Scheduling on Firm Real-Time Systems

by

Yuanxin Wang

A thesis submitted to
the Faculty of Graduate Studies and Research
in partial fulfillment of
the requirements for the degree of Master of Science

School of Computer Science
Carleton University,
Ottawa, Ontario, Canada

September, 2004

Copyright 2004, Yuanxin Wang

Library and
Archives Canada

Bibliothèque et
Archives Canada

Published Heritage
Branch

Direction du
Patrimoine de l'édition

395 Wellington Street
Ottawa ON K1A 0N4
Canada

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence

ISBN: 0-612-97486-3

Our file Notre référence

ISBN: 0-612-97486-3

The author has granted a non-exclusive license allowing the Library and Archives Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque et Archives Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

In compliance with the Canadian Privacy Act some supporting forms may have been removed from this thesis.

Conformément à la loi canadienne sur la protection de la vie privée, quelques formulaires secondaires ont été enlevés de cette thèse.

While these forms may be included in the document page count, their removal does not represent any loss of content from the thesis.

Bien que ces formulaires aient inclus dans la pagination, il n'y aura aucun contenu manquant.

Canada

Abstract

This thesis focuses on firm real-time systems in which it is acceptable to allow a low percentage of task instances to miss their deadlines. Three scheduling algorithms, the Pre-Execution Estimation Based Scheduling (PES-A and PES-B) Algorithms and the Tardy Task First (TTF) algorithm are proposed for a uni-processor based firm real-time system.

A number of experiments were conducted to evaluate the performance of these algorithms based on four performance indices: 1) guaranteed ratio for meeting deadlines, 2) average ratio for meeting deadlines, 3) inter-task fairness metric and 4) useful CPU utilization achieved. The experimental results indicate that, for most of the workloads experimented with, the proposed algorithms performed better than RMS, primarily at higher system loads.

Acknowledgments

This work is inspired and supervised by Professor Shikharesh Majumdar. No words can fully express my gratefulness to him. His knowledge on real-time systems, insights in technical issues and ability to grasp the essential of a problem guided the direction on every stage of the research. Without his advice and help, I can not imagine having this work successfully done.

The support of Nortel Networks during the initial part of the research is gratefully acknowledged. Thanks to my husband Feng who never failed to be there with me. Also to little Tao and Paul who always cheer me up. Thanks to all others who have helped me on my research.

Table of Contents

<i>Abstract</i>	ii
<i>Acknowledgments</i>	iii
List of Figures	viii
List of Tables	xiii
<i>Chapter 1 Introduction</i>	1
1.1 Background and Motivation	1
1.1.1 Real-time System and Scheduling	1
1.1.2 Hard Real Time System and Rate Monotonic Scheduling	3
1.1.3 Soft/Firm Real-time System Scheduling	7
1.2 Goals of This Research	9
1.3 Thesis Contributions	10
1.4 Organization	11
<i>Chapter 2 Proposed Scheduling Algorithms</i>	12
2.1 Input Parameters for the System	12
2.2 Performance Metrics	13
2.3 Pre-Execution Estimation Based Scheduling Algorithms	14
2.3.1 General Introduction	14
2.3.2 Discussion of Pre-Execution Estimation Based Scheduling Algorithms	16
2.3.3 Pre-Execution Estimation Based Scheduling Algorithm A	20
2.3.4 Pre-Execution Estimation Based Scheduling Algorithm B	29

2.4	Tardy Task First Algorithm	33
<i>Chapter 3 Experimental Environment</i>		35
3.1	Design and Implementation of the Prototype Environment	35
3.1.1	Task Spawning Routine	36
3.1.2	Monitoring and Synchronization Routine	37
3.1.3	Periodic Interrupt Routine	37
3.2	System Environment	38
3.3	Verification of Measurement	39
3.4	System Characteristics	39
3.4.1	Number of Tasks in the System	39
3.4.2	Period for Each Task	39
3.4.3	Computation Time of Each Task	40
3.4.4	Super-Period of the Estimation Task	41
3.4.5	Duration of an Experiment	42
3.4.6	Threshold Ratio for Meeting Deadlines of Each Task	43
3.5	Performance of Proposed Algorithms	44
3.6	Analysis of Overheads	47
3.6.1	Analysis of Overheads	47
3.6.2	Experiments on Overhead	48
<i>Chapter 4 Results of Experiments</i>		50
4.1	Same Utilization of Resources by Tasks	50
4.1.1	The Ratio of Task Periods between 1 and 2 in a Non-overloaded System	50
4.1.2	The Ratio of Periods between 1 and 2 in an Overloaded System	58

4.1.3 The Ratio of Longest and Shortest Task Periods No Less Than 2 in a Non-overloaded System	65
4.1.4 The Ratio of Longest and Shortest Task Periods No Less Than 2 in an Overloaded System	67
4.1.5 Equal Periods for All Tasks in a Non-overloaded System	68
4.1.6 Equal Periods for All Tasks in an Overloaded System	71
4.2 CPU Utilizations Requested by All Tasks Inversely Proportional to Periods of Tasks	71
4.2.1 The Ratio of Task Periods between 1 and 2 in a Non-overloaded System	72
4.2.2 The Ratio of Task Periods between 1 and 2 in an Overloaded System	77
4.3 CPU Utilizations of Tasks Directly Proportional to Periods of Tasks	78
4.3.1 The Ratio of Task Periods between 1 and 2 in a Non-overloaded System	78
4.3.2 The Ratio of Periods between 1 and 2 in an Overloaded System	80
4.4 Analysis of Overhead	80
4.4.1 Overhead for Various CPU Utilization Requested by All Tasks	81
4.4.2 Overhead for Various Numbers of Tasks	82
4.4.3 Overhead for Various Periods of All Tasks	83
4.4.4 Overhead for Various Super-Periods for the Estimation Task	83
4.4.5 Discussion on Overhead	85
4.5 Summary of Results and Insights into System Behavior	85
4.6 Discussion	88
Chapter 5 Conclusions	90
5.1 Summary	90
5.2 Insights into System Performance	92

5.3	Future Work	94
	References	95
	<i>Appendix A Additional Results for Systems with Same CPU Utilizations Requested by Each Task</i>	100
	<i>Appendix B Additional Results for Systems with CPU Utilizations of Tasks Inversely Proportional to Task Periods</i>	105
	<i>Appendix C Additional Results for Systems with CPU Utilizations of Tasks Directly Proportional to Task Periods</i>	114

List of Figures

Figure 2-1 Multiple invocations of task τ_i within a single period of task τ_i	21
Figure 2-2 Case I: Total Overlap	21
Figure 2-3 Case II: Partial Overlap with Same Start Time	22
Figure 2-4 Case II: Partial Overlap with Same End Time	22
Figure 2-5 Case III: Partial Overlap with Different Start Times	22
Figure 2-6 Timing Diagram for PES-A	28
Figure 2-7 Timing Diagram for RMS	28
Figure 2-8 Computing time available by PES-B	30
Figure 4-1 Comparison of Guaranteed Ratios for Meeting Deadlines Achieved by PES-A, PES-B, TTF and RMS on a Non-overloaded System when $1 < T_n/T_1 < 2$	51
Figure 4-2 Comparison of Average Ratios for Meeting Deadlines Achieved by PES-A, PES-B, TTF and RMS on a Non-overloaded System when $1 < T_n/T_1 < 2$	54
Figure 4-3 Comparison of Useful CPU Utilizations Achieved by PES-A, PES-B, TTF and RMS on a Non-overloaded System when $1 < T_n/T_1 < 2$	56
Figure 4-4 Comparison of Inter-task Fairness Metric Achieved by PES-A, PES-B, TTF and RMS on a Non-overloaded System when $1 < T_n/T_1 < 2$	58
Figure 4-5 Comparison of Guaranteed Ratios for Meeting Deadlines Achieved by PES-A, PES-B, TTF and RMS on an Overloaded System when $1 < T_n/T_1 < 2$	59
Figure 4-6 Comparison of Average Ratios for Meeting Deadlines Achieved by PES-A, PES-B, TTF and RMS on an Overloaded System when $1 < T_n/T_1 < 2$	62
Figure 4-7 Comparison of Useful CPU Utilizations Achieved by PES-A, PES-B, TTF and RMS on an Overloaded System when $1 < T_n/T_1 < 2$	63
Figure 4-8 Comparison of Inter-task Fairness Metric Achieved by PES-A, PES-B, TTF and RMS on an Overloaded System when $1 < T_n/T_1 < 2$	64
Figure 4-9 Comparison of Guaranteed Ratios for Meeting Deadlines Achieved by PES-A, PES-B, TTF and RMS on a Non-overloaded System when $T_n/T_1 \geq 2$	66

Figure 4-10 Comparison of Inter-task Fairness Metric Achieved by PES-A, PES-B, TTF and RMS on a Non-overloaded System when $T_n/T_1 \geq 2$	67
Figure 4-11 Comparison of Guaranteed Ratios for Meeting Deadlines Achieved by PES-A, PES-B, TTF and RMS on a Non-overloaded System when Task Periods are Equal to One Another	68
Figure 4-12 Comparison of Average Ratios for Meeting Deadlines Achieved by PES-A, PES-B, TTF and RMS on a Non-overloaded System when Task Periods are Equal to One Another	69
Figure 4-13 Comparison of Useful CPU Utilizations Achieved by PES-A, PES-B, TTF and RMS on a Non-overloaded System when Task Periods are Equal to One Another	70
Figure 4-14 Comparison of Inter-task Fairness Metric Achieved by PES-A, PES-B, TTF and RMS on a Non-overloaded System when Task Periods are Equal to One Another	70
Figure 4-15 Comparison of Guaranteed Ratios for Meeting Deadlines Achieved by PES-A, PES-B, TTF and RMS on a Non-overloaded System when $C/T = O(1/T^6)$	72
Figure 4-16 Comparison of Average Ratios for Meeting Deadlines Achieved by PES-A, PES-B, TTF and RMS on a Non-overloaded System when $C/T = O(1/T^6)$	74
Figure 4-17 Comparison of Average Ratios for Meeting Deadlines Achieved by PES-A, PES-B, TTF and RMS on a Non-overloaded System when $C/T = O(1/T^2)$	74
Figure 4-18 Comparison of Average Ratios for Meeting Deadlines Achieved by PES-A, PES-B, TTF and RMS on a Non-overloaded System when $C/T = O(1/T^4)$	75
Figure 4-19 Comparison of Useful CPU Utilizations Achieved by PES-A, PES-B, TTF and RMS on a Non-overloaded System when $C/T = O(1/T^6)$	76
Figure 4-20 Comparison of Inter-task Fairness Metric Achieved by PES-A, PES-B, TTF and RMS on a Non-overloaded System when $C/T = O(1/T^6)$	77
Figure 4-21 Comparison of Average Ratios for Meeting Deadlines Achieved by PES-A, PES-B, TTF and RMS on a Non-overloaded System when $C/T = O(T^4)$	79
Figure 4-22 Comparison of Useful CPU Utilizations Achieved by PES-A, PES-B, TTF and RMS on a Non-overloaded System when $C/T = O(T^4)$	80
Figure 4-23 Overhead Introduced by the Proposed Algorithms	81
Figure 4-24 Overhead Introduced by Different Proposed Algorithms with Various Number of Tasks	82

Figure 4-25 Overhead Introduced by Different Proposed Algorithms with Various Periods of the Tasks	83
Figure 4-26 Overhead Introduced by Different Proposed Algorithms with Various Super-periods	84
Figure A-1 Comparison of Average Ratios for Meeting Deadlines Achieved by PES-A, PES-B, TTF and RMS on a Non-overloaded System when $T_n/T_1 \geq 2$	100
Figure A-2 Comparison of Useful CPU Utilizations Achieved by PES-A, PES-B, TTF and RMS on a Non-overloaded System when $T_n/T_1 \geq 2$	100
Figure A-3 Comparison of Guaranteed Ratios for Meeting Deadlines Achieved by PES-A, PES-B, TTF and RMS on an Overloaded System when $T_n/T_1 \geq 2$	101
Figure A-4 Comparison of Average Ratios for Meeting Deadlines Achieved by PES-A, PES-B, TTF and RMS on an Overloaded System when $T_n/T_1 \geq 2$	101
Figure A-5 Comparison of Useful CPU Utilizations Achieved by PES-A, PES-B, TTF and RMS on an Overloaded System when $T_n/T_1 \geq 2$	102
Figure A-6 Comparison of Inter-task Fairness Metric Achieved by PES-A, PES-B, TTF and RMS on an Overloaded System when $T_n/T_1 \geq 2$	102
Figure A-7 Comparison of Guaranteed Ratios for Meeting Deadlines Achieved by PES-A, PES-B, TTF and RMS on an Overloaded System when Task Periods are Equal to One Another	103
Figure A-8 Comparison of Average Ratios for Meeting Deadlines Achieved by PES-A, PES-B, TTF and RMS on an Overloaded System when Task Periods are Equal to One Another	103
Figure A-9 Comparison of Useful CPU Utilizations Achieved by PES-A, PES-B, TTF and RMS on an Overloaded System when Task Periods are Equal to One Another	104
Figure A-10 Comparison of Inter-task Fairness Metric Achieved by PES-A, PES-B, TTF and RMS on an Overloaded System when Task Periods are Equal to One Another	104
Figure B-1 Comparison of Guaranteed Ratios for Meeting Deadlines Achieved by PES-A, PES-B, TTF and RMS on a Non-overloaded System when $C/T = O(1/T^2)$	105
Figure B-2 Comparison of Useful CPU Utilizations Achieved by PES-A, PES-B, TTF and RMS on a Non-overloaded System when $C/T = O(1/T^2)$	105
Figure B-3 Comparison of Inter-task Fairness Metric Achieved by PES-A, PES-B, TTF and RMS on a Non-overloaded System when $C/T = O(1/T^2)$	106

