.

$$\begin{aligned}
\mathbf{C} &= [[(C,0,9,\frac{9}{10}),(E,0,9,0),(A,1,9,1),(C,1,24,7\frac{1}{10})],[(E,1,24,1)],\\
&\quad [(D,14,24,1\frac{5}{6}),(B,16,24,1),(C,16,40,0),(D,16,40,2\frac{1}{6})],[(E,16,45,2)]]\\
\delta &= \frac{1}{10}\\
\mathbf{C} &= [[(C,0,9,\underline{1})],[(E,0,9,0)],[(A,1,9,1)],[(C,1,24,\underline{7})],[(E,1,24,1)],\\
&\quad [(D,14,24,1\underline{\frac{14}{15}}),(B,16,24,1),(C,16,40,0),(D,16,40,2\underline{\frac{1}{15}})],[(E,16,45,2)]]\\
\delta &= \frac{1}{15}\\
\mathbf{C} &= [[(C,0,9,1)],[(E,0,9,0)],[(A,1,9,1)],[(C,1,24,7)],[(E,1,24,1)],\\
&\quad [(D,14,24,\underline{2})],[(B,16,24,1)],[(C,16,40,0)],[(D,16,40,\underline{2})],[(E,16,45,2)]]
\end{aligned}$$

Concatenate **C** and eliminate executives with zero execution times, and the result is the pre-schedule **E** shown below, (which is the same as shown in Example 2).

$$\mathbf{E} = [(C,0,9,1),(A,1,9,1),(C,1,24,7),(E,1,24,1),(D,14,24,2),(B,16,24,1),(D,16,40,2),(E,16,45,2)]$$

4.3 Analysis

We assume that the input of the transformer is a valid pre-schedule on the domain of rational numbers. The rational-to-integer transformer has the following properties. (1) Termination: The transformer terminates within $O(n^3)$, where n is the number of jobs in **J** (Theorem 3). (2) Validity: The transformer produces a valid pre-schedule (Theorem 4); (3) Integralization: The transformer produces a pre-schedule in the domain of integers (Theorem 4). We prove these properties in this section.

Lemma 2 *The output pre-schedule of Algorithm 10 is valid.*

Proof: Let X and X' represent some entity X before and after a swap $SWAP(J_x, J_y)$. We only need to prove that $\mathbf{E}'$ is a valid pre-schedule. Recall that the validity of pre-schedule is defined in Section 3.2.

Non-negative and scope constraints are obviously true in $\mathbf{E}'$, since the lowest execution time that could be assigned to an executive is 0 and valid scopes of executives are not modified by a swap. Demand constraints are explicitly maintained by constraints (4) and (5) in the definition of swap.

Now we prove that the supply constraints are also satisfied by $\mathbf{E}'$. According to Lemma 1, we only need to prove that supply constraints on critical constraints are all satisfied. Let I be a critical time interval, and let $\mathbf{W}(I)$ be the set of all executives within I: an executive E is in $\mathbf{W}(I)$ if and only if either $I.b \leq E.r$ and $E.d \leq I.e$, or $I.b + P \leq E.r$ and $E.d + P \leq I.e$. Notice that since swap does not change the valid scope of executives, E' is in $\mathbf{W}(I)$ if and only if E is in $\mathbf{W}(I)$. We only need to prove that $\sum_{E' \in \mathbf{W}(I)} E'.c \leq \sum_{E \in \mathbf{W}(I)} E.c$. Consider any overlapping pair of executives E_u of J_x and E_v of J_y, in $SWAP(J_x, J_y)$. For presentation convenience, we define $C(E_u, E_v)$ ($C'(E_u, E_v)$) as the contribution of this overlapping pair to $\sum_{E \in \mathbf{W}(I)} E.c$ ($\sum_{E' \in \mathbf{W}(I)} E'.c$). There are four cases. (1) Both E_u or E_v are in $W(I)$; then $C(E_u, E_v) = E_u.c + E_v.c$; (2) None of E_u or E_v is in $W(I)$: $C(E_u, E_v) = 0$; (3) E_u is in $W(I)$ and E_v is not: $C(E_u, E_v) = E_u.c$; (4) E_u is not in $W(I)$ and E_v is: $C(E_u, E_v) = E_v.c$; We only need to prove the following claim.

Claim 1: $C'(E_u, E_v) \leq C(E_u, E_v)$.

Consider the four cases. Constraint (3) in the definition of swap requires $E_u.c + E_v.c = E'_u.c + E'_v.c$. Therefore Claim 1 is true for Case (1). Claim 1 is trivially true under Case (2). Under Case (3), E_u and E_v is the last overlapping pair in $\mathbf{O}(J_x, J_y)$, therefore $E'_u.c \leq E_u.c$ by the definition of swap. Under Case (4), J_x is before J_y, E_u and E_v is the first overlapping pair in $\mathbf{O}(J_x, J_y)$, therefore, $E'_v.c \leq E_v.c$ by the definition of swap. ■

Lemma 3 *The output pre-schedule of Algorithm 10 is regular.*

Proof: Let x, y and z be indexes of jobs in $\mathbf{J}$ and $x < y < z$.

Claim 1: Right after $SWAP(J_x, J_y)$, $\mathbf{O}(J_x, J_y)$ is regular.

Claim 2: If $\mathbf{O}(J_x, J_y)$ is regular, after $SWAP(J_x, J_z)$, $\mathbf{O}(J_x, J_y)$ is still regular.

Claim 3: If $\mathbf{O}(J_x, J_y)$ and $\mathbf{O}(J_x, J_z)$ are regular, then after $SWAP(J_y, J_z)$, (1) $\mathbf{O}(J_x, J_y)$ is still regular, and (2) $\mathbf{O}(J_x, J_z)$ is still regular.

Now consider an arbitrary pair of jobs J_x and J_y in $\mathbf{J}$ such that $x < y$. According to Claim 1, right after $SWAP(J_x, J_y)$, $\mathbf{O}(J_x, J_y)$ is regular. According to Algorithm 10, the swaps thereafter in the same inner loop are in the form of $SWAP(J_w, J_y)$, where $x < w < y$. According to (2) of Claim 3, after $SWAP(J_w, J_y)$, $\mathbf{O}(J_x, J_y)$ is still regular. Then for any subsequent outer loop $i = z$, $SWAP(J_x, J_z)$ is executed first, then $SWAP(J_y, J_z)$ is executed. According to Claim 2 and (1) of Claim 3, $\mathbf{O}(J_x, J_y)$ is still regular by the end of Algorithm 10. We do not make any specific assumptions on x and y, therefore this result is true for any pair of jobs in $\mathbf{J}$. ■

In the following lemmas, we prove that if the input of a round-and-compensate $\mathbf{E}$ is a valid and regular pre-schedule, the output $\mathbf{E}'$ is also a valid and regular pre-schedule. It is trivial to prove that non-negative and scope constraints are still true in $\mathbf{E}'$. Other properties are proved in Lemma 9, 10, and 11.

For presentation convenience, we introduce the concept of *in-flow* and *out-flow* in a round-and-compensate. For every scope $[E_b, E_e]$ with more than one executive, E_b (E_e) has an in-flow (out-flow) during the round-and-compensate. Any other executive has neither in-flow nor out-flow. We use in/out-flow to represent "either an in-flow or an out-flow".

By the definition of coverage and in/out-flows, the following properties of in/out-flows hold. Let E_x and E_y be executives in $\mathbf{E}$ and $x < y$.

- Property 1: if any two of the following statements are true, then the third one is also true: (1) E_x has an in-flow. (2) E_y has an out-flow. (3) The aggregate execution time of all executives in $[E_x, E_y]$ is an integer.

- Property 2: if any two of the following statements are true, the third one is also true: (1) E_x has an out-flow. (2) E_y has an in-flow. (3) The aggregate execution time of all executives in (E_x, E_y) is an integer.

- Property 3: if any two of the following statements are true, the third one is also true: (1) E_x has an in-flow. (2) E_y has an in-flow. (3) The aggregate execution time of all executives in $[E_x, E_y)$ is an integer.

Now we prove the demand constraints are still satisfied by $\mathbf{E}'$. The strategy of proof is as follows. First, an important property of regular pre-schedule is proved in Lemma 4. Then we prove that the in-flow and out-flow executives of a job must strictly interleave each other by Lemma 5 and 6; i.e., an in-flow executive of a job J is either the last in/out-flow executive of J, or the next in/out-flow executive of J is an out-flow executive; and vice versa. Then we prove that if the first in/out-flow executive of J has an in-flow (out-flow), then the last in/out-flow executive of J must have an out-flow (in-flow) by Lemma 7 and 8. Therefore, the number of in-flows of J must be equal to the number of out-flows of J. Because all moves in the same round-and-compensate has the same adjustment value δ, the aggregate execution time of all executives of J does not change. ■

Recall that we assume that the pre-schedule is valid and regular.

