Response-time Analysis and Overload Management in Real-time Systems

by

Sriram Murali

B. E., Electronics and Communications Engineering, Anna University, 2008

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Applied Science

in

THE FACULTY OF GRADUATE STUDIES
(Electrical and Computer Engineering)

The University Of British Columbia
(Vancouver)

February 2012

Abstract

We provide two approaches to handling overload in shared resources offering real-time guarantees. We first provide a technique (based on mathematical optimization) for identifying the possible causes for an overload situation by computing the worst-case demand of the system, depending upon the amount of requests serviced. Worst-case analysis of response time has a pseudo-polynomial time complexity, and when there is no knowledge about the workload, the complexity further increases. We provide polynomial-time heuristics to reduce the computation time of the algorithm. Further, we evaluate it against other techniques using stochastic analysis to stress on the accuracy and ease of estimation of the result. The scheduling policy based on the approach is useful to detect an overload in the resource and to allow us to make responsible decisions on it. Secondly, we present a scheduling policy (obtained through stochastic approximation) to handle overload in real-time systems. Competitive analysis of online algorithms has commonly been applied to understand the behavior of real-time systems during overload conditions. While competitive analysis provides insight into the behavior of certain algorithms, it is hard to make inferences about the performance of those algorithms in practice. Similar on-line scheduling approaches tend to function differently in practice due to factors. Further, most work on handling overload in real-time systems does not consider using information regarding the distribution of arrival rates of jobs and execution times to make scheduling decisions. With some information about the workload, we aim to improve the revenue earned by the service provider, in a scenario when each successful job completion results in revenue accrual. We prove that the policy we outline does lead to increased revenue when compared to a class of scheduling policies that make static resource allocations to different ser-

vice classes. We also use empirical evidence to underscore the fact that this policy performs better than a variety of other scheduling policies. The ideas presented can be applied to several soft real-time systems, specifically systems with multiple service classes.

Preface

Chapter 4 consists of work continued by Sriram Murali, based on the initial research conducted by Dr. Sathish Gopalakrishnan[1].

[1] Supervisor, Assistant Professor, Department of Electrical and Computer Engineering, University of British Columbia

Table of Contents

List of Tables

List of Figures

Glossary

FOGS	The Faculty of Graduate Studies
UGF	University Graduate Fellowship
EDF	Earliest Deadline First
EPU	Effective Processor Utilization
FAP	Fractional Allocation Policy
FPS	Fixed-Priority Scheduling
FPPS	Fixed-Priority Preemptive Scheduling
FPTAS	Fully Polynomial-Time Approximation Scheme
MPEG	Motion Picture Experts Group
GTM	General Task Model
IRIS	Increased Reward with Increased Service
LP	Linear Program
MF	Multiframe Task Model
PTAS	Polynomial-Time Approximation Scheme
QOS	Quality-of-Service
QRAM	QoS-based Resource Allocation Model

RMA	Rate Monotonic Analysis
RMS	Rate Monotonic Scheduling
RTA	Response-time Analysis
RTQT	Real-time Queuing Theory
SDP	Stochastic Dynamic Programming
SLA	Service Level Agreement
WCRT	Worst-case Response-time

Acknowledgments

I extend my gratitude to faculty and students at UBC and my friends, who inspired me to write the thesis. I am sincerely thankful to my advisor, and mentor Dr. Sathish Gopalakrishnan, without whom I would not be the person I am today. His guidance helped me address several hard research problems, as well as gain a good perspective of the growing job industry.

My sincere thanks to all my course supervisors from Electrical and Computer Engineering and Computer Science Departments, for their support throughout the course of my degree. I am also thankful for the past and present members of Radical lab, whose company made me feel at home, and willing to learn. Debojit, Theepan, Maliha, Karim and Bader were always around, whether for a relaxing conversation, or a heated discussion after the Paper Reading seminar. I am grateful for having Madhu, Mrigank, Karthik, Jagath and Shankar beside me during the times of need. Finally, my special regards to my parents, grand-parents, Bharath (brother), and cousins, who helped me at different stages of my life. I am grateful to The Faculty of Graduate Studies (FOGS), University of British Columbia (UBC) for granting the University Graduate Fellowship (UGF)[2] for pursuing my degree.

I dedicate this thesis to my guru Prof. N Venkateswaran (Waran), who guided me to pursue this career.

[2]UBC Annual fellowship awards for meritorious students pursuing graduate degree

Chapter 1

Introduction

Large scale Internet-based operators provide a variety of services today. These services range from simple HTML content retrieval to sophisticated infrastructure services. Amazon.com, for example, offers a storage service (S3) for developing flexible data storage capabilities, a database with support for real-time queries over structured data (SimpleDB), and a computation cloud for web-scale computing (Elastic Cloud) [2]. Such services are offered at a basic support level, and at premium support levels with more stringent Service Level Agreement (SLA). These SLAs specify the availability, reliability, and response times that customers can expect for the services provided. Further, several services are offered on a pay-for-use model rather than on the basis of long-term contracts.

Whereas most service providers size their systems to meet the normal demand and some spikes in workload, studies on Internet service workload have noted that peak-to-average ratio of workload varies from 1.6:1 to 6:1 [12]. This large variation makes it exceedingly difficult for service providers to size their systems to handle all possible workload scenarios. Systems should, therefore, be designed to gracefully degrade under overload conditions. Web services are illustrative of systems that need to handle heavy workload and respond to requests within bounded durations to adhere to SLAs with clients.

These services are like any other application, characterized by a group of requests competing for a shared or distributed resource. However, the important difference is that they have predefined priority levels, and have deadlines to be met

based on the offered SLA. *Soft real-time systems* are a class of such systems in which the deadlines can be missed, but results in the loss of revenue to the service providers. The revenue earned directly depends upon the requests serviced within the expected response times. Since many service providers have abundant resources, extra resources can be provisioned for the clients who require more. However, during times of severe overload, or when there is a massive outage of system resources, alternative approaches such as dropping the jobs that are requested by clients with lower Quality of Service (QoS) guarantees.

1.1 Overload Management in Real-time Systems

In the case of a large amount of requests, the system administrator might provision additional resources to serve the clients, or use a scheduling policy to drop certain requests and (preferentially) provide service to requests from clients that offer better revenue (have chosen a higher QoS. Finding the exact characteristics of the scheduling algorithm gives a few metrics to compare, but are not ideal for handling hugely varying workloads [12]. Approaches towards assuming the behavior of the workload, and making scheduling decisions based on it have proven to be efficient, but are based on stochastic modeling. Finding the best possible solution among the two is hard, but the knowledge of both is ideal for making meaningful decisions.

It is important because, even though the SLA may indicate peak workload, the average workload might be much lower than the peak workload. In order to conserve resources, often the total available processing power of the system would be significantly lower than the cumulative requirement of all the tasks serviced over a period of time. Since most of the requests are short lived, the resources gets freed-up very quickly. Service providers also multiplex service among many clients, and this increases the need for managing situations when the requests from clients overload the system; the duration of the overload maybe a few minutes to a few hours and a good scheduling policy will lead to optimal (or near-optimal) revenue for the service provider per unit time.

By using on-line schedulability tests, these policies can be made efficiently. Schedulability analysis is essential to validate if the requests can meet their deadlines. The system resource has a peak utilization of 100%, that is shared among a

stream of requests. These requests comprise of a stream of jobs, having an arrival time that is periodic, or arbitrary, a deadline relative to its arrival time D_i, an execution time requirement e_i, and a fixed reward for successful completion. These parameters would be part of the SLA between the client and the service provider.

1.2 Overview of our Work

We now discuss two important contributions of this work, in which we address the Overload management problem in real-time systems. First, in a system agnostic of the workload, we show that the possibility of an overload can be identified by understanding the worst-case performance of the system. Consequently, system resources can be increased to accommodate the overload. In the case of limited system resources, we try to we make use of the knowledge about the workload, and develop scheduling policies for maximizing the amount of requests serviced. We also analyze the optimality of these policies based on empirical evaluations and worst-case performance.

1.2.1 On-line Scheduling Approach to Detect Overload

On-line scheduling using different scheduling models have been studied widely. In a real-time scenario, where we do not have much information about the system, it is only possible to determine the task schedulability for worst-case assumptions i.e. the one which yields maximum response time. We restrict our scope to Rate Monotonic Scheduling (RMS), which is the most common scheduling algorithm used, in which the tasks arrive periodically, and are prioritized based on the rate of arrival.

Even though Rate Monotonic Analysis (RMA) of tasks scheduled under RMS have pseudo-polynomial time complexity, they are acceptable if the analysis can be done prior to the system deployment (off-line test). These tests were traditionally used for designing small real-time systems such as sensor networks, or systems with known requirements, in case of aviation systems. However, in the case of web services with soft real time requirements, and workload varying over time, the Worst-case Response-time (WCRT) analysis is different.

We consider the problem of task schedulability on a *uniprocessor* system with

dynamic task assumptions. To solve this problem, we make use of mathematical optimization to obtain the *worst-case task set* for the given system, where the task periods are known. Further, any knowledge about task computation times can be easily encoded into the problem to determine a suitable solution. The task will have a certain worst-case execution time $e_{i,max}$, and a best-case execution time $e_{i,min}$, between which the resulting task set lies. We determine the absolute WCRT of all possible task streams, and show that if the system requires additional requirements than available already. This helps the system designer set the basic characteristics of the application, so as to maximize the system utilization. In the case of dynamic applications, it helps to check the feasibility of tasks during run-time, since it is sufficient to find the WCRT of the lowest-priority task in the system.

In sum, we have attempted to address the following issues:

- Provide a new approach to find the absolute WCRT for a system of client requests to be scheduled in a *uniprocessor* system.
- Provide an approximation scheme to identify the worst-case fractional resource allocation, such that the requests are schedulable under RMS, if the processor capacity is known at the time of design. When some information about workload is available, it can also be programmed in the model.
- Devise a scheduling policy based on WCRT analysis, for the use in dynamic real-time applications.
- Analyze stochastically, the approximation scheme with respect to other efforts. This evaluation is important, because the amount of deviation of response time from the average response time of all possible combination of resource allocations is a useful metric for evaluation.

1.2.2 Maximizing Revenue when the System is Overloaded

When the system resource cannot be adapted to handle worst-case workload demands, it is necessary to have some requests discarded. Even though there are some heuristics that perform well under such circumstances, it is hard to prove the tightness of the bounds on the actual amount of requests serviced. In order to

address that, we make use of empirical evidence to show that maximum resource utilization is possible. Several scheduling policies have been proposed to schedule the request effectively on the available resource to reduce the frequency of deadline-misses, and thereby maximize the revenue a service provider may gain.

These policies are based on either

1. the occupance of an event, such as job arrival, or completion, or if it a deadline is missed, or

2. work-conservation (or) minimizing the idle time of the system.

We consider the behavior of the scheduling policy over an infinite horizon. Note that a short overload duration (5 – 10 minutes) is sufficiently long to motivate the use of infinite horizon policies when a system is receiving several hundred (or thousand) service requests per second, as is common for many Internet services. If a system were to respond to 100 service requests per minute, a 10-minute interval would yield 60000 jobs. We also aim to maximize the average reward earned per time step; this is closely related to maximizing the total reward obtained. This model relies on the use of micro-payments, which are becoming a popular pricing design, and other pricing schemes can be approximated using micro-payments.

In sum, we have attempted to address the following issues:

- To develop a scheduling policy to improve revenues when a system is overloaded, if we knew, a priori, some information about future workload.

- Provable effectiveness of such a scheduling policy.

- Evaluate the policy against other efforts aiming to handle overload situations, using empirical analysis, and determine the optimality (or near-optimality) of the policy.

- Analyze the importance of having any knowledge about future job arrivals.

The scheduling policy we have derived is based on a stochastic improvement approach, and this approach is likely to be useful in a variety of other real-time scheduling problems.

1.3 Outline

We discuss different response time analysis techniques that we analyze in varying details in the Chapter 2. We provide the worst-case bounds on response times, and derive the worst possible request allocation to detect possible system overload in the Chapter 3. In Chapter 4, we discuss the scheduling policy that handles real-time systems under overload and maximize the revenue for the service provider. Finally, we provide conclusions and possible extensions in Chapter 5

Chapter 2

Background in Real-time Scheduling

We discuss *Schedulability Tests for Periodic Task Scheduling*, particularly about the on-line and off-line schedulability tests from Real-time Systems by Jane W.S. Liu [34]. Further, we discuss the fundamentals of *Combinatorial Optimization* from the Papadimitriou and Steiglitz [39], and explain the formulation of Linear Programming in some detail. This section can be skipped if the reader has fundamental knowledge of Scheduling Theory, and Linear Programming.

2.1 Basics of Schedulability Theory

A schedulability test of a real-time system is the verification of a request or a set of requests to be permitted on a processor for execution, resulting in a yes/no answer, i.e. the requests are accepted or not accepted. For requests scheduled using dynamic-priority scheduling like Earliest Deadline First (EDF), the schedulability test is simple. If the total utilization of all the requests is less than the processor capacity, the requests are schedulable. However for Fixed-Priority Scheduling (FPS), such as RMS, the schedulability analysis is little more complicated. The main advantage of using FPS is the simplicity of deployment on an embedded device, where the timeliness of a real-time system is a much sought after quality. In RMS the requests with higher priorities can preempt the ones with lower priorities and

hence the scheduling is known as Fixed-Priority Preemptive Scheduling (FPPS). We will discuss the properties of FPPS and the efforts to provide a schedulability test for FPPS.

Some of the most common scheduling tests [10, 16, 28, 33, 38, 41], as explained below, check for feasible scheduling of periodic task sets.

2.1.1 Simple Schedulability Tests

Liu and Layland Bound

The initial work on scheduling theory was done by Liu and Layland in 1973, which provides an off-line schedulability test. Some of the assumptions are, (1) the deadline of the tasks are equal to their periods, (2) the tasks are independent and periodic, with fixed computation times, and (3) there is negligible release jitter.

Theorem 1. *(Theorem 5 in [33]). The task set* $\Gamma = \{\tau_1, \ldots, \tau_n\}$ *is schedulable* ***iff****:*

$$\sum_{i=1}^{n} \frac{e_i}{P_i} \leq n(\sqrt[n]{2} - 1) = U_{bound}. \tag{2.1}$$

The schedulability analysis is very easy in this case with a complexity of $O(n)$. The bound provided by the work is a sufficient, but not necessary bound, leaving a large room for improvement. It results in very low utilization bound U_{bound} for large task sets as shown:

$$\lim_{n\to\infty} n(\sqrt[n]{2} - 1) = \ln 2 \simeq 0.6931 \tag{2.2}$$

This however inspired further research in schedulability analysis.

Hyperbolic Bound

The hyperbolic bound [10] is an improvement over the Liu and Layland bound having a larger region of coverage, but using the same assumptions as before. The algorithm has a complexity of O(n), and uses the processor utilization of each task to evaluate their feasibility. The theorem also assumes that the tasks suffer from a worst-case processor bound when initiated at the critical instant. The condition for

schedulability is given by,

$$\prod_{1=1}^{n}(U_i+1) \leq 2 \tag{2.3}$$

The simple schedulability tests are often pessimistic, and as a result will not permit a lot of tasks that can potentially be scheduled successfully. Therefore, better tests have been sought after. The optimal test for schedulability in fixed-priority scheduling makes use of WCRT analysis.

2.1.2 WCRT Analysis of Periodic Task Systems

Liu and Layland [33] first showed that the task τ_i has its maximum possible response time, when released along with the higher priority tasks. This instant is called critical instant, and the response time R_i will be the WCRT of the task. Worst-case Response-time Analysis (RTA) is a sufficient and necessary schedulability test proposed by Lehoczky et al. [32]. It is based the processor time-demand analysis, where the total processor time requirement for each task is computed, and check if the requirement is met before the deadline of the task expires. Naturally, the lower priority tasks must also account for the time demand of the higher priority tasks. The workload for the tasks are given by the relation

$$W_i(t) = e_i + \sum_{j<i} \left\lceil \frac{t}{P_k} \right\rceil e_k, \tag{2.4}$$

$$\forall \ 0 < t \leq P_i$$

If a task τ_i meets its deadline, according to the time demand analysis, then it is implied that any higher priority task will also meet its deadline. In order to guarantee the feasibility of the task set, the tasks must be checked for the inequality $W_i(t) \leq t$ at all the points in between $(0, D_i]$. Lehoczky et al. [32] proved that it is necessary and sufficient to check for the condition at specific points, referred to as *preemption points* for τ_i. The number of preemption points, depends upon the size of the inputs, making the running time of the algorithm, pseudo-polynomial.

2.1.3 WCRT Analysis of Tasks with Arbitrary Deadlines

When the tasks have deadline greater than the periods, then additional factors must be considered while performing the time demand analysis. This is because, the processor time demand increases with the invocation of every additional job. In essence, the analysis must be extended until the end of a period called *level-i busy period*. Lehoczky [29] defined the *level-i busy period* of a task τ_i, as the total time the processor is busy, until the k^{th} invocation of τ_i is completed. The processor demand function for *level-i busy period* which ends at t when k jobs of τ_i are scheduled is given as follows:

$$W_i(k,t) = ke_i + \sum_{j<i} \left\lceil \frac{t_k}{P_j} \right\rceil e_j \leq t \tag{2.5}$$

The *level-i busy period* ends when the computation requirement of τ_i is completed. The WCRT of the task set is given by the maximum among the response times of the individual jobs.