Figure B-4 Comparison of Guaranteed Ratios for Meeting Deadlines Achieved by PES-A, PES-B, TTF and RMS on a Non-overloaded System when $C/T = O(1/T^4)$	106
Figure B-5 Comparison of Useful CPU Utilizations Achieved by PES-A, PES-B, TTF and RMS on a Non-overloaded System when $C/T = O(1/T^4)$	107
Figure B-6 Comparison of Inter-task Fairness Metric Achieved by PES-A, PES-B, TTF and RMS on a Non-overloaded System when $C/T = O(1/T^4)$	107
Figure B-7 Comparison of Guaranteed Ratios for Meeting Deadlines Achieved by PES-A, PES-B, TTF and RMS on an Overloaded System when $C/T = O(1/T^2)$	108
Figure B-8 Comparison of Average Ratios for Meeting Deadlines Achieved by PES-A, PES-B, TTF and RMS on an Overloaded System when $C/T = O(1/T^2)$	108
Figure B-9 Comparison of Useful CPU Utilizations Achieved by PES-A, PES-B, TTF and RMS on an Overloaded System when $C/T = O(1/T^2)$	109
Figure B-10 Comparison of Inter-task Fairness Metric Achieved by PES-A, PES-B, TTF and RMS on an Overloaded System when $C/T = O(1/T^2)$	109
Figure B-11 Comparison of Guaranteed Ratios for Meeting Deadlines Achieved by PES-A, PES-B, TTF and RMS on an Overloaded System when $C/T = O(1/T^4)$	110
Figure B-12 Comparison of Average Ratios for Meeting Deadlines Achieved by PES-A, PES-B, TTF and RMS on an Overloaded System when $C/T = O(1/T^4)$	110
Figure B-13 Comparison of Useful CPU Utilizations Achieved by PES-A, PES-B, TTF and RMS on an Overloaded System when $C/T = O(1/T^4)$	111
Figure B-14 Comparison of Inter-task Fairness Metric Achieved by PES-A, PES-B, TTF and RMS on an Overloaded System when $C/T = O(1/T^4)$	111
Figure B-15 Comparison of Guaranteed Ratios for Meeting Deadlines Achieved by PES-A, PES-B, TTF and RMS on an Overloaded System when $C/T = O(1/T^6)$	112
Figure B-16 Comparison of Average Ratios for Meeting Deadlines Achieved by PES-A, PES-B, TTF and RMS on an Overloaded System when $C/T = O(1/T^6)$	112
Figure B-17 Comparison of Useful CPU Utilizations Achieved by PES-A, PES-B, TTF and RMS on an Overloaded System when $C/T = O(1/T^6)$	113
Figure B-18 Comparison of Inter-task Fairness Metric Achieved by PES-A, PES-B, TTF and RMS on an Overloaded System when $C/T = O(1/T^6)$	113
Figure C-1 Comparison of Guaranteed Ratios for Meeting Deadlines Achieved by PES-A, PES-B, TTF and RMS on a Non-overloaded System when $C/T = O(T^4)$	114

Figure C-2 Comparison of Inter-task Fairness Metric Achieved by PES-A, PES-B, TTF and RMS on a Non-overloaded System when $C/T = O(T^4)$	114
Figure C-3 Comparison of Guaranteed Ratios for Meeting Deadlines Achieved by PES-A, PES-B, TTF and RMS on an Overloaded System when $C/T = O(T^4)$	115
Figure C-4 Comparison of Average Ratios for Meeting Deadlines Achieved by PES-A, PES-B, TTF and RMS on an Overloaded System when $C/T = O(T^4)$	115
Figure C-5 Comparison of Useful CPU Utilizations Achieved by PES-A, PES-B, TTF and RMS on an Overloaded System when $C/T = O(T^4)$	116
Figure C-6 Comparison of Inter-task Fairness Metric Achieved by PES-A, PES-B, TTF and RMS on an Overloaded System when $C/T = O(T^4)$	116

List of Tables

Table 2-1 Estimation Results and Priorities for τ_1	25
Table 2-2 Estimation Results and Priorities for τ_2	25
Table 3-1 Computation Times and Periods for Tasks	42

Chapter 1 Introduction

1.1 Background and Motivation

This thesis concerns scheduling of tasks in real time systems. An introduction to the research domain including previous work done in the area is presented.

1.1.1 Real-time System and Scheduling

In real-time systems, the correctness of the logical results of computation does not guarantee the correctness of the system; whether these results have been successfully produced with a certain time constraint also need to be checked. Examples of this type of systems include flight control programs, air traffic control systems, control systems for nuclear power plants, patient monitoring systems, and multimedia communication systems. Based on the consequences of failure to meet certain time constraints, real-time systems can be categorized into three types [16, 17]:

- Hard real-time systems, where failure to meet response time constraints leads to catastrophic consequences on the system. In this case, any instance of the task should be guaranteed to meet its time constraints in the worst-case scenario. Examples are automated factories and flight control systems [17].
- Soft real-time systems, where failure to meet response time constraints leads to degraded performance, but the system will not fail to function completely by such kind of failures. An example is GUI systems that reflect traffic situations for traffic control systems [16].
- Firm real-time systems, which are characterized by a set of firm real-time tasks. With these, it is acceptable to have a low percentage of task instances occasionally missing

their deadlines. In this case, any instance of the task should be either guaranteed to complete within its deadline or entirely rejected. Some applications in telephone switches and other communication systems are examples of such systems [14].

In addition to handling concurrency and synchronisation, task scheduling is very important in the context of real-time systems.

Scheduling can be either pre-emptive or non-pre-emptive [34].

- Pre-emptive scheduling means that tasks that are running can be temporarily suspended and the resource is allocated to another task.
- Non-pre-emptive scheduling means that tasks that are running will run to completion. New scheduling decisions are made when the running task is completed.

Scheduling can also be categorized into static and dynamic scheduling based on prescribed run-time behaviours.

- Static scheduling algorithms make their decisions before the system starts running.
- Dynamic scheduling algorithms make their decisions at run time.

Currently, the most widely applied scheduling algorithms in hard real-time systems are Rate Monotonic Scheduling (RMS) [25], Deadline Monotonic Scheduling (DMS) [24] and Earliest Deadline First Scheduling (EDF) [31].

RMS and DMS are pre-emptive static scheduling algorithms. They are called fixed-priority scheduling algorithms since they assign priorities to each task before the set of tasks starts execution, and priorities for all tasks remain unchanged from that point. RMS is a scheduling technique for tasks with their deadlines equal to their periods. Tasks with shorter periods are assigned higher priorities. DMS is an extension of RMS in which tasks can have relative deadlines that are less than their periods [11]. The relative deadline is

each request's deadline for a task minus its start time. It is constant for each request. Tasks with shorter relative deadlines have higher priorities.

EDF is a pre-emptive dynamic scheduling technique. It is also called a dynamic-priority scheduling algorithm since it assigns a priority to each task before every request during run-time. Tasks with earlier deadlines are assigned higher priorities. The priority of each task may be changed from request to request based on the deadline of the current request.

Many of the modifications of these scheduling algorithms have been applied to meet the requirements of soft or firm real-time systems. Since the research discussed in this thesis is based on RMS, a detailed discussion of the recent developments of the RMS technique is presented in the following sections.

1.1.2 Hard Real Time System and Rate Monotonic Scheduling

RMS, proposed by Liu and Layland in 1973 is a pre-emptive fixed priority-based static scheduling algorithm. It is based on the following five assumptions [25]:

- All critical tasks are periodic with constant intervals. This indicates that all tasks send their requests at regular constant intervals of time.
- Each request for a task must be completed before the next request for the same task occurs.
- All tasks are independent. They do not need to communicate with each other.
- Processing time of each task is a constant.
- Any non-periodic task in the system is non-critical.

In RMS, each task is assigned a priority according to its period based on the rule that a task with a shorter period is assigned a higher priority. These priorities are assigned before execution and do not change from request to request. All tasks are scheduled at run-time

in accordance with a priority-based pre-emptive scheduling algorithm. Only when no tasks with higher priorities are ready to execute and the CPU is idle, can a task with a lower priority execute. However, the currently executing task with a lower priority can be pre-empted by a newly arrived task with a higher priority.

RMS guarantees that if the total processor utilization does not exceed $n(2^{1/n}-1)$, an arbitrary set of n periodic tasks is schedulable, which means that all tasks in the hard real-time system can meet their deadlines [25]. Based on this formula, the upper bound of processor utilization for guaranteeing schedulability is 0.69 when n approaches infinity. It was shown by Liu that if the processor utilization exceeds the upper bound for a given n , the worst case scenario occurs when $T_1 < T_n < 2 \cdot T_1$. Later Lehoczky *et al.* improved the bound to approximately 0.88 for a set of n tasks on a simulation study [21].

It has been noted that the schedulability condition is sufficient but not necessary. This indicates that if a set of tasks satisfies the schedulability condition, it must be schedulable. For certain systems, the set of tasks may still be schedulable even if the condition is not met [6].

RMS is an effective algorithm for hard real-time systems because of its strong theoretical foundation and the simplicity of its computation. However, as indicated in the following list, in a number of systems the five assumptions underlying the RMS model described earlier may not represent the real situation.

- A system may have hybrid critical tasks, whose periods are periodic or aperiodic.
- The tasks in a system are very rarely independent. Indeed, they are often reactive to each other, and inter-process communication is required.

- A task may have variable processing time so that a transient overload can occur in the system.

For these reasons, many researchers have eased the restrictions of the RMS model to make it more applicable to practical hard real-time systems.

To handle aperiodic tasks, some studies have modified RMS and added fixed-priority servers and dynamic-priority servers. Each server is treated as a normal periodic task having its own period and computation time.

The fixed-priority servers are assigned priorities based on static scheduling algorithms such as RMS. These include polling server, deferrable server and sporadic server. When the polling server [20, 30] is invoked at the start of every period, it is assigned a fixed slot of time for serving pending aperiodic tasks. If no aperiodic requests are pending, the server suspends itself until the beginning of its next period, and the computation time originally allocated for aperiodic service is not preserved for aperiodic execution but is used by periodic tasks. The computation time is replenished at the beginning of the next period.

The deferrable server [20, 32] is the same as the polling server, except that it preserves its computation time if no requests are pending upon its invocation. The sporadic server [30] is the same as the deferrable server, except that the computation time is replenished only after it has been consumed by aperiodic task executions.

Unlike the methods described in the previous paragraphs, slack stealing [23] creates a passive task called Slack Stealer, which attempts to make time for serving aperiodic tasks by “stealing” all the slack time it can from the periodic tasks without causing their deadlines to be missed. Hence, when an aperiodic request arrives, the Slack Stealer steals

the available slack from periodic tasks and uses it to execute aperiodic requests as soon as possible.

The dynamic-priority servers are similar to the fixed-priority servers, except priorities are assigned based on dynamic scheduling algorithms such as EDF [27, 28].

To handle problems of unbounded priority inversion and deadlock caused by dependent tasks, the priority inheritance protocol (PIP), the priority ceiling protocol (PCP) and others have been proposed. In unbounded priority inversion, a task with a low priority may lock a resource and be pre-empted by a task with a high priority, which in turn gets blocked when it requests the resource that is locked by the task with the low priority. A task with a medium priority that does not require the resource pre-empts the task with the low priority again. As a result, the task with the high priority has to wait for the task with the medium priority to complete before the task with the low priority gets a chance to release the resource. PIP [29] can solve the problem. In PIP, if a task with a higher priority is blocked because a resource is being held by another task with a lower priority, the task with the lower priority inherits the higher priority of the blocked task and is executed until the lock on the requested resource is released. At this point, the task regains its low priority.

PCP [29] handles the deadlock that occurs when more than one task requires more than one resource to enter a critical section but each task is currently locking different resources. In this protocol, each resource is assigned a priority ceiling, which is the highest priority among all tasks that could lock it. A task X is allowed to enter a critical section only if its priority is higher than all the priority ceilings of the resources currently locked

by other tasks with lower priorities. Otherwise, the other tasks locking the requested resources will inherit higher priority of the task X, and run to completion.

1.1.3 Soft/Firm Real-time System Scheduling

As discussed in Section 1.1.1, a soft or firm real-time system does not require all jobs of every task in the system to meet their deadlines. A job corresponds to a request for a task execution that is generated once in every task period. This is different from a hard real-time system in which all jobs of every task must meet their deadlines. Directly using hard real-time scheduling algorithms such as RMS, DMS or EDF in a soft or firm real-time system may lead to more jobs missing deadlines, unfairness for some tasks with low priorities, and wasted resource utilization. This is the case because tasks with higher priorities are consuming CPU unnecessarily even when they cannot meet their deadlines or are not required to meet their deadlines 100% of the time. Meanwhile, other tasks with lower priorities get fewer or no chances to be executed. Thus, the application of RMS, DMS, or EDF to soft/firm real-time systems can lead to poor performance. This situation becomes more acute when the system becomes overloaded.

In recent years, some researchers have made efforts to modify hard real-time scheduling techniques to make them applicable to soft/firm real-time systems. These research efforts are aimed at providing scheduling algorithms that meet the following goals especially when the system is overloaded and the jobs of each task have variable computation times:

- Improvement of the number of jobs for all tasks meeting their deadlines in a system.
- Improvement of the fairness for all tasks in a system.
- *A priori* prediction of system performance before it becomes operational.

- Improvement of the resource utilization of a system.

The research presented in [13, 35] is related to soft real-time system scheduling on a uni-processor. Tia, *et al.* proposed a method called Probabilistic Time Demand Analysis (PTDA) to transform the original task into a periodic task followed by a sporadic task [35]. In this system, there is a threshold of computation time given by the system designer to determine if a sporadic server is needed for an arriving job. If the computation time of the arriving job does not exceed the threshold, it can be served as a periodic task and is guaranteed to meet its deadline; otherwise, it is split into a periodic job whose computation time equals the threshold, and an aperiodic job with the remaining computation time. The periodic job is guaranteed to meet its deadline, while the aperiodic job may or may not meet its deadline. This method has the advantage of improving resource utilization. However, because a sporadic server is used in FIFO order, jobs that do not need to meet deadlines may be served first, and unfairness for tasks may result. Gardner *et al.* proposed the Stochastic Time Demand Analysis (STDA) technique [13] in order to provide a prediction of system performance by calculating a lower bound on the ratio of the number of jobs meeting deadlines and the total number of jobs.