Lemma 4 *Let E_b and E_e be non-zero executives of job J, $b < e$, and there does not exist non-zero executive of job J in (E_b, E_e). The aggregate execution time of all executives in (E_b, E_e) is an integer.*

Proof: For any job J^{other} other than job J, if there exists a non-zero executive of J^{other} in (E_b, E_e), then all non-zero executives of J^{other} is in (E_b, E_e). The aggregate execution time of all executives of J^{other} must be integer by its demand constraint. ■

Lemma 5 *Assume that E_b is an executive of job J with an out-flow, E_e is an executive of job J with a non-integer execution time, $b < e$, and for any executive E_x of job J such as $b < x < e$, $E_x.c$ is an integer. E_e must have an in-flow.*

Proof: According to Lemma 4, the aggregate execution time of all executives in (E_b, E_e) is an integer. According to Property 2 of in/out-flows, this lemma is true. ■

Lemma 6 *Assume that E_b is an executive of job J with an in-flow. At least one of the following cases is true: (1) There exists no executive E_e of job J, such that $b < e$ and E_e has an in/out-flow; or (2) there exists an executive E_e of job J, $b < e$, E_e has an out-flow, and there exists no executive E_x of job J such that $b < x < e$ and E_x has an in/out-flow.*

Proof: Assume the opposite: There exists an executive E_e of job J, $b < e$, E_e has an in-flow, and there exists no executive E_x of job J such that $b < x < e$ and E_x has an in/out-flow.

According to Property 3 of in/out flows, the aggregate execution time of all executives in $[E_b, E_e)$ is an integer. $E_b.c$ is not an integer, (otherwise it will not have an in-flow), then the aggregate execution time of all executives in (E_b, E_e) is not an integer. According to Lemma 4, there must exist executive(s) of J with non-integer execution times in (E_b, E_e). Let E_x be the last one of such executives. According to Lemma 4, the aggregate execution time of all executives in (E_x, E_e) is an integer. According to Property 2 of in/out flows, E_x has an out-flow. Contradiction. ■

Lemma 7 *Let E_f and E_l be the first and last executives of job J which have in/out-flows. If E_f has an in-flow, E_l has an out-flow.*

Proof: Claim 1: There exists no executive E_v of job J such that $v < f$ and $E_v.c$ is non-integer.

Otherwise, let E_v be the one with the largest index among such executives. According to Lemma 4, the aggregate execution time of all executives in (E_v, E_f) is an integer. According to Property 2 of in/out flows, E_v has an out-flow, contradiction to the lemma assumption.

Claim 2: There must exist executive(s) of J after E_f with non-integer execution time.

Because of the demand constraint, the aggregate execution time of all executives of J is equal to $J.c$, which is an integer. Because $E_f.c$ is not an integer and Claim 1, Claim 2 is true.

Let E_l be the last non-integer executive of J. Because of Claim 2, $f \neq l$.

Claim 3: E_l has an out-flow.

According to Claim 1 and the definition of E_l, the aggregate execution time of all executives of J in $[E_f, E_l]$ is an integer. According to Lemma 4, the aggregate execution time of all executives in $[E_f, E_l]$ is an integer. According to Property 1 of in/out flows, Claim 3 is true. ■

Lemma 8 *Let E_f and E_l be the first and last executives of job J which have in/out-flows. If E_f has an out-flow, E_l has an in-flow.*

Proof: Claim 1: The aggregate execution time of executives of J in $[E_0, E_f]$ is not an integer.

Assume that Claim 1 is false. Let E_v be the first executive with non-integer execution time of J. According to Lemma 4, the aggregate execution time for all executives in $[E_v, E_f]$ is an integer. According to Property 1 of in/out flows, E_v has an in-flow. It contradicts with the assumption on E_f.

Claim 2: There exists one or more non-integer executives of task J in $(E_f, E_{n-1}]$, where n is the number of executives in $\mathbf{E}$.

This claim follows Claim 1 and the demand constraint.

Claim 3: Let E_w be the first executive with non-integer execution time of J after E_f in $\mathbf{E}$. E_w has an in-flow.

The aggregate execution time of all executives in (E_f, E_w) is an integer, and E_f has an out-flow. Claim 3 follows Property 2 of in/out-flows.

If E_w is the last executive of J with an in/out-flow, lemma is proved. Otherwise, assume the opposite: the last executive of J with and in/out-flow is E_l and it has an out-flow. According to Property 1 of in/out-flows, the aggregate execution times of all executives in $[E_w, E_l]$ is an integer. Because $\mathbf{E}$ is regular, according to Lemma 4 the aggregate execution time of all executives of jobs other than J between and including $[E_w, E_l]$ is an integer. Therefore, the aggregate execution time of all executives of J between and including $[E_w, E_l]$ is an integer. According to Claim 1, there exists an executive E_v of J with non-integer execution time, and $l < v$. Without losing generality, let E_v be the one with lowest index among such executives. According to Lemma 4, the aggregate execution time of all executives of jobs other than J in (E_v, E_l) is an integer. According to the definition of E_v and E_l, the aggregate execution time of all executives of J in (E_v, E_l) is also an integer. Therefore, the aggregate execution time of all executives in (E_v, E_l) is an integer. According to Property 2 of in/out-flows, E_v has an in-flow. Contradiction to the assumption made on E_f. ■

Lemma 9 *The pre-schedule after a round-and-compensate still satisfies demand constraints.*

Proof: It follows Lemma 4 to Lemma 8. ■

Lemma 10 *The pre-schedule after a round-and-compensate still satisfies all supply constraints.*

Proof: According to Lemma 1, If supply constraints on critical intervals are satisfied, supply constraints on all intervals are satisfied. Let I be a critical interval.

Case 1: $0 \leq I.r$ and $I.d \leq P$. The supply constraint on I is

$$\sum_{I.b \leq E.r \text{ and } E.d \leq I.e} E.c \leq B(I)$$

Let E_b and E_e be the first and last executives within I. Let $E_x \leftarrow E_y(\delta)$ be a move. if $x < b$ and $b \leq y \leq e$, then it is a move *from* I; if $b \leq x \leq e$ and $e < y$, then this is a move *to* I. According to the definition of round-and-compensate, the number of moves from I is 0 or 1, and the number of moves to I is 0 or 1. If the number of moves to I is equal to the number of moves from I, then the aggregate execution time of executives within I does not change, then the supply constraint on I is still true. If the number of moves to I is 0 and the number of moves from I is 1, then the aggregate execution time of executives within I decreases, then the supply constraint on I is still true.

Assume the number of moves to I is 1 and the number of moves from I is 0. Let the move to I be $E_x \leftarrow E_y(\delta)$, where $b < x < e$. Let A be the aggregate execution time of all executives in $[E_b, E_x)$. Because there is no move from I, E_b must have an in-flow, therefore $A = A'$. Since both E_b and E_x have in-flows, A is an integer. (Recall Property 3 of in/out-flows). Let C be the aggregate execution time of all executives in $[E_x, E_e]$. According to the definition of coverage in round-and-compensate, C must be a non-integer. According to the definition of δ in round-and-compensate, $C' \leq \lceil C \rceil$.

$\mathbf{E}$ is a valid pre-schedule, so $A + C \leq B(I)$, so $A' + C' \leq \lceil B(I) \rceil$. Since the pre-scheduling problem is defined on the domain of integers, $B(I)$ is an integer. Therefore, $\lceil B(I) \rceil = B(I)$. Then $A' + C' \leq B(I)$.

Case 2: $0 \leq I.b < P < I.e$. Recall that under this case, the supply constraint over I is defined as follows:

$$\sum_{I.b \leq E.r \text{ or } E.d+P \leq I.e} E.c \leq B(I)$$

Let E_b be the first executive such that $I.b \leq E_b.r$, and let E_e be the last executive such that $(E_e.d + P \leq I.e)$. Similar to Case 1, The proof is non-trivial only when (1) there exists a move $E_u \leftarrow E_w(\delta)$, where $0 < u < e < b$, and (2) there exists no move $E_x \leftarrow E_y(\delta)$, where $e < x < b < y$. Again similar to Case 1, the increase of aggregate execution time within I does not across the integer boundary of $B(I)$. Therefore the supply constraint still holds. ■

Lemma 11 *The pre-schedule after a round-and-compensate is regular.*

A round-and-compensate does not create or delete executives, and it does not change the order of executives. A round-and-compensate does not change the execution time if an execution time has been an integer. Particularly, a round-and-compensate does not change a zero executive to a non-zero executive.

Case 1: J_a is before J_b, or J_a is parallel to J_b, and $a < b$. Let E_x be the last non-zero executive of J_a, and let E_y be the first non-zero executive of J_b. Since $\mathbf{E}$ is regular, $x < y$. Since a round-and-compensate does not change an zero executive to an non-zero executive, all executives of J_a after E_x remain zero executives in $\mathbf{E}'$, and all executives of J_b before E_y remain zero executives in $\mathbf{E}'$. Therefore $\mathbf{O}'(J_a, J_b)$ is still regular in $\mathbf{E}'$.