2.1.4 Competitiveness of On-line Scheduling Algorithms for Real-time Systems

On-line scheduling algorithms in real-time systems decides if a new job of a task can be scheduled dynamically (during the run time). When a hard real-time system is overloaded, one or more requests tend to miss its deadlines, thereby resulting in timing failures. In soft-real time systems, missing such deadlines does not imply timing faults, but merely loss of revenue. So the aim of the algorithm is to maximize the revenue (value v_i)accrued from successful completion of jobs. A useful metric for comparing the *goodness* of an on-line scheduling algorithm is its *competitive factor* $0 < r < 1$. If an algorithm has a *competitiveness* of r, it means that the algorithm is guaranteed to achieve a cumulative value of $V^{\zeta} = r.V^{\zeta^*}$, where V^{ζ^*} is the cumulative value of an off-line (or clairvoyant).

Real-time Queuing Network Theory [31] has been used to model heavy traffic in real-time systems. In a $M/M/1$ queuing theory, one assumes a Poisson(λ) task arrival process to a single server, and each task stream has an exponentially(μ) dis-

tributed computation requirements. If one takes the number of tasks in the system as a state variable, then this state is a Markov Chain having equilibrium distribution given by $\pi_i = (1-\rho)\rho^i, i \geq 0$, where $\rho = \lambda/\mu$, is the traffic intensity. The task streams are modeled as a Markov process, using *lead-time*, the time until its deadline. If *lead-time* is negative, then the task missed its deadline. In the case of constant deadlines, the equilibrium distribution of the queuing system can be obtained in a closed form.

2.2 Basics of Combinatorial Optimization

Combinatorial Optimization problems are the class of problems with discrete solutions, in which we minimize or maximize an *objective function*, with respect to one or more constraints. An instance of an optimization problem is defined in [39], as a pair (F,c), with is any set of feasible points F, and a cost function c, given by a mapping

$$c : F \to \mathbb{R}$$

The problem is to find an $f \in F$ for which

$$c(f) \leq c(y) | \, \forall y \in F$$

The point f is called a *globally optimal* solution to the given instance. Linear Programming in particular is a technique to obtain the best outcome for the given objective function, subject to linear equality and inequality constraints. If a Linear Program (LP) has a feasible solution, then it is bounded by the constraints, thereby forming a convex polyhedron. The solution of the LP is either the minimum or maximum possible point within the convex region. An instance of LP is defined by

$$F = \{x : x \in \mathbb{R}^n, \, Ax = b, \, x \geq 0\} c : x \to c'x$$

Even though this linear program formulation is a continuous optimization problem, it is considered as a Combinatorial Optimization problem because any instance I of a LP is bounded by a set of constraints, without which there is no feasible solution for the problem. For example, in Figure 2.1, we minimize the objective function $c'x = c_1x_1 + c_2x_2 + c_3x_3$. The corners of the triangle formed by the plane $c'x$ will

contain the maximum or minimum possible values of our LP. Linear Programs have a polynomial time complexity, and finds optimal solution in a finite number of steps.

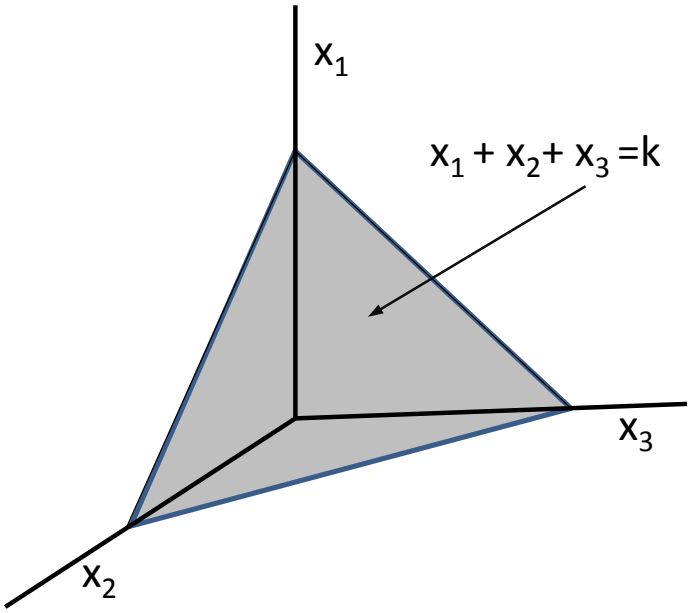

Figure 2.1: Binding Constraints in a LP Problem

Chapter 3

On-line Scheduling Policy for Overload Management

Scheduling of real-time applications with *dynamic task assumptions* fall under an important class of FPS. There has been no prior effort towards determining absolute WCRT bound for such a system. This result is highly useful in the situations, where we need the detect system overload possibilities, in order to manage it effectively. This motivated us towards finding the WCRT for such systems.

3.1 Related Work

Liu and Layland bound [33], Hyperbolic Bound [10], are simple $\Theta(n)$ algorithms for schedulability, but are very pessimistic. Exact schedulability test, using the necessary and sufficient condition for RMA [32], is computationally complex, leading to a compromise to use approximation schemes. Extensive research has been done on Polynomial-Time Approximation Scheme (PTAS) for Exact RTA [16, 20, 22, 38], using a variety of techniques, such as simplification of the problem domain, approximating the solution for future job arrivals etc. Resource augmentation analysis and, approximation analysis are used to bound the accuracy of such algorithms. Tei-Wei Kuo et al. [16], provided harmonic-period based test useful for on-line analysis and admission control. However, it is restricted only for multimedia streaming service where the task streams have periods harmonic to

the other streams. The scope of RMA is restricted in such applications, and the analysis cannot be applied to other applications.

Albers and Slomka [1], and later Fisher and Baruah [20] proposed a Fully Polynomial-Time Approximation Scheme (FPTAS) for static-priority feasibility analysis for response analysis test on EDF and RMS respectively using backed by a Resource Augmentation analysis [38]. According to the technique, the accuracy of the approximate test is lower-bounded by the exact test performed on a system with half the computing power. Several derivatives these approximations on the exact Response-time characterization given by 2.4 has been proposed [18, 20, 38], which are polynomial time algorithms. Some of them also account for release jitter J_i of a task, which is the most significant factor affecting the computation times.

However, when there is limited or no information about the workload, at the time of the design, such analysis are not possible. Empirical analysis to get a good measure of the worst-case performance is time consuming, and defeats the purpose of the PTAS. The Multiframe Task Model (MF) task model has been studied in the real-time systems community after the initial work by Mok and Chen [37]. Under this model, the task execution times are periodically changed according to a pattern. Therefore, schedulability analysis have been performed for such systems with *dynamic task assumptions*. This is the closest available real-time systems model to our work. However, we further relax the constraints of the MF, by allowing the computation time of the jobs to be any feasible value, rather than following a pattern. We define this model as the General Task Model (GTM), and will consider the model for the design and analysis of our PTAS.

Lee et al. [28] provided a schedulability test for real-time systems with *dynamic task assumptions* using mathematical optimization. Tasks streams are considered to have a predefined period corresponding a Quality-of-Service (QOS) option in Motion Picture Experts Group (MPEG) or $H.264$ video streaming or surveillance radar systems application. They provided a polynomial-time heuristic, leading to a sub-optimal solution (a sufficient test). While using mathematical optimization, tightness bounds cannot be provided for a general case, making the algorithm not easily comparable to the actual schedulability test.

Majority of the approximation techniques, such as the aforementioned cases have the task parameters pre-defined. Particularly, the task computation times are

used in the computation of the bounds. This is useful for many cases, in which the schedulability test is done prior to the job execution, making it inefficient for dynamic applications [35]. Particularly, in servers, where large number of requests with QOS guarantees are received, efficient on-line schedulability test is essential. Furthermore, the task assignment based on worst-case conditions leads to reduced CPU utilization, since more tasks cannot be accommodated. By settling to average-case conditions, a reasonable task assignment is possible. But during high CPU load, or when there are interfering tasks, some tasks may miss their deadlines. Therefore, the approximation algorithms we consider, assumes arbitrary task computation times.

3.2 System and Task Model

3.2.1 The GTM

We define the real-time system with *dynamic task assumptions* with a GTM, as follows:

Definition 1. *A general task τ_i is represented by a tuple $(\mathbb{E}_i, P_i)$, where $\mathbb{E}_i$ is a tuple $(e_{i,min}, e_{i,max})$, where the elements are minimum and maximum permissible execution times for the task, respectively, and P is the task period.*

The average execution time of the task τ_i is $\frac{e_{i,min}+e_{i,max}}{2}$. But, instead of using the average execution time of the task, we relax the minimum and maximum limits to 0 and P_iU, respectively, where U is the system utilization. We consider a *uniprocessor* system with *n general tasks*, represented by $\Gamma = \{\tau_1, \tau_2, \ldots, \tau_n\}$ competing for a shared resource. We also assume that each task τ_i has a relative deadline D_i, before which the execution must complete. Since we assume RMS, the tasks are ordered according to increasing order of periods. They have decreasing order of priorities, such that a task τ_i can be preempted by any higher-priority task τ_j, where $j < i$. Each task utilizes a portion of the system resource, represented by task utilization, $u_i = \frac{e_i}{P_i}$. Therefore, the total system utilization is given by $U = \sum_{i=1}^{n} u_i = \sum_{i=1}^{n} \frac{e_i}{P_i}$.

The total time between the release of τ_i and its completion time is given by its Response-time R_i. We assume that the tasks are scheduled at the critical instant, i.e.

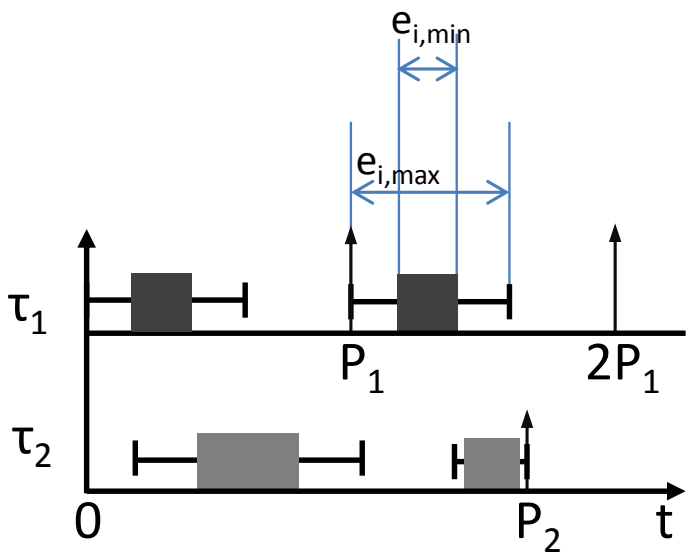

Figure 3.1: Illustration of GTM for Two Tasks

all tasks are released simultaneously, resulting in the WCRT of task i. We assume that the tasks are independent, where each task can be preempted by higher priority tasks. Further, they have negligible context switch overhead and release jitter J_i, but can be easily extended to account them.

3.2.2 Simple Task Scheduling Model using RMS

We now revisit the exact test for schedulability of tasks scheduled under RMS by Lehoczky et al. [32] in Section 2.1.2. The theorem gives both sufficient and necessary conditions for task schedulability under RMS, and hence called an exact schedulability test. The algorithm performs a processor time-demand analysis for the task set, at specific time instants, to determine if the processor is free at any of those points. Formally, the theorem is stated as:

Theorem 2. *Theorem 1.2 in Lehoczky et al. [32]: Given the task set* $\Gamma = \{\tau_1, \ldots, \tau_n\}$

1. *The task* τ_i *can be scheduled for all task phasings using the* RMS ***if and only***

if

$$R_i = \min_{t \in \mathbb{S}_i} \left\{ T : W_i(t) \leq t \right\} \tag{3.1}$$

where $W_i(t)$ is the processor time-demand at the instant t, given by Equation (2.4). *$\mathbb{S}_i$ represents a finite set of* preemption points *for τ_i that increases the processor time-demand, before the instant R_i. it is given by:*

$$\mathbb{S}_i = \left\{ kP_j |\ j = 1, \ldots, i;\ k = 1, \ldots, \left\lfloor \frac{R_i}{P_j} \right\rfloor \right\} \cup \{R_i\} \tag{3.2}$$

2. *The entire task set is schedulable for all task phasings using* RMS ***if and only if***

$$R = \max_{t \in \mathbb{S}_i, 1 \leq i \leq n} R_i \leq t \tag{3.3}$$

When the tasks are released with worst-case phasing,i.e. along with the other tasks, Liu and Layland [33], showed that the tasks are released at the critical instant, and the Response-time of the task is equal to the WCRT. Therefore, we also consider the first job invocation of the tasks are released together, at time 0. Theorem 1 provides the exact test for finding the WCRT R_i, when the tasks are released at the critical instant.

The set $\mathbb{S}_i$ is a subset of all the points of time $t, 0 < t < P_i$, forming a piece-wise monotonically increasing processor time-demand function, marked by the *pre-emption points*. For example, consider the example task set in Figure **??**, $\Gamma = \{(1,3),\ (1.5,5),\ (1.25,7),\ (0.5,9)\}$. The time instants when the processor demand increases, with all the four tasks present in the system is given by $\mathbb{S}_4 = \{3,5,7,6,9\}$. Theorem 2 proves that the task set is schedulable under RMS, and gives the value of WCRT as 9. It is to be noted that the total utilization of the task set is 0.8674, which is much greater than that of the Liu-Layland bound of 0.7568 for a task set having four tasks. This gives sufficient evidence that the RTA of tasks will provide an exact schedulability test, and yield the WCRT of the lowest-priority task in the system. The point of intersection of the processor time-demand and the line $f(t) = t$, is the WCRT of the task set.

3.2.3 Practical Considerations for Scheduling in a GTM

There are several advantages of using the RTA to find the response bounds for a system such as GTM. As opposed to use a worst-case scenario or average-scenario of probabilistic models for Exact Test [32] or Approximation Schemes [38, 48], a single analysis of GTM tasks is computationally efficient. In order to obtain a relatively good estimate of the worst-case task assumption, $10,000$ iterations are needed, and an algorithm to uniformly assign task utilizations to generate a random task set. Evaluations took time in the order of several minutes, for the Exact Test [32], and in the order of few minutes for the Approximation Schemes [38, 48].

We will show that the WCRT obtained for a task system under GTM is sufficiently accurate for large task utilizations, and large number of tasks. While designing an embedded device, using this model, the system designer should have a deadline greater than R_i^{ub}, for a τ_i, so that it doesn't suffer a timing fault. This deadline would be at least as big as the Response-time of any feasible task set for the system, and therefore needs proper planning before deciding the task periods. When the ratio of the smallest task period to the largest task period in the system is huge, the WCRT is very large, while in the case of harmonic task periods [16], the WCRT is acceptably small. The benefit of using RTA is that, once we perform it for task τ_i and determine feasibility, it is implied that all the higher priority tasks $\tau_j, j < i$ follow suit. For simplicity of calculations, we limit our scope to the interference caused by higher priority tasks, and not the blocking of system resources by lower priority tasks in the system.

3.3 WCRT Analysis of a General Task Model

In this section, we propose a new WCRT analysis technique using mathematical optimization, based on the utilization bound estimation technique, previously discussed in [28, 41]. We make use of the definition of a *barely-schedulable task set*, which has a utilization U^{ub}, and any task with utilization $U > U^{ub}$ is unschedulable with the same processor.

3.3.1 Problem Description

Liu and Layland proved that the WCRT of a task is found when performing RMA on tasks released at the critical instant. For a GTM, there are many possible WCRTs, corresponding to every possible task assumption. We propose a new approach to determine Response-time bound. It is defined as

Definition 2. *The upper-bound on Response-times, R_i^{ub} is the maximum Response-time among all tasks sets of* GTM*, when scheduled using* RMS.

$$R_i^{ub} = \max_{\mathbb{E}_j, \forall j \leq i} R_i \tag{3.4}$$

For a given system utilization, $\mathfrak{U}$, we aim to find R_i^{ub}. In order to achieve that, we need a mathematical optimization approach, such as LP to maximize R_i over the possible values of $\mathbb{E}_j, \forall j \leq i$. However, this approach has a few drawbacks:

1. We must always check the Response-time of all the jobs in the *level-i busy period*, and find its maximum to determine the WCRT by finding.

2. We must use an exit mechanism for the algorithm, when the end of the *level-i busy period* is reached. In the work of Park et al. [41], the algorithm exits, when there is no feasible solution.

Therefore, we obtain WCRT with an alternative representation, using a LP. Here, we find a *barely-schedulable task set* for an assumed Response-time bound R_i^{ub}. The task set will have a total utilization equal to the bound U^{ub}. Any task set with a smaller utilization will be schedulable. Therefore, U^{ub} and R_i^{ub} forms the upper bound on utilization and WCRT respectively. We then compare it with the given system utilization, $\mathfrak{U}$, and re-iterate the LP with a near optimal value of R_i^{ub}, until we find a sufficiently accurate Response-time bound, close to $\mathfrak{R}$.

Constraints of the LP

We now show the sufficient and necessary conditions required for the feasibility of a task, and how it directly relates to the formulation of LP problem to find

utilization bound. A level-i utilization bound U_i^{ub} is said to be sufficient if the total utilization $U \leq U_i^{ub}$. Further, the processor time demand must be greater than the available computing power, at all times, until the task's Response-time. These constraints are listed below:

1. The processor time-demand function of all the tasks $\tau_1, \ldots, \tau_i$ meets the line $f(t) = t$ at R_i^{ub}. This is the WCRT of the low priority task τ_i. It is the cumulative processor usage of all tasks, when initiated at the critical instant. The first constraint is therefore given by:

$$W_i(t) = e_i + \sum_{j<i} \left\lceil \frac{R_i^{ub}}{P_j} \right\rceil e_j = R_i^{ub} \tag{3.5}$$

   This condition is only a sufficient condition for schedulability, such that if R_i^{ub} is smaller than the task's deadline, the task is feasible. It doesn't say that R_i^{ub} is the WCRT, because we haven't verified if the system is idle in between time $(0, R_i^{ub}]$. The next condition will solve that particular issue.