Other research related to a uni-processor based firm real-time system is presented in [2, 4]. Statistical Rate Monotonic Scheduling (SRMS), an extension of RMS, is proposed in [1, 3, 4]. The model underlying this method contains a set of periodic tasks with variable computation times in a firm real-time system. Every task has its period and super-period. A task is invoked every period. Its super-period is the period of the immediate next task with a longer period. The “allowance value” of each task is calculated by its computation time multiplied by the ratio of its super-period and its period. A job is admitted to the

system if and only if its computation time does not exceed the time remaining in the period after all tasks with higher priorities have claimed their “allowance values”. Otherwise, its priority is dropped to a background level.

Slack Stealing Job Admission Control (SSJAC) is proposed in [2]. In this algorithm, an arriving job is admitted into the system and is guaranteed to meet its deadline if its computation time does not exceed a threshold given by the system designer. If this condition is not satisfied, slack stealing [11, 23] is used for accurate computation of the remaining time. If there is sufficient time remaining, the job is allowed into the system and guaranteed to meet the deadline; otherwise, it is dropped.

1.2 Goals of This Research

A lot of work has been done on scheduling algorithms for hard real-time systems. Comparatively little research exists, however, on scheduling algorithms for soft or firm real-time systems.

Soft or firm real-time systems allow some jobs of each task to miss their deadlines as long as the percentage of missed deadlines is below a certain specified level. For this reason, RMS and other hard real-time system scheduling algorithms may not be effective for soft or firm real-time systems. These algorithms try to meet deadlines for all jobs of all tasks. Jobs from tasks with higher priorities consume as much CPU as they can, even if there is insufficient time to meet deadlines or it is unnecessary to meet deadlines 100% of the time. This leads to jobs from tasks with lower priorities having fewer chances to run. Consequently, these systems have low number of jobs meeting deadlines for all tasks and are unfair to tasks with low priorities.

There are not many existing scheduling algorithms for soft real-time systems, and even fewer scheduling algorithms are available for firm real-time systems. This thesis attempts to fill this gap by proposing modified RMS algorithms for uni-processor based firm real-time systems.

This thesis focuses on firm real time systems characterized by periodic tasks with fixed execution times. The goals of this thesis are to develop effective scheduling strategies in firm real time systems, and to analyse their performances under various workloads. Such a performance analysis can lead to insights regarding the relative performances of the scheduling strategies. A part of this research is motivated by Nortel Networks that is interested in scheduling on telecommunication systems.

1.3 Thesis Contributions

The contributions of this thesis include the following.

- Three modified RMS algorithms are proposed in this thesis. These scheduling algorithms are proposed for a uni-processor based firm real-time system.
- In most of the cases analysed in this thesis, the proposed algorithms improve the proportion of jobs that meet their deadlines in comparison with RMS.
- In most of the cases analysed in this thesis, the proposed scheduling algorithms improve system fairness in comparison with RMS.
- In most of the cases analysed in this thesis, the proposed scheduling algorithms improve the CPU utilization in comparison with RMS.
- The thesis provides a set of detailed insights into the relative performances of the scheduling algorithms. Systems under both normal load and overload are analysed.

1.4 Organization

This thesis is divided into five chapters. Chapter 1 provides an introduction to the topic. Chapter 2 describes the three proposed scheduling algorithms. Chapter 3 presents a detailed description of the implementations used in the performance analysis, and the design of the experiments performed on the prototype. Chapter 4 focuses on the analysis of results of these experiments, whereas Chapter 5 presents the conclusions of this thesis and some open issues for future research.

Chapter 2 Proposed Scheduling Algorithms

Three scheduling algorithms are proposed in this research. There are two Pre-Execution Estimation Based Scheduling algorithms and the Tardy Task First algorithm. These will be discussed in detail in this Chapter.

Before discussing the scheduling algorithms the input system parameters and the performance metrics are introduced.

2.1 Input Parameters for the System

n - The number of tasks in the system. It does not include the estimation task used by the proposed algorithms.

C_i - Computation time for each task τ_i . In this thesis the task execution times are fixed and every job of a given task τ_i consumes C_i units of CPU time.

T_i - The period for each task τ_i .

T_0 - Super-period which is a period of the estimation task τ_0 introduced by the PES algorithms and the TTF algorithm.

$P(\tau_i)$ - Priority of task τ_i .

maxrun - The time duration of an experiment.

Threshold ratio for meeting deadlines, a_i - A threshold ratio of the number of jobs meeting deadlines and the total number of jobs in task τ_i . It is used to decide on the priority of a job and compute the super-period. All the tasks have the same a_i in each experiment.

CPU utilization requested by each task τ_i - It is defined as C_i / T_i .

CPU utilization requested by all tasks - It is defined as $\sum_{i=1}^n (C_i / T_i)$.

2.2 Performance Metrics

Ratio for meeting deadlines for each task τ_i , A_i - Ratio of the jobs that have met their deadlines and the total number of jobs for each task τ_i .

Guaranteed Ratio for meeting deadlines, G - The minimum ratio for meeting deadlines achieved by a task of the system. Note that an expected set of A_i 's and/or an expected G may be specified by the user of the system.

Average ratio for meeting deadlines, $\bar{A}$ - The average of the ratios for meeting deadlines for all tasks in the system. It is defined as:

$$(\sum_{i=1}^n A_i) / n$$

Depending on the system, G or $\bar{A}$ is compared with the expected ratio for meeting deadlines. If G or $\bar{A}$ is greater than or equal to this expected value, the system is considered to operate satisfactorily.

Useful CPU utilization achieved by each task τ_i - The proportion of CPU time consumed by the jobs of task τ_i that met deadlines. It is defined as:

$$(the\ number\ of\ jobs\ meeting\ deadlines\ for\ task\ \tau_i \cdot C_i) / maxrun$$

Useful CPU utilization achieved, U - The proportion of CPU time consumed by the jobs of all tasks that met their deadlines. It is the sum of the useful *CPU utilization achieved* by each task.

Inter-task fairness metric, f - This is used to evaluate the fairness of a scheduling strategy.

It is defined as:

$$f = \sqrt{\frac{\sum_{i=1}^n (A_i - \bar{A})^2}{n-1}} * 100\%$$

Some scheduling strategies that are not fair may favor a certain set of tasks that will achieve higher ratios for meeting deadlines in comparison with other tasks in the system. This measure is used to evaluate the fairness characteristic of a scheduling strategy. The higher the value of f , the more unfair is the strategy. Fairness is a desirable characteristic of a scheduling strategy in many systems (see [4] for example). With a fair strategy, the A_i values achieved with the different tasks are expected to be close to one another giving rise to a small f . An unfairness factor has been used in [4] to characterize the fairness of scheduling strategies.

2.3 Pre-Execution Estimation Based Scheduling Algorithms

2.3.1 General Introduction

As discussed in Section 1.1.3 of Chapter 1, classical Rate Monotonic Scheduling may not perform well for firm real-time systems. This is the case because RMS is a fixed-priority algorithm, which always assigns higher priorities to tasks with shorter periods than to tasks with longer periods. Once the priorities are assigned, they are fixed so that tasks with shorter periods will never yield to tasks with longer periods.

As a result, if RMS is used for a firm real-time system, the tasks with shorter periods still consume CPU unnecessarily even when they cannot meet their deadlines or are not required to meet their deadlines 100% of the time. At the same time, other tasks with longer periods have fewer or no chances to be executed. Hence, the number of jobs meeting deadlines in such tasks can be low. RMS can lead to unfairness: although many

jobs of tasks with shorter periods meet their deadlines, only a few jobs of tasks with longer periods may be able to do so. If too many jobs of tasks with longer periods miss their deadlines, the average ratio for meeting deadlines will be low. Therefore, the use of RMS in a firm real-time system may result in poor performance.

In this thesis Pre-Execution Estimation Based Scheduling algorithms are proposed for a firm real-time system. PES algorithms are modified RMS algorithms, which overcome the disadvantages of RMS, and are more suitable for a firm real-time system. Unlike RMS, in which the priority of a task is fixed once it is assigned, in PES algorithms the priority of a task may change from one job to another. For example, assume there are two tasks in a firm real time system. Initially, a job of a task with a shorter period has a higher priority than a job of a task with a longer period. However, there are two conditions under which the job with the shorter period will be assigned a lower priority than the job with the longer period. The first condition is that the job of the task with shorter period cannot meet its deadline because of insufficient time remaining before the deadline and the job of the task with the longer period has the ability to meet its deadline if it is run instead of the job of the task with a shorter period. The second is that the task with the shorter period has currently satisfied its requirement for meeting deadlines, while the task with the longer period has not. Since tasks with longer periods get more chances to meet their deadlines in comparison with RMS, the use of PES algorithms is likely to increase the guaranteed ratio for meeting deadlines. Similarly, the inter-task fairness metric is likely to be improved. The average ratio for meeting deadlines is likely to be increased as well.

2.3.2 Discussion of Pre-Execution Estimation Based Scheduling Algorithms

In PES algorithms, an extra task called estimation is added to the system and assigned the highest priority. The purpose of estimation is to make the necessary computation that will be used to determine task priorities. Based on the information produced, an appropriate priority is assigned to the job before it is dispatched to CPU.

Since a firm real-time system does not require that all tasks meet their deadlines 100% of the time, the concepts of expected ratio for meeting deadlines and threshold ratio for meeting deadlines are introduced in PES algorithms (See Section 2.1, 2.2 for definitions). The expected ratio for meeting deadlines is a user-expected average ratio of meeting deadlines in a system. The threshold ratio for meeting deadlines is a predefined ratio for meeting deadlines. It is used to decide the priority of a job. For example, before a job is dispatched to the CPU, the current ratio for meeting deadlines of a task is calculated and checked to determine whether it has achieved the threshold ratio for meeting deadlines. If it has, the priority of the job is decreased. In PES algorithms analysed in this thesis, the threshold ratio for meeting deadlines is same for all the tasks in the system.

Since different threshold ratios for meeting deadlines affect the priorities of tasks, different average ratios for meeting deadlines will be obtained. Some of them may achieve the expected ratio for meeting deadlines; others may not. Consequently, it is important to effectively determine the threshold ratio for meeting deadlines in a system. If the chosen ratio is too high, tasks with shorter periods will occupy much more CPU than tasks with longer periods, and vice versa. In either case, a low average ratio for meeting deadlines will be obtained.

In this thesis, the threshold ratio for meeting deadlines used by the PES algorithms is selected in the following manner. We first measure the average ratio for meeting deadlines, X , achieved with RMS, and then the PES algorithms are executed with a number of threshold ratios that lie in the range $[X-10\%, X+10\%]$. The threshold ratios start with $X - 10\%$, followed by $X - 9\%$, $X - 8\%$ and so on. The threshold ratio that produces the best average ratio for meeting deadlines is chosen.

An important component of the PES algorithms, the estimation task, is discussed next.

Assume a set of n tasks $(\tau_1 \dots \tau_n)$ in a firm real-time system. The period of task τ_i is T_i . The tasks in the list $(\tau_1 \dots \tau_n)$ are ordered such that $T_{i-1} < T_i$ ($i = 1 \dots n$)

Characteristics of the estimation task:

Like any task τ_i , the additional estimation task has a computation time and a period. The period of this task is called the super-period. The length of the super-period (T_0) is determined in two steps. The first step determines the value of an initial super-period called *init- T_0* which is given by,

$$init-T_0 = \max\{1 \leq i \leq n \mid k \cdot T_i\} \quad (1)$$

k is the smallest integer satisfying the condition that $m = a \cdot k$ where m is an integer and a is the threshold ratio for meeting deadlines. For example, if the threshold ratio for meeting deadlines is 0.8, m is 4 and k is 5.

The second step determines the super-period by using equation (1). In general, the super-period (T_0) should be long enough to contain a sufficient number of jobs for each task for the estimation to be precise. We define,

$$T_0 = c \cdot init-T_0 \text{ (where } c \text{ is an Integer)} \quad (2)$$

The constant c is an enlargement factor used to expand $init-T_0$. In this thesis, the value of c is determined through experiments described in Section 3.4.4 of Chapter 3.

The function of estimation:

The estimation task always has the highest priority. It runs first at the beginning of each super-period. The estimated ratio for meeting deadlines for a task is computed for every job in the super-period. It is computed for job k of a task τ_i in a super-period in the following way. Let x be the number of jobs that arrive before job k and these jobs are estimated to meet their deadlines. Then the required estimated ratio for meeting deadlines is given by x/k . The ratio is reset to be 0 at the start of every super-period.

As in RMS, all tasks $\tau_1 \dots \tau_n$ are first assigned priorities based on their periods. The longer the period of a task, lower is its priority. Then, following the order of priority of each task from high to low, every job in each task in the current super-period is checked to determine whether or not it has sufficient time to meet its deadline. If a job has sufficient time to meet its deadline, it is included in the set of tasks estimated to meet their deadlines. If a job has insufficient time to meet its deadline, or the estimated ratio for meeting deadlines is equal to or higher than threshold ratio for meeting deadlines, the index of the job is stored in a list L . This list is used to assign priorities to tasks during this super-period, and updated at the start of each super-period by the estimation task.

In this thesis, two PES algorithms are proposed. They are Pre-execution Estimation based Scheduling algorithm A (PES-A) and Pre-execution Estimation based Scheduling algorithm B (PES-B). These algorithms will be described in detail in Section 2.3.3 and Section 2.3.4.

Assignment of priorities to tasks:

Three groups of priorities are defined in the PES algorithms. On a system with n tasks, each group has n priority values. The priority values range from 1 to n in the first group, from $n+1$ to $2n$ in the second group, and from $2n+1$ to $3n$ in the third group. The smaller the number, the higher is the priority. 0 is the highest priority and is always assigned to the estimation task. At each group, task periods are used for assigning priorities.