Case 2: J_a contains J_b. Let E_x and E_y be the first and last non-zero executive of J_b. Since $\mathbf{E}$ is regular, all executives of J_a in (E_x, E_y) are zero executives. The rest of the proof is similar to that of Case 1. ∎

Theorem 3 *The complexity of the transformer is $O(n^3)$, where n is the number of jobs in $\mathbf{J}$.*

Proof: The complexity of each swap or round-and-compensate is $O(n)$. Because of the structure of double loops in Algorithm 10, the number of swaps is $O(n^2)$. Every round-and-compensate increases the number of scopes in coverage $\mathbf{C}$. The number of executives in all scopes in $\mathbf{C}$ does not change during round-and-compensates and it is upper bounded by n^2, Therefore the number of round-and-compensate transformations is bounded by $O(n^2)$. ∎

Theorem 4 *The rational-to-integer transformer produces a valid pre-schedule in the domain of integers.*

Proof: According to Lemma 2, 3, 9, 10, and 11, the sequence of swaps produces a valid and regular pre-schedule, then every round-and-compensate transforms a valid and regular pre-schedule into another valid and regular pre-schedule. Therefore the result of the transformer is a valid pre-schedule. At the termination of round-and-compensate transformations, every simple integral scope contains a single executive, so the execution time of every executive must be an integer. ■

4.4 Direct LP Approach

As shown in Chapter 3 and 4, a basic pre-scheduling problem can be transformed to an LP problem and solved on the domain of rational numbers; then, given the pre-scheduling problem defined on the domain of integers, this solution can be transformed to the domain of integers. In this section, we propose an alternative approach *without* explicit rational-to-integer transformation, which we call *direct LP* approach. By direct LP approach, we simply transform the pre-scheduling problem to an LP problem with an objective function. We can prove that any optimal solution to this LP problem must be on the domain of integers.

4.4.1 The Algorithm

In direct LP solution, Step One is the same as defined in the basic LP solution in Subsection 3.3.1. In Step Two, the non-negative constraints, demand constraints, and supply constraints are defined the same as in the basic LP solution in Subsection 3.3.2. However, in direct LP solution, We define an objective function o as follows. Let $x_{i,j}$ be the execution time of the j^{th} executive of job J_i in $\mathbf{E}$. $o = \sum c_{i,j} \cdot x_{i,j}$, where $c_{i,j}$ is the coefficient of $x_{i,j}$ in the objective function. The coefficients are defined by the following algorithm:

Algorithm 11: Defining Objective Function Coefficients

```
i := n − 1;
d_i := 1;
while i > 0
    let m be the number of executives of J_i in E;
    foreach τ_i ∈ T
        foreach j ∈ [0..m − 1]
            c_{i,j} = d_i · j;
        d_{i−1} := d_i · m_i;
        i := i − 1;
```

Then we seek a solution to minimize this objective function, subject to the sets of constraints listed in Sub-section 3.3.2.

Example 10 **J** *and* **F** *are defined in Example 1 and 3 respectively. The non-negative, demand and supply constraints are defined in Example 5. Define the objective function, and show a solution to minimize the objective function, subject to the constraints.*

The computation of Algorithm 11 is illustrated in Table 4.1. Every line in the table corresponds to an iteration of the loop in Algorithm 11.

Table 4.1: The Computation of Coefficients in the Objective Function

i	d_i	$c_{i,j}$		
4	1	$c_{E,0} = 0;$	$c_{E,1} = 1;$	$c_{E,2} = 2$
3	3	$c_{D,0} = 0;$	$c_{D,1} = 3$	
2	6	$c_{C,0} = 0;$	$c_{C,1} = 6;$	$c_{C,2} = 12$
1	12	$c_{B,0} = 0$		
0	12	$c_{A,0} = 0$		

Therefore, the objective function is defined as follows:

$$o = 6x_{C,1} + 12x_{C,2} + 3x_{D,1} + 1x_{E,1} + 2x_{E,2}$$

An optimal solution to this LP problem is as follows:

$$\begin{aligned}
x_{A,0} &= 1, \\
x_{B,0} &= 1, \\
x_{C,0} &= 6, \quad x_{C,1} = 2, \quad x_{C,2} = 0, \\
x_{D,0} &= 3, \quad x_{D,1} = 1, \\
x_{E,0} &= 0, \quad x_{E,1} = 0, \quad x_{E,2} = 3
\end{aligned}$$

The pre-schedule corresponding to this solution is defined as follows:

$$\begin{aligned}
\mathbf{E} &= [(C,0,9,6),(A,1,9,1),(C,1,24,2),(D,14,24,3), \\
&\quad (B,16,24,1),(D,16,40,1),(E,16,45,3)]
\end{aligned}$$

4.4.2 Analysis

According to Theorem 2, a solution to the extended LP problem exists if and only if a valid pre-schedule exists. We only need to prove Theorem 5 defined as follows.

Theorem 5 *Given a pre-scheduling problem defined on the domain of integers, an optimal solution to the extended LP problem is always on the domain of integers.*

Proof: Assume that $\mathbf{E}$ is a valid non-integral pre-schedule. We shall prove that there exists a better pre-schedule $\mathbf{E}'$, such that $o^{\mathbf{E}} < o^{\mathbf{E}'}$, where $o^{\mathbf{E}}$ and $o^{\mathbf{E}'}$ represent the values of the objective function o corresponding to $\mathbf{E}$ and $\mathbf{E}'$. There are two cases.

Case 1: $\mathbf{E}$ is not regular. (Recall that regularity is defined in Section 4.2.1.)

There exist a pair of jobs J_i and J_j, $i < j$, and $\mathbf{O}(J_i, J_j)$ is not regular. We define $\mathbf{E}'$ as the result of $SWAP(J_i, J_j)$. Let o and o' be the values of the objective function corresponding to $\mathbf{E}$ and $\mathbf{E}'$.

Claim: $o' < o$.

Let i_k be the index in $\mathbf{E}$ for the k^{th} executive of job J_i. According to the definition of regularity and $SWAP$, the following must be true.

- There exists the c^{th} executive of job J_i in $\mathbf{E}$, such that for every executive E_{i_k} of job J_i, if $i_k \leq i_c$, $E_{i_k}.c \leq E'_{i_k}.c$, otherwise, $E_{i_k}.c \geq E'_{i_k}.c$.
- There exists an executive E_{j_c}, such that for every executive E_{j_k} of job J_j, if $j_k \leq j_c$, $E_{j_k}.c \geq E'_{j_k}.c$, otherwise, $E_{j_k}.c \leq E'_{j_k}.c$.
- Let $\Delta = \sum_{0 \leq k \leq c} E'_{i_k}.c - E_{i_k}.c$. $\sum c_{i,k} \cdot (E'_{i_k} - E_{i_k}) \leq -\Delta \cdot d_i$, and $\sum_{c_{j,k} \cdot (E'_{j_k} - E_{j_k})} \leq \Delta \cdot d_j \cdot (m-1)$, where m is the total number of executives of J_j.
- The execution times of executives of jobs other than J_i and J_j do not change.

According to the definition of the objective function in Subsection 4.4.1,

$$\begin{aligned} o' - o &= \sum c_{i,k} \cdot (E'_{i_k} - E_{i_k}) + \sum c_{j,k} \cdot (E'_{j_k} - E_{j_k}) \\ &\leq \Delta \cdot ((m-1) \cdot d_j - d_i) \end{aligned}$$

According to the definition of d in Algorithm 11 and the assumption of $i < j$,

$$(m-1) \cdot d_j < d_i$$

Therefore,

$$o' < o$$

Case 2: $\mathbf{E}$ is regular.

In this case, we can always construct $\mathbf{E}'$ with a less value of objective function. The construction is defined as follows.

First, find a simple integral scope coverage $\mathbf{C}$ of $\mathbf{E}$ as defined in Subsection 4.2.2. Let i be the lowest index in $\mathbf{J}$ such that an executive of J_i has is at the boundary a simple integral scope in $\mathbf{C}$; i.e., there exists $[E_{b_k}..E_{e_k}] \in \mathbf{C}$, such that either E_{b_k} or E_{e_k} is the executive of job J_i with the lowest index in $\mathbf{E}$. Then, one of the following two cases is true.

Case 2.1: E_{b_k} is the executive of job J_i with the lowest index in $\mathbf{E}$.

Then $\mathbf{E}'$ is constructed by round-and-compensate. For job i, in-flows and out-flows of any job strictly alternate, and the last in/out flow must be an out-flow, as proved in Lemma 6, Lemma 7, Lemma 8, therefore,

$$\sum_k E'_{i_k}.c - E_{i_k}.c \leq -\delta \cdot d_i$$

For each job J_j other than job J_i, let m_j be the number of executives of job J_j,

$$\sum_k E'_{j_k}.c - E_{j_k}.c \leq \delta \cdot d_j \cdot m_j$$

By the assumption of i, $d_i > \sum_{j>i} d_j \cdot m_j$. Therefore, $o' < o$.

Case 2.2: E_{e_k} is the executive of job J_i with the lowest index in $\mathbf{E}$.

Then $\mathbf{E}'$ is constructed by a "counter" round-and-compensate defined as follows.

1. Compute δ as follows. For any executive E_x in $\mathbf{E}$, if $E_x.c$ is an integer, $\Delta(E_x) = \infty$. Otherwise, there must exist k where $E_x \in [E_{b_k}, E_{e_k}]$, which is a scope in $\mathbf{C}$. $\Delta(E_x)$ is computed as follows:

$$\Delta(E_x) = \lceil \sum_{E_y \in [E_x, E_{e_i}]} E_y.c \rceil - \sum_{E_y \in [E_x, E_{e_i}]} E_y.c$$

   Let δ be the minimum of $\Delta(E_x)$ for any executive E_x in $\mathbf{E}$.