2. At any time instant $t < R_i^{ub}$, the processor time demand is always greater the resource. In other words, $W_i(t) > t$. This is a continuous constraint, in which one must check at each instant in $(0, R_i^{ub}]$. Lehoczky et al. [32] showed that it is *sufficient and necessary* to check for this condition only at the *preemption points* given by $\mathbb{S}_i$, rather than all the time instants. If we only check if the processor time demand is higher than the computing power, at the points of time in $\mathbb{S}_i - \{R_i^{ub}\}$, we will have the necessary conditions for schedulability.

$$W_i(t) > t; \forall t \in \mathbb{S}_i - \{R_i^{ub}\} \tag{3.6}$$

   These set of conditions will not check if the processor time demand will over-flow after time R_i^{ub}.

We must therefore look at both the conditions above in order to find schedulability test. Consequently, the Response-time R_i^{ub}, for which these conditions are satisfied will be the WCRT of the task set. Also, the LP, that aims to find the *barely-schedulable task set* satisfying these conditions, will have a utilization equal to the level-i *barely-schedulable utilization bound* U_i^{ub}, as shown in [28], we can

refer to the WCRT of any feasible task set as the level-i *comfortably-schedulable Response-time bound*. This is because, by the definition of the LP, any task set with utilization $U \leq U_i^{ub}$, will be schedulable.

Definition 3. *The level-i comfortably-schedulable Response-time bound for any task set, with utilization $U \leq U_i^{ub}$ is given by R_i^{ub}.*

By picking the value of R_i^{ub} from the possible values, we get an initial feasible U_i^{ub}. By comparing it with the given system utilization bound $\mathfrak{U}$ and searching linearly for the $\mathfrak{R}$ that yields the actual solution, we obtain the sufficiently accurate value for WCRT.

Hardness of the Problem

The proposed LP must satisfy several constraints that the affect the task utilizations. We will prove that we must satisfy a pseudo-polynomial number of constraints in order to get the exact WCRT. Further, the number of jobs to be verified within the *level-i busy period* is large, when the system utilization $\mathfrak{U}$ is arbitrarily large. These factors make the problem **Pseudo-polynomial**, and simpler polynomial-time formulations must be used, when we need to use this technique for on-line schedulability tests.

3.3.2 Exact WCRT Analysis

Determining the Barely Schedulable Task Set

The tight level-i utilization bound U_i^{ub} is the utilization of the *barely-schedulable task set*, given the input Response-time R_i^{ub}. It is first defined by Lee et al. [28] as the utilization bound that guarantees schedulability of tasks $\tau_1, \ldots \tau_i$. The objective function of the LP is therefore given by

$$\sum_{j=1}^{i} \frac{e_j}{P_j} = U_i^{ub} \tag{3.7}$$

The *level-i utilization bound* has an interesting property, which allows us to restrict the number of LPs required to obtain U^{ub} to 1, instead of i, when there

are i tasks in the system. It is shown by Lee et al. [28] that the *system-level utilization bound* U^{ub} as the smallest value among all of the *level-k utilization bounds*, where $k = 1, \ldots, i$ utilization. The following lemma determines the *system-level utilization bound*, under a special condition.

Lemma 1. *If the Response-time of the task τ_i yields a level-i utilization bound U_i^{ub}, the following condition holds:*

$$U_j^{ub} \leq U_i^{ub}, \forall\, R_j^{ub} \leq R_i^{ub} \tag{3.8}$$

Proof. Let us consider $j = i-1$, be a next highest priority task to τ_i. The *level-(i-1) comfortably-schedulable Response-time bound* is R_{i-1}^{ub}, and the corresponding utilization bound is U_{i-1}^{ub}. Since τ_{i-1} has higher priority than τ_i, the period $P_{i-1} \leq P_i$. The processor time demand functions for the two tasks are given as follows:

$$W(t_i) : \sum_{j \leq i} e_j \left\lceil \frac{t_i}{P_j} \right\rceil \;\geq\; t_i$$

$$W(t_{i-1}) : \sum_{j < i} e_j \left\lceil \frac{t_{i-1}}{P_j} \right\rceil \;\geq\; t_{i-1}$$

$$\forall t_i = \left\{ kP_m |\; m = 1, \ldots, i;\; k = 1, \ldots, \left\lfloor \frac{R_i^{ub}}{P_m} \right\rfloor \right\}$$

$$\forall t_{i-1} = \left\{ lP_n |\; n = 1, \ldots, i-1;\; l = 1, \ldots, \left\lfloor \frac{R_{i-1}^{ub}}{P_n} \right\rfloor \right\}$$

This is a special case of problem addressed in [23] in which the condition $W(t_{i-1})$ is redundant. The first equation in Equation (3.3.2) dominates the second one, i.e. if $\frac{\Sigma_{j<i} \left\lceil \frac{t_i}{P_j} \right\rceil e_j}{t_i} \geq \frac{\Sigma_{j<i} \left\lceil \frac{t_{i-1}}{P_j} \right\rceil e_j}{t_{i-1}}; \forall j \leq i-1$. Due to the presence of i^{th} task, additional terms are possible for $W(t_i)$, as shown in Equation (3.3.2). Since, we considered that $P_{i-1} \leq P_i$, the corresponding Response-time bounds $R_{i-1}^{ub} \leq R_i^{ub}$. The relation between the processor demand functions of the tasks can be derived based on their dominance, assuming $\Xi(i) = \frac{W(t_i)}{t_i}$ as follows:

$$\Xi(i) = \frac{\sum_{j \le i} \left\lceil \frac{t_i}{P_j} \right\rceil e_j}{t_i}$$
$$= \frac{\sum_{j<i} \left\lceil \frac{t_i}{P_j} \right\rceil e_j}{t_i} + \frac{\left\lceil \frac{t_i}{P_i} \right\rceil e_i}{t_i}$$
$$\Xi(i-1) = \frac{\sum_{j<i} \left\lceil \frac{t_{i-1}}{P_j} \right\rceil e_j}{t_{i-1}}$$

Here, the first term of $\Xi(i)$ is always greater than $\Xi(i-1)$, thereby making the former dominate over the latter. The additional terms of $\Xi(i)$ only makes our case much stronger, thereby eliminating the necessity of checking $\Xi(i-1)$, while having i tasks in the system. Therefore, the utilization upper bound U_i^{ub} is also an upper bound for utilization for a system with $j < i$ tasks. □

We now give the LP formulation to minimize U_i^{ub} and find the *barely-schedulable task set*, using the constraints mentioned in Section 3.3.

$$\begin{aligned} &\textit{Minimize} && U_i^{ub} = \sum_{j=1}^{i} \frac{e_j}{P_j} \\ &\textit{Subject to} && \tau_j, j \le i \textit{ is schedulable under } \text{RMS}(3.5) \\ &&& \textit{Processor is not fully utilized at } R_i^{ub}(3.5) \\ &&& \textit{Processor is fully utilized at } t \in \mathbb{S}_i - \{R_i^{ub}\}(3.6) \end{aligned} \tag{3.9}$$

Since the Response-time of the task set is R_i^{ub}, processor is not fully utilized at that instant. This is a sufficient condition for schedulability of τ_i. The constraints that expect the processor to be fully utilized at $t \in \mathbb{S}_i$ provides the necessary but not sufficient conditions for task schedulability. Therefore, we require both in order to find the optimal utilization bound U_i^{ub}.

Sufficient Conditions for Utilization Bound

The utilization upper bound, obtained by only using the sufficient condition, given in Equation (3.5) is evidently smaller than the one obtained using all the constraints. We call the utilization upper bound, as the minimum possible upper bound $U_i^{ub,min}$. The following Lemma, first proved by Lee et al. [28] provides the suf-

ficient condition for schedulability, and in turn finds the minimum possible level-i utilization bound $U_i^{ub,min}$ corresponding to a *barely-schedulable task set*.

Lemma 2. *The sufficient condition for schedulability of a* barely-schedulable task set *is given by the following constraint.*

$$\sum_{j<1} \left\lceil \frac{R_i^{ub}}{P_j} \right\rceil e_j + e_i = R_i^{ub} \tag{3.10}$$

Proof. Equation (3.10) shows that, when a task set Γ is not feasible with a *barely-schedulable utilization bound* U^{ub}, it is possible to schedule a task set by changing the task computation times. If the processor demand at time $W(R_i^{ub}) < R_i^{ub}$, then infinite number of schedulable task sets are available, with utilization as low as 0. Therefore, we do not consider that case. Let the task set is barely schedulable, and its processor time demand is greater than R_i^{ub}. In this case, at least some amount of work for a task $\tau_j, j < i$ is done after R_i^{ub}. Let δ_j be extra work done after R_i^{ub}.

$$\begin{aligned} \sum_{j=1}^{i} \left\lceil \frac{R_i^{ub}}{P_j} \right\rceil e_j + e_i &> R_i^{ub} \\ \sum_{j=1}^{i} \left\lceil \frac{R_i^{ub}}{P_j} \right\rceil e_j + e_i &= R_i^{ub} + \delta_j \end{aligned}$$

Let us consider a new task set, that is barely schedulable Γ' with processor time demand at R_i^{ub} is R_i^{ub}. This can be created by rewriting from the task set, we arrive at a new task set Γ', with $e'_j = e_j - \frac{\delta_j}{\left\lceil \frac{R_i^{ub}}{P_j} \right\rceil}$ as follows.

$$\begin{aligned} \left(\left\lceil \frac{R_i^{ub}}{P_j} \right\rceil e_j - \delta\right) + \sum_{k\neq j,k=1}^{i} \left\lceil \frac{R_i^{ub}}{P_k} \right\rceil e_k + e_i &= R_i^{ub} \\ \left(\left\lceil \frac{R_i^{ub}}{P_j} \right\rceil e'_j\right) + \sum_{k\neq j,k=1}^{i} \left\lceil \frac{R_i^{ub}}{P_k} \right\rceil e_k + e_i &= R_i^{ub} \end{aligned}$$

When there is an overflow of δ_j, the overall utilization is reduced by δ_j/P_j, because the overflow is eliminated. In order to fill up the idle time, more computation can be added, which is equal to $\lfloor P_i/P_j \rfloor \delta_j$, and correspondingly, the utilization is increased by$(\lfloor P_i/P_j \rfloor \delta_j)/P_j$, which is less than δ_j/P_j. Therefore, the utilization

of the task set Γ' is less than that of Γ. We can conclude that, when we use the sufficient condition for schedulability defined by the Equation (3.10) in a LP to find $U_i^{ub,min}$, will return a *barely-schedulable task set*. □

From the Figure 3.2 (a), we can see that, if we only check for completion of task τ_3 at time 8, the test will pass, but still have idle times in between 0 and 8. This will make the WCRT to be less than 8, and result in the insufficiency of RTA. The LP using only the constraint from Equation (3.5) will give the minimum uti-

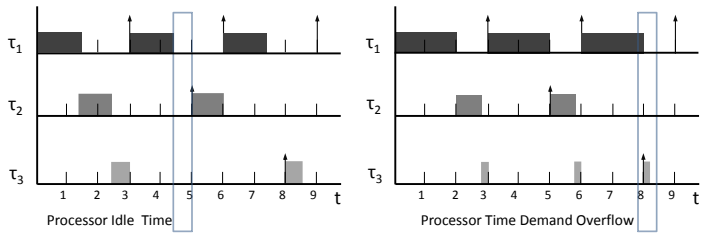

Figure 3.2: Occurrence of (a)Processor Idle Time, with Sufficient Test for Schedulability, and (b) Processor Time Demand Overflow with Necessary Tests for Schedulability

lization bound $U_i^{ub,min}$, while the LP using all the constraints from Equations (3.5) and (3.6) will give rise to the maximum possible utilization bound U_i^{ub}. This result is important, because the number of constraints in Equation (3.6) is pseudo-polynomial with respect to the number tasks. Therefore, in order to provide the worst-case bounds on the U_i^{ub} or correspondingly the WCRT, the $U_i^{ub,min}$ is helpful.

Necessary and Sufficient Conditions for Utilization Bound

We now provide the LP using all the constraints specified in (3.9) that gives the *barely-schedulable task set*, and the corresponding upper bound on utilization U_i^{ub}. In addition to having the processor complete all its computation requirements by R_i^{ub}, it must also be fully utilized until the time $t < R_i^{ub}$. The condition is given

by Equation (3.6). For example, if we only consider $\sum_{j<i}\lceil\frac{t}{P_j}\rceil e_j + e_i > t, \forall t \in \mathbb{S}_i$, in Figure 3.2 (b), we will not be able to detect the overflow, thereby failing the schedulability test. Therefore, in order to show that the processor demand meets line $f(t) = t$ only at time $t = R_i^{ub}$, we must use both the conditions and have an exact schedulability test. However, the LP formulation can be relaxed to accept anything in between the sufficient and exact test. This means that, the utilization bound obtained with a set of constraints, can be within the maximum limit of U_i^{ub} when the exact schedulability test is used, or $U_i^{ub,min}$, when only the sufficient constraints are used. The corresponding WCRT will be greater when only the sufficient constraints are used, as opposed to all the constraints. This is given by the following theorem.

Theorem 3. *The solution of following* LP *will always be in the limit* $[U_i^{ub}, U_i^{ub,min}]$

$$\begin{aligned} \textit{Minimize} \quad & \sum_{j\leq i}\frac{e_j}{P_j} \\ \textit{Subject to} \quad & \sum_{j<i}\left\lceil\frac{R_i^{ub}}{P_j}\right\rceil e_j + e_i \quad = R_i^{ub} \\ & \sum_{j<i}\left\lceil\frac{\Phi}{P_j}\right\rceil e_j + e_i \quad \geq \Phi \mid \Phi \subseteq \{kP_m\}, \forall_{m=1}^{i}, \forall_{k=1}^{\lfloor\frac{R_i^{ub}}{P_m}\rfloor} \end{aligned} \tag{3.11}$$

Proof. According to Lemma 2, the utilization of the task set using the Sufficient Test will be lower than the actual utilization, because of idle time between 0 and R_i^{ub}. But the total processor demand will be less than R_i^{ub}, making the task set schedulable. Let this task set be Γ' with a utilization $U_i^{ub,min}$. Correspondingly, let Γ be the task set that is barely schedulable with all the constraints, giving rise to U_i^{ub}. By establishing the upper and lower bounds, for the utilization, we can say that the utilization approximation caused by the choosing a few of the necessary constraints will be within the bounds. □

Equation (3.11), when using all the sufficient and necessary constraints will give the Exact utilization bound for our assumption of R_i^{ub}. We illustrate the importance of sufficient and necessary conditions with an example. Consider a simple four-task set with *dynamic task assumptions*, ordered based on their priority ac-

cording and scheduled by RMS. We do not know the computation requirements of each task, since we use a GTM. Table 3.1 gives the worst-case scheduling points given according to criterion given in the equation (in lemma).

τ_i	P_i	R_i^{ub}	$\mathbb{S}_i$
τ_1	5		
τ_2	14		
τ_3	27		
τ_4	35	31	$\{$ 5, 10, 14, 15, 20, 25, 27, 28, 30, 31$\}$

Table 3.1: Task Schedulability Points $\mathbb{S}_i$

In order to confirm that the Response-time of the task set is R_i^{ub}, when there is no information about the computation requirements, we need to verify the list of constraints, obtained from Equation. The following conditions are to be satisfied, for proving the Response-time of R_4^{ub} to be 31

$$\left.\begin{array}{lll} e_1+e_2+e_3 & >5 & P_1 \\ 2e_1+e_2+e_3 & >10 & 2*P_1 \\ 3e_1+e_2+e_3 & >14 & P_2 \\ 3e_1+2e_2+e_3 & >15 & 3*P_1 \\ 4e_1+2e_2+e_3 & >20 & 4*P_1 \\ 5e_1+2e_2+e_3 & >25 & 5*P_1 \\ 6e_1+2e_2+e_3 & >27 & P_3 \\ 6e_1+2e_2+e_3 & >28 & 2*P_2 \\ 6e_1+3e_2+e_3 & >30 & 6*P_1 \\ 7e_1+3e_2+e_3 & =31 & R_4^{ub} \end{array}\right\} \quad (3.12)$$

where $\}$ denotes logical AND operation, making all the conditions necessary to prove that R_4^{ub} as the Response-time of τ_4. Further, the task with least priority τ_4 is schedulable under RMS if and only if $R_4^{ub} \leq D_4$.

Tightness of the Estimated Utilization Bound

Following the proof of Lemma 2, we can say that any LP that uses the sufficient condition in Equation (3.5), and one or more necessary conditions from the Equation (3.6), will give a utilization upper bound $U_i^{ub,approx}$, which is within the range $[U_i^{ub,min}, U_i^{ub}]$. Figure 3.3 shows the plot of Utilization Bounds against Response-Time. Utilizations are obtained using the LPs that used (1) only the sufficient conditions for schedulability, and (2) both sufficient and necessary conditions for schedulability.