The priorities are assigned before a job is dispatched to CPU based on information provided by estimation. If the index of a job is not in the list L , it means that the job has been estimated to have sufficient time to meet its deadline, and the estimated ratio for meeting deadlines is lower than the threshold ratio for meeting deadlines. This job is assigned a priority value from the first group so that it is guaranteed to meet its deadline.

The index of a job may be in the list in two cases. In one case, the estimated ratio for meeting deadlines of the task is lower than the threshold ratio for meeting deadlines, and the job is calculated to have insufficient time for meeting its deadline. In the other case, the estimated ratio for meeting deadlines of the task is equal to or higher than the threshold ratio regardless of whether or not the job has sufficient time to meet its deadline.

Note that due to the approximations used in computing, the estimated ratio for meeting deadlines may be different from the actual current ratio for meeting deadlines. The current ratio for meeting deadlines for a task has a similar definition as the estimated ratio for meeting deadlines except that it is calculated dynamically when the system is running. Therefore, before a job is dispatched to the CPU, the current ratio for meeting deadlines is checked in both cases while priority assignment is made. If the index of a job is in the list, and the current ratio for meeting deadlines of a task is lower than the threshold ratio for meeting deadlines, the job is assigned a priority value from the second group. With this

priority there is still a possibility for the job to meet its deadline if it has sufficient time before the expiry of its deadline. If the current ratio for meeting deadlines of a task is equal to or higher than the threshold ratio, the job is assigned a priority value from the third group. Such a job still has a chance to meet its deadline if other tasks with higher priorities have achieved the threshold ratio for meeting deadlines.

It can be seen that the PES algorithms assign priorities to tasks not only in accordance with their periods but also by comparing the current/estimated ratio for meeting deadlines with the threshold ratio for meeting deadlines, as well as by calculating the time available before the deadline for each job of any task.

Sometimes it is possible for a job to be released in one super-period (called start super-period) and for the corresponding deadline to occur in the next super-period (called end super-period). In estimation, the task period of this type of job is shrunk. Its deadline is advanced to the end of “start super-period”. Estimation is greatly simplified in this way, but at the cost of a certain loss of accuracy. However, if the super-period is long enough and the number of jobs included in the super-period is large enough, the effect of inaccuracy can be negligibly small.

2.3.3 Pre-Execution Estimation Based Scheduling Algorithm A

A system with a set of n tasks $\tau_1, \tau_2 \dots \tau_n$ is considered. The periods of these tasks are $T(\tau_1) < T(\tau_2) < \dots < T(\tau_n)$. Initially, the priorities of these tasks are $P(\tau_1) > P(\tau_2) > \dots > P(\tau_n)$. Under priority-based scheduling, a job τ_i will be pre-empted only by jobs with higher priorities, and it will pre-empt any jobs with lower priorities. Thus, to compute the time available before the occurrence of the deadline for τ_i , it is necessary to consider the amount of time consumed by $\tau_1, \dots, \tau_{i-1}$ during a period of τ_i . We use Figure 2-1 to

describe the method of computing the time available for τ_i before its deadline in the estimation task.

$t_{ik}^{consumed}$ is the time consumed by $\tau_1, \dots, \tau_{i-1}$, during the given period of the k^{th} job for task τ_i . During a period of the k^{th} job of τ_i , multiple jobs of τ_j ($j < i$) may execute (See Figure 2-1). Consider two tasks τ_i and τ_j with $j = i-1$ and $T_i > T_j$. In order to compute N_j , the number of invocations of τ_j within a single period of τ_i that corresponds to the k^{th} job, three cases are to be considered.

Figure 2-1 Multiple invocations of task τ_j within a single period of task τ_i

Case I: Total overlap

In this case, T_i is an exact multiple of T_j (See Figure 2-2), and $N_j = T_i / T_j$.

Figure 2-2 Case I: Total Overlap

Case II: Partial overlap with the same start time

In this case, a job of τ_j arrives at the same time as the k^{th} job of τ_i (See Figure 2-3). In this case, $T_i / T_j = I + x$, where I is an integer and $x < 1$. For such a system $N_j = \lceil T_i / T_j \rceil$.

Figure 2-3 Case II: Partial Overlap with Same Start Time

Note that this case is similar to the case in which the arrival time of the $k+1^{th}$ job of task τ_i synchronizes with the arrival of a job for task τ_j (See Figure 2-4).

Figure 2-4 Case II: Partial Overlap with Same End Time

Case III: Partial overlap with different start time

In this case the arrivals of jobs for τ_j do not synchronize with the arrivals of jobs for τ_i (See Figure 2-5). In this case, $N_j = \lceil T_i / T_j \rceil + 1$.

Figure 2-5 Case III: Partial Overlap with Different Start Times

From the above three cases it is clear that $N_j = \lceil T_i / T_j \rceil + 1$. Since we do not know whether or not the arrival of jobs of different tasks are synchronized, the maximum value of N_j is used in the computation of $t_{ik}^{consumed}$. So the maximum time consumed by $\tau_1, \dots, \tau_{i-1}$ during a period of the k^{th} job for task τ_i is

$$\sum_{j=1}^{i-1} (\lceil T_i / T_j \rceil + 1) \cdot C_j, \text{ where } C_j \text{ is computation time for task } \tau_j$$

It is possible that some jobs for $\tau_1, \dots, \tau_{i-1}$ are assigned a priority value from the second or third group since the indices of these jobs are stored in the list L by the estimation task.

So when $t_{ik}^{consumed}$ is calculated, these jobs should not be considered. In this algorithm, the number of such lost jobs is $lost_j$. $lost_j$ can be computed by searching the list L. Note that while computing $lost_j$ only those jobs of τ_j that can potentially execute in the interval between $start(\tau_{ik})$ and $end(\tau_{ik})$ are considered. Thus,

$$t_{ik}^{consumed} = \sum_{j=1}^{i-1} (\lceil T_i / T_j \rceil + 1 - lost_j) \cdot C_j \quad (1 \leq i \leq n, j < i) \quad (3)$$

The time available before the deadline for τ_{ik} is the period of τ_i minus the consumed time.

Thus,

$$\begin{aligned} t_{ik}^{avail} &= T_i - t_{ik}^{consumed} \\ &= T_i - \sum_{j=1}^{i-1} (\lceil T_i / T_j \rceil + 1 - lost_j) \cdot C_j \quad (1 \leq i \leq n, j < i) \end{aligned} \quad (4)$$

Since $t_{ik}^{consumed}$ is the maximum possible consumed time, it may exceed the actual consumed time. As a result, if the time available before deadline for a job is calculated to be equal to or greater than C_i , then this job can meet its deadline. The index of this job is

thus not stored in the list L. So this job is assigned a priority value from the first group, and is guaranteed to meet its deadline. Otherwise, the index of the job is stored in the list, although in fact this job may have sufficient time for meeting its deadline. The job is assigned a priority value from the second group or third group and is not guaranteed to meet its deadline.

The following is an example of using PES-A. Consider a system with a set of two tasks: $C_1=4, T_1=5; C_2=1, T_2=8$. With the first job of τ_2 (τ_{21}), the time available for job τ_{21} is:

$$\begin{aligned} t_{21}^{avail} &= T_2 - t_{ik}^{consumed} \\ &= T_2 - \sum_{j=1}^1 (\lceil T_2/T_1 \rceil + 1 - lost_j) \cdot C_1 \\ &= 8 - (\lceil 8/5 \rceil + 1 - 0) \cdot 4 = 8 - 12 = -4 < C_2 \end{aligned}$$

τ_{21} is deemed to not have enough time before the deadline. This job is, nevertheless, assigned a priority value from the second group and is not guaranteed to meet its deadline.

2.3.3.1 An Example of PES-A

This section provides an example of PES-A to illustrate the mechanism of the proposed algorithms. Consider a firm real-time system in which

$$a_1 = a_2 = 0.7$$

$$\tau_1 : C_1 = 1, T_1 = 2$$

$$\tau_2 : C_2 = 2, T_2 = 3$$

The steps executed by the PES-A algorithm for this system are presented.

1. Calculating super-period of the estimation task

Using equation (1), since $a_1 = a_2 = 0.7$: k is 10 and $init-T_0 = \max \{10 \cdot T_1, 10 \cdot T_2\} = \max \{10 \cdot 2, 10 \cdot 3\} = 30$

Using equation (2) and assuming c is 1, $T_0 = c \cdot \text{init} - T_0 = 1 \cdot 30 = 30$

2. Function performed by the estimation task

Table 2-1 Estimation Results and Priorities for τ_1

Jobs	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Est ratio(%)	0	50	66.7	75	60	66.7	71.4	62.5	66.7	70	63.6	66.7	69.2	71.4	80
list L	No	No	No	Yes	No	No	Yes	No	No	Yes	No	No	No	Yes	No
Time Avail	1	2	2	2	2	2	2	2	2	2	2	2	2	2	2
Curr ratio(%)	0	50	66.7	75	60	66.7	71.4	62.5	66.7	70	63.6	66.7	69.2	71.4	80
Priority	1	1	1	5	1	1	5	1	1	5	1	1	1	5	1

Table 2-2 Estimation Results and Priorities for τ_2

Jobs	1	2	3	4	5	6	7	8	9	10
Est ratio(%)	0	0	0	0	0	0	0	0	0	0
list L	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Time Avail	0	0	0	0	0	0	0	0	0	0
Curr ratio(%)	0	0	33.3	50	60	66.7	71.4	62.5	66.7	70
Priority	4	4	4	4	4	4	6	4	4	6

In estimation, the time available before the deadline for each job during the current super-period and the estimated ratio for meeting deadlines of each task are calculated. They are displayed in Table 2-1 and Table 2-2. In these tables, The 'Est ratio' indicates the estimated ratio for meeting deadlines. The 'Yes/No' in the second row indicates if the index of a job is in list L. The 'Time avail' indicates the time available before a job's deadline. The 'Curr ratio' indicates the current ratio for meeting deadlines. It is dynamically calculated on the running system. In this example, because of the approximations used in computing consumed time in PES-A, some current ratios for meeting deadlines are different from estimated ratios for meeting deadlines. In this example, the computation time of estimation task is supposed to be 1 time unit.

For τ_1 , the estimated ratio for meeting deadlines is 0% for the first job of task τ_1 . It indicates if the first job does not meet its deadline, the estimated ratio for meeting

deadlines is lower than the threshold ratio for meeting deadlines 70%. Since the computation time of estimation task is 1 and periods of τ_1 is 2, the time available before deadline for this job is 1. Since computation time for τ_1 is 1, the job has sufficient time to meet its deadline. Thus, the index of 1st job is not stored in the list L.

When the 2nd job of τ_1 is estimated, the number of jobs meeting deadlines is 1 and the total number of jobs is 2. So the estimated ratio for meeting deadlines is $1/2 = 50\%$. It indicates if the second job does not meet its deadline, the estimated ratio for meeting deadlines is lower than threshold ratio for meeting deadlines. Since the time available before deadline is 2 and computation time for τ_1 is 1, the job has sufficient time to meet its deadline. Thus, the index of 2nd job is not stored in the list L. The 3rd, 5th, 6th, 8th, 9th, 11th, 12th, 13th and 15th jobs give rise to a similar situation as the 2nd job.

Before the 4th job is estimated, the number of jobs meeting deadlines is 3 and the total number of jobs is 4. So the estimated ratio for meeting deadlines is $3/4 = 75\%$. It indicates that if the 4th job misses its deadline, the estimated ratio is still higher than the threshold ratio, so the index of 4th job is stored in the list L. The 7th, 10th and 14th jobs give rise to a similar situation as the 4th job.

For τ_2 , the estimated ratio for meeting deadlines is 0% for the first job that is lower than threshold ratio for meeting deadlines 70%. However, based on equation (3) the time consumed by task τ_1 is $(\lceil 3/2 \rceil + 1) \cdot 1 = 3$. The period of task τ_2 is 3 units. So there is not enough time for this job to meet its deadlines. Thus, the index of 1st job for τ_2 is stored in a list L. The following jobs from 2nd to 10th give rise to a similar situation as the 1st job.

3. Assignment of priorities to tasks

As described in Section 2.3.2 there are three priority groups. The number of priority values in each group depends on the number of tasks. In this example, since the system has two application tasks, and the estimation task has the highest priority of 0, the first priority group includes priority values 1 and 2, the second priority group includes priority values 3 and 4, and the third priority group includes priority values 5 and 6. As noted earlier, a smaller number indicates higher priority. If jobs of two tasks are assigned the same priority group, the job of τ_1 is assigned a higher priority than the job of τ_2 since τ_1 has shorter period. So task τ_1 can have priorities of 1, 3 and 5, task τ_2 can have priorities of 2, 4 and 6.

For τ_1 the jobs whose indices are not in list L are assigned a priority of 1 from the first group. The 10th and 14th jobs are assigned a priority of 5 from the third group since their current ratios for meeting deadlines are higher than the threshold ratio. For τ_2 the jobs 1st, 2nd, 3rd, 4th, 5th, 6th, 8th and 9th are assigned a priority of 4 from the second group since their current ratios for meeting deadlines are lower than the threshold ratio and there are not enough time before their deadlines. The 7th and 10th jobs are assigned a priority of 6 from the third group since their current ratios for meeting deadlines are higher than the threshold ratio.

The time history of the system during execution using PES-A is shown in Figure 2-6. The time history for the same system using RMS algorithm is presented in Figure 2-7 for comparison.