2. For every scope $[E_{b_k}, E_{e_k}]$ in $\mathbf{C}$, conduct *counter execution time move* $E_{b_k} \rightarrow E_{e_k}(\delta)$, which is defined as $E_{b_k}.c := E_{b_k}.c - \delta$ and $E_{e_k}.c := E_{e_k}.c + \delta$.

First, a counter round-and-compensate produces a valid pre-schedule, and the proof is similar to that of Lemma 9 and Lemma 10. Second, since in-flows and out-flows are reversed in counter round-and-compensate, Therefore the first in/out-flow of job J_i is an in-flow. Third, similar to round-and-compensate, Therefore, similar to Case 2.1, $o' < o$.

The value of an objective function is non-negative, Therefore, there must exists a solution with a minimal value of objective function. By all cases, if a solution is not on the domain of integers, there exists a better solution. Therefore, an optimal solution must be on the domain of integers. ■

4.4.3 Discussion

Indeed, the direct LP approach is equivalent to the explicit round-and-compensate approach. By the definition of the objective function o, the direct LP approach requires the following transformations must be taken: (1) If a solution is not regular, then there exists a swapping transformation to improve the value of the objective function; (2) If a regular solution is not on the domain of integers, then a round-and-compensate can be done to improve the value of the objective function. Therefore, the objective function leads a generic LP solver to an integer solution.

However, by Algorithm 11, the values of the co-efficients in the objective function increase exponentially with the number of jobs in $\mathbf{J}$, and the memory requirement to store the co-efficients grows linear with the number of jobs. This will cause two problems: First, the upper bounds of representation of integers in programming languages and computer architectures; e.g., some architectures require that integers are represented by 32 bits, Although special treatments on huge integers are possible, they are also expensive. For instance, existing LP solvers may not support that. Second, the complexity of relevant arithmetic operations, such as additions and multiplications, grows quadratic with the length of operants. Therefore, the direct LP approach proposed here is not as efficient as the explicit round-and-compensate approach. Actually, since the explicit round-and-compensate approach is efficient, we don't see much incentive to improve the efficiency of the direct LP approach. We'd rather consider that it provides us an insight on the pre-scheduling problem.

4.5 Related Works

LP problems on rational numbers can be solved in polynomial time [13, 15], but Integral Linear Programming (ILP) is NP-Complete in the strong sense [2, 14]. Some approximate approaches to ILP problems are described in [24]. Chapter 3 of [24] is entitled "Using Linear Programming to Solve Integer Programs". Specifically, Section 3.3 of [24] is entitled "Obtaining Integer Programming Solutions by Rounding Linear Programming Solutions". By this naive approach, an integer programming problem is "relaxed" to its corresponding linear programming problem, and the results on the domain of rational numbers are rounded to the integers close to them. By this naive approach, linear constraints may be violated, and the objective function might be sub-optimal. The round-and-compensate approach is significantly different: none of the constraints of a valid pre-schedule will be violated during the procedure. Therefore, the transformer produces a valid pre-schedule on the domain of integers if the pre-scheduling problem is defined on the domain of integers and a valid pre-schedule on the domain of rational numbers is given as input.

4.6 Summary

This chapter focuses on a rational-to-integral transformer of valid pre-schedules, which is polynomial to the size of pre-schedule (number of executives). Combined with the basic LP-based pre-scheduler on the domain of rational numbers in Chapter 3, a generalized, sound, complete, PTIME and *integral* pre-scheduler is devised, which is practical for scheduling preemptive resources with context switch overheads. We also show a direct LP approach, which essentially implements round-and-compensate but devising the objective function of LP problem.

Chapter 5

Resource Supply Analysis

The interface between a pre-scheduled component and the system is defined by an online supply function and an off-line supply contract. The process of generating the supply contract is called "resource supply analysis". Since resource supply to a pre-scheduled component is a result of resource competition of all components within a system, resource supply analysis depends on the understanding of following items: (1) the pre-scheduled component, including its component schedulers and workload; (2) competing components, including their component schedulers and workloads; (3) the coordinator mechanisms. Since the variety of these items, there is no universal process for doing resource supply analysis. In this chapter, we exemplify the resource supply analysis with two cases of typical real-time system settings.

5.1 Case Study One: Scheduling A Combination of Time-Driven and Event-Driven Workloads with CEDF

As we mentioned earlier in the introduction of Chapter 3, a combination of time-driven and event-driven workloads to one resource is common in contemporary real-time systems. In this section, we provide a pre-scheduling solution for such systems, with a focus on how to define the supply contract.

The time-driven workload is still modeled as a set of periodic jobs $\mathbf{J}$ as defined in Section 3.2, and it is allocated in a component to be pre-scheduled.

Event-driven workloads are modeled as a set of sporadic tasks $\mathbf{T^S}$. Recall that sporadic task is defined in Subsection 2.4.1. a sporadic task T is an infinite sequence of jobs, and it is defined by a tuple: (c, p, d), where c is the execution time, p defines the minimal length of the time interval between two consecutive jobs, and d is the maximal relative delay. The actual ready time of any job of a sporadic task is unknown *a priori*. The event-driven workload is therefore modeled as a set of sporadic tasks

We define the hyper period P to be a common multiple of the periods of all sporadic tasks in $\mathbf{T^S}$, because we want the supply contract to be recursive by the hyper period P. (Recall that the recursiveness is defined in Section 3.2). We assume that the coordinating algorithm is CEDF defined in Section 3.5.

We define the computation of supply contract B. Given any time interval (b, e) such that $e - b$ is less than or equal to P, $B(b, e)$ is defined as follows. Let l be $e - b$, which is the length of the time interval. Let function $n(T, l)$ be the maximal number of jobs of sporadic task T that must be completely scheduled within a time interval with length l: If $l - \lfloor \frac{l}{T.p} \rfloor \cdot T.p < T.d$, $n(T, l) = \lfloor \frac{l}{T.p} \rfloor$; otherwise, $n(T, l) = \lfloor \frac{l}{T.p} \rfloor + 1$. The lower bound of the maximal aggregate time that must be scheduled for the sporadic tasks between a time length of l is $\sum_{T \in \mathbf{T^S}} T.c \cdot n(T, l)$. Then $B(b, e)$ is computed as follows.

$$\begin{aligned} O(b,e) &= (e-b) - \sum_{T \in \mathbf{T^S}} T.c \cdot n(T,(b,e)) \\ B(b,e) &= min\{O(b,x) | e \leq x \leq b+P\} \end{aligned}$$

Example 11 *The workload to be pre-scheduled is defined in Example 1.* $\mathbf{T^S}$ *is*

Table 5.1: Supply Contract $B(I)$ on Critical Intervals for Example 11

I.b	I.e	9	24	40	45	54
0		6	17	29	30	
1		5	16	28	30	
14			7	19	20	29
16			5	17	19	27

defined as follows. Compute supply contract B on critical intervals.

$$\mathbf{T^S} = \{(3, 45, 3), (4, 15, 15)\}$$

Supply contract $B(b, e)$ is shown in Table 5.1.

5.2 Case Study Two: Scheduling A Combination of Time-Driven and Event-Driven Workloads with FP

In this case study, we make the same assumptions as in Section 5.1, except that the coordinator approach is FP instead of CEDF. By FP, each component is assigned to a fixed priority. If there is a resource competition, the component with a higher priority wins. We assume that the pre-scheduled component is set at the lowest priority.

The supply contract is obtained by *saturated test* of all sporadic tasks in $\mathbf{T^S}$. In a saturated test, we assume that for every sporadic task T in $\mathbf{T^S}$, the first job of T arrives at time 0, and subsequent jobs of T arrives at the minimal interval, which is defined by $T.p$. The arrived jobs are scheduled by FP. The resource is *idle* at a time t if all arrived jobs have been satisfied at time t. Given any time interval I with length l, $B(I)$ is defined as the aggregate length of idle time between time interval $(0, l)$ during the saturated test.

Example 12 *The workload to be pre-scheduled is defined in Example 1. Competing workload $\mathbf{T^S}$ is defined in Example 12. Compute supply contract B on critical*

Table 5.2: Supply Contract $B(I)$ on Critical Intervals for Example 12

I.b	I.e	9	24	40	45	54
0		2	13	25	30	
1		1	12	24	29	
14			3	15	19	25
16			1	13	18	23

intervals.

The execution of the saturated test is illustrated in Figure 5.1. The un-shadowed time intervals are idle in the saturated test. The supply contract B on critical intervals is defined in Table 5.2.

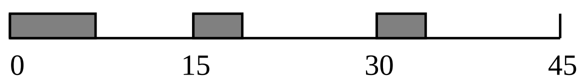

Figure 5.1: Execution of Saturated Test

■

The sporadic task set is the same in Example 11 and 12. However, due to the different coordinating algorithms, the supply constraints imposed to the prescheduled component are different.

Chapter 6

Implementation and Experiments

In Chapter 3, we proved the soundness and completeness of the basic LP-based approach. In Chapter 4, we showed that the pre-scheduling problem can be solved on the domain of integers with practical computational cost. However, there are still a number of interesting questions to be studied by experiments. This chapter reports our implementation and experiments on pre-scheduling. Details of the implementation is described in Section 6.1. Then the objectives and results of experiments are reported in Section 6.2.