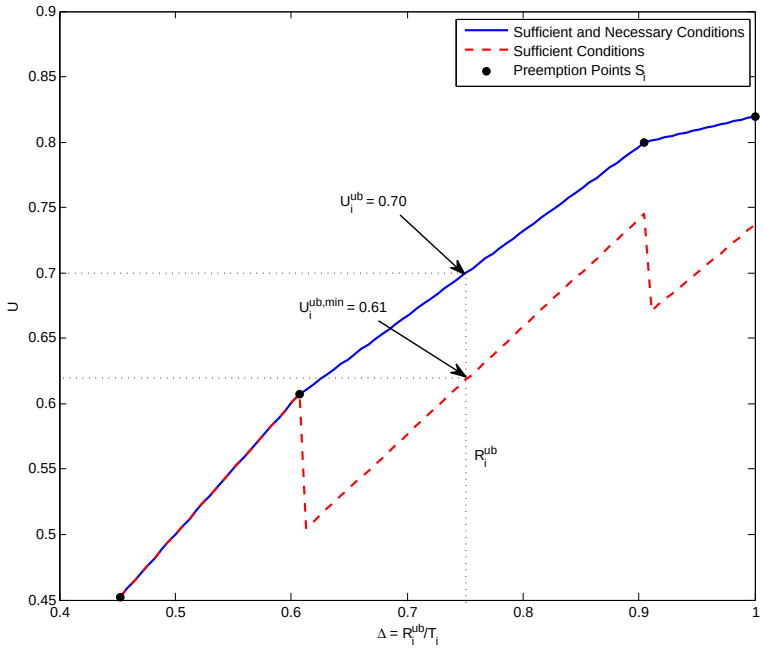

Figure 3.3: Comparison of Sufficient and Exact Test for Schedulability

Finding the Exact WCRT

The formulation of our LP problem guarantees a one-to-one correspondence between the utilization bound and response time bound. For instance, a task set which is barely schedulable under RMS has a utilization bound U_i^{ub}, and has as Response-time R_i^{ub}. Given the maximum possible utilization for the system $\mathfrak{U}$, the system designer should be able to obtain the corresponding WCRT bound. Therefore, in order to obtain the $\mathfrak{R}$ among all possible jobs having utilization $\mathfrak{U}$, we must search the solution space the LP. The bounds of the space can be as large as the *hyperperiod* of the tasks in the system, i.e. $(0, LCM(\tau_1, \ldots, \tau_i)]$. Since this is a huge space to search from, we narrow down the scope to an smaller space, corresponding to a smaller number of points from $\mathbb{S}_i$.

Once we narrow down the scope to these points, the solution close to $\mathfrak{R}$ can be obtained by further search. If we use Binary search, the algorithm converges in polynomial-time, with a complexity of $\Theta(\log \frac{1}{\varepsilon})$, for an approximation of $(U) \pm \varepsilon$.

We show, by brute-force, how the WCRT corresponding to a Utilization bound can be determined. Table 3.2 shows the relative response time bound and utilization bounds for a two task set $P_1 = 46, P_2 = 65$. We calculated the utilization bounds for each time instant in the sample range $[46, 92]$ using the LP model discussed in Section 3.3.2.

If we have a task set Γ with utilization $\mathfrak{U} = 0.863$, then we have to find the closest approximation $R_i^{ub} = 71$, which corresponds to the $U^{ub} = 0.8665$, such that $\mathfrak{U} \leq U^{ub}$. Here, R^{ub} will give the WCRT bound for the any task set, with utilization. The advantage of searching the space of LP solutions is that, it provides only the *barely-schedulable task sets*, and the solution is piece-wise linear, between the time instants marked by the *preemption points* $\mathbb{S}_i$.

3.3.3 Off-line Schedulability Test with Exact WCRT

We now provide an off-line schedulability test using the exact WCRT obtained by incrementally searching the LP solutions on the space of $\mathbb{S}_i$. Since we do not have any limitations with respect to the complexity of computation, we give the Exact and Optimal LP formulation. This is acceptable for off-line schedulability tests, when the system designer is only cares about the WCRT of the GTM, rather than

R_2^{ub}	U_2^{ub}	R_2^{ub}	U_2^{ub}	R_2^{ub}	U_2^{ub}
46	0.707692	62	0.809365	78	0.911037
47	0.714047	63	0.815719	79	0.917391
48	0.720401	64	0.822074	80	0.923746
49	0.726756	65	0.828428	81	0.9301
50	0.73311	66	0.834783	82	0.936455
51	0.739465	67	0.841137	83	0.942809
52	0.745819	68	0.847492	84	0.949164
53	0.752174	69	0.853846	85	0.955518
54	0.758528	70	0.860201	86	0.961873
55	0.764883	**71**	**0.866555**	87	0.968227
56	0.771237	72	0.87291	88	0.974582
57	0.777592	73	0.879264	89	0.980936
58	0.783946	74	0.885619	90	0.987291
59	0.790301	75	0.891973	91	0.993645
60	0.796656	76	0.898328	92	1
61	0.80301	77	0.904682		

Table 3.2: Utilization and Response-time Bounds for a Barely Schedulable Task Set

the amount of time it takes to compute.

Once we are sufficiently close to the solution, we use Binary Search between the two adjacent points on $\mathbb{S}_i$, that yields the utilization bound. We also provide the complexity analysis of the algorithm, and show how we can improve upon it. The LP formulation to find the Exact WCRT bound involves all the constraints mentioned in Section 3.3.1, because an exact schedulability test encompasses sufficient and necessary tests.

We also account for *level-i busy period*, when the input R_i^{ub} to the LP is larger than P_i. This is because, if the Response-time is greater than the period, it must be the largest of all the Response-times in the *level-i busy period*. Since we are performing an Off-line test, we can determine the WCRT within the busy period without any problem. The solution to the LP is the *level-i barely schedulable task set*, and the WCRT, FT corresponding to that task set can be easily obtained using the Equation (2.5), for each job $j = 1, \ldots$ until the busy period ends. If FT is

within the input Response-time R_i^{ub} of the LP formulation, then we can exit the algorithm. Again, this algorithm extends until the end of level-i busy period, and is of pseudo-polynomial time complexity. It is only acceptable when it is an off-line schedulability test, and approximation schemes are to be followed to approximate the solution in the case of on-line schedulability test.

Algorithm 1 SearchExact($\mathfrak{U}, <P_1, P_2, \ldots, P_n>$)

Result: Find the WCRT of the system

Data: $\mathbb{P} \leftarrow <P_1, P_2, \ldots, P_n>$ `// Task Periods`

$\Phi \leftarrow <>$

while *(1)* **do**

  $i \leftarrow 1$

  for *(*$j \leftarrow 1$ **to** n*)* **do**

    if *(*$i \% P_j$ ***is*** 0*)* **then**

      $\mathbb{S}_n \leftarrow \mathbb{S}_n \cup \{i\}$

      $id \leftarrow \mathbb{S}_n.last()$ `// id forms the` R_n^{ub} `input for LP`

      Break `// To add redundant constraints only once`

  `// Call LP to find` U^{ub}

  $U_n^{ub} \leftarrow$ **FindUtilBound**$(\mathbb{P}, \mathbb{S}_n, id)$

  if *(*$U_n^{ub} \geq \mathfrak{U}$*)* **then**

    $R_n^{ub} \leftarrow$ **BSearch**$(\mathfrak{U}, \mathbb{P}, \mathbb{S}_n, i-1, i, (1+\varepsilon))$ `// Call Search Routine`

    return R_n^{ub} `// Found the WCRT`

  `// When` R_n^{ub} `is not found, continues here`

  if *(*$i > P_n$*)* **then**

    `// If the Response-time exceeds` P_n`, find end of level-n busy period`

    $FT \leftarrow$ **LevelNPeriodEnd**$(\mathbb{E}, \mathbb{P})$

    if *(*$FT \leq P_n$*)* **then**

      $R_n^{ub} \leftarrow$ **BSearch**$(\mathfrak{U}, \mathbb{P}, \mathbb{S}_n, i-1, i, (1+\varepsilon))$ `// Call Search Routine`

      return R_n^{ub} `// Found the WCRT`

    else

      Continue

  `// When` R_n^{ub} `is not found, increment` i

  $i \leftarrow i+1$

Searching for Optimal WCRT with Binary Search

We have calls to a BSearch routine(2) upon finding a WCRT sufficiently close to the solution. It helps narrowing down the WCRT between the two recent points of time $i-1, i$. The worst-case running time of the Binary Search algorithm can be estimated in terms of the accuracy factor ε, and the size of the search space $[t_1, t_2]$. But we first need to establish the fact that we have a continuous function between those two points. Therefore, we show that the function of U^{ub} has no discontinuities in the region $[t_1, t_2]$, and show the maximum possible distance between t_1 and t_2, in order to perform binary search. The definition of the worst-case processor demand bound function is important for the proof.

Definition 4. *For a sub task set $\Gamma = \{\tau_1, \tau_2, \ldots, \tau_i\}$, the worst-case processor demand bound at time t is given by*

$$\Omega_i(t) = \sum_{j=1}^{i} \left\lceil \frac{t}{P_j} \right\rceil e_j - t \tag{3.13}$$

The interpretation of the processor demand bound function is given by Figure 3.4. We can clearly see that the processor usage decreases over a period of time, when the tasks get computed. However, there are discontinuities at the time instances which are multiples of the task periods, because newer jobs are introduced in the system. The Response-time of the lowest priority task R_i is the time when the function $\Omega_i(t) = 0$, or the processor is idle.

Lemma 3. *BSearch(2) Algorithm has a worst-case running time of $O(log(n/\varepsilon))$, for an accuracy of ε in the range (0,1), $n = R_b - R_a$, the difference between the minimum and maximum range of search for the algorithm.*

Proof. We know that the processor demand $\sum_{j=1}^{i} \left\lceil \frac{t}{P_j} \right\rceil e_j > t$, until the jobs of tasks $\tau_j, j \leq i$ are completed. It is clear from the Processor Demand Bound in Figure 3.4(a), has local maxima at time when new jobs are introduced in the system, and is idle at R_i. The only discontinuities in the processor demand bound function are at points $\mathbb{S}_i = kP_j \mid \forall j \leq i, k = 1, 2, \ldots$. The utilization bound U_i^{ub} is found at these time instants, using Equation (3.11) with time $t = R_i^{ub}$ as input. We see that the utilization bound function, U_i^{ub} has a maxima at the points represented

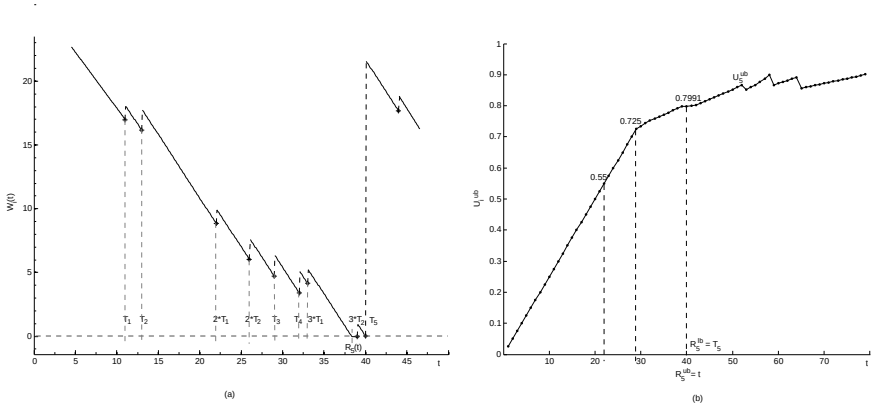

Figure 3.4: (a)Worst-case Processor Demand Bound, and (b) Corresponding Utilization Bound, U_i^{ub}, for a Barely Schedulable Task Set

by $\mathbb{S}_i$. Since we check for the utilization bound at $\mathbb{S}_i$, we have a function that is continuous in the interval $(t_a, t_{a+1}], \forall t \in \mathbb{S}_i$. The open interval after time instant in $\mathbb{S}_i$ can be neglected while performing the binary search, since we are looking for the first occurrence of $\mathfrak{U}$, and subsequent occurrence smaller than any occurrence can be neglected. Figure 3.4 (b) shows the utilization bound function for the same task set. The minimum and maximum range for the Binary Search algorithm is therefore continuous $[R_a, R_b]$, and the binary search will run in a time polynomial with respect to n and ε. □

The complexity of computing the Response-Time bounds by searching for the sub-optimal solution using binary search is $O(log(\frac{t}{\varepsilon}))$, where t is the distance between two points t_a and t_{a+1}. We showed, by Lemma 3 that the maximum distance between those two points can be the distance between the consecutive points in $\mathbb{S}_i$, where the discontinuities in the processor demand occurs.

Algorithm 2 Helper Functions for Exact WCRT Estimation

```
// Binary Search to return Response-time bound
```

Procedure **BSearch**$(\mathfrak{U}, \mathbb{P}, \mathbb{S}_n, min, max, \delta)$:

if $(min \leq max)$ **then**

  $R_n^{ub} \leftarrow (min + max)/2$

  $U_n^{ub} \leftarrow$ **FindUtilBound**$(\mathbb{P}, \mathbb{S}_n, R_n^{ub})$

  if $(U_n^{ub} \geq \mathfrak{U} + \delta)$ **then**

    $max \leftarrow$ **BSearch**$(\mathfrak{U}, \mathbb{P}, \mathbb{S}_n, R_n^{ub}, max, \delta)$

  else if $(U_n^{ub} \leq \mathfrak{U} - \delta)$ **then**

    $min \leftarrow$ **BSearch**$(\mathfrak{U}, \mathbb{P}, \mathbb{S}_n, min, R_n^{ub}, \delta)$

  else return R_n^{ub}

return *0*

```
// Find the time when Level-n Busy Period ends
```

Procedure **LevelNPeriodEnd**$(\mathbb{E}, \mathbb{P})$:

$t \leftarrow P_n$

while *1* **do**

  $j \leftarrow \lfloor t/P_n \rfloor$

  $CT \leftarrow \sum_{k=1}^{n} \left\lceil \frac{t}{P_k} \right\rceil e_k$ `// Completion time of job` j

  if $CT \leq t$ **then**

    $FT \leftarrow CT - (j-1)P_n$

    return FT `// End of busy period for the task` $(\mathbb{E}, \mathbb{P})$

  $t \leftarrow t + P_n$

The search space of BSearch 2 is shown in Figure 3.5, for the same five-task set. Let us start with the preliminary solution for U_i^{ub} at the two points t_a and t_{a+1} of time, using $R_i^{ub} = t_a$ and $R_i^{ub} = t_{a+1}$ respectively. We continue searching for the solution, until we reach a sufficiently accurate solution $\mathfrak{U} \pm \varepsilon$ using BSearch. The BSearch module has a polynomial running time in both t and ε, and therefore, has a very fast convergence rate.

The LP formulation in the algorithm makes use of job number as well, in order

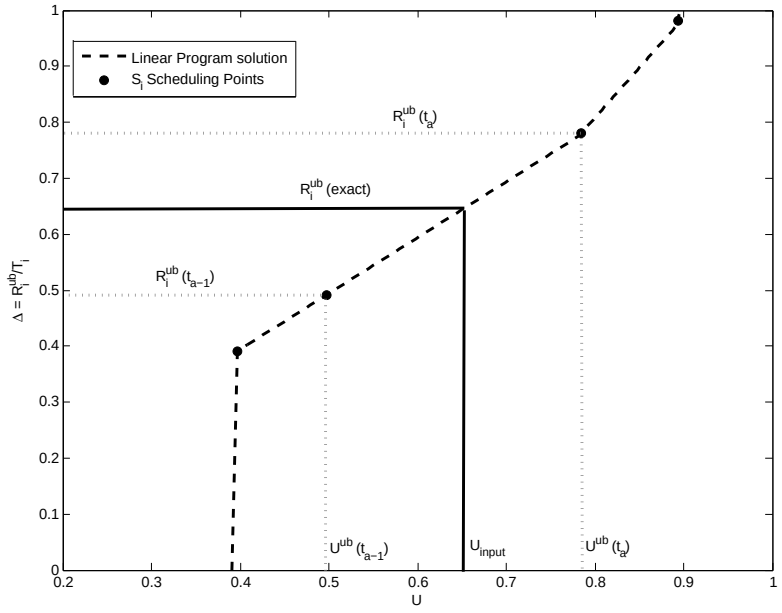

Figure 3.5: Space of Binary Search to find R_i^{ub}

to account for all the jobs in the *level-i busy period*. This is useful for finding the WCRT of task sets when the system utilization is arbitrarily high. Particularly for systems with high computing power, and when off-line scheduling is possible, this formulation is ideal.

Algorithm 3 LP Formulation to find Utilization Bound

```
// Returns minimum utilization
```

Procedure **FindUtilBound**$(\mathbb{P}, \mathbb{S}_n, R_n^{ub})$:

Minimize $\sum_{i=1}^{n} \frac{e_i}{P_i}$

subject to $\sum_{k=1}^{n-1} \left\lceil \frac{j}{P_k} \right\rceil e_k + \left\lfloor \frac{j}{c_n} \right\rfloor e_n \geq j \mid \forall j \in \mathbb{S}_n$, and

$$\sum_{k=1}^{n-1} \left\lceil \frac{R_n^{ub}}{P_k} \right\rceil e_k + \left\lfloor \frac{j}{c_n} \right\rfloor e_n = R_n^{ub}$$

return $\mathfrak{U}_n^{ub}$

3.4 Reducing the Complexity of WCRT Analysis

So far we have discussed about the Exact test for finding the Utilization upper bound U_i^{ub}, and a binary search technique to find the WCRT from the space of LP solutions. This test is very accurate, and provides the absolute WCRT of a task set, given a utilization bound $\mathfrak{U}$. The test is often done off-line, i.e. prior to the execution of the tasks. However, for applications with varying computation requirements, an approximate algorithm is essential.