From the time history shown in Figure 2-6 and Figure 2-7, it is observed that:

With PES algorithms, for τ_1 all jobs meet their deadlines except the 4th and 7th jobs. For τ_2 all jobs meet their deadlines except 1st, 7th and 10th jobs. So the ratio for meeting deadlines achieved by τ_1 is $13/15 = 86.7\%$. The ratio for meeting deadlines achieved by τ_2

is $7/10 = 70\%$. The average ratio for meeting deadlines is $(86.7\% + 70\%)/2 = 78.4\%$. The guaranteed ratio for meeting deadlines is 70% . The useful CPU utilization achieved (defined in Section 2.2) is $(1.13 + 2.7)/30 = 90\%$. The inter-task fairness metric is given by:

$$f = \sqrt{\frac{\sum_{i=1}^2 (A_i - \bar{A})^2}{2-1}} * 100\% = 12\%$$

Figure 2-6 Timing Diagram for PES-A

Figure 2-7 Timing Diagram for RMS

With RMS, all jobs of τ_1 meet their deadlines. For τ_2 only 2nd, 4th, 6th, 8th and 10th jobs meet their deadlines. So the ratio for meeting deadlines achieved by τ_1 is 100%, while τ_2 has achieved a ratio for meeting deadlines of $5/10 = 50\%$. The average ratio for meeting deadlines is $(100\% + 50\%)/2 = 75\%$. The guaranteed ratio for meeting deadlines is 50%.

The useful CPU utilization achieved is $(110 + 2.5)/30 = 83.3\%$. The inter-task fairness metric is 35%.

It can be summarized from above that for PES-A, the average ratio for meeting deadlines is 3.4% higher than that for RMS. The guaranteed ratio for meeting deadlines is 20% higher than that for RMS. The useful CPU utilization achieved is 6.7% higher than that for RMS. The inter-task fairness metric is much lower than that for RMS. This means that the use of PES-A results in a fairer treatment of the tasks in comparison to RMS.

The improved performance produced by PES algorithms is the result of each job being dynamically assigned a proper priority in accordance with the result of the pre-execution estimation. The accuracy of an estimated result is dependent on the accuracy with which the time available before deadline is computed for a job.

2.3.4 Pre-Execution Estimation Based Scheduling Algorithm B

As with the PES-A algorithm described in Section 2.3.3, a system with a set of n tasks $\tau_1, \tau_2 \dots \tau_n$ is considered. The periods of these tasks are $T(\tau_1) < T(\tau_2) < \dots < T(\tau_n)$. The priorities of these tasks are $P(\tau_1) > P(\tau_2) > \dots > P(\tau_n)$ initially. Under priority-based scheduling, a job of τ_i will be pre-empted only by jobs with higher-priorities, and it will pre-empt any job with a lower priority. Thus, to compute the time available before deadline of τ_i , the time consumed by $\tau_1, \dots, \tau_{i-1}$ during a period of τ_i must be considered. We use Figure 2-8 to describe the method of computing the time available for τ_i before its deadline. $t_{ik}^{consumed}$ is the time consumed by $\tau_1, \dots, \tau_{i-1}$, during the period of the k^{th} job for task τ_i . The only difference between PES-B and PES-A is that PES-B can compute the consumed time more accurately.

Figure 2-8 Computing time available by PES-B

In this algorithm, the start time of the k^{th} job for τ_i is called $start(\tau_{ik})$, and the time at which its deadline occurs is called $end(\tau_{ik})$. This job can be pre-empted by any τ_j with a higher priority.

For any τ_j with a higher priority, the length of time between $start(\tau_{ik})$ and $end(\tau_{ik})$ can be divided into three parts: $left_j$, $middle_j$ and $right_j$. Based on the following equations, $left_j$ and $right_j$ are less than the period of τ_j . $middle_j$ can be a multiple of the period of τ_j . In

Figure 2-8, $j = i - 1$ for τ_{i-1} and $j = i - 2$ for τ_{i-2} .

$$left_j = \lceil start(\tau_{ik}) / T_j \rceil \cdot T_j - start(\tau_{ik}),$$

$$right_j = end(\tau_{ik}) - \lfloor end(\tau_{ik}) / T_j \rfloor \cdot T_j,$$

$$middle_j = T_i - left_j - right_j$$

For any higher priority task τ_j , since $left_j$ is less than the period of τ_j , there is at most one job of τ_j to be considered during estimation. Assume that $lost_l_j$ is the number of jobs for τ_j that has been estimated to be missed during $left_j$. The value can be 0 or 1. If $left_j$ is 0, it indicates that there is no job during $left_j$. So $lost_l_j$ must be 0; otherwise, $lost_l_j$ can be 0 or 1. If the job of task τ_j during $left_j$ is estimated to be lost, it will be found in the list L. Such a job is assigned a lower priority than the current job of τ_i . Note that while computing $lost_j$ only those jobs of τ_j that can potentially execute in the interval between $start(\tau_{ik})$ and $start(\tau_{ik}) + left_j$ are considered. Thus, the maximum possible time consumed by τ_j during $left_j$ is given by:

$$t_j^{consumed} = (\lceil left_j / T_j \rceil - lost_l_j) \cdot C_j \quad (5)$$

where C_j is the computation time for task $j(\tau_j)$,

Since j can vary from 1 to $i-1$ the total time consumed during $left_j$ is given by:

$$t_j^{total_consumed} = \sum_{j=1}^{i-1} (\lceil left_j / T_j \rceil - lost_l_j) \cdot C_j \quad (6)$$

Since $t_{ik}^{consumed_left}$ is the time consumed between $start(\tau_{ik})$ and the latest end time for a $left_j$ ($j = 1 \dots i-1$), it cannot exceed $\max\{left_j \mid 1 \leq j \leq i-1\}$. Because of conservative assumption used in equation (6), the computed $t_j^{total_consumed}$ can exceed $\max\{left_j \mid 1 \leq j \leq i-1\}$. Therefore, the total time consumed by the higher priority tasks τ_j ($j = 1 \dots i-1$) during the period $left_j$ is given by:

$$t_{ik}^{consumed_left} = \min\left\{\sum_{j=1}^{i-1} (\lceil left_j / T_j \rceil - lost_l_j) \cdot C_j, \max\{left_j \mid 1 \leq j \leq i-1\}\right\} \quad (7)$$

The analysis is similar for $right_j$ and $middle_j$. Assume that $lost_r_j$ is the number of jobs for τ_j that has been estimated to be missed during $right_j$. Since $t_{ik}^{consumed_right}$ is the time

consumed between the earliest start time for a $right_j$ ($j = 1 \dots i-1$) and $end(\tau_{ik})$ it cannot exceed $\max\{right_j | 1 \leq j \leq i-1\}$. Thus, the total time consumed by the higher priority tasks τ_j ($j = 1 \dots i-1$) during the period $right_j$ is given by:

$$t_{ik}^{consumed_right} = \min\left\{\sum_{j=1}^{i-1} \lceil right_j / T_j \rceil - lost_r_j\right\} \cdot C_p \cdot \max\{right_j | 1 \leq j \leq i-1\} \quad (8)$$

Since the duration of $middle_j$ is divisible by τ_j without a remainder, the total time consumed by the higher priority tasks τ_j ($j = 1 \dots i-1$) during the period $middle_j$ is given by:

$$t_{ik}^{consumed_middle} = \sum_{j=1}^{i-1} (middle_j / T_j - lost_m_j) \cdot C_p \quad (9)$$

where $lost_m_j$ is the number of jobs for τ_j that has been estimated to be missed during $middle_j$.

Thus, the time available before the occurrence of the deadline for τ_{ik} is given by:

$$t_{ik}^{avail} = T_i - t_{ik}^{consumed_left} - t_{ik}^{consumed_middle} - t_{ik}^{consumed_right}$$

Using equation (7), (8) and (9), it is given by:

$$\begin{aligned} t_{ik}^{avail} = T_i - & \min\left\{\sum_{j=1}^{i-1} \lceil left_j / T_j \rceil - lost_l_j\right\} \cdot C_p \cdot \max\{left_j | 1 \leq j \leq i-1\} \\ & - \sum_{j=1}^{i-1} (middle_j / T_j - lost_m_j) \cdot C_p \\ & - \min\left\{\sum_{j=1}^{i-1} \lceil right_j / T_j \rceil - lost_r_j\right\} \cdot C_p \cdot \max\{right_j | 1 \leq j \leq i-1\} \quad (1 \leq i \leq n) \end{aligned} \quad (10)$$

In comparison with PES-A, PES-B has potentially more accurate results for time available because it does not simply use a mathematical ceiling to obtain the consumed time, but also selects a minimum value between the calculated value and the maximum value for each part.

The same example as in Section 2.3.3 is used for calculating the time available for a job. Consider a system with a set of two tasks: $C_1=4$, $T_1=5$; $C_2=1$, $T_2=8$. With the first job of $\tau_2(\tau_{21})$, we get:

$$left_1 = \lceil start(\tau_{21})/T_1 \rceil \cdot T_1 - start(\tau_{21}) = 0,$$

$$right_1 = end(\tau_{21}) - \lfloor end(\tau_{21})/T_1 \rfloor \cdot T_1 = 8 - \lfloor 8/5 \rfloor \cdot 5 = 3,$$

$$middle_1 = T_2 - 0 - 3 = 5.$$

If $lost_l_1$, $lost_m_1$ and $lost_r_1$ are 0,

$$\begin{aligned} t_{21}^{avail} &= 8 - \min\{\lceil left_1/T_1 \rceil \cdot C_1, left_1\} \\ &\quad - middle_1/T_1 \cdot C_1 \\ &\quad - \min\{\lceil right_1/T_1 \rceil \cdot C_1, right_1\} \\ &= 8 - \min\{0, 0\} - 4 - \min\{4, 3\} \\ &= 8 - 0 - 4 - 3 = 1 \end{aligned}$$

Since C_2 is 1, τ_{21} can meet its deadline. This job will be assigned a priority value from the first group and is guaranteed to meet its deadline.

2.4 Tardy Task First Algorithm

Tardy Task First (TTF) is an algorithm in which priorities are assigned based on results obtained from the past history of task executions. In TTF, as in PES algorithms, an additional estimation task is added to the real-time system and assigned the highest priority, and its period is called the super-period.

At the beginning of each super-period, the estimation task computes the ratio for meeting deadlines for each task using the results of the immediately preceding super period. Each task is then assigned a priority based on its ratio for meeting deadlines. The lower the ratio for meeting deadlines, the higher is the task priority.

In the first super-period, since there is no history of task execution, the ratio for meeting deadlines is 0 for each task. All are, therefore, awarded the same priority. It may be seen that in TTF, the priorities of all tasks may be changed at the start of each super-period. During a super-period, however, the task priorities remain fixed; they are not changed from one job to another.

Unlike RMS, in which the shorter-period tasks are always considered first, TTF always considers the tasks that have lower ratios for meeting deadlines. Consequently, since these tasks are assigned higher priorities their ratios for meeting deadlines are improved.

Chapter 3 Experimental Environment

Three techniques are employed for performance evaluation: analytical modelling, simulation, and measurement. Both analytical modelling and measurement are used to perform a quantitative analysis on the algorithms proposed in this research. Analytical modelling provides the best insight into different algorithms that are presented in Chapter 2. Measurement offers greater detail and a higher level of accuracy. The results of the performance analysis can be considered to be valid if the results produced by both techniques match. A performance prototype is built for evaluating the performances of the scheduling algorithms described in the previous chapters. The prototype is used with synthetic applications and the performances of the algorithms are measured. A detailed description of the prototype implementation is presented in the first part of the chapter. Experiments conducted to analyse the performance of the scheduling algorithms are discussed in the second part of the chapter.

3.1 Design and Implementation of the Prototype Environment

In the prototype, both RMS and the proposed algorithms are implemented. As described in Chapter 2, the proposed algorithms use an additional task, called estimation. Other application tasks are common to all the algorithms and are termed as normal tasks. For a system having n normal tasks, there are $n+1$ routines for spawning tasks called taskSpawn routines. Each of the routines creates a specific task τ_i ($i = 0 \dots n$). The task τ_0 is estimation task. The tasks $\tau_1 \dots \tau_n$ are normal tasks. A job for each task is created in every period of the task. The system uses a system clock characterized by a tick of 1/60 second. The clock is used in the various measurements performed. The job of task τ_i executes a for

loop for a given number of times to achieve the computation time. The execution times for all jobs of a given task are the same.

There is one monitoring and synchronization routine, as well as $n+1$ periodic-interrupt routines called taskPeriodISR routines. Each taskPeriodISR generates a job of estimation task every super-period or of a specific normal task every period. The taskSpawn routine and taskPeriodISR routine are synchronized with the monitoring and synchronization routine. The monitoring and synchronization routine is called once every tick. It guarantees all tasks are created simultaneously at the start of an experiment. It also records the CPU time consumed so far by a task. At the end of experiments the performance metrics such as G , $\bar{A}$, U and f are calculated.

The C language is used in implementing the prototype. During experiments, printed messages are displayed to make sure that the program runs properly. For example, a message that a task is spawned is displayed for each task at the start of an experiment. At the end of an experiment, a list of index of jobs missing deadlines is displayed.

The details of these routines are described in the following sections.

3.1.1 Task Spawning Routine

Each task has its own taskSpawn routine. When it is triggered, it keeps executing a while loop until the experiment is completed. In the while loop, the routine waits for the periodic signal from the corresponding taskPeriodISR routine. When the signal is received, the following operations are performed.

First, the current ratio for meeting deadlines is calculated before a new job is created.

Second, before the new job is created, its priority is assigned based on the scheduling algorithm.

Third, if the previous job is unfinished, it is killed before the new job is created.

Fourth, a new job is created. Such a job is kept alive by performing an empty loop that can end in two ways. If the job is finished before its deadline, a flag is set to exit from the loop and the job departs from the system. Alternately, if the job has not completed before its deadline occurs, it is killed before the next job is started. A job of each task is assumed to have a constant execution time.

3.1.2 Monitoring and Synchronization Routine

A system has only one monitoring and synchronization routine, which is called once every tick. The time consumed by the routine is so small that the routine can be finished in less than one tick. This routine has three responsibilities.

First, it enforces the creation of a job and taskPeriodISR to synchronize with a tick.

Second, it guarantees that all tasks spawn their first jobs during the same tick.

Third, this routine increments the execution time of a job which has the highest priority by one tick. If the time consumed by the job is equal to its execution time, this routine informs the job to depart from the system.

It should also be noted that it takes much less than one tick to create and delete one job, so the time consumed for creating and deleting a job is included in its execution time.