6.1 Implementation of The Pre-Scheduler

The algorithm of the pre-scheduler is defined in Chapter 3. We describe the implementation and experiments specifics here.

The workload in pre-scheduled component is modeled as a set of periodic job $\mathbf{J}$ as defined in Section 3.2, and the workload in competing component is modeled as a set of sporadic tasks $\mathbf{T^S}$ as defined in Section 5.1. The pre-scheduler obtains the definitions of $\mathbf{J}$ and $\mathbf{T^S}$ from a text file. The the pre-scheduler establishes the internal data structures, such as the sorted list of jobs and the sorted list of executives, as defined in Section 3.2.

The supply constraints are computed according to the supply analysis algorithm defined in Section 5.1. The number of supply constraints is $\Theta(n^2)$, where n is the number of jobs in $\mathbf{J}$. However, in many cases, the number of non-trivial constraints is much less than n^2. In our implementation, we applied several simple mechanisms to eliminate obviously trivial constraints.

We use *lp_solve_4.0*, which is a general purpose LP solving program, to solve the execution times. *lp_solve_4.0* provides a set of function calls as interface to user programs. The pre-scheduler interacts with *lp_solve_4.0* by the following scenario. First, the LP problem is established by function call *make_lp*; the demand constraints and supply constraints are added into the internal presentation of the LP problem by calling *add_constraint*; Then function *solve* is called, which commands the LP solver to produce a solution; Finally the pre-scheduler retrieves the solution from the LP solver by calling *get_variables*.

6.2 Experiments and Results

6.2.1 Success Rates

The following situation is not rare in previous real-time scheduling research and engineering: Approach A is proved to be optimal and approach B is proved to be sub-optimal; However, in practice, B is *almost* as good as A, and B is actually more popular than A because of its simplicity. A simple way of pre-scheduling is to produce a static pre-schedule based on a pseudo constant supply rate, then test if this pre-schedule works with the real supply contract. This is by and large the common practice before we propose the LP-based pre-scheduler. One of the objectives of our experiments is to find out if there is a significant difference between the success rates of the naive approach and those of LP-based approach.

We compare the success rates of the LP-based pre-scheduler with those of an EDF-based pre-scheduling algorithm which is sound and complete under constant supply rate assumption. EDF can be extended to the following straight-forward pre-scheduler. Schedule the subject workload according to EDF in one hyper interval, assuming that there is no competing component. There will be a sequence of time intervals in the output schedule, and a job is assigned to the resource during each of these time intervals. Then we construct a pre-schedule according to the schedule as follows. For each time interval in the schedule, we create an executive. The corresponding job of an executive is the same as the job scheduled in its corresponding time interval, the ready time and deadline of each executive are the start and the end of its corresponding time interval, and the execution time is the length of the time interval. Then we minimize the ready-times and maximize the deadlines of executives under the following constraints: The sequence of all ready-times and the sequence of all deadlines are both non-decreasing, and the ready-time and deadline of each executive is within the valid scope of its corresponding job. Under the assumption of constant and predictable resource supply rate, this EDF-based algorithm produces a valid pre-schedule if and only if one exists. Therefore we deem it a reasonable pre-scheduler for a fair comparison with the LP-based pre-scheduler.

In our performance measurement, competing components are modeled as a set of sporadic tasks, and the online composition mechanism is CEDF as defined in Section 5.1; i.e., the subject component obtains the resource when the deadline of the current executive is earlier than the earliest deadline of all pending sporadic jobs representing competing components. We measure the success rates of both LP-based and EDF-based pre-schedulers on eight groups of test cases. There are 100 cases for each group. In each test case, the jobs in the subject component and the sporadic tasks representing the competing components are both randomly generated under the following constraints. The aggregate utilization rate of competing workload is set between 10% and 20%. The relative deadline of each sporadic task is between its execution time and its period. The number of jobs in subject workload is set between 50 and 100. The utilization rates in subject component are set to different ranges in the test groups as shown in Table 6.1.

Experiments show that when system utilization rate is not extremely low, the success rate of LP-based pre-scheduler is significantly higher than that of EDF-based pre-scheduler. Take the last group as an example: When the system utilization rate is between 80% and 100% (70% to 80% subject component utilization plus 10% to 20% competing workload utilization), LP-based pre-scheduler can produce valid pre-schedules for 89 cases out of 100 cases, while EDF-based pre-scheduler can produce valid pre-schedules for only 28 cases.

Table 6.1: Success Rate Comparisons: LP-Based vs. EDF-Based Pre-Schedulers

Pre-scheduled Component Utl. (%)	LP-Based Success Rate(%)	EDF-Based Success Rate(%)
0.01-10	100	100
10-20	99	96
20-30	97	77
30-40	98	57
40-50	98	35
50-60	97	33
60-70	97	29
70-80	89	28

6.2.2 Fragmentation and Computation Time

By our assumptions, a job could be pre-scheduled to multiple executives. This is called fragmentation. For systems with context-switch overhead, fragmentation shall be reduced if possible. The non-preemptive scheduling problem, even with constant supply rate assumption, is well-known to be NP-hard [8]. Since the problem of minimizing the number of executives covers the non-preemptive scheduling problem, it is also NP-hard. By our LP-based pre-scheduler, the number of executives in a pre-schedule is $\Theta(n^2)$. We will investigate the average cases of the number of executives by experiments.

The dominant factor of the computational complexity of the LP-based pre-scheduler is that of the LP solver. LP problem is proved to be polynomial [13]. People don't exactly know the tide upper bound of it, and LP solver usually perform much better than the known upper bound for most of the cases. This fact leaves us some interest in investigating the execution time of the LP-based pre-scheduler by experiments. The dominating factor in the number of constraints in the LP problem is the number of supply constraints, which is $O(n^2)$. However, in practice, most of the supply constraints are *trivial*, in the sense that they are satisfied if other constraints are satisfied. We also investigate the average cases for the number of non-trivial supply constraints.

We conduct three groups of experiments, and the number of periodic jobs are controlled as follows. the number of jobs in $\mathbf{J}$ is set between 50-100 in Group 1, 100-200 in Group 2, and 200-400 in Group 3. The same utilization ranges are set in all groups. The aggregate utilization of subject workload is set between 70% to 80%, and the competing workload utilization is set between 10% to 20%. Therefore, the system utilization rate is between 80% and 100%. The experiments are executed on Sun Ultra 5, with 360MHz Ultra PARC-IIi CPU and 128 Megabytes memory.

The experimental results are shown in Table 6.2 to Table 6.3. We run LP-based pre-scheduler on a test case only if it passes a schedulability test; otherwise it is marked as "un-schedulable" in the tables. The "number of executives" refers to the total number of executives in $\mathbf{F}$ as defined by Step 1 (Subsection 3.3.1), and the "number of non-zero executives" refers the number of executives with non-zero execution times in **E**, which is the pre-schedule produced by the LP solver in Step Two (Subsection 3.3.2). If the problem is not pre-schedulable, it is so written under the column of "number of non-zero executives".

In Group 1, Most of the cases are pre-scheduled successfully, and the execution times vary from few seconds to hundreds of seconds.

In Group 2, 71 unique cases are generated. 14 cases out of these 71 cases are not even schedulable, therefore they are not pre-scheduled. For the rest of 57 cases, the aggregate execution times of adding constraints spans from a few seconds to more than 24 hours. For 53 cases out of the 57 cases, constraints can be completely added within 3 hours, and the LP problem can be solved within another couple of hours. For the other 4 exceptional cases, constraints can't be completely loaded within 24 hours. For these cases, we use "$> x$" to indicate the number of added constraints at the time of termination is x; The "execution time for $lp\_solve()$" and "number of non-zero executives" are unknown, therefore marked as "*". During the execution of the exceptional cases, the disk of the computer of the experiments starts constant reading and writing after first few hours, which indicates that the memory of the computer is not big enough to hold the internal presentation of the constraints. The swapping between disk and memory slows down the computation drastically.

The cases in Group 3 are either trivial, which can be pre-scheduled within seconds, or the constraints can't be completely added within 24 hours.

The experiments shows the following results: (1)In all cases in our experiments, the numbers of executives is lower than $5 \cdot n$, where n is the number of periodic jobs, . This is much lower than the theoretical bound of $\Theta(n^2)$. (2) The numbers of constraints added to the LP solver vary drastically from case to case between the order of n to the order of n^2. (3) The execution times of LP solver grow about linearly to the number of executives and about quadratically to the number of constraints.