We first describe a polynomial-time algorithm that will result in a sub-optimal solution of the WCRT. The accuracy of the algorithm can be bounded from below with the sufficient test, defined already. Therefore, the approximation schemes will be better than this worst-case scenario. We evaluate the performance of the algorithm with other comparable bounds using empirical studies. Finally, we provide insights into improving the accuracy of the algorithm by using a near-optimal algorithm based on the work related to pruning the constraints of schedulability test, by Bini and Buttazzo [8].

The running time of the ideal search algorithm to obtain R_i^{ub} is pseudo-polynomial, and it is acceptable for polynomial-time algorithms that delivers near optimal results. The factors responsible for making the algorithm pseudo-polynomial are:

1. Pseudo-polynomial nature of the problem, where the problem size is dependent on the value of the inputs.

2. The overhead of LP to find the bounds, and

3. The running time of the search algorithm to find the correct solution.

Since we need to search the solution space of several LPs to identify the WCRT, it is hard to quantify it mathematically. Therefore, we provide a simple polynomial-time heuristic to minimize the complexity of the problem. By modifying the implementation of the search algorithm to identify R_i^{ub}, we can obtain a sub-optimal solution running in time polynomial to the number of tasks. Depending upon the accuracy of the output expected, the polynomial-time search approximation is be implemented using approximation factor $\delta = (1 + \frac{\varepsilon}{2N})$, for a task set with N tasks. Also, the number of constraints for the LP, given by the array $\mathbb{S}_n$ makes the algorithm run in pseudo-polynomial time. Therefore, we can reduce them to one per task, when an approximation factor $\gamma = 0,1$ is set to 1. In order to use the search algorithm to use all the constraints, we set $\gamma = 0$.

3.4.1 Approximated WCRT Analysis Formulation

The two important factors that contributes to the hardness of the problem formulation are the number of constraints required for the LP formulation, and the total search space of all jobs that must be checked in the *level-i busy period*. We reduce the number of jobs to be checked to a polynomial number by using the approximate request bound function, first proposed by Fisher and Baruah [20]. It is given by

$$\delta(\tau_i, t) = \begin{cases} \left\lceil \frac{t}{P_i} \right\rceil & \text{for } t \leq (k-1)P_i \\ (t + P_i)U_i & \text{otherwise} \end{cases} \quad (3.14)$$

where $k = \lceil 1/\varepsilon \rceil - 1$, for an approximation factor $0 < \varepsilon < 1$. The approximate cumulative processor demand function is given for each job τ_i, l as

$$W_{i,l} = le_i + \sum_{j<i} \delta(\tau_j, t) \quad (3.15)$$

By reducing the number of points to check in the level-i busy period, we introduce an approximation to the actual WCRT. The Approximation Algorithm 4 runs in polynomial-time to the number of inputs. We also enforce a policy to reduce the number of constraints to be checked for the LP with the conditions provided by Bini et al. [8].

Algorithm 4 SearchApproxResponseTimes($\mathfrak{U}, <P_1, P_2, \ldots, P_n>, \varepsilon$)

$\mathbb{P} \leftarrow <P_1, P_2, \ldots, P_n>$

$\mathbb{J}_n \leftarrow$ **AcquireSearchRange**$(\mathbb{P}, P_n, \varepsilon)$

$\mathbb{S}_n \leftarrow$ **AcquireReducedConstraints**$(\mathbb{P}, P_n)$

for $i \in \mathbb{J}_n$ **do**

  $U_n^{ub} \leftarrow$ **FindUtilBound**$\mathbb{P}, \mathbb{S}_n, i$

  if $U_n^{ub} \geq \mathfrak{U}$ **then**

    $\mathfrak{R}_n^{ub} \leftarrow$ **BSearch**$(\mathfrak{U}, , \mathbb{P}, \mathbb{S}_n, i-1, i, (1+\varepsilon))$

Procedure **AcquireSearchRange**$(\mathbb{P}, t, \varepsilon)$:

$\mathbb{J}_n \leftarrow <>$

$k \leftarrow \lceil 1/\varepsilon \rceil - 1$

for *j ← 1 to k - 1* **do**

  $\mathbb{J}_n \leftarrow \mathbb{J}_n \cup \{j.P_n\}$

return $\mathbb{J}_n$

In order to provide a tighter bound, close to the exact Response-time bound provided by Lehoczky et al. [32], we modify the algorithm to make use of an optimal subset of the total constraints possible. Prior work in analyzing the space of RMS schedulability shows that, for the entire list of scheduling points, $\mathbb{S}_n$, there exists a reduced scheduling point set $\mathbb{P}_n \subseteq \mathbb{S}_n$ [8, 36]. It is formally stated as: *The reduced scheduling set $\mathbb{P}_i$ is defined by the recurrent expression:*

$$\mathbb{P}_1(t) = t \tag{3.16}$$

$$\mathbb{P}_i(t) = \mathbb{P}_{i-1}\left(\left\lfloor \frac{t}{P_i} \right\rfloor P_i\right) \cup \mathbb{P}_{i-1}(t) \tag{3.17}$$

Despite being complex, the equation is inherently simple. In a system of n tasks,

by simply including the scheduling points that lie immediately before a scheduling of a low priority task, we can fill the list $\mathbb{P}_n$. This means that the scheduling points $\mathbb{P}_j$ depends on $\mathbb{P}_i, j < i$. We make use of the reduced set of constraints for our LP formulation to obtain a tighter bound for Response-Time $\mathfrak{R}^{ub}$. Again, this formulation doesn't provide necessary and sufficient conditions for schedulability, as discussed earlier.

We illustrate it with a simple example from Table 3.1. The exhaustive set of constraints are given by $\mathbb{S}_4 = \{5, 10, 14, 15, 20, 25, 27, 28, 30, 31\}$. By recursively calculating the constraints for higher priority tasks τ_j, with the knowledge of constraints for the lower priority tasks $\tau_i, j < i$, we can find out the set $\mathbb{P}_4 = \{10, 14, 25, 27, 28, 30, 31\}$. The algorithm to determine the reduced set has a complexity of $O(n)$, and therefore is polynomial in the number of tasks n.

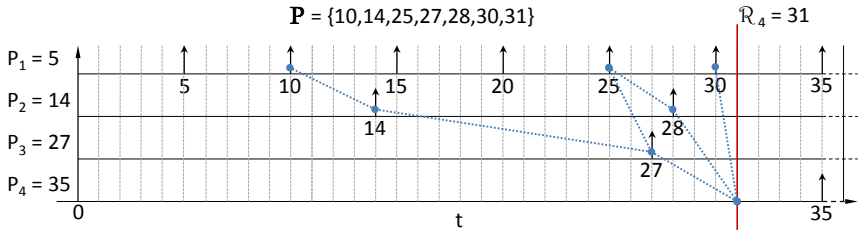

Figure 3.6: Timeline of $\mathbb{P}_j$ for the Example in Table 3.1

The only modification to the SearchApprox (4) is in the procedure AcquireReducedConstraints. The procedure AcquireReducedConstraints(5) eliminates the need for a γ option to choose minimal constraints or not. The algorithm recursively find the constraints of higher priority tasks, and runs in polynomial time. It is therefore ideal to use the method to determine the Response-time bound.

Algorithm 5 Procedure to Acquire Reduced Number of Constraints for the LP

Procedure **AcquireReducedConstraints**$(\mathbb{P}, t)$:

Data: $\mathbb{P}_n \leftarrow < t >$

for *i* ← *n - 1 to 1* **do**
    $\mathbb{L}_i \leftarrow <>$ `// $\mathbb{L}_i$ is the list of constraints for task $\tau_i$`
    for *j in* P_{i+1} **do**
        $\mathbb{L}_i \leftarrow$ **AddConstraints**$(i, j, \mathbb{P})$
    $\mathbb{P}_n \leftarrow \mathbb{P}_n \cup \{\mathbb{L}_i\}$
return $\mathbb{P}_n$

`// Adds constraints for task $\tau_k$`

Procedure **AddConstraints**$(k, time, \mathbb{P})$:

$Constraint = \left\lfloor \frac{time}{P_k} \right\rfloor P_k$

return *Constraint*

3.4.2 Empirical Evaluation of Approximated WCRT Analysis

The tightness of the algorithm can be verified with respect to conventional approaches to finding Response-times with known task computation times [18, 38, 45, 48]. For a set of given task periods, we pick computation times that are uniformly distributed and find the Response-time bound for all the samples and compare it with the Response-time bound obtained using SearchApprox(4). Though the approach of generating synthetic task sets doesn't reflect the application's characteristics, it is ideal to test the average case performance of a RTA technique, given the input utilization bound $\mathfrak{U}$. Several other algorithms to obtain Uniform Distribution of task utilizations are available, some of which are listed in [9]. We use UUnifast [7] for its simplicity in formulation and adaptability for larger task sets. We use UUnifast to obtain the task sets, and determine the Response-time bounds using other RTA. $\mathfrak{R}^{ub}$, obtained by procedure SearchApprox (4) is obtained for the Task periods assumed, for comparison. We calculate the exact formulation of Response-time bounds using the formulation given by Lehoczky et al. [32] to verify if the value is less than $\mathfrak{R}^{ub}$. A well known bound that could be computed

efficiently, known as the Fluid-flow model, proposed by [48] is also evaluated. The formulation of Response-time bound using Fluid-flow model is by expanding the ceiling function in the exact characterization as follows:

$$R_i = e_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{P_j} \right\rceil e_j$$

$$R_i \leq e_i + \sum_{j \in hp(i)} \left(\frac{R_i}{P_j} + 1 \right) e_j$$

$$R_i \leq \frac{\sum_{j \in hp(i)} e_j}{1 - \sum_{j \in hp(i)} \frac{e_j}{P_j}} = R_i^{fluid} \tag{3.18}$$

Despite its simplicity, the Fluid-flow model has a very bad worst-case performance, which we portray in the experiment. Therefore, we also compare $\mathfrak{R}^{ub}$ with another computationally efficient method proposed by Bini et al. [11], which has a bounded worst-case performance. The related works that are used in the evaluation, and the Response-time bound formulation, are listed in Table 3.3. Figure

Cited article	Response-time Model	R^{ub}
[32]	Exact Characterization	$R_i^{ub} = e_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i^{ub}}{P_j} \right\rceil e_j$
[48]	Fluid-flow Model	$R_i^{fluid} = \frac{\sum_{j \in hp(i)} e_j}{1 - \sum_{j \in hp(i)} U_j}$
[11]	Linear-approximation Model	$R_i^{bound} = \frac{c_i + \sum_{j \in hp(i)} e_j (1 - U_j)}{1 - \sum_{j \in hp(i)} U_j}$
This work	LP based Optimization	

Table 3.3: Comparison of Tight Upper Bounds on Response-time

3.7 shows the comparison of various techniques to obtain Response-time bounds that we discussed against our bound. We show only a small range of samples, of

the whole experiment for clarity. It is found that the Response-time bound using the algorithm specified in our work is close to the asymptotic value of the exact characterization. This gives us confidence that the upper bound is a sufficient value to determine task schedulability.

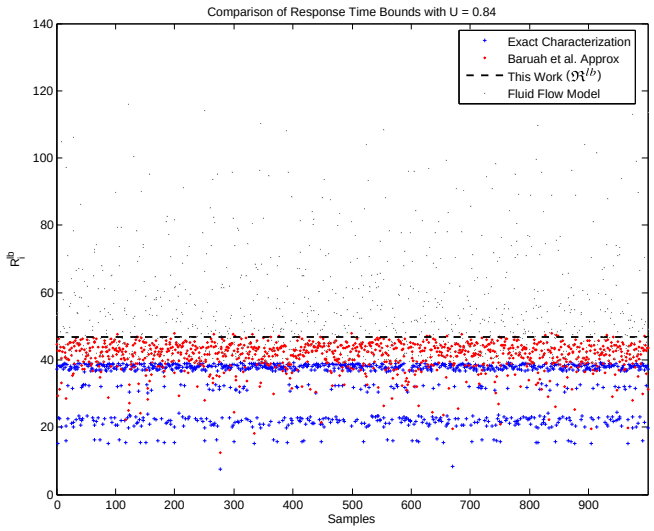

Figure 3.7: Empirical evaluation of Approximated WCRT Analysis

3.4.3 Discussion

Our experimental analysis shows that the WCRT bound obtained using LP approach is a useful number to determine the system characteristics, allowing the system designers to assign a fraction of the resource, and still be feasible. In applications like Web services, where we assume a stream of requests arriving at a shared resource, WCRT bound , and thereby loss of QOS guarantees can be estimated using our approach. The LP accepts a minimum and maximum solution

for the task computation times, which can be represented as $\frac{min}{i} \leq e_i \leq e_i^{max}$, for a task τ_i. By enforcing the constraints of the computation times, the LP will give a feasible Response-time bound that suits GTM. There are studies discussing utilization bounds as a feasibility test for similar applications such as MF, indicated by Lee et al. [28]. This ensures that there are other applications in which the WCRT Analysis can be used for.

An on-line scheduling algorithm is the only option in a system whose future workload is unpredictable, because it can accommodate dynamic variations in user demands and resource availability. The benefit of having a single metric to detect the worst-case Response-time of a group of requests in a system is itself useful for various applications with dynamic application requirements. The lack of knowledge of the workload is not an issue in the case of GTM. On-line Schedulability tests on the approximate WCRT Analysis, shown in our work is important to determine the system load whenever a new request is received.

Even though fast schedulability tests using Liu and Layland bound [33], or Hyperbolic Bound [10] are available, they are pessimistic and unsuitable for systems processing a large number of incoming requests. We proposed a new WCRT Analysis technique for a system of tasks, when the system does not have information about the workload. We showed that the complexity of the technique is pseudo-polynomial, and proposed approximation techniques to minimize the computational complexity. The proposed approximation involves some heuristics to reduce the number of parameters involved in the computation of WCRT, to make it suitable for On-line scheduling test. Approximate WCRT is comparable to the average performance of the best known Response-time Bound analysis techniques. The major benefit of the model is that it is applicable for a GTM, and predicts the worst-case scenario of the system, when large number of requests are received. Any scheduling policy built on top of the idea will help the system administrator make responsible decisions for handling overload in real-time systems.

Chapter 4

Scheduling to Improve Rewards during Overload

4.1 Related Work

In the context of soft real-time systems, where real-time jobs can be executed with some flexibility, many techniques have been presented for maximizing a utility function subject to schedulability constraints. While Buttazzo, et al. [13] provide a detailed exposition on soft real-time systems, some approaches that are more closely related to the work described in this work involve the imprecise computation [17] and the Increased Reward with Increased Service (IRIS) [19] task models. In these models, a real-time job is split into a mandatory portion and an optional portion. The mandatory portion provides the basic (minimal) quality of service needed by a task; the mandatory portion has to be completed before the job's deadline. The optional part can be executed if the system has spare capacity, but it too must be completed before the job's deadline. The optional portion results in a reward, and the longer the optional portion can execute the greater is the reward garnered. The reward for executing the optional portion is described using a function of the extent to which the option portion is executed. Along these lines, Aydin, et al. presented techniques for optimal reward based scheduling for periodic real-time tasks [3]. Other techniques for maximizing utility (which can be considered as revenue/rewards) include the use of linear and non-linear optimization [47], and

heuristic resource allocation techniques such as QoS-based Resource Allocation Model (QRAM) [43, 44].

Our work is distinct from the imprecise computation model or the IRIS model because jobs in our task model do not have a mandatory or an optional portion. Further, a fixed revenue accrues with each job completion and this is unlike prior work we have highlighted where the reward is a function of the optional portion.

Overload in real-time systems has also received attention. Baruah and Haritsa described the ROBUST scheduling policy for handling overload [6]. Baruah and Haritsa used the *effective processor utilization* as a measure of the "goodness" of a scheduling policy. The Effective Processor Utilization (EPU) is the fraction of time during an overload that the system executes tasks that complete by their deadlines. When the EPU is used as a metric for measuring the performance of a scheduling policy the task model is a *special case* of scheduling to improve rewards: in this model the reward for a job completion is equal to the execution time of the job. The task model studied by Baruah and Haritsa made no assumptions about the arrival rates of jobs. Each job was characterized by its arrival times, its execution time and its deadline. The ROBUST scheduler is an optimal online scheduler among schedulers with no knowledge of future arrivals. Baruah, et al. established that no online scheduler is guaranteed to achieve an EPU greater than 0.25 [4]. When the value of a job need not be related to the execution length, Baruah, et al. [5] provided a general result that the competitive ratio for an online scheduling policy cannot be guaranteed to be better than $\frac{1}{(\sqrt{k}+1)^2}$ where k is the ratio of the largest to smallest *value density* among jobs to be scheduled. The value density of a job is its value-to-execution length ratio.

For systems where a job's value need not be directly related to its execution length, Koren and Shasha developed the D^{over} online scheduling policy [24], which provides the best possible competitive ratio relative to an off-line (or clairvoyant) scheduling policy. Koren and Shasha also developed the Skip^{over} scheduling approach [25] to deal with task sets where certain jobs can be skipped to ensure schedulability at the cost of lower quality of service. While Skip^{over} was developed as a mechanism for dealing with overload, it is not suited to the application scenarios we have described earlier.

Hajek studied another special case when all jobs are unit length and concluded

that the competitive ratio for online scheduling of such jobs lies in the interval $[0.5, \phi]$ where $\phi = \frac{\sqrt{5}-1}{2} \approx 0.618$, the inverse of the golden ratio [21].