3.1.3 Periodic Interrupt Routine

Each task has its own taskPeriodISR routine. It interrupts and restarts the timer for the estimation task at the beginning of every super-period. The timer is restarted in every period for a normal task. When the timer for a specific task τ_i ($i = 0 \dots n$) expires, a signal is sent to the corresponding taskSpawn routine to create a job for the task. Once the job is created, the timer is restarted.

Note that each of the algorithms proposed in this thesis computes priorities of tasks/jobs for every super-period. The priority computation tries to achieve a high ratio for meeting deadlines. It is possible, however, that a low priority job may be in execution when the next super-period starts. This incomplete job is removed from the system at the beginning of the super-period. In certain systems partial execution of a task may not be acceptable. Such systems need to compare the execution times of jobs with the time left in the super-period and not dispatch a job to the CPU if it cannot be completed within the current super-period.

3.2 System Environment

The prototype is implemented on a VxWork simulator, called VxSim, developed by WindRiver. VxSim is a port of the VxWork operating system to the host environment. The host is a PC with a 1.70GHz, Pentium-IV CPU, 256M RAM, 40GB hard disk running under a Windows 2000 operating system. VxSim includes the following major components [37]:

- VxWorks real-time operating system for running time-critical or embedded applications. It provides facilities for real-time scheduling (pre-emptive, priority based) and inter-task communication. A number of other standard real-time facilities such as mechanisms for achieving mutual exclusion are included.
- Development tools that aid in testing, timing and debugging. These tools include an editor, a compiler and a loader. Facilities for source level debugging and performance monitoring are also included.

3.3 Verification of Measurement

The system was run on test cases for a short period of time. The measured performance metrics were compared with hand-computed metrics. The measured values were formed to agree with the hand-computed values.

The primary objective of the experiments performed in this research is to analyse the performance of the algorithms introduced in Chapter 2 and to compare them with the RMS algorithm. All experiments are performed on a dedicated system without any interference from other applications. The second part of this chapter discusses the experiments to be performed and their parameters.

3.4 System Characteristics

This section addresses the method by which system parameters are computed.

3.4.1 Number of Tasks in the System

The number of tasks in the system is an integer. Depending on the test, the system can consist of four, eight or ten tasks. An additional estimation task is required by the proposed algorithms. Using such a number of tasks leads to a reasonable time for completing all the experiments. Moreover with such a synthetic application it is possible to understand the relative performances of the scheduling algorithms.

3.4.2 Period for Each Task

The experiments employ a set of n periodic tasks, $\tau_1 \dots \tau_n$ which are listed in increasing order of their periods. As discussed in Chapter 1, it was shown by Liu that if the processor utilization exceeds the upper bound for a given n , the worst case scenario occurs when T_1

$< T_n < 2 \cdot T_1$. For this reason, two sets of task periods are used. For the first set, the condition of $1 < T_n/T_1 < 2$ is satisfied. The second set satisfies the condition of $T_n/T_1 \geq 2$.

The periods of the tasks are generated in the following way. For a set of n tasks, let r_i be the ratio T_{i+1}/T_1 ($i = 1 \dots n$). A set of r ($1 < r < 2$ or $r \geq 2$) is generated randomly by using a uniform distribution. When the period T_1 of τ_1 is known, a set of periods T_i for all tasks can be obtained. For example, if there are four tasks in a system, a sequence of three r values is obtained ($r_1 < r_2 < r_3$). So if the period T_1 of τ_1 is given, the period T_2 of τ_2 is $T_1 \cdot r_1$, and so on. The period of T_1 is a parameter specified by the user.

3.4.3 Computation Time of Each Task

The computation time for the estimation task depends on a number of factors including the number of tasks in the system. In all the experiments described in this thesis, the computation time for the estimation task is less than one tick. To achieve a conservative design the computation time of the estimation task is assumed to be one tick in the algorithms described in this thesis.

For other normal tasks, three cases need to be considered for determining the computation time.

Case 1) All tasks demand the same CPU utilization. That is, the ratios C_i/T_i for all τ_i are equal to one another. The C_i/T_i is computed as (CPU utilization requested by all tasks)/ n . For example, for a set of 4 tasks, if the CPU utilization requested by all tasks is 1.0, then the utilization for each task is 0.25. Since the period of each task is known from Section 3.4.2, the computation time of each task can be obtained. The CPU utilization requested by all tasks is an input parameter specified by the user and varies from 0.8 to 1.4 in the experiments described in this thesis. Note that although the CPU utilization

requested by all tasks can be higher than 1, the useful CPU utilization achieved can never exceed 1.

Case 2) The CPU utilization requested by each task is inversely proportional to T_i . Thus, the longer the period of a task, the smaller is its computation time. The computation time of each task is computed by using the following two equations:

$$C_i/T_i = b/(T_i^\alpha), \text{ where } 1 \leq i \leq n, b \in I, \alpha \in I \quad (10)$$

$$\sum_{i=1}^n (C_i/T_i) = \text{CPU utilization requested by all tasks} \quad (11)$$

In this thesis, α is held at 2, 4 or 6. Using the above two equations we get

$$\sum_{i=1}^n (C_i/T_i) = \sum_{i=1}^n (b/T_i^\alpha) = \text{CPU utilization requested by all tasks} \quad (12)$$

Given CPU utilization requested by all tasks, α and T_i , b can be computed from equation (12). Once b , α and T_i are known, C_i can be computed from equation (10).

Case 3) The CPU utilization requested by each task is directly proportional to T_i . Based on the following equations the computation time of each task can be computed using a procedure similar to that described earlier. α is held at 4 for the experiments described in this thesis.

$$C_i/T_i = b \cdot (T_i^\alpha), \text{ where } 1 \leq i \leq n, b \in I, \alpha \in I, \text{ and}$$

$$\sum_{i=1}^n (C_i/T_i) = \text{CPU utilization requested by all tasks.}$$

3.4.4 Super-Period of the Estimation Task

As shown in Section 2.3.2 of Chapter 2 the super-period of the estimation task is given by $T_0 = c \cdot \text{init-}T_0$ (See equation (2) in Chapter 2). We conducted a number of experiments

to determine a suitable value of c . Task periods and computation times are listed in Table 3-1.

Table 3-1 Computation Times and Periods for Tasks

	CPU utilization requested by all tasks = 0.9	CPU utilization requested by all tasks = 1.1
τ_1	$C_1 = 13 \quad T_1 = 60$	$C_1 = 16 \quad T_1 = 60$
τ_2	$C_2 = 19 \quad T_2 = 84$	$C_2 = 23 \quad T_2 = 84$
τ_3	$C_3 = 23 \quad T_3 = 102$	$C_3 = 28 \quad T_3 = 102$
τ_4	$C_4 = 27 \quad T_4 = 118$	$C_4 = 33 \quad T_4 = 118$

The threshold ratio for meeting deadlines is set at 90%. For any of the strategies PES-A, PES-B, TTF and a given CPU utilization requested by all tasks (0.9 or 1.1), c was varied from 1 to 8.

As the value of c is increased the average ratio for meeting deadlines for a given strategy was observed to fluctuate. However, the fluctuation seemed to be reduced to 0.1% or lower when a $c=3$ for any given scheduling algorithm and any given CPU utilization requested by all tasks. Therefore, in all the experiments repeated in Chapter 4, the value of c is fixed at 3.

3.4.5 Duration of an Experiment

The length of an experiment is called maxrun. The average ratio for meeting deadlines is computed by changing maxrun from $2 \cdot T_0$, $4 \cdot T_0$, $6 \cdot T_0$, ... to $16 \cdot T_0$. For each value of maxrun experiments were conducted for the parameters described in Table 3-1 and c was held at 3. The average ratios for meeting deadlines are computed for PES-A, PES-B and TTF for different values of maxrun. Note that the experiment is repeated 10 times for each maxrun and a mean of the average ratio for meeting deadlines is computed. This average ratio for meeting deadlines for a given algorithm and parameter set changes

initially when maxrun is increased. The difference between the average value obtained with $\text{maxrun} = N \cdot T_0$ and $\text{maxrun} = (N-1) \cdot T_0$ is observed to decrease as N increases. When maxrun is $10 \cdot T_0$ the difference in average ratios for meeting deadlines achieved with a given scheduling algorithm and a given CPU utilization requested by all tasks is 0.1% or lower for two successive values of maxrun. Thus, maxrun is held at $10 \cdot T_0$ in our experiments described in Chapter 4.

3.4.6 Threshold Ratio for Meeting Deadlines of Each Task

As discussed in Section 2.3.2 of Chapter 2, the threshold ratio for meeting deadlines is used to determine the priority of each job. Since different threshold ratios for meeting deadlines will affect the priorities of tasks, different average ratios for meeting deadlines will be obtained. Thus, the threshold ratio for meeting deadlines can affect system performance. As discussed in Section 2.3.2 of Chapter 2, the average ratio for meeting deadlines achieved with RMS $\pm 10\%$ is used as the range of threshold ratios for meeting deadlines in PES algorithms. For example, if the average ratio for meeting deadlines is 88% using RMS at CPU utilization requested by all tasks of 1.0, the range of threshold ratio for meeting deadlines is chosen from 98% to 78%. Since the best average ratio for meeting deadlines occurs at a threshold ratio of 87%, the threshold ratio for meeting deadlines in PES algorithms is fixed at 87%. Note that if the average ratio for meeting deadlines with RMS exceeds 90%, the upper limit of the threshold ratio range is capped at 100%.

3.5 Performance of Proposed Algorithms

In this thesis, there are seven test cases for comparing the performance of the different algorithms. They may be divided into three categories based on the distribution of the CPU utilization requested by each task.

In the first category, the CPU utilization requested by each task is the same, i.e. C_i/T_i has the same value for each task. C_i/T_i is computed as (CPU utilization requested by all tasks)/ n . This category includes three test cases:

- 1) For the first test case, the ratio of periods is between 1 and 2, i.e. $1 < T_n / T_1 < 2$.
- 2) For the second test case, the ratio of the longest and shortest periods is no less than 2, i.e. $T_n / T_1 \geq 2$.
- 3) For the third test case, the periods of all tasks are equal, i.e. $T_1 = T_2 = \dots = T_n$.

In the second category, the CPU utilization requested by each task is inversely proportional to the periods, i.e. $C_i/T_i = b / T_i^\alpha$. In other words, the longer the periods of tasks, the shorter are their computation times. When $1 < T_n / T_1 < 2$, there are three test cases based on the factor α :

- i) Low α : for the first test case $\alpha = 2$.
- ii) Medium α : for the second test case $\alpha = 4$.
- iii) High α : For the third test case $\alpha = 6$.

In the third category, the CPU utilization requested by each task is directly proportional to the periods, i.e. $C_i/T_i = b \cdot T_i^\alpha$. When $1 < T_n / T_1 < 2$ there is only one case with $\alpha = 4$.

Generally speaking, a system is designed for a normal situation instead of an overloaded situation to avoid wasting resources such as CPU. However, it is possible that sometimes a system becomes temporarily overloaded because of a failure or due to the

large variability in computation times of tasks in the system [4]. Although overloads on real systems are temporary we have used system models that are continuously subjected to a workload characterized by a CPU utilization requested by all tasks that is higher than 1. The scheduling policies that perform well in such an overloaded situation are expected to effectively handle temporary overloads. Therefore, in this thesis both a non-overloaded system and an overloaded system are considered.

Systems such as telephone switches are designed to handle higher loads experienced at the “busy hour” of a day. Sometimes the occurrence of unusual events creates a temporary increase in load on the system. For example, a natural calamity like an earth quake or a popular sporting event or a TV program can give rise to a large increase in the call intensity of a telephone switch. System designers would often design their systems to effectively handle the worst case scenario. For the experiments on the non-overloaded systems we have investigated systems with high loads. The CPU utilization requested by all tasks is varied from 0.8 to 1.0. For the experiments on the overloaded systems the CPU utilization requested by all tasks is varied from 1.05 to 1.4. Similar values of CPU utilization requested by all tasks are used by other researchers as well (See [4] for example). We set the start point at 0.8, since the equation shows that RMS is guaranteed to achieve a 100% ratio of meeting deadlines when the CPU utilization requested by all tasks is no more than 0.75 for the first category, and 0.72 for the other two categories. This is obtained by setting $n = 4$ for the first category and $n = 8$ for the other two categories in the expression for the upper bound on CPU utilization: $n(2^{1/n} - 1)$ presented in [25]. This utilization bound value is known to be conservative and is usually exceeded in practice [21]. So we use a start point of 0.8 for CPU utilization requested by all tasks.

The parameters used in the different experiments are briefly described.

- CPU utilization requested by all tasks is varied from 0.8 to 1.4.
- The number of tasks is 4 for the first category of test cases, and 8 for the other two.
- Super-period T_0 is fixed $3 \cdot \text{init-}T_0$.
- Maxrun is fixed at $10 \cdot \text{super-period } T_0$.
- Threshold ratio for meeting deadlines depends on the CPU utilization requested by all tasks.

Input parameters for RMS are the same as those for the proposed algorithms, except that super-period and the threshold ratio for meeting deadlines are not used in RMS.

The performance metrics computed for the proposed algorithms and RMS include:

- Guaranteed ratio for meeting deadlines, G
- Average ratio for meeting deadlines, $\bar{A}$
- Inter-task fairness metric, f
- Useful CPU utilization achieved, U

PES-A, PES-B, TTF and RMS are evaluated for each test case. Each experiment is repeated 10 times. The maximum difference between any average ratio for meeting deadlines and the mean computed over the 10 runs or any guaranteed ratio for meeting deadlines and the corresponding mean are observed to be less than or equal to 0.1%. The maximum difference between any CPU utilization achieved and the mean computed over the 10 runs is below 0.1% as well. The maximum difference between any inter-task fairness metrics and the corresponding mean is no more than 1%. The difference between any observation X and the mean M is computed as $(X - M) / M * 100\%$. The values of G ,

$\bar{A}$, U and f for a given parameter set are the average of the 10 values achieved in the 10 experiments.