Table 6.2: Fragmentation and Execution Time – Group 1

case#	number of periodic jobs	number of executives	number of supply constraints	execution time for add_constraints()	execution time for lp_solve()	number of non-zero executives
50-10000	66	110	86	1s	0s	66
50-10001	63	147	1146	2s	8s	91
50-10002	78	294	5572	241s	322s	109
50-10003	66	129	4154	62s	68s	126
50-10004	56	196	3066	55s	72s	103
50-10005	65	165	1912	7s	23s	101
50-10006	95	250	2019	11s	28s	106
50-10007	90	329	6739	372s	441s	129
50-10008	81	194	4164	84s	105s	113
50-10009	74	390	5270	289s	350s	123
50-10010	68	109	68	0s	0s	68
50-10011	72	260	4353	110s	167s	112
50-10012	93	189	290	1s	0	102
50-10013	un-schedulable					
50-10014	74	174	919	1s	4s	85
50-10015	53	104	2698	20s	27s	95
50-10016	91	189	5863	159s	137s	109
50-10017	96	462	9004	1024s	1130s	171
50-10018	81	210	6385	228s	246s	143
50-10019	53	161	1868	14s	28s	82
50-10020	57	164	2990	39s	53s	97
50-10021	51	147	2523	25s	35s	93
50-10022	80	260	5999	243s	255s	126
50-10023	70	126	1950	5s	17s	112
50-10024	86	192	3033	23s	59s	133
50-10025	50	97	2243	12s	19s	88
50-10026	71	193	3686	63s	71s	95
50-10027	99	315	5039	119s	158s	135
50-10028	80	156	4224	77s	75s	105
50-10029	50	86	175	0s	1s	55
50-10030	71	134	236	0s	1s	90
50-10031	59	245	2996	55s	80s	89
50-10032	89	231	6597	247s	216s	140
50-10033	89	231	6568	253s	287s	141

Table 6.3: Fragmentation and Execution Time – Group 1 (Continued)

case#	number of periodic jobs	number of executives	number of supply constraints	execution time for add_constraints()	execution time for lp_solve()	number of non-zero executives
50-10034	70	130	4255	63s	68s	116
50-10035	78	201	2546	27s	41s	94
50-10036	52	52	52	0s	0s	52
50-10037	56	110	2745	20s	27s	93
50-10038	98	175	4597	65s	113s	167
50-10039	91	91	91	0s	1s	91
50-10040	93	273	8415	518s	388s	166
50-10041	92	182	6026	170s	122s	120
50-10042	un-schedulable					
50-10043	82	218	5813	187s	206s	131
50-10044	88	260	2787	23s	63s	131
50-10045	92	182	7120	242s	192s	not pre-schedulable
50-10046	85	325	6182	314s	361s	139
50-10047	un-schedulable					
50-10048	99	195	9210	435s	297s	172
50-10049	68	260	4384	130s	139s	113
50-10050	90	215	3171	30s	65s	130
50-10051	un-schedulable					
50-10052	54	104	2557	17s	23s	89
50-10053	50	98	2352	14s	19s	86
50-10054	73	159	2018	10s	26s	106
50-10055	90	220	7251	309s	316s	149
50-10056	un-schedulable					
50-10057	un-schedulable					
50-10058	87	231	3355	36s	68s	131
50-10059	79	280	6114	281s	338s	138
50-10060	un-schedulable					
50-10061	81	224	6279	241s	273s	152
50-10062	91	130	116	0s	0s	91
50-10063	un-schedulable					
50-10064	57	164	3007	38s	61s	99
50-10065	77	77	77	0s	0s	77
50-10066	83	192	896	0s	5s	116
50-10067	91	231	2227	13s	34s	133

Table 6.4: Fragmentation and Execution Time – Group 1 (Continued)

case#	number of periodic jobs	number of executives	number of supply constraints	execution time for add_ constraints()	execution time for lp_solve()	number of non-zero executives
50-10067	91	231	2227	13s	34s	133
50-10068	99	220	4753	120s	134s	155
50-10069	83	231	6118	211s	339s	136
50-10070	70	196	4578	108s	135s	114
50-10071	82	252	2182	10s	34s	112
50-10072	89	231	7662	364s	399s	160
50-10073	82	234	6422	253s	232s	138
50-10074	67	154	2831	30s	46s	98
50-10075	78	154	5695	130s	109s	131
50-10076	78	198	2888	51s	54s	93
50-10077	66	299	3996	122s	236s	100
50-10078	76	150	5273	106s	91s	123
50-10079	un-schedulable					
50-10080	70	130	1821	7s	15s	109
50-10081	67	164	1538	3s	15s	100
50-10082	92	259	7538	381s	374s	131
50-10083	98	308	2978	33s	66s	123
50-10084	79	156	5402	114s	97s	122
50-10085	88	195	1842	5s	22s	124
50-10086	56	156	2568	25s	36s	80
50-10087	89	198	7434	284s	215s	136
50-10088	85	385	6982	503s	608s	147
50-10089	70	195	1775	10s	26s	96
50-10090	67	195	4205	90s	97s	116
50-10091	61	146	3403	45s	60s	102
50-10092	77	165	1157	1s	9s	110
50-10093	80	232	5222	208s	140s	not pre-schedulable
50-10094	53	103	2085	11s	13s	63
50-10095	79	189	3579	49s	76s	123
50-10096	62	98	262	0s	0s	80
50-10097	78	130	349	0s	0s	92
50-10098	93	180	6979	225s	171s	146
50-10099	83	190	2712	25s	42s	123

Table 6.5: Fragmentation and Execution Time – Group 2

case#	number of periodic jobs	number of executives	number of supply constraints	execution time for add_constraints()	execution time for lp_solve()	number of non-zero executives
10000	155	363	21326	4320s	2091s	248
10001	103	198	3627	101s	92s	109
10002	119	266	1530	2s	20s	not pre-schedulable
10003	104	169	1792	4s	16s	not pre-schedulable
10004	167	495	>24966	> 3 hours	*	*
10005	111	315	11046	1020s	715s	180
10006	140	140	0	0s	1	140
10007	145	429	18118	4027s	1873s	not pre-schedulable
10008	un-schedulable					
10009	144	312	845	4s	22s	177
10010	169	169	0	1s	0s	169
10011	196	676	>20137	>24 hours	*	*
10012	144	286	19384	2883s	1292s	219
10013	127	436	14701	2627s	1825s	229
10014	un-schedulable					
10015	148	384	15045	2542s	1520s	not pre-schedulable
10016	145	429	18347	3943s	2026s	259
10017	un-schedulable					
10019	115	440	11823	1697s	1479s	200
10020	un-schedulable					
10023	168	420	12310	1716s	1176s	225
10024	198	458	22570	6073s	3373s	252
10025	127	306	965	3s	16s	133
10026	un-schedulable					
10030	un-schedulable					
10034	un-schedulable					
10039	166	461	23163	5928s	4104s	267
10040	119	297	2145	3s	24s	162
10041	un-schedulable					
10046	104	182	747	1s	3s	156

Table 6.6: Fragmentation and Execution Time – Group 2 (Continued)

case#	number of periodic jobs	number of executives	number of supply constraints	execution time for add_ constraints()	execution time for lp_solve()	number of non-zero executives
10047	169	472	18260	4485s	1952s	210
10048	132	242	1866	7s	29s	187
10049	161	440	24586	7481s	5424s	286
10050	176	231	22	1s	1s	187
10051	135	260	12669	1140s	722s	203
10052	141	658	17346	5145s	4123s	212
10053	102	300	10033	832s	544s	185
10054	un-schedulable					
10057	136	340	15474	2155s	1088s	196
10058	114	548	10623	1531s	1598s	187
10059	un-schedulable					
10064	106	210	10662	645s	397s	185
10065	un-schedulable					
10069	Un-schedulable					
10073	162	364	15705	2639s	1502s	211
10074	166	330	25368	5859s	3120s	297
10075	144	286	18426	2674s	1374s	not pre-schedulable
10076	170	320	2422	21s	56s	190
10077	144	286	19756	3004s	1405s	251
10078	un-schedulable					
10080	198	830	>16091	>24 hours	*	*
10081	166	429	23804	6164s	5050s	270
10082	121	220	4479	46s	109s	not pre-schedulable
10083	133	257	9564	610s	354s	164
10084	160	776	>17683	> 24 hours	*	*
10085	192	379	17139	3614s	1408s	216
10086	124	483	12403	1687s	1213s	186
10087	118	273	58	1s	0s	120
10088	121	351	12735	1413s	886s	205
10089	178	420	15295	2448s	1592s	264
10090	166	450	23590	6870s	5649s	292

Table 6.7: Fragmentation and Execution Time – Group 2 (Continued)

case#	number of periodic jobs	number of executives	number of supply constraints	execution time for add_constraints()	execution time for lp_solve()	number of non-zero executives
10091	un-schedulable					
10092	134	484	14195	2193s	1615s	207
10093	102	300	7684	461s	506s	149
10094	145	429	15909	2970s	2018s	228
10095	125	230	11765	897s	598s	not pre-schedulable
10096	176	558	23005	8482s	19653s	219
10097	181	506	21712	5613s	3746s	296
10098	108	254	9622	640s	489s	160
10099	144	473	12279	1744s	1298s	208

Table 6.8: Fragmentation and Execution Time – Group 3

case#	number of periodic jobs	number of executives	number of supply constraints	execution time for add_constraints()	execution time for lp_solve()	number of non-zero executives
20000	286	286	286	2	1	286
20001	291	572	$>$ 23967	$>$ 24 hours	*	*
20002	371	1362	$>$ 12623	$>$ 24 hours	*	*
20003	341	990	$>$ 16717	$>$ 24 hours	*	*
20004	396	726	5603	10s	115s	561
20005	288	779	$>$ 21984	$>$ 24 hours	*	*
20006	un-schedulable					
20007	255	390	270	1s	1s	255
20008	un-schedulable					
20009	200	330	3498	3s	51s	300
20010	333	881	$>$ 18030	$>$ 24 hours	*	*

Chapter 7

More Types of Constraints in Real-Time Systems

In Section 3.2, we defined that a valid pre-schedule shall satisfy a set of constraints, namely non-negative constraints, valid scope constraints, demand constraints, and supply constraints. Later in Chapter 4, the integral constraints are added into the definition. In fact, there are other types of constraints that might be required for real-time systems, and a variety of pre-scheduling problems can be defined based on which subset of those constraints is covered. In this chapter, we discuss several more types of constraints. Section 7.1 addresses precedence constraints, which can be solved in polynomial time in pre-scheduling problem. Section 7.2 addresses mutual exclusive constraints, distance constraints and locality constraints, which are all NP-hard.