Competitive analysis of scheduling policies provides us good insight into the behavior of different policies but does not address all issues. The job arrival pattern that leads to poor performance of a policy ζ may be extremely rare in real systems. Additionally, two online algorithms with the same competitive ratio might have significantly varied performance in practice. Koutsoupias and Papadimitriou discuss the limitations of competitive analysis and suggest some refinements that could make problem formulation more realistic [26]. The limitations of competitive analysis have spurred investigations into several heuristics that offer good performance in most settings. For example, Buttazzo, et al. have described experiences with robust versions of the earliest deadline first algorithm [14, 15].

With regard to prior work on handling overload in real-time systems, we study a general revenue model where the revenue earned on completing a job need not be related to the execution time of the job. Moreover, we propose a scheduling policy that has limited awareness of the characteristics of the workload. While in prior work ([4, 6, 14, 15, 24]) no assumptions were made about future job arrivals, we use estimates of arrival rates to make better decisions. Such information can easily be measured, or specified, in a system, and is often described in the service level agreements between service providers and customers. This information is, therefore, not unreasonable to expect for the class of systems that we are interested in. Furthermore, Stankovic, et al. [49] have stressed the need to incorporate more information about the workload. Writing about competitive analysis for overload scheduling ([49], p. 17) they note that "More work is needed to derive other bounds based on more knowledge of the task set." Although our work does not lead to deriving bounds on competitive performance of online scheduling policies, we use information concerning the task streams to develop a scheduling policy to improve revenues in the presence of overload.

Lam, et al. [27] have presented a scheme that uses faster processors to handle overload. We have proposed a scheme that is suited to situations where extra resources may not easily be available, or cannot be deployed quickly, to ameliorate overload.

Finally, we note that we use stochastic models for soft real-time systems. Real-

time queueing theory [30] deals with probabilistic guarantees for real-time systems but Real-time Queuing Theory (RTQT) does not provide tools either for analyzing overload conditions or for maximizing rewards in a real-time system.

4.2 System and Task Model

The system and task model that we consider is that of n streams, $\{S_1, \ldots, S_n\}$, with *preemptible* jobs; all jobs are executed on a *uniprocessor* system. Within a particular stream S_i jobs arrive with a mean inter-arrival time P_i; the inter-arrival times are governed by a Poisson process with rate $r_i = \frac{1}{P_i}$.[1] The execution time of each job may also vary; for stream S_i we consider the execution time of jobs to be governed by an exponential distribution with mean e_i. Each job also has a deadline; the deadlines for jobs of S_i follow an exponential distribution with mean D_i. When a job belonging to S_i is completed prior to its deadline expiring a fixed revenue of $v_i(>0)$ is earned. We will use the terms revenue, value and reward interchangeably for the rest of this work.

In this work, we provide a method for achieving high average revenue over an infinite time horizon. An optimal scheduling policy, ζ^*, is one that will achieve the supremum

$$V^{\zeta^*} = \limsup_{t\to\infty}\{\frac{V^{\zeta}(t)}{t+1}\}$$

where $V^{\zeta}(t)$ is the revenue obtained using policy ζ over the interval $[0,t)$.

The scheduling policies of interest are *non-idling*, or work conserving, policies that make decisions whenever the state of the system changes: when a new job arrives, when a job finishes, or when a deadline expires.

This model also generalizes the traditional periodic task model studied by Liu and Layland. No relationship need exist between the deadlines and the rates of the tasks.

4.3 Identifying a Good Scheduling Policy

Before we develop some intuition regarding scheduling policies that optimize the average revenue earned over a long run of the system, we note that this discussion

[1] The inter-arrival times correspond to peak workload.

is particularly relevant for overloaded systems, i.e., for systems where $\sum_{i=1}^{n} \frac{e_i}{P_i} = \sum_{i=1}^{n} e_i r_i > 1$. If the system was under-utilized then such a policy is optimal and would generate an average revenue of $\sum_{i=1}^{n} v_i r_i$; the earliest deadline first policy, in fact, emulates this allocation when the utilization is ≤ 1.

Whenever the system is not overloaded, we will assume the use of the EDF policy. Notice that a system is guaranteed to meet all deadlines when $\sum_i e_i/D_i \leq 1$. In general, when a system operates at utilization less than 100% then EDF maximizes the average revenue earned.

We shall identify an ideal policy by first determining an optimal static allocation of the processor among the different job streams, and then improving that allocation at each decision step. Our first goal is to determine fractional allocations of the processor among the n streams. Essentially we seek a vector $\mathbf{f} = \{f_1, f_2, \ldots, f_n\}$ such that f_i represents the proportion of processor time allocated to stream S_i. In other words, such a static allocation would allocate an f_i fraction of each time unit to task stream S_i. Although this may be an impractical policy – because of the excessive context switching overhead – we shall use this as an initial step to obtaining a more practical policy.

4.3.1 Optimal Fractional Resource Allocation

We would like to partition the processor's efforts among the n streams to optimize the revenue earned. f_i represents that long-run fraction of time spent by the processor servicing jobs of stream S_i.

When dealing with systems subject to overload, job queue lengths may grow rapidly but the system is kept stable by the fact that jobs have deadlines. We let $L_i(t)$ represent the length of the queue of jobs from S_i at time instant t. The n queue lengths are stochastic processes that evolve depending on the scheduling policy chosen; further the queue lengths are independent of each other because each queue is guaranteed a fraction of the processor. The queue length $L_i(t)$ is, therefore, a simple birth-death process with the rate of arrivals to the queue being r_i and the departure rate being $\frac{f_i}{e_i} + \frac{l}{D_i}$ [influenced by job completions and deadline expirations] when the state of the queue, the queue length, is l. If we use terms that are more common to queueing systems, then the service rate $s_i = \frac{1}{e_i}$, the deadline

miss rate $d_i = \frac{1}{D_i}$, and the departure rate for the queue length process is $f_i s_i + l d_i$.

Applying some standard results concerning birth-death processes [40], the stationary distribution for $L_i(t)$, when stream S_i is allotted an f_i proportion of the processor, is given by

$$\Pi_i(l, f_i) = \frac{(r_i)^l}{\prod_{m=1}^{l}(s_i f_i + m d_i)} \Pi_0(r_i, s_i f_i, d_i), \tag{4.1}$$

where l is the state of queue i and

$$\Pi_0(r_i, s_i f_i, d_i) = \left(\sum_{l=0}^{\infty} \frac{(r_i)^l}{\prod_{m=1}^{l}(s_i f_i + m d_i)} \right)^{-1}. \tag{4.2}$$

The average revenue obtained using scheduling policy $\zeta_{\mathbf{f}}$ that allocates f_i proportion of the processor to stream S_i is

$$V_{\mathbf{f}} = \sum_{i=1}^{n} v_i s_i f_i \left[1 - \Pi_0(r_i, s_i f_i, d_i)\right] \tag{4.3}$$

and the optimal fractional allocation policy ζ^* is that policy that picks the maximizing vector $\mathbf{f}$:

$$V^* = \max\{V_{\mathbf{f}} | f_i \geq 0, \sum_{i=1}^{n} f_i = 1\}. \tag{4.4}$$

We will initially assume that we have obtained the optimal fractional allocation policy and suggest a mechanism to improve on policies that pre-allocate processor shares. We will refer to ζ^*, the optimal fractional allocation policy, as Fractional Allocation Policy (FAP). Further, we noted earlier that the fractional allocation policy might require each time step to divided among all queues, which might lead to unacceptable overhead. The improvement step will result in a policy that can be applied at every time instant when the state of the system changes, i.e., whenever a new job arrives, or when a job is completed, or when a job misses its deadline.

4.3.2 An Improved Policy for Online Job Selection

We will improve upon a fractional allocation policy $\zeta_{\mathbf{f}}$ by defining a priority index $Z_i(l_i)$ that indicates the priority of a stream when there are l_i queued jobs belonging to that stream. Then, at any time t when the scheduler needs to make a decision, the scheduler will activate a job from the stream with the highest priority index; thus stream S_i will be chosen if and only if

$$Z_i(l_i) \quad = \quad \max_k \{Z_k(l_k)\}. \tag{4.5}$$

A scheduling decision is made whenever the state of any of the queues changes. The approach underlying our improved policy is to assume that at every decision instant a particular job is scheduled and that from the next decision instant policy FAP will be applied; the selection of the job at the first decision instant is based on improving the revenue in comparison to a consistent use of FAP. By applying the improvement step (as dictated by the priority indices) at each decision instant we can obtain consistently better performance than FAP. This approach can be re-stated as follows:

- If $t = 0$ is the first decision instant then we will select a job and execute it till the second decision instant.
- Assume that FAP will be used from the second decision instant. Therefore, pick a job at $t = 0$ that will lead to an improved revenue when compared with the use of FAP from $t = 0$.
- If we treat every decision instant exactly like the first decision instant then the modified policy will consistently outperform FAP.

In this work we shall denote the policy that uses the above priority index as Policy Z.We shall now state the main theorem and then proceed to prove this theorem.

Theorem 4. *The scheduling policy that improves upon the fractional allocation policy ζ_f is the policy that chooses to service task stream i when $l_i > 0$ and*

$$Z_i(l_i) = \max_{k;l_k>0} \{Z_k(l_k)\}$$

where

$$Z_i(l_i) = v_i s_i \left[1 - \frac{(s_i f_i)\Pi_0(r_i, s_i f_i, d_i)}{(s_i f_i + l_i d_i)\Pi_0(r_i, s_i f_i + l_i d_i, d_i)} \right].$$

Understanding the modified policy. The prioritization suggested by the updated scheduling policy is greedy. This is expected when scheduling tasks with deadlines. The priorities are based on the highest possible revenue rate ($v_i s_i$). At the same times, the priority attempts to delay those streams that typically have longer deadlines; draining queues that have jobs that can wait would, at later time instant, lead to serving jobs that do not yield high revenues and this is reflected by the zero probability term $\Pi_0(r_i, s_i f_i, d_i)$. However, if a queue is sufficiently long then we can serve jobs in that queue without worrying about draining that queue and this is reflected by the $\Pi_0(r_i, s_i f_i + l_i d_i, d_i)$ term. Also, when deadlines are short the deadline miss rate (d_i) is high and this is captured by the term $(s_i f_i + l_i d_i)^{-1}$ that boosts the priority of streams with shorter deadlines.

Whenever a scheduling decision is to be made, the optimal choice would depend on whether executing a job now is better than deferring its execution. The penalty that one may incur by deferring the execution of a job is that the job may miss its deadline thereby resulting in no revenue. We denote the expectation of the revenue earned from S_i by applying the fractional allocation policy when the state of queue i is l_i as $V_{\mathbf{f},i}(l_i)$. The priority of each stream can then be computed as

$$Z_i(l_i) \quad = \quad s_i[v_i - \{V_{\mathbf{f},i}(l_i) - V_{\mathbf{f},i}(l_i - 1)\}]. \tag{4.6}$$

Proof outline. In computing the priorities we essentially account for the potential loss in revenue if we defer the execution of a job to a later time instant. The highest priority job is that job that will result in the maximum loss if its execution were to be deferred and its deadline were to expire as a consequence of the deferral. It becomes essential to compute the expected change in revenue, $V_{\mathbf{f},i}(l_i) - V_{\mathbf{f},i}(l_i - 1)$ before we can determine the priority of a job. The rest of this section is dedicated to a discussion on how we can recover this quantity.

To understand the long-run average reward obtained from a particular class of workload, we consider the evolution of the queue $\{L_i(t), t \in \mathbb{R}^+\}$ with initial condition $L_i(0) = l_i$ and being awarded a fraction f_i of processing time. The queue

length will evolve as a birth-death process with birth rate r_i and death rate $s_i f_i + l d_i$ at time t with $l \in \mathbb{Z}^+, l = L_i(t)$.

A scheduling policy that apportions fractional processing to different job streams is guaranteed an average revenue of $f_i s_i v_i$ from stream i as long as queue i is never empty. If we have determined the optimal fractional allocations then a scheduling policy can attain high value by not allowing queues to empty: jobs that provide high revenue and have short deadlines may be preferred. We will, therefore, understand the variation in the emptying time of a queue if a job is processed at time instant t or at a later time instant.

The remainder of the proof is devoted to identifying the quantity $V_{\mathbf{f},i}(l_i) - V_{\mathbf{f},i}(l_i - 1)$.

Proof. The stopping time for the birth-death process $\{L_i(t), t \in \mathbb{R}^+\}$ when the scheduling policy uses fractional allocations defined by the vector $\mathbf{f}$ is defined as

$$\tau_{\mathbf{f},i}(l_i) \quad := \quad \inf\{t \mid t > 0 \textit{ and } L_i(t) = 0\}. \tag{4.7}$$

The expected value obtained from queue i in the interval $[0, \tau_{\mathbf{f},i}(l_i))$ is denoted $\hat{V}_{\mathbf{f},i}(l_i)$. Further, we denote the expectation for the stopping time as

$$\overline{T}_{\mathbf{f},i}(l_i) \quad := \quad E[\tau_{\mathbf{f},i}(l_i)]. \tag{4.8}$$

From standard results concerning Markov Decision Processes [42], we can establish that the 0 state is a regeneration point for the queuing process $\{L_i(t), t \in \mathbb{R}^+\}$. We can then obtain

$$V_{\mathbf{f},i}(l_i) - V_{\mathbf{f},i}(l_i - 1) = \hat{V}_{\mathbf{f},i}(l_i) - \hat{V}_{\mathbf{f},i}(l_i - 1) - [\overline{T}_{\mathbf{f},i}(l_i) - \overline{T}_{\mathbf{f},i}(l_i - 1)] V_{\mathbf{f},i} \quad \forall l_i \in \mathbb{Z}^+, \quad 1 \leq i \leq \mathbb{N}. \tag{4.9}$$

Notice that if we define an alternative stopping time

$$\hat{\tau}_{\mathbf{f},i}(l_i) \quad := \quad \inf\{t \mid t > 0 \textit{ and } L_i(t) = l_i - 1\} \tag{4.10}$$

then $\hat{V}_{\mathbf{f},i}(l_i) - \hat{V}_{\mathbf{f},i}(l_i - 1)$ is the value derived from servicing queue i, which is gov-

erned by the MDP $\{L_i(t), t \in \mathbb{R}^+\}$ with $L_i(0) = l_i$ during the interval $[0, \hat{\tau}_{\mathbf{f},i}(l_i))$.

Also,

$$\overline{T}_{\mathbf{f},i}(l_i) - \overline{T}_{\mathbf{f},i}(l_i - 1) \quad = \quad E[\hat{\tau}_{\mathbf{f},i}(l_i)]. \tag{4.11}$$

We shall now introduce a shadow process $\{\hat{L}_i(t), t \in \mathbb{R}^+\}$ to ease our analysis. This process *shadows* the queueing process $\{L_i(t), t \in \mathbb{R}^+\}$ with some subtle differences. The shadow process is a birth-death process with birth rate r_i and death rate $s_i f_i + (l_i + m)d_i$ in state $(l_i + m), m \in \mathbb{N}$. The death rate is 0 in states where the queue length is less than l_i. The initial state of the shadow process is $\hat{L}_i(0) = l_i$. The shadow process is identical to the original queue length process $\{L_i(t), t \in \mathbb{R}^+\}$ when the queue length is greater than $l_i - 1$ but the shadow process cannot enter the state where the queue length is $l_i - 2$. The shadow process has as its regeneration point the state $l_i - 1$ and the reward derived from the shadow process per unit time is

$$\tilde{V}_{\mathbf{f},i} \quad = \quad \frac{\hat{V}_{\mathbf{f},i}(l_i) - \hat{V}_{\mathbf{f},i}(l_i - 1)}{r_i^{-1} + \overline{T}_{\mathbf{f},i}(l_i) - \overline{T}_{\mathbf{f},i}(l_i - 1)}. \tag{4.12}$$

In the expression for $\tilde{V}_{\mathbf{f},i}$, the numerator represents the reward earned when the original MDP transitions from state l_i to l_{i-1}; the denominator is the expected duration for the shadow process to return to its initial state, i.e., start from the initial state of l_i, transition to state l_{i-1} and then return to state l_i.

From standard results regarding birth-death processes [40] we can obtain the stationary distribution for $\{\hat{L}_{\mathbf{f},i}(t), t \in \mathbb{R}^+\}$ as

$$\hat{\Pi}_i(l) = \begin{cases} r_i^{l-l_i+1} \left\{ \frac{\Pi_0(r_i, s_i f_i + (l_i - 1)d_i, d_i)}{\Pi_{m=l_i}^{l}(s_i f_i + m d_i)} \right\}, & l \geq l_i - 1 \\ 0, & l \leq l_i - 2 \end{cases} \tag{4.13}$$

The value obtained per unit time for the shadow process, which does not earn any revenue in state $l_i - 1$, is given by

$$v_i s_i f_i (1 - \hat{\Pi}_i(l_i - 1)) \quad = \quad v_i s_i f_i [1 - \Pi_0(r_i, s_i f_i + (l_i - 1)d_i, d_i)]. \tag{4.14}$$

Further, we can use (4.3) and (4.12) to infer that

$$\frac{(\hat{V}_{\mathbf{f},i}(l_i) - \hat{V}_{\mathbf{f},i}(l_i-1))}{r_i^{-1} + \overline{T}_{\mathbf{f},i}(l_i) - \overline{T}_{\mathbf{f},i}(l_i-1)} \quad = \quad v_i s_i f_i [1 - \Pi_0(r_i, s_i f_i + (l_i-1)d_i, d_i)] \tag{4.15}$$

and that

$$\frac{(\overline{T}_{\mathbf{f},i}(l_i) - \overline{T}_{\mathbf{f},i}(l_i-1))}{r_i^{-1} + \overline{T}_{\mathbf{f},i}(l_i) - \overline{T}_{\mathbf{f},i}(l_i-1)} \quad = \quad 1 - \Pi_0(r_i, s_i f_i + (l_i-1)d_i, d_i). \tag{4.16}$$

We can now combine (4.9), (4.14), (4.15) and (4.16) to conclude that

$$V_{\mathbf{f},i}(l_i) - V_{\mathbf{f},i}(l_i-1) \quad = \quad \frac{[v_i s_i f_i][\Pi_0(r_i, s_i f_i, d_i)]}{[s_i f_i + l_i d_i][\Pi_0(r_i, s_i f_i + l_i d_i, d_i)]}. \tag{4.17}$$

Finally, we use (4.17) in (4.6) to complete the theorem.