3.6 Analysis of Overheads

The overhead introduced by the proposed algorithms is an important issue. From Chapter 2, it is known that all the proposed algorithms use an estimation task. This additional task will perform time estimation and other related operations during each super period. The ratio of the CPU time consumed by these additional operations and the whole length of an experiment is considered as overhead.

3.6.1 Analysis of Overheads

Based on the discussions of the PES algorithms presented in Section 2.3.2, Section 2.3.3 and Section 2.3.4 of Chapter 2, the overhead introduced by PES algorithms consists of three components. The first component is estimation, the calculation of time remaining for each job of the tasks at the beginning of each super period. This overhead component is increased if the number of jobs in a system is increased. The total number of jobs in turn depends on the total number of tasks and the enlargement factor for the super-period.

The second component is searching the list L described in Section 2.3.2. The size of L depends on the number of jobs that may miss their deadlines as estimated by the estimation task. When the number of tasks is increased, the CPU utilization requested by all tasks is increased, or the enlargement factor c for super-period is increased, the number of jobs missing deadlines can increase, thus affecting this overhead component.

The third component is computing the current ratio for meeting deadlines for each task at the beginning of every period. During each super-period, if the number of jobs is increased, this computation time is increased. Both the enlargement factor c for super-

period and the number of tasks affect the number of jobs in a super-period. So these factors affect this overhead component.

Based on the discussion of TTF algorithm in Section 2.4 of Chapter 2, at the beginning of each super-period, the estimation task computes the ratio for meeting deadlines of each task based on the result of the preceding super period. All tasks are sorted, in turn, based on their ratios for meeting deadlines. A higher priority is assigned to the task with lower ratio for meeting deadlines, and the converse. Thus, the possible overhead introduced by TTF algorithm consists of two components during each super-period.

The first component is computing the ratio for meeting deadlines for each task at the start of the super-period. As mentioned earlier, the enlargement factor c for super-period and the number of tasks affect this overhead component.

The second component is sorting the tasks by using their ratios for meet deadlines at the start of the super-period. The number of tasks in the system affects this overhead component.

Thus, the overhead introduced by the proposed algorithms depends on the number of tasks in the system and the enlargement factor c for super-period. The overheads of the PES algorithms are also affected by the CPU utilization requested by all tasks. When the CPU utilization requested by all tasks is very high, the number of jobs that are estimated to miss their deadlines is likely to increase. Since PES algorithms use a list of indices of jobs that may miss their deadlines, more time may be consumed during the searching of the list.

3.6.2 Experiments on Overhead

A number of experiments are conducted to analyse the overheads.

- 1) Experiment to analyse the effect of CPU utilization requested by all tasks on overhead:

In this experiment, the input parameters are:

- CPU utilization requested by all tasks is varied from 0.8 to 1.4.
- The number of tasks is 10.
- Super-period T_0 is $3 \cdot \text{init-}T_0$.
- Maxrun is $10 \cdot T_0$.
- Threshold ratio for meeting deadlines depends on the CPU utilization requested by all tasks.

- 1) Experiment to analyse the effect of the total number of tasks in the system on overhead:

In this experiment, the input parameters are the same as in 1), except that the number of the tasks is varied from 5 to 50, and the CPU utilization requested by all tasks is fixed at 0.9.

- 2) Experiment to analyse the effect of the periods of the tasks on overhead:

The C_i and T_i for each task τ_i are changed from one experiment to another. These values are varied in such a way that the CPU utilization requested by all tasks remains the same for each experiment.

- 3) Experiment to analyse the effect of the duration of the super-period on overhead:

The purpose of the experiment is to check whether changing the enlargement factor c affects the overhead. In this experiment, the input parameters are same as 1) except that the CPU utilization requested by all tasks is 0.9 and enlargement factor c is varied from 1 to 10.

Chapter 4 Results of Experiments

As explained in Chapter 3, two types of experiments were performed: experiments that focus on system performance and experiments that focus on the scheduling overhead. In this chapter, the results of these experiments are presented and analyzed.

Sections 4.1, 4.2 and 4.3 present the performance of all the proposed algorithms and RMS for various CPU utilizations requested by all tasks. Four performance indices (defined in Section 2.2 of Chapter 2) are used: guaranteed ratio for meeting deadlines, average ratio for meeting deadlines, inter-task fairness metric and useful CPU utilization achieved. Each of these sections presents the analysis of a non-overloaded system and an overloaded system. Section 4.4 focuses on the analysis of scheduling overhead. Section 4.5 summarizes the insights into system behavior gained from the results of the experiments.

4.1 Same Utilization of Resources by Tasks

In this section three test cases are described based on the periods of all the tasks. In the first test case, the ratio of periods is between 1 and 2. In the second test case, the ratio of the longest and shortest periods is no less than 2. In the third test case, the periods of all tasks are equal. For each test case, non-overloaded and overloaded systems are discussed separately.

4.1.1 The Ratio of Task Periods between 1 and 2 in a Non-overloaded System

4.1.1.1 Guaranteed Ratio for Meeting Deadlines

As shown in Figure 4-1, in comparison to RMS the guaranteed ratios for meeting deadlines for PES-A, PES-B and TTF are improved by 2.3%, 2.6% and 3.2% respectively

when the CPU utilization requested by all tasks is 0.9; by 12.2%, 12.8% and 19% when the CPU utilization requested by all tasks is 0.95; and by 24.7%, 26.2% and 30.5% when the CPU utilization requested by all tasks is 1.0. The results of all the proposed algorithms and RMS are comparable when the CPU utilization requested by all tasks is less than 0.9 since all tasks meet deadlines 100% of the time.

Among all the proposed algorithms, at higher values of CPU utilizations requested by all tasks, the TTF algorithm achieves a better guaranteed ratio for meeting deadlines than PES-A and PES-B. The performances of PES-A and PES-B are comparable.

Figure 4-1 Comparison of Guaranteed Ratios for Meeting Deadlines Achieved by PES-A, PES-B, TTF and RMS on a Non-overloaded System when $1 < T_n/T_1 < 2$

All the proposed algorithms have a much higher G than RMS at higher values of CPU utilizations requested by all tasks. RMS has a low G because it assigns the priorities to all tasks dependent on their periods only. For RMS, the tasks with shorter periods are always assigned higher priorities than the tasks with longer periods. Once the priorities are assigned, they are fixed. Even if the tasks cannot finish before their deadlines or have achieved threshold ratio for meeting deadlines, they still consume CPU. As a result, the

tasks with longer periods have fewer chances to run, and achieve low ratios for meeting deadlines.

The proposed algorithms attempt to overcome the disadvantage of RMS. For the PES algorithms, as discussed in Chapter 2, the estimation task estimates each job of all tasks at the start of each super-period. If the job of one task is observed to have insufficient time before its deadline, or the estimated ratio for meeting deadlines of a task is equal to or higher than its threshold ratio for meeting deadlines, the index of the job is stored in a list L.

When the job is dispatched to CPU, the list is checked and a different group priority may be assigned to it. The first-group priorities are higher than the second-group priorities, and the second-group priorities are higher than third-group priorities. If the job is not in the list L, it is assigned a priority value from the first group. If it is in the list L and the current ratio for meeting deadlines of the task is lower than the threshold ratio for meeting deadlines, it is assigned a priority value from the second group. If it is in the list L and the current ratio for meeting deadlines of the task is equal to or higher than the threshold ratio for meeting deadlines, it is assigned a priority value from the third group. In this way, the priority of a job can be lowered irrespective of its period when the PES algorithms are applied. As a result, CPU is first allocated to the tasks not achieving the threshold ratio for meeting deadlines if their jobs are estimated to have enough time to meet deadlines.

TTF considers both tasks with longer periods and shorter periods. A priority is assigned to each task based on its ratio for meeting deadlines for each task in the previous

super-period. A higher priority is always assigned to the task, which has a lower ratio for meeting deadlines in the previous super-period.

Among all the proposed algorithms, it is observed that TTF has the highest guaranteed ratio for meeting deadlines, and PES-B has a slightly higher guaranteed ratio for meeting deadlines than PES-A at the higher CPU utilizations requested by all tasks. As noted earlier, TTF algorithm assigns the highest priority to the task that has the lowest ratio for meeting deadlines. So the task with the lowest ratio for meeting deadlines always has a chance to increase its ratio for meeting deadlines during the time of an experiment. Thus, the guaranteed ratio for meeting deadlines, which is the lowest ratio for meeting deadlines among all tasks in a system, is increased.

The PES algorithms assign the priorities based on three factors: whether or not the index of the job is in the list L, current ratio for meeting deadlines, and the period of a task. Because of these factors the task with the lowest ratio for meeting deadlines may not be assigned a higher priority. As a result, the task having the lowest ratio for meeting deadlines may not be able to increase the ratio for meeting deadlines. PES-B demonstrates a slightly better result than PES-A in that PES-B uses a more precise estimation.

4.1.1.2 Average Ratio for Meeting Deadlines

As shown in Figure 4-2, the average ratios for meeting deadlines achieved by PES-A and PES-B are marginally better than that for RMS when the CPU utilizations requested by all tasks are 0.95 and 1.0. Their results are comparable with that of RMS when the CPU utilization requested by all tasks is less than 0.9. The average ratio for meeting deadlines achieved by TTF is 1.9% lower than that for RMS when the CPU utilization requested by all tasks is 1.0. TTF has a comparable average ratio for meeting deadlines

with RMS when the CPU utilization requested by all tasks is less than 1.0. Among all the proposed algorithms, PES-B has a slightly higher average ratio for meeting deadlines at higher values of CPU utilizations requested by all tasks.

Figure 4-2 Comparison of Average Ratios for Meeting Deadlines Achieved by PES-A, PES-B, TTF and RMS on a Non-overloaded System when $1 < T_n/T_1 < 2$

The PES algorithms obtain better results than RMS at higher values of CPU utilizations requested by all tasks since RMS always favors tasks with shorter periods. At higher values of CPU utilizations requested by all tasks, the tasks with longer periods have low ratios for meeting deadlines when RMS is used. Using the PES algorithms, the ratios for meeting deadlines of tasks with longer periods are increased, while the ratios for meeting deadlines of tasks with shorter periods may still achieve at least the threshold ratio for meeting deadlines. Thus, in the whole system, the average ratio for meeting deadlines achieved by the PES algorithms is slightly better than that of RMS at higher values of CPU utilizations requested by all tasks.

The TTF algorithm has a lower average ratio for meeting deadlines than RMS at CPU utilization requested by all tasks of 1.0. In TTF, tasks with lower ratios for meeting

deadlines during the previous super-period are assigned higher priorities in the current super-period. Thus, tasks with both shorter and longer periods may be assigned higher priorities and have opportunities to achieve higher ratios for meeting deadlines. Once the tasks are assigned higher priorities, they do not yield to other tasks irrespective of whether or not they can meet the deadlines or have already achieved the threshold ratio for meeting deadlines. So in the current super-period, while the tasks with higher priorities attain high ratios for meeting deadlines, other tasks may achieve low ratios for meeting deadlines.

RMS is different in that it always favors the tasks with shorter periods. So in general, the ratios for meeting deadlines of the tasks with longer periods achieved by TTF are higher than that by RMS. But the ratios for meeting deadlines of the tasks with shorter periods achieved by RMS are much higher than TTF. The overall effect is that TTF has a lower average ratio for meeting deadlines than RMS.

4.1.1.3 Useful CPU Utilization Achieved

As shown in Figure 4-3, the useful CPU utilizations achieved for PES-A and PES-B are marginally better than that for RMS when the CPU utilizations requested by all tasks are 0.95 and 1.0. Their results are comparable with that of RMS when the CPU utilization requested by all tasks is less than 0.9. The useful CPU utilization achieved for TTF is 1.7% lower than that for RMS when the CPU utilization requested by all tasks is 1.0. TTF has a useful CPU utilization achieved comparable to that of RMS when the CPU utilization requested by all tasks is less than 1.0.

In this test case, since the CPU utilization requested by each task is the same, the tasks with longer periods have greater computation times than the tasks with shorter periods. As discussed in Section 4.1.1.2, at higher values of CPU utilizations requested by all tasks,

more jobs of the tasks with longer periods meet their deadlines under the PES algorithms than under RMS. Therefore, the useful CPU utilizations achieved for the tasks with longer periods under the PES algorithms are higher than that under RMS. The number of jobs for the tasks with shorter periods that meet their deadlines may have slightly decreased under the PES algorithms because they yield to the tasks with longer periods. Therefore, the useful CPU utilizations achieved for the tasks with shorter periods under the PES algorithms is slightly lower than those achieved under RMS. Overall, the useful CPU utilizations achieved under the PES algorithms are higher than that under RMS at higher values of CPU utilization requested by all tasks.

Figure 4-3 Comparison of Useful CPU Utilizations Achieved by PES-A, PES-B, TTF and RMS on a Non-overloaded System when $1 < T_n/T_1 < 2$

The TTF algorithm produces a lower useful CPU utilization achieved than that achieved by RMS. As mentioned earlier, the CPU utilization requested by each task is the same. TTF has higher useful CPU utilizations achieved for the tasks with longer periods than RMS, because the ratios for meeting deadlines of these tasks are higher. However, since RMS has much higher ratios for meeting deadlines of the tasks with shorter periods

than TTF, the CPU utilizations achieved for the tasks with shorter periods achieved by RMS are higher than that achieved by TTF. Overall, TTF produces a lower useful CPU utilization achieved than that achieved by RMS.

4.1.1.4 Inter-task Fairness Metric

As shown in Figure 4-4, the inter-task fairness metric for PES-A, PES-B are comparable with RMS, and TTF is marginally better than RMS when the CPU utilization requested by all tasks is 0.9. In comparison to RMS, f for PES-A, PES-B and TTF are improved by 6%, 7% and 12% when the CPU utilization requested by all tasks is 0.95; and by 16%, 17% and 21% when the CPU utilization requested by all tasks is 1.0. When the CPU utilization requested by all tasks is less than 0.9, since all tasks can meet their deadlines, the values of inter-task fairness metric achieved by all the proposed algorithms and RMS are 0.