7.1 Precedence Constraints

A *precedence constraint* between a pair of jobs is represented as $J_x \rightarrow J_y$, which reads "J_x precedes J_y". It defines that the instance of job J_x shall be scheduled before the instance of job J_y in every hyper interval. Precedence constraints are common in real-time systems. The set of all precedence constraints is represented as $\mathbf{P}$. A *precedence graph* can be constructed according to $\mathbf{P}$ as follows. We consider every job J_x in $\mathbf{J}$ as a vertex, and every precedence constraint $J_x \rightarrow J_y$ as a directed link from vertex J_x to vertex J_y. If there exists a circle in this graph, then the precedence constraints are not satisfiable. Otherwise, the precedence graph is a set of Directed Acyclic Graphs (DAGs).

Example 13 $\mathbf{J}$ *is defined in Example 1. A set of precedence constraints* $\mathbf{P}$ *is defined as follows.* $\mathbf{P}$ *is also illustrated in Figure 7.1.*

$$\mathbf{P} = [A \rightarrow E, C \rightarrow E, C \rightarrow D]$$

■

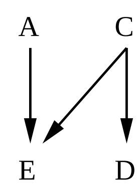

Figure 7.1: A DAG of Precedence Constraints **P**

We present how to solve precedence constraints in pre-scheduling. The basic LP-based pre-scheduler defined in Section 3.3 is still used. However, we add two extra steps, Step 0, and Step 3, before and after the execution of Step 1 and 2 in the basic LP-based pre-scheduler.

Step 0 transforms $\mathbf{J}$ according to the precedence constraints. First, the valid scopes of jobs in $\mathbf{J}$ is maximized under the following constraints: (1) The valid scope of any job J' is within the valid scope of J: $J.r \leq J'.r$ and $J'.d \leq J.d$; (2) For every precedence $J_x \rightarrow J_y$ in $\mathbf{P}$, J'_x is before or parallel to J'_y. This could be implemented by changing the ready time of jobs while traversing the precedence DAGs top-down, and changing the deadlines of jobs while traversing the DAGs bottom-up. Second, **J** is sorted such that the following condition is true: If J_x is before or contained by J_y, or J_x is parallel to J_y and $J_x \to J_y$, $x < y$. The sorting algorithm is obvious.

Taking the transformed **J** as input, Step 1 and 2 of the basic pre-scheduler, as defined in Section 3.3, are executed. After these two steps, we execute one more step, Step 3, to enforce the precedence constraints.

Step 3 is to conduct Algorithm 10 defined in Subsection 4.2.1.

Example 14 **J** *is defined in Example 1, supply function is defined by Table 5.1, and the set of precedence constraints* **P** *is defined in Example 13. Produce a valid pre-schedule that satisfies the precedence constraints.*

Step 0 transforms **J** to the following. Notice that the ready time of job E is changed.

$$\mathbf{J} = [A : (1, 9, 1), B : (16, 24, 1), C : (0, 40, 8), D : (14, 40, 4), E : (\underline{1}, 45, 3)]$$

J is illustrated in Figure 7.2. Assume that pre-schedule **E** produced by Step 1 and 2

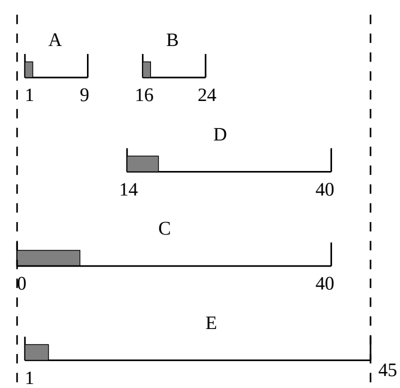

Figure 7.2: **J** After Step 0

is as follows:

$$\mathbf{E} \;=\; [(A, 1, 9, 1), (C, 1, 24, 1), (E, 1, 24, 1), (D, 14, 24, 2), (B, 16, 24, 1),$$

$(C, 16, 40, 7), (D, 16, 40, 2), (E, 16, 45, 2)]$

Step 3 transforms $\mathbf{E}$ to the following:

$$\mathbf{E} = [(A, 1, 9, 1), (C, 1, 24, 4), (B, 16, 24, 1), (C, 16, 40, 4), (D, 16, 40, 4), (E, 16, 45, 3)]$$

■

We show the correctness of the precedence solving steps. Let $J_x \to J_y$ be a precedence constraint in $\mathbf{P}$. After Step 0, $J_{x'}$ is either before $J_{y'}$ or parallel to $J_{y'}$, and $x' < y'$. After Step 1 and 2, For each executive E_u of $J_{x'}$, one of the following cases must be true: (1) E_u is before all executives of $J_{y'}$; (2) or E_u and an executive E_v of $J_{y'}$ form an overlapping pair, and $u < v$. Then after Step 3, all non-zero executives of $J_{x'}$ are before all non-zero executives of $J_{y'}$ in $\mathbf{E}'$. Therefore, precedence constraints are satisfied.

7.2 NP-hard Constraints

There are several other common types of constraints in real-time systems — mutual exclusions, distance constraints, and locality constraints. We briefly discuss them.

A pair of jobs J_x and J_y are *mutually exclusive* if the following contraint is required: in each hyper interval, either the instance of job J_x is completely scheduled before the instance of job J_y, or vise versa. *Non-preemption* of a job is a special case of mutual exclusion, where the job is mutually exclusive with every other job.

A *distance* constraint can be defined between the start time or end time of time intervals scheduled to a pair of jobs. For instance, a distance constraint may define that job J_x shall not be started until 5 time units after the completion of job J_y.

In this dissertation, we have assumed that there is one resource to be scheduled. Now we consider the case of multiple homogeneous resources (For instance, multiple CPUs). If an instance of a job must be scheduled to one resource, or there is a cost of migration between resources, then pre-scheduling problem is NP-hard in general, even with the constant supply rate assumption.

Static schedule generation with mutual exclusions, distance constraints or locality constraints is NP-hard even with the assumption of constant supply rate. A number of NP-hard schedule problems with these constraints are listed in the appendixes of [8]. However, effective searching algorithms have been invented to solve large and practical problems with both mutual exclusions and distance constraints with the assumption of constant resource supply rate [27].

Chapter 8

Conclusion

Once again, we turn to the grand picture of scheduler composition. Let's assume there is a complex real-time system to be designed. Assume that the resource assignment problem is complex enough such that the designer decides to apply some coordinator/component scheduler composition scheme. There are two layers of considerations: the layer of coordinating mechanisms and the layer of component construction. There are a number of approaches that have been researched and published on both layers, some fancier than the rest, but the designer will probably start with some *simple* approaches. First, we consider the layer of coordinating mechanisms. The designer may try a round robbin or a fixed temporal partition first. If these simple solutions do not provide sufficient flexibility, then try a fixed priority scheme; If fixed priority scheme is still not good enough in utilization, then CCC might be considered. Second, we consider the layer of component construction. Consider a component of time-driven workload. If the assumption of resource supply at a constant rate serves well, then off-line EDF can be applied for pre-schedule generation; otherwise, consider LP-based pre-schedule generation. If pre-schedule can't be generated because of supply constraints, then more dynamic schedulers, such as EDF, might be applied as online scheduler. Therefore, on each of the two layers, there are a spectrum of design choices, for simple to complex, in the following aspects. (1)The logic complexity: how difficult it is to describe, comprehend, and implement. (2) The computational complexity, especially, the online part. (3) The amount of information required. For instance, pre-scheduling required a supply contract instead of a constant supply rate, therefore pre-scheduling is more complex then static scheduling from the perspective of information hiding. Generally, on one hand, the more specific information the correctness is based on, the more vulnerable the design is for change; on the other hand, more complex design may provide extra power.

The mission of real-time scheduling research is to provide solutions over the spectrum from simpler to more powerful. This dissertation reviewed the major contributions of my research on two layers: in the layer of coordinating mechanism, we defined Class-based Component Composition (CCC); in the layer of component construction, we defined a variety of LP-based pre-scheduling algorithms. CCC is a generalization of fixed priority scheduling, and LP-based pre-scheduling is a generalization of the static scheduling. Comparing with their counter-parts, both CCC and LP-based pre-scheduler provide finer grain control over resource and require more information.