□

4.4 Empirical Evaluation

Having described the structure of a policy for job selection to maximize rewards, we shall now describe simulation results that compare the performance of our policy with other approaches.

Before elaborating on empirical evaluation, we emphasize that it is extremely difficult to exhaustively evaluate, via simulation, different scheduling policies, especially when rewards can be assigned arbitrarily. The proof that Policy Z can yield strong, and increased, revenue (Theorem 4) is what should suggest the "goodness" of the policy. The empirical evaluations are only indicative of the general applicability of that result.

4.4.1 Comparison with Stochastic Dynamic Programming (SDP)

Optimal solutions to the scheduling problem of interest can be recovered using stochastic dynamic programming [46]. Stochastic dynamic programming is, however, computationally expensive and is not practical for most applications. For a simple workload with at most *two* task streams it is computationally feasible to resort to SDP; we used this case to compare the performance of the proposed policy

with the optimal policy.

We begin by making two comparisons:

1. Optimal fractional allocation (FAP) vs. Policy Z, and

2. Policy Z vs. the optimal policy via SDP.

For these comparisons we used many task streams, and we present the results from a representative set of simulation runs (parameters in Table 4.1). Each run consisted of two task streams, and the simulations were performed for $9,00,000$ time steps. Each task stream had the same average inter-arrival time of 350 time units, and the revenue earned for every job of task stream 2 was 1.0, i.e., $v_2 = 1.0$. We also kept the same mean deadline for each task stream, $D = D_1 = D_2$. For some simulation runs the mean execution time was longer than the mean deadline, making scheduling decisions even harder.

Experiment	D	e_1	e_2	v_1	f_1^*
E_1	1000	600	600	1.0	0.50
E_2	1000	620	725	1.1	0.54
E_3	1000	580	790	1.2	0.57
E_4	1000	545	855	1.3	0.61
E_5	1000	520	925	1.4	0.64
E_6	1000	500	1010	1.5	0.67
E_7	500	610	735	1.1	0.55
E_8	500	530	900	1.3	0.63
E_9	500	475	1110	1.5	0.70
E_{10}	250	590	765	1.1	0.56
E_{11}	250	495	1020	1.3	0.67
E_{12}	250	435	1430	1.5	0.77
E_{13}	165	575	785	1.1	0.58
E_{14}	165	465	1170	1.3	0.72
E_{15}	165	400	2000	1.5	0.84

Table 4.1: Task Stream Parameters to Compare the Performance of the Proposed Policy with Other Policies (Two Task Streams)

We describe our results for each experiment (Figure 4.1). The optimal fractional allocation is described with other parameters (Table 4.1). Recall that $f_2^* =$

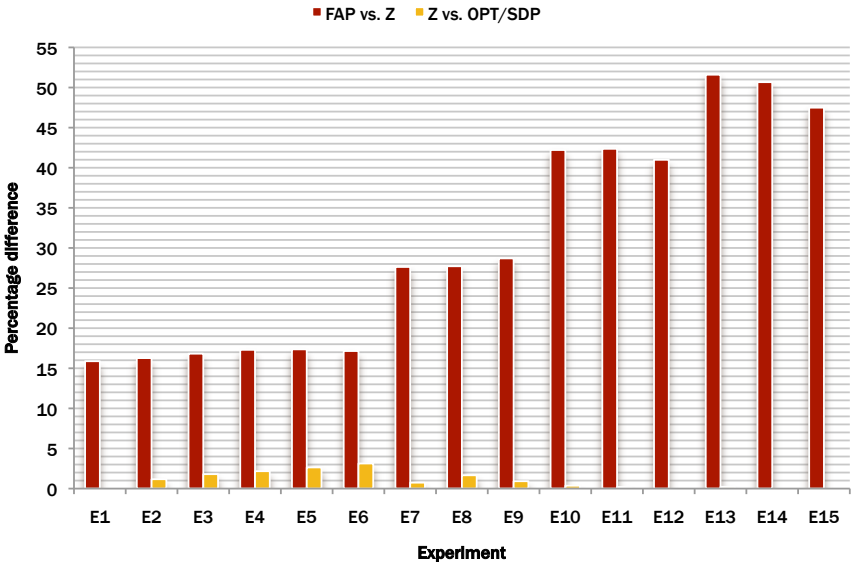

Figure 4.1: Performance of Policy Z Compared with the Optimal Fractional Policy and SDP

$1 - f_1^*$. Policy Z clearly improves over FAP; the *percentage improvement* in average revenue is at least 15% (red bars in the graph). We compute the percentage improvement as follows: If V_Z was the revenue accrued by Policy Z at the end of an experiment and if V_{FAP} was the reward accrued using FAP, then the percentage improvement is $100 \times \frac{V_Z - V_{FAP}}{V_{FAP}}$.

In comparison to the optimal policy recovered using SDP, we determined the *loss in average revenue* (percentage loss = $100 \times \frac{V_{SDP} - V_Z}{V_{SDP}}$) using policy Z (yellow bars); the maximum loss was not more than 4%. This confirms the dramatic improvement that can be obtained over the FAP and indicates that the suggested policy has a performance that is very close to the optimal SDP policy. The performance of Policy Z improves when the rate of deadline misses increases.

4.4.2 Comparison with ROBUST

Baruah and Haritsa developed the ROBUST scheduler [6] for achieving near-optimal performance during overload for a specific class of systems where

- The value of a job is equal to its execution length, and
- Each job has a slack of at least s, i.e., $\frac{D_i}{e_i} \geq s$.

The performance of the ROBUST scheduler is near-optimal in the sense that it can, asymptotically, match the performance of the optimal online scheduling policy for the mentioned class of systems. They showed that the best performance that an online scheduler can guarantee is an EPU of $\lceil s \rceil / (\lceil s \rceil + 1)$ and that the ROBUST scheduler guarantees an EPU that is at most $2/s(s+1)$ fractionally off from the optimum [6].

We provide a brief description of the ROBUST scheduler before detailing some empirical comparisons between the Policy Z and ROBUST. The ROBUST scheduler partitions an overloaded interval into an even number of contiguous phases (Phases-$1, \ldots, 2a$). The length of each even numbered phase is equal to a $1/(s-1)$ fraction of the length of the preceding odd numbered phase. At the start of an odd phase, the algorithm selects the longest eligible job and executes it non-preemptively. This job may have been executed in the previous even numbered phase; the length of the odd numbered phase is equal to the execution time remaining for that job. An odd phase concludes with the termination of the chosen job. During an even numbered phase, the scheduler selects a job with maximum length; this job may be preempted if another job arrives with longer execution length.

To compare Policy Z with the ROBUST scheduler, we used several simulations. For two sets of simulated runs, we chose a fixed slack factor of 2; for the other two sets of runs we chose a slack factor of 4. Each simulated run lasted 1,000,000 time units and involved four task streams. The execution time for jobs belonging to the same task stream were drawn from the same exponential distribution (the mean execution times for the four task streams were 50, 100, 200 and 400 respectively); the deadline for each job was set based on the slack factor. For simplicity we chose the same arrival rate for all streams; based on the desired workload intensity the arrival rate was determined.We did perform a variety of simulation studies with different

arrival rates for different task streams. The performance of the scheduling policies when the arrival rates for different streams are different is similar to the results reported in this work. Only Policy Z is concerned with task streams; the ROBUST scheduler simply schedules on a job-by-job basis. The reward for completing a job successfully was equal to the execution time for that task stream. We do not intend this empirical analysis to be exhaustive but merely indicative of the benefits of using stochastic approximation to derive scheduling policies. For each data point, we averaged 50 independent simulation runs and compared the behavior of the two policies.

We found that Policy Z outperformed ROBUST in all scenarios (Figures 4.2 and 4.3). Policy Z is not clairvoyant, but the awareness of potential future arrivals enables it to make better decisions. With a slack factor of 2 ($s = 2$), we were able to improve the per-time step rewards in excess of 15% in some cases. When the slack factor increases ($s = 4$), Policy Z was able improve revenue per time step but the increases are smaller. When the slack factor is high, most policies will be able to recover from a poor decision and still generate near-optimal revenue.

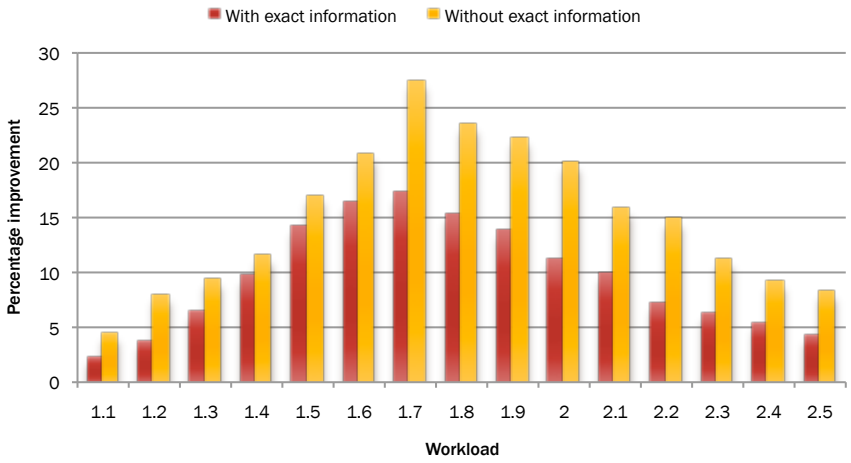

Figure 4.2: Performance of Policy Z Compared with the ROBUST Policy when Slack Factor is 2

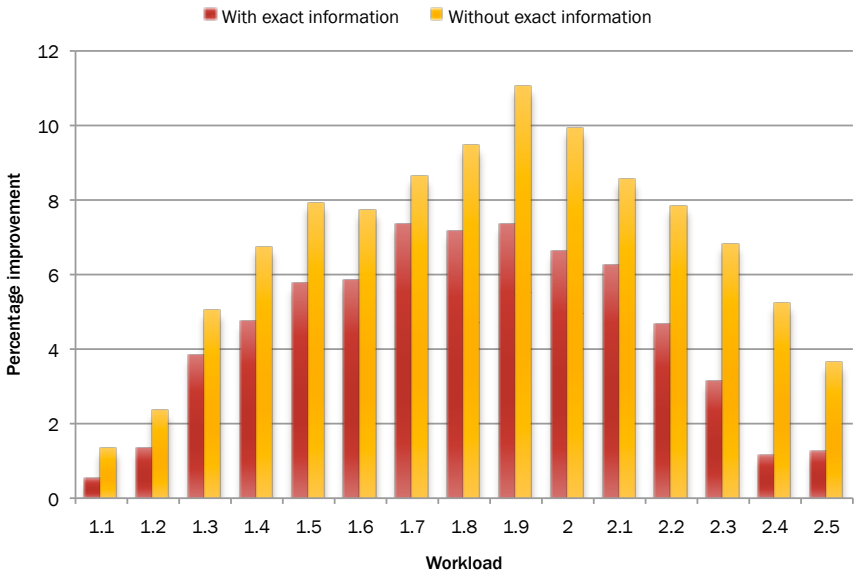

Figure 4.3: Performance of Policy Z Compared with the `ROBUST` Policy when the Slack Factor is 4

The `ROBUST` scheduler requires accurate knowledge of the execution times of jobs and their deadlines. Policy Z is obtained via stochastic approximation and is more tolerant of errors in the parameters. When the `ROBUST` scheduler is only provided with the mean execution time for a job its performance drops significantly and the improvement noticed by using Policy Z is more pronounced. (The red bars in Figures 4.2 and 4.3 are based on the `ROBUST` scheduler using exact information; the orange bars are based on approximate information.)

Another observation is that when the extent of overload is small, both policies perform equally well (or equally poorly). Similarly, when the system experiences heavy overload, most choices are equally good and the two policies have smaller differences.

4.4.3 Comparison with REDF

The ROBUST scheduler is targeted at systems with known slack factors and with a job's value being equal to its execution time. The policy we have developed, however, is also suited to arbitrary reward assignments and to situations when jobs do not have a guaranteed slack.

To understand the performance of Policy Z under general workloads we compared its performance with the performance offered by the Robust EDF heuristic [14, 15]. The REDF policy is identical to EDF when the system is not overloaded. Whenever a job arrives a check is performed to determine if the system is overloaded. (If tasks are scheduled using EDF and $e_i/D_i \leq 1$ then the system is not overloaded.) When an overload is detected, the least value task that can prevent the system from being overloaded is removed from the queue of pending jobs to a reject queue.[2] If some job completes ahead of time then jobs from the reject queue whose deadlines have not expired may be brought back to the pending queue. Buttazzo, Spuri and Sensini showed that REDF is well behaved during overloads [15], and we used additional simulations to understand the performance of REDF and Policy Z, and to contrast the two approaches.

For these simulations, we used the task streams similar to those in our comparisons with ROBUST. For each run we used four task streams, S_1, S_2, S_3, S_4, with mean execution times of 150, 100, 200 and 400 respectively. The deadlines for jobs of the four task streams were drawn from exponential distributions with mean 600, 800, 1600 and 3200 respectively. The arrival rate was chosen to generate the required workload. Similar to the previous evaluation, each stream had the same arrival rate.

We compared the performance of REDF with Policy Z under two reward models:

- The rewards associated with jobs of the four streams were 150, 300, 400 and 200 respectively. These were chosen to represent a *random* ranking of task streams in terms of value.

- The reward associated with each stream was inversely related to the mean

[2] This policy can be modified and a smart search strategy might remove multiple jobs of low value to prevent overload. We have not implemented this approach in our evaluation.

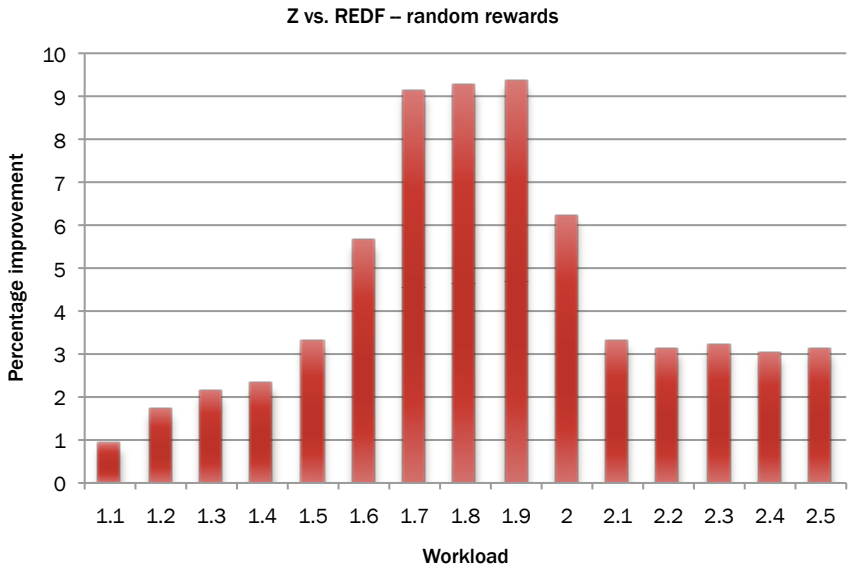

Figure 4.4: Performance of Policy Z Compared with the `REDF` Policy (Random Rewards)

deadline for that stream, i.e., shorter the deadline greater the reward. The rewards associated with $S_1, \ldots, S_4$ were 450, 300, 200 and 100 respectively. This reward model was intended to be approximately *linear* in terms of job deadlines.

We note that Policy Z results in measurable improvements in revenue when compared with `REDF` using both the *random* (Figure 4.4) and the *linear* (Figure 4.5) reward models. The linear reward model indicates greater differences because `REDF` has to choose to drop jobs that may yield high rewards (because higher rewards are connected to higher utilization, and one job providing high reward may dropped in place of multiple jobs that jointly yield a smaller reward) to ensure that other jobs meet their deadlines.