Now we consider the techniques we applied in our research. LP techniques are relatively less frequently used in previous researches in real-time scheduling community. LP is effective in dealing with a number of constraints at design time. However, some other types of constraints, such as mutual exclusions, distance constraints, and processor locality constraints in multi-processor systems, are non-linear. For scheduling problems with these constraints, search techniques are norm. LP-based techniques and search-based techniques might be combined to effectively schedule systems with both linear and non-linear constraints. The following ideas might be exploited in the future. First, We can design the objective function to guide LP solver toward a solution that *might* also satisfy some non-linear constraints, which is similar to the direct LP approach described in Section 4.4. Second, we may use the result of a LP solver to improve the search efficiency. Consider there are a number of non-linear constraints. Each non-linear constraint can be translated to a set of possible scheduling choices to make. A choice can often be presented as a set of linear constraints. For instance, consider job A and B are mutually exclusive in a pre-scheduling problem. once we choose A to be scheduled before B, then the execution times of the executives of A after the last executive of B are set to zero. In searching algorithms, each constraint might be considered as a layer in a search tree. When a branch in the tree is proved to be infeasible, the searching algorithm draws back to certain layer and looks for other choices. At a node in a search tree, we may compute if there is still a feasible solution for all linear constraints and the all choices that have made so far over non-linear constraints. Third, LP solver algorithms and searching algorithms might even be coupled internally. For instance, consider simplex method in solving the LP algorithm. A solution to the LP problem is a value assignment to the set of variables. The procedure of simplex method is a sequence of iterations, and the value assignment is changed in each iteration to improve over the objective function. We may set extra constraints to the change of value assignment according to those non-linear constraints.

In summary, the research in scheduler composition can be continued and extended in the following two directions. Horizontally, we may provide more design choices covering more problems with practical interests. Vertically, we may invent better algorithms based on deeper understandings.

Bibliography

[1] T. P. Baker, A. Shaw. The cyclic executive model and Ada. Real-Time Systems Symposium, pp.120-129, 1988.

[2] I. Borosh, L. B. Treybig. Bounds on Positive Integral Solutions of Linear Diophantine Equations. Proc. Amer. Math. Soc. 55, 299-304, 1976.

[3] R. Cayssials, J. Orozco, J. Santos and R. Santos. Rate Monotonic Schedule of Real-Time Control Systems with the Minimum Number of Priority Levels, Euromicro Conference on Real Time Systems, pp. 54-59, 1999.

[4] Z. Deng and J. Liu. Scheduling Real-Time Applications in an Open Environment. Real-Time Systems Symposium, pp. 308-319, 1997.

[5] J. Erschler, F. Fontan, C. Merce, F. Roubellat. A New Dominance Concept in Scheduling n Jobs on a Single Machine with Ready Times and Due Dates. Operations Research, 31:114-127.

[6] G. Fohler. PhD Thesis. Technisch-Naturwissenschaftliche Fakultaet, Technische Universitaet Wien, Austria, April 1994.

[7] G. Fohler. Joint Scheduling of Distributed Complex Periodic and Hard Aperiodic Tasks in Statically Scheduled Systems. Real-Time Systems Symposium, pp. 152-161, 1995.

[8] M.Garey and D. Johnson. Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman and Company, 1979.

[9] R. Gerber, W. Pugh, and M. Saksena. Parametric Dispatching of Hard Real-Time Tasks. IEEE Trans. on Computers, Vol.44, No.3, pp. 471-479, Mar 1995.

[10] IEEE. Portable Operating System Interface(POSIX)—Part 1: Application Program Interface(API) [C Language] —Amendment: Realtime Extensions, IEEE 1-55937-375-X.

[11] International Organization for Standardization. ISO/PRF 11898-4. Road vehicles – Controller area network (CAN) – Part 4: Time-triggered communication.

[12] D. Isovic, G. Fohler. Handling Sporadic Tasks in Off-line Scheduled Distributed Real-Time Systems. EUROMICRO Conference on Real-Time Systems, pp. 60-67, 1999.

[13] N. Karmarkar. A New Polynomial-Time Algorithm for Linear Programming. Combinatorica 4(1984), 373-395.

[14] R. M. Karp. Reducibility among Combinatorial Problems. Complexity of Computer Computations, Plenum Press, New York, 85-103, 1976.

[15] L. G. Khachian. A Polynomial Algorithm in Linear Programming. Dokl. Akad. Nauk. SSSR 244 (1979), 1093-1096(in Russian). English translation in Soviet Math. Dokl. 20(1979), 191-194.

[16] H. Kopetz. Real-Time Systems: Design Principles for Distributed Embedded Applications. Kluwer Academic Publishers, ISBN 0-7923-9894-7, 1997.

[17] G. Lipari, J. Carpenter, S. Baruah. A Framework for Achieving Inter-Application Isolation in Multiprogrammed, Hard Real-Time Environment. Real-Time Systems Symposium, pp. 217-226, 2000.

[18] C. L. Liu and J. W. Layland. Scheduling Algorithms for Multi-programming in Hard Real-time Environment. Journal of ACM 20(1), 1973.

[19] A. K. Mok. Fundamental Design Problems of Distributed Systems for the Hard-Real-Time Environment. Ph.D. thesis. MIT. 1983.

[20] A. K. Mok, X. Feng. Towards Compositionality in Real-Time Resource Partitioning Based on Regularity Bounds. Real-Time Systems Symposium, pp. 129-138, 2001.

[21] D.-T. Peng, K. G. Shin, T. F. Abdelzaher. Assignment and Scheduling Communicating Periodic Tasks in Distributed Real-Time Systems. IEEE Transactions on Software Engineering, Volume 23 , Issue 12, pp. 745 - 758 , December 1997.

[22] K. Ramamritham. Allocation and Scheduling of Precedence-Related Periodic Tasks. IEEE Transactions on Parallel and Distributed Systems, Vol 6, No 4, pp. 412-420, April 1995.

[23] J. Regehr, J. A. Stankovic. HLS: A Framework for Composing Soft Real-Time Schedulers. Real-Time Systems Symposium, pp. 3-14, 2001.

[24] H. M. Salkin, K. Mathur. Foundations of Integer Programming. Elsevier Science Publishing Co., Inc. ISBN 0-444-01231-1.

[25] I. Shin, I. Lee. Periodic Resource Model for Compositional Real-Time Guarantees. Real-Time Systems Symposium, pp. 2-13, 2003.

[26] M. Spuri, G. Buttazzo, Scheduling Aperiodic Tasks in Dynamic Priority Systems. Real-Time Systems Journal, Vol,10, pp. 179-210, 1996.

[27] D.-C. Tsou. Execution Environment for Real-Time Rule-Based Decision Systems. PhD thesis, Department of Computer Sciences, The University of Texas at Austin, 1997.

[28] W. Wang, A. K. Mok. On the Composition of Real-Time Schedulers. Real-Time and Embedded Computing Systems and Applications, LNCS 2968, pp. 18-37, 2003.

[29] W. Wang, A. K. Mok, G. Fohler. Pre-Scheduling: Integrating Off-line and On-line Scheduling Techniques. The Conference on Embedded Software, LNCS 2855, pp. 356-372, 2003.

[30] W. Wang, A. K. Mok, G. Fohler. Pre-Scheduling: Integrating Off-line and On-line Scheduling Techniques. UTCS Technical Report, RTS-PS-TR-03-01, 2003.

[31] A Class-Based Approach to the Composition of Real-Time Software Components, Weirong Wang and Aloysius K. Mok. Technical Report: RTS-CC-TR-03-01, 2003.

[32] W. Wang, A. K. Mok, G. Fohler. Generalized Pre-Scheduler. Euromicro Conference on Real-Time Systems (ECRTS), 2004.

[33] W. Wang, A. K. Mok, G. Fohler. Pre-Scheduling on The Domain of Integers. Real-Time Systems Symposium, 2004.

[34] W. Wang, A. K. Mok, G. Fohler. Generalized Pre-Scheduler. UTCS Technical Report RTS-PS-TR-04-01, 2004.

[35] W. Wang, A. K. Mok, G. Fohler. Pre-Scheduling. UTCS Technical Report RTS-PS-TR-04-02, 2004.

[36] W. Wang, A. K. Mok, G. Fohler. Pre-Scheduling on The Domain of Integers. UTCS Technical Report RTS-PS-TR-04-03, 2004.

[37] X. Yuan, M. C. Saksena, A. K. Agrawala. A Decomposition Approach to Non-Preemptive Real-Time Scheduling. Real-Time Systems, Vol. 6, No. 1, pp. 7-35, 1994.

Index

Vita

Weirong Wang graduated with a B.E. degree in Computer Engineering in 1992, from Beijing University of Technology, which was also translated as "Beijing Polytechnic University". He then worked for SIEMENS for 15 months as a junior programmer. He then worked for Motorola as a software engineer and project lead for three years. He studied in the Department of Computer Engineering in Arizona State University as a graduate student in Spring 1997. In the Fall of 1997, he transferred to the Department of Computer Sciences, University of Texas at Austin, where he obtained the degree of Master of Art in Computer Sciences in 1998, under the advising of Professor Aloysius K. Mok. Thereafter he has been working on his Ph.D degree under the advising of Professor Mok.

Permanent Address: None

This dissertation was typeset with LaTeX 2_ε[1] by the author.

[1] LaTeX 2_ε is an extension of LaTeX. LaTeX is a collection of macros for TeX. TeX is a trademark of the American Mathematical Society. The macros used in formatting this dissertation were written by Dinesh Das, Department of Computer Sciences, The University of Texas at Austin, and extended by Bert Kay and James A. Bednar.