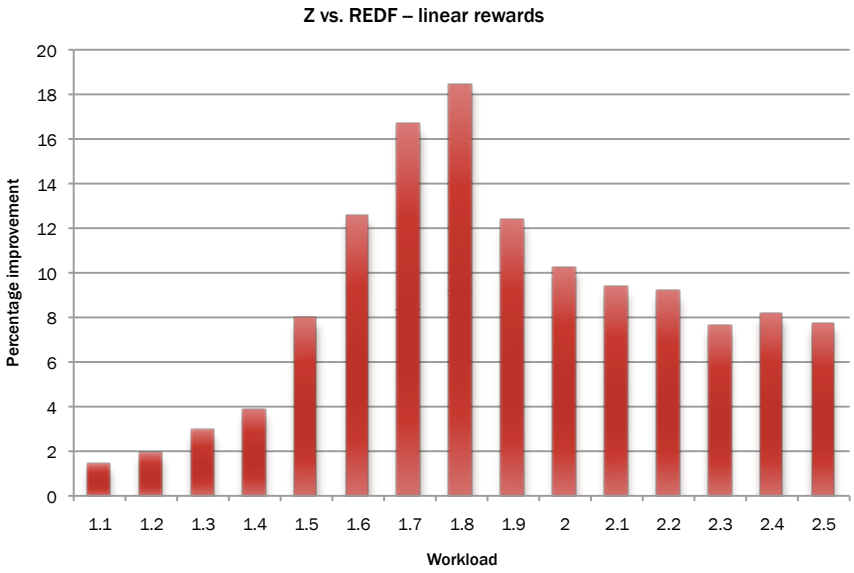

Figure 4.5: Performance of Policy *Z* Compared with the REDF Policy (Linear Rewards)

4.4.4 Discussion

On the basis of the three different comparisons (with SDP, with ROBUST, with REDF), we were able to ascertain the uniformly improved performance that the proposed scheduling approach (Policy *Z*) is able to offer. These comparisons strongly indicate that using knowledge of future workload does increase the revenue one can earn. The improvement in revenue can be at least 10%, and is likely higher when perfect information regarding the temporal requirements of jobs is not available. The improvements in revenue obtained using Policy *Z* diminish when the system is extremely overloaded; this hints at the possibility that most scheduling decisions are likely to be reasonable in those situations.

We speculate that if Policy *Z* is near optimal (as is the case when there are two task streams – see Figure 4.1) then other scheduling policies (e.g., ROBUST, REDF) are also likely to be only about 20 to 25% away from optimality (even less in some cases) in practice, and that is an encouraging result concerning the

practical applicability of those policies.

The structure of the priority index for Policy Z is intuitive and can form the basis for obtaining good scheduling heuristics even when workload might not conform to simple probability distributions.

Implementation considerations. Policy Z requires a priority for each class of requests, and this dynamic priority depends on the length of the corresponding queues. It is possible to compute the priorities at different queue lengths off-line and use a table lookup to identify the priorities of tasks online. This makes the proposed policy easy to implement. We also need to identify the optimal fractional allocation policy, and this is also an offline operation. Identifying the optimal fractional allocation is an optimization problem in itself and we use a search over the space of possible allocations to determine the optimal allocation. This is feasible when the number of service classes is limited. It is likely that some sub-optimal initial allocations may not affect the behavior of Policy Z significantly but this notion requires further study.

Chapter 5

Conclusions and Future Improvements

Overload in certain soft real-time systems (such as Internet-based services) is often unavoidable because the costs of provisioning for peak load are significantly greater than the costs of handling typical load. In such systems, service providers need to provide the best possible service to customers who demand higher quality of service and are willing to pay more for better QoS.

In this thesis, we have addressed the issue of handling overloads in Real-time Systems, and have taken two different approaches. We initially proposed a way of find the conditions causing overload of the system, and avert it by provisioning extra resources, or dropping requests. Subsequently, we proposed a fractional allocation policy based on stochastic approximation to make a decision to drop requests, while maximizing the revenue at the same time.

5.1 Summary of the Contributions

5.1.1 On-line Scheduling Policy to Detect Overload

The problem of finding the WCRT for Rate Monotonic Scheduling is a challenging problem. The past and current efforts on Rate Monotonic Analysis have been attempted at finding the best match of accuracy and computational complexity, but

not when there is no information about the workload. Our work shows that it is possible to find an upper bound on WCRT for any possible distribution of resources among the task streams, with reduced complexity. We showed the hardness of the problem when Stochastic Analysis is done, even when PTAS are used to minimize the complexity. By using mathematical optimization techniques, we proved that the maximum WCRT can be determined with lesser complexity, but it is still not applicable for embedded devices with little processing power.

In order to address the issue of complexity in the proposed WCRT analysis, we provided heuristics to minimize the problem size at different stages of the analysis. Therefore, we showed that by using polynomial number of constraints, the Linear Program converges to a suboptimal solution faster. In order to bound the worst-case accuracy of the approximation technique, we established worst-case limits on the possible solutions of the Linear Program formulation. Then, we followed an approximation scheme proposed by Fisher and Baruah [20] to restrict the search space of the solution, instead of incrementally checking all possible future arrivals of the task. WCRT gives a good indication of the possible overload to the system administrator, depending on the arrival of the requests and allows for handling overloads efficiently.

5.1.2 Overload Management to Improve Rewards by Stochastic Approximation

When the resources available are stringent, we made assumptions on the workload characteristics and presented a scheduling policy for handling overload conditions while improving the revenue earned. The policy that we present, Policy Z, is based on stochastic approximation, using the information about future job arrivals. It is an on-line scheduling policy and therefore, approximate information about future workload is sufficient to make meaningful decisions. Empirical evidence suggests that Policy Z has an excellent performance when compared with other scheduling policies for value maximization in the presence of overload.

5.2 Potential Future Improvements

In large-scale systems having multiple processing units to serve the requests from different clients are prevalent. Schedulability analysis, and fractional allocation are far more challenging in such scenarios. With more cores being added in the same processor in embedded systems, makes it more compelling to have the schedulability analysis extended for multiprocessor /multicore systems. The problem of scheduling of these resources, based on allocating jobs to different cores(bin packing) is widely studied, and estimated to be a *NP Complete* problem. Therefore, identification of WCRT for *dynamic task assumptions* is an interesting area for research.

Our scheduling policy, using stochastic approximation is sufficiently general and can be used in *multiprocessor* systems as well. However, we have restricted the discussion in this article to *uniprocessor* systems, but it is possible to use the policy in a system with m processors by selecting the top m jobs based on their priority indices.

Even though we make some minor assumptions in our formulation, it can be removed and the model can be extended to such cases. In the case of Schedulability analysis, the assumptions such as periodic arrival of tasks, task independence, lack of release jitter induced delays, etc. can be eliminated, since the Linear Program formulation is fairly generalized. Similarly, we make some assumptions about job arrival rates and deadlines, we believe that the approach of generating an initial policy and then improving upon that policy (as we do with the optimal fractional allocation policy and Policy Z) is a useful tool for decision making in real-time systems that can be generalized and applied to other problems as well.

Bibliography

[1] K. Albers and F. Slomka. An event stream driven approximation for the analysis of real-time systems. In *ECRTS*, pages 187–195, 2004. → pages 14

[2] Amazon.com. Amazon web services. http://www.amazon.com/, May 2008. → pages 1

[3] H. Aydin, R. Melhem, D. Mossé, and P. Mejía-Alvarez. Optimal reward-based scheduling for periodic real-time tasks. *IEEE Transactions on Computers*, 50(2):111–130, February 2001. ISSN 0018-9340. doi:http://dx.doi.org/10.1109/12.908988. URL http://portal.acm.org/ft_gateway.cfm?id=365334&type=external&coll=Portal&dl=GUIDE&CFID=68485383&CFTOKEN=40244428. → pages 45

[4] S. Baruah, G. Koren, B. Mishra, A. Raghunathan, L. Rosier, and D. Shasha. On-line scheduling in the presence of overload. In *Proceedings of the Symposium on Foundations of Computer Science*, pages 100–110, October 1991. ISBN 0-8186-2445-0. → pages 46, 47

[5] S. Baruah, G. Koren, D. Mao, B. Mishra, A. Raghunathan, L. Rosier, D. Shasha, and F. Wang. On the competitiveness of on-line real-time task scheduling. *Real-Time Systems*, 4(2):125–144, May 1992. ISSN 0922-6443. doi:http://dx.doi.org/10.1007/BF00365406. → pages 46

[6] S. K. Baruah and J. R. Haritsa. Scheduling for overload in real-time systems. *IEEE Transactions on Computers*, 46(9):1034–1039, September 1997. ISSN 0018-9340. doi:http://dx.doi.org/10.1109/12.620484. URL http://portal.acm.org/ft_gateway.cfm?id=265217&type=external&coll=Portal&dl=GUIDE&CFID=68485383&CFTOKEN=40244428. → pages 46, 47, 58

[7] E. Bini. *The Design Domain of Real-Time Systems*. PhD thesis, Scuola Superiore S.Anna, Pisa, 2004. → pages 41

[8] E. Bini and G. C. Buttazzo. The space of rate monotonic schedulability. In *IEEE Real-Time Systems Symposium*, pages 169–178, 2002. → pages 37, 39

[9] E. Bini and G. C. Buttazzo. Biasing effects in schedulability measures. In *ECRTS*. IEEE Computer Society, 2004. ISBN 0-7695-2176-2. → pages 41

[10] E. Bini, G. C. Buttazzo, and G. Buttazzo. Rate monotonic analysis: The hyperbolic bound. *IEEE Trans. Computers*, 52(7):933–942, 2003. → pages 8, 13, 44

[11] E. Bini, T. Nguyen, P. Richard, and S. Baruah. A response-time bound in fixed-priority scheduling with arbitrary deadlines. *IEEE Trans. Computers*, 58:279–286, 2009. → pages 42

[12] E. A. Brewer. Lessons from giant-scale services. *IEEE Internet Computing*, 5(4):46–55, Jul./Aug. 2001. → pages 1, 2

[13] G. Buttazzo, G. Lipari, L. Abeni, and M. Caccamo. *Soft Real-Time Systems: Predictability vs. Efficiency*. Springer, 2005. → pages 45

[14] G. C. Buttazzo and J. A. Stankovic. RED: a robust earliest deadline scheduling algorithm. In *Proceedings of the International Workshop on Responsive Computing Systems*, pages 100–111, September 1993. → pages 47, 61

[15] G. C. Buttazzo, M. Spuri, and F. Sensini. Value vs. deadline scheduling in overload conditions. In *Proceedings of the IEEE Real-Time Systems Symposium*, pages 90–99, December 1995. → pages 47, 61

[16] D. Chen, A. K. Mok, and T.-W. Kuo. Utilization bound re-visited. In *RTCSA*, pages 295–302, 1999. → pages 8, 13, 18

[17] J.-Y. Chung, J. W. S. Liu, and K.-J. Lin. Scheduling periodic jobs that allow imprecise results. *IEEE Transactions on Computers*, 39(9):1156–1174, September 1990. ISSN 0018-9340. doi:http://dx.doi.org/10.1109/12.57057. URL http://portal.acm.org/ft_gateway.cfm?id=102826&type=external&coll=Portal&dl=GUIDE&CFID=68484521&CFTOKEN=56294152. → pages 45

[18] R. Davis and A. Burns. Response time upper bounds for fixed priority real-time systems. In *Real-Time Systems Symposium (RTSS)*, pages 407–418. IEEE Computer Society, December 2008. → pages 14, 41

[19] J. K. Dey, J. Kurose, and D. Towsley. On-line scheduling policies for a class of IRIS (increasing reward with increasing service) real-time tasks. *IEEE*

Transactions on Computers, 45(7):802–813, July 1996. doi:http://dx.doi.org/10.1109/12.508319. URL http://portal.acm.org/ft_gateway.cfm?id=627153&type=external&coll=Portal&dl=GUIDE&CFID=68485383&CFTOKEN=40244428. → pages 45

[20] N. Fisher and S. Baruah. A polynomial-time approximation scheme for feasibility analysis in static-priority systems with arbitrary relative deadlines. In *In Proceedings of the 13th International Conference on Real-Time Systems*, pages 117–126. IEEE Computer Society Press, 2005. → pages 13, 14, 38, 66

[21] B. Hajek. On the competitiveness of on-line scheduling of unit-length packets with hard deadlines in slotted time. In *Proceedings of the Conference on Information Sciences and Systems*, pages 434–439, March 2001. → pages 47

[22] C.-C. Han and H.-Y. Tyan. A better polynomial-time schedulability test for real-time fixed-priority scheduling algorithm. In *IEEE Real-Time Systems Symposium*, pages 36–45, 1997. → pages 13

[23] X. S. Hu and G. Quan. Fast performance prediction for periodic task systems. In *CODES*, pages 72–76, 2000. → pages 22

[24] G. Koren and D. Shasha. D^{over}: an optimal on-line scheduling algorithm for overloaded uniprocessor real-time systems. *SIAM Journal of Computing*, 24 (2):318–339, April 1995. → pages 46, 47

[25] G. Koren and D. Shasha. Skip^{Over}: algorithms and complexity for overloaded systems that allow skips. In *Proceedings of the IEEE Real-Time Systems Symposium*, pages 110–119, December 1995. ISBN 0-8186-7337-0. URL http://portal.acm.org/ft_gateway.cfm?id=828929&type=external&coll=Portal&dl=GUIDE&CFID=29442874&CFTOKEN=47928360. → pages 46

[26] E. Koutsoupias and C. H. Papadimitriou. Beyond competitive analysis. *SIAM Journal of Computing*, 30(1):300–317, January 2000. ISSN 0097-5397. doi:http://dx.doi.org/10.1137/S0097539796299540. URL http://portal.acm.org/ft_gateway.cfm?id=587029&type=external&coll=Portal&dl=GUIDE&CFID=29061345&CFTOKEN=21883824. → pages 47

[27] T.-W. Lam, T.-W. J. Ngan, and K.-K. To. Performance guarantee for EDF under overload. *Journal of Algorithms*, 52(2):193–206, August 2004. ISSN 0196-6774. doi:http://dx.doi.org/10.1016/j.jalgor.2003.10.004. → pages 47

[28] C.-G. Lee, L. Sha, and A. Peddi. Enhanced utilization bounds for qos management. *IEEE Trans. Computers*, 53(2):187–200, 2004. → pages 8, 14, 18, 20, 21, 22, 23, 44

[29] J. P. Lehoczky. Fixed priority scheduling of periodic task sets with arbitrary deadlines. In *IEEE Real-Time Systems Symposium*, pages 201–213, 1990. → pages 10

[30] J. P. Lehoczky. Real-time queuing theory. In *Proceedings of the IEEE Real-Time Systems Symposium*, pages 186 – 195, Dec. 1996. → pages 48

[31] J. P. Lehoczky. Real-time queuing network theory. In *Proceedings of the IEEE Real-Time Systems Symposium*, pages 58–67, Dec. 1997. → pages 10

[32] J. P. Lehoczky, L. Sha, and Y. Ding. The rate monotonic scheduling algorithm: Exact characterization and average case behavior. In *IEEE Real-Time Systems Symposium*, pages 166–171, 1989. → pages 9, 13, 16, 18, 20, 39, 41, 42

[33] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM*, 20(1):46–61, 1973. → pages 8, 9, 13, 17, 44

[34] J. W.-S. Liu. *Real-time systems*. Prentice Hall, 2000. ISBN 978-0-13-099651-0. → pages 7

[35] R. MacKenzie, D. Hands, and T. O'Farrell. Packet handling strategies to improve video qos over 802.11e wlans. In *PIMRC*, pages 1173–1177, 2009. → pages 15

[36] Y. Manabe and S. Aoyagi. A feasibility decision algorithm for rate monotonic and deadline monotonic scheduling. *Real-Time Systems*, 14(2): 171–181, 1998. → pages 39

[37] A. K. Mok and D. Chen. A multiframe model for real-time tasks. *IEEE Trans. Software Eng.*, 23(10):635–645, 1997. → pages 14

[38] T. Nguyen, P. Richard, and E. Bini. Approximation techniques for response-time analysis of static-priority tasks. *Real-Time Systems*, 43: 147–176, 2009. ISSN 0922-6443. → pages 8, 13, 14, 18, 41

[39] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall, 1982. ISBN 0-13-152462-3. → pages 7, 11

[40] A. Papoulis and S. U. Pillai. *Probability, Random Variables and Stochastic Processes*. McGraw-Hill, 4 edition, 2002. → pages 50, 54

[41] D.-W. Park, S. Natarajan, A. Kanevsky, and M. J. Kim. A generalized utilization bound test for fixed-priority real-time scheduling. In *RTCSA*, pages 73–, 1995. → pages 8, 18, 19

[42] M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley and Sons, New York, 5 edition, 2005. → pages 53

[43] R. Rajkumar, C. Lee, J. P. Lehoczky, and D. Siewiorek. A resource allocation model for QoS management. In *Proceedings of the IEEE Real-Time Systems Symposium*, pages 298–307, Dec. 1997. → pages 46

[44] R. Rajkumar, C. Lee, J. P. Lehoczky, and D. Siewiorek. Practical solutions for QoS-based resource allocation. In *Proceedings of the IEEE Real-Time Systems Symposium*, pages 296–306, Dec. 1998. → pages 46

[45] P. Richard and J. Goossens. Approximating response times of static-priority tasks with release jitters. In *In the WiP segment of the 18th Euromicro Conference on Real-Time Systems*, 2006. → pages 41

[46] S. M. Ross. *Introduction to stochastic dynamic programming*. Academic Press, 1995. → pages 55

[47] D. Seto, J. P. Lehoczky, and L. Sha. Task period selection and schedulability in real-time systems. In *Proceedings of the IEEE Real-Time Systems Symposium*, pages 188–198, Dec. 1998. → pages 45

[48] M. Sjödin and H. Hansson. Improved response-time analysis calculations. In *IEEE Real-Time Systems Symposium*, pages 399–408, 1998. → pages 18, 41, 42

[49] J. A. Stankovic, M. Spuri, M. D. Natale, and G. C. Buttazzo. Implications of classical scheduling results for real-time systems. *Computer*, 28(6):16–25, June 1995. ISSN 0018-9162. doi:http://dx.doi.org/10.1109/2.386982. URL http://portal.acm.org/ft_gateway.cfm?id=620236&type=external&coll=Portal&dl=GUIDE&CFID=28250344&CFTOKEN=38174394. → pages 47