Among all the proposed algorithms, at higher values of CPU utilizations requested by all tasks, the TTF algorithm achieves a better inter-task fairness metric than PES-A and PES-B. PES-B achieved better inter-task fairness metric than PES-A.

All the proposed algorithms are fairer than RMS since they do not favor tasks with shorter periods as much.

Among all the proposed algorithms, TTF achieves the best result. The TTF algorithm always assigns higher priorities to the tasks with lower ratios for meeting deadlines. Hence, the ratio for meeting deadlines for each task is closer to the mean value. The PES algorithms differ from TTF in that they assign higher priorities to the tasks not achieving the threshold ratio for meeting deadlines if their jobs are estimated to have enough time to meet deadlines. These kinds of tasks may not have the lowest ratios for meeting deadlines.

PES-B achieves better results than PES-A because it has more precise estimation than PES-A.

Figure 4-4 Comparison of Inter-task Fairness Metric Achieved by PES-A, PES-B, TTF and RMS on a Non-overloaded System when $1 < T_n/T_1 < 2$

4.1.2 The Ratio of Periods between 1 and 2 in an Overloaded System

4.1.2.1 Guaranteed Ratio for Meeting Deadlines

As shown in Figure 4-5, the guaranteed ratios for meeting deadlines for PES-A, PES-B and TTF are improved 50% to 60% in comparison to that of RMS. When the CPU utilization requested by all tasks is equal to or above 1.2, the guaranteed ratio for meeting deadlines is 0 for RMS.

Among all the proposed algorithms, when the CPU utilization requested by all tasks is below or equal to 1.30, PES-B achieves a guaranteed ratio for meeting deadlines 2% to 6% higher than TTF and 1% to 3% higher than PES-A. When the CPU utilization requested by all tasks is 1.35 or 1.40, TTF achieves a guaranteed ratio for meeting deadlines that is 2% to 6% higher than the PES algorithms.

In an overloaded system, the disadvantage of RMS, which favors the tasks with shorter periods, is more obvious than in a non-overloaded system. Tasks with longer periods have fewer chances to occupy the CPU. When the CPU utilization requested by all tasks is very high, such tasks have little chance to run. So their ratios for meeting deadlines are very low, even zero. This accounts for the very low, even zero, guaranteed ratio for meeting deadlines achieved by RMS.

Figure 4-5 Comparison of Guaranteed Ratios for Meeting Deadlines Achieved by PES-A, PES-B, TTF and RMS on an Overloaded System when $1 < T_n/T_1 < 2$

For all the proposed algorithms, the ratio for meeting deadlines for each task is lower in an overloaded system than in a non-overloaded system. This is because in an overloaded system, all tasks are struggling to finish their jobs in time. However, the proposed algorithms attempt to treat each task fairly. As a result, their minimum ratios for meeting deadlines remain much higher than that achieved by RMS. This accounts for the higher guaranteed ratios for meeting deadlines achieved by the proposed algorithms.

Among all the proposed algorithms, when the CPU utilization requested by all tasks is below or equal to 1.30, PES-B achieves the best result. This is different from the result in

the non-overloaded system, in which TTF achieves the best result. For the TTF algorithm, when the tasks are assigned higher priorities in a given super-period, they achieve high ratios for meeting deadlines in that super period. However, they are assigned lower priorities in the next super-period because of their high ratios for meeting deadlines. At this point, the tasks with lower priorities still have a good chance to meet their deadlines in a non-overloaded system. This is the case because in a non-overloaded system, after tasks with higher priorities have finished their jobs, some time remains for running the tasks with lower priorities. Consequently, during the experiment the overall ratio for meeting deadlines for each task can remain high. This leads to a high G that is the minimum ratio for meeting deadlines achieved by a task in a non-overloaded system.

In an overloaded system, since the tasks with higher priorities are also struggling to finish their jobs in time, the tasks with lower priorities have fewer chances to run. This accounts for the low G achieved by TTF in an overloaded system. Since the tasks with longer periods have greater computation times, they are affected more seriously than the tasks with shorter periods when they are assigned lower priorities.

For the PES algorithms, the tasks with longer periods have a better chance to run than in the case of TTF. The reason is that when the tasks with shorter periods have achieved the threshold ratio for meeting deadlines, their priorities are lowered so that CPU is yielded to the tasks with longer periods. Thus, tasks with longer periods have chances to improve their ratios for meeting deadlines. This explains why PES-B achieves a better-guaranteed ratio for meeting deadlines than TTF.

It is noted that when the CPU utilizations requested by all tasks are 1.35 and 1.4, TTF has a higher guaranteed ratio for meeting deadlines than the PES algorithms. When a

system is overloaded so heavily, the computation times for tasks with both longer and shorter periods are very large. For the PES algorithms in a heavily overloaded situation, even the tasks with shorter periods may not be able to achieve the given threshold ratio for meeting deadlines. Thus the priorities of the tasks with longer periods remain at low levels. There is little time left for the tasks with longer periods to finish their work in time. It leads to low ratios for meeting deadlines for these tasks. Thus, the guaranteed ratio for meeting deadlines can be very low.

For TTF, a task that exhibits a low ratio for meeting deadlines in one super-period is assigned a higher priority in the next super-period during which it produces a higher ratio for meeting deadlines. As a result, the variability in the ratios for meeting deadlines exhibited by the tasks is smaller than that achieved by PES-B. Thus TTF demonstrates a higher performance for CPU utilizations requested by all tasks of 1.35 and 1.4.

4.1.2.2 Average Ratio for Meeting Deadlines

As shown in Figure 4-6, the average ratios for meeting deadlines achieved by PES-A, PES-B and TTF are 1% to 4% higher than that achieved by RMS, except when the CPU utilization requested by all tasks is 1.05. The average ratio for meeting deadlines achieved by TTF is 1.9% lower than that achieved by RMS when the CPU utilization requested by all tasks is 1.05.

For all values of CPU utilizations requested by all tasks, PES-B achieves an average ratio for meeting deadlines that is approximately 2% higher than that of TTF, and 1% higher than that of PES-A.

As discussed in Section 4.1.2.1, in an overloaded system the disadvantage of RMS, which favors the tasks with shorter periods, is more obvious than in a non-overloaded

system. The tasks with longer periods have fewer chances to run when the CPU utilization requested by all tasks is very high. So their ratios for meeting deadlines are very low, even zero. This explains why the average ratio for meeting deadlines achieved by RMS is lower than that achieved by the proposed algorithms.

Figure 4-6 Comparison of Average Ratios for Meeting Deadlines Achieved by PES-A, PES-B, TTF and RMS on an Overloaded System when $1 < T_n/T_1 < 2$

For all the proposed algorithms, the ratio for meeting deadlines for each task is lower in an overloaded system than in a non-overloaded system since in an overloaded system all tasks are struggling to finish their jobs in time. However, in comparison to RMS, the proposed algorithms still try to improve the ratios for meeting deadlines for the tasks with longer periods. Thus in the whole system, the average ratio for meeting deadlines achieved by the proposed algorithms is better than that achieved by RMS. The improvement made by the proposed algorithms in an overloaded system is greater than that in a non-overloaded system.

Among the proposed algorithms, PES-B performs better than PES-A because of its more precise estimation technique. The PES algorithms demonstrate better results than

TTF, which assigns higher and lower priorities to the tasks alternately. In TTF, once the tasks are assigned higher priorities, they never yield to other tasks with lower priorities in the current super-period. Consequently, a task can achieve very high ratio for meeting deadlines in the current super-period, and very low ratio for meeting deadlines in the next super-period. The PES algorithms attempt to make all tasks achieve the threshold ratio for meeting deadlines. Overall, the average ratios for meeting deadlines achieved by the PES algorithms are observed to be better than that achieved by TTF.

4.1.2.3 Useful CPU Utilization Achieved

As shown in Figure 4-7, in comparison to RMS, the CPU utilizations achieved by PES-A, PES-B and TTF are improved marginally, except when the CPU utilization requested by all tasks is 1.05. In this case, the useful CPU utilization achieved by TTF is comparable with that achieved by RMS.

Figure 4-7 Comparison of Useful CPU Utilizations Achieved by PES-A, PES-B, TTF and RMS on an Overloaded System when $1 < T_n/T_1 < 2$

Among all the proposed algorithms, the CPU utilizations achieved by PES-B are approximately 1% to 4% higher than those achieved by TTF and 1% to 2% higher than those achieved by PES-A.

In this test case since CPU utilization requested by each task is same, the tasks with longer periods have greater computation times than the tasks with shorter periods. Therefore, the useful CPU utilization achieved demonstrates the same trend as the average ratio for meeting deadlines. In other words, a high average ratio for meeting deadlines indicates more jobs meeting deadlines and a high useful CPU utilization achieved.

4.1.2.4 Inter-task Fairness Metric

Figure 4-8 Comparison of Inter-task Fairness Metric Achieved by PES-A, PES-B, TTF and RMS on an Overloaded System when $1 < T_n/T_1 < 2$

As shown in Figure 4-8, in comparison to RMS, the inter-task fairness metrics achieved by PES-A, PES-B and TTF show a 26% to 47% improvement. When CPU utilization requested by all tasks is 1.40, the inter-task fairness metric achieved by TTF is 5% better than those obtained by PES-A and PES-B. The inter-task fairness metrics achieved by the proposed algorithms are comparable when CPU utilization requested by all tasks is below

1.40. As discussed in Section 4.1.1.4, all the proposed algorithms attempt to treat all tasks fairly, so they are fairer than RMS.

4.1.3 The Ratio of Longest and Shortest Task Periods No Less Than 2 in a Non-overloaded System

The average ratio for meeting deadlines and useful CPU utilization achieved will not be discussed here because the relative performances of the algorithms are similar to those described in Section 4.1.1.2 and Section 4.1.1.3. The related graphs are included in Appendix A, Figure A-1, and Figure A-2.

4.1.3.1 Guaranteed Ratio for Meeting Deadlines

As shown in Figure 4-9, the guaranteed ratios for meeting deadlines for PES-A, PES-B and TTF are marginally better than that achieved by RMS when the CPU utilization requested by all tasks is 0.9. In comparison to RMS, G achieved by PES-A, PES-B and TTF are improved by 17.1%, 17.5% and 16.5% when the CPU utilization requested by all tasks is 0.95; and by 31%, 32% and 28.4% when the CPU utilization requested by all tasks is 1.0. The performances of all the proposed algorithms and RMS are comparable when the CPU utilization requested by all tasks is below 0.9.

Among all the proposed algorithms, at higher values of CPU utilization requested by all tasks, the PES algorithms achieve guaranteed ratios for meeting deadlines that are 1% to 3% better than those achieved by TTF. The results for PES-A and PES-B are comparable.

All the proposed algorithms achieve higher guaranteed ratios for meeting deadlines than that achieved by RMS for the same reasons discussed in Section 4.1.1.1.

In this test case, the guaranteed ratios for meeting deadlines achieved by the PES algorithms is slightly higher than those achieved by TTF. This is different from the test

case of $1 < T_n / T_l < 2$ described in Section 4.1.1.1 in which TTF achieves the best result. The results indicate that for higher variability in task periods the PES algorithms demonstrate a better performance than TTF.

Figure 4-9 Comparison of Guaranteed Ratios for Meeting Deadlines Achieved by PES-A, PES-B, TTF and RMS on a Non-overloaded System when $T_n/T_l \geq 2$

4.1.3.2 Inter-task Fairness Metric

As shown in Figure 4-10, the inter-task fairness metrics for PES-A, PES-B and TTF are comparable with RMS when the CPU utilization requested by all tasks is 0.9. In comparison to RMS, f for PES-A, PES-B and TTF demonstrate improvement of 10%, 10% and 12% when the CPU utilization requested by all tasks is 0.95; and of 20%, 21% and 21% when the CPU utilization requested by all tasks is 1.0. The inter-task fairness metrics for all the proposed algorithms and RMS are comparable when the CPU utilization requested by all tasks is less than 0.9.

All the proposed algorithms have better f at higher values of CPU utilization requested by all tasks in comparison to RMS since they try to consider all tasks, while RMS always favors the tasks with shorter periods. The lower values of CPU utilization requested by all

tasks has no effect on the result, since all tasks can meet deadlines 100%. However, at higher values of CPU utilization requested by all tasks, since all tasks cannot meet deadlines 100%, the disadvantage of RMS is exposed.

Figure 4-10 Comparison of Inter-task Fairness Metric Achieved by PES-A, PES-B, TTF and RMS on a Non-overloaded System when $T_n/T_1 \geq 2$

In this test case, the values of f achieved by each of the proposed algorithms are comparable. This is different from the test case for $1 < T_n / T_1 < 2$ described in Section 4.1.1.4 in that TTF achieves the best result. Thus a higher variability in task periods seems to be detrimental for the TTF algorithm.

4.1.4 The Ratio of Longest and Shortest Task Periods No Less Than 2 in an Overloaded System

The relative performances of the scheduling algorithms are the same as in the test case described in Section 4.1.3. No further discussion is thus provided. The graphs for the guaranteed ratio for meeting deadlines, average ratio for meeting deadlines, useful CPU utilization achieved and inter-task fairness metric are included in Appendix A, Figure A-3 to Figure A-6.

4.1.5 Equal Periods for All Tasks in a Non-overloaded System

4.1.5.1 Guaranteed Ratio for Meeting Deadlines

As shown in Figure 4-11, the guaranteed ratios for meeting deadlines achieved by all algorithms are the same when the CPU utilization requested by all tasks is below 1.0. When the CPU utilization requested by all tasks is 1.0, the guaranteed ratios for meeting deadlines achieved by the proposed algorithms are 0.1% lower than those achieved by RMS.

Figure 4-11 Comparison of Guaranteed Ratios for Meeting Deadlines Achieved by PES-A, PES-B, TTF and RMS on a Non-overloaded System when Task Periods are Equal to One Another

Since the CPU utilization requested by each task is the same and period of each task is the same, all tasks are expected to meet their deadlines 100% of the time when the CPU utilization requested by all tasks is below or equal to 1.0. Thus G achieved by RMS is 100%. For the proposed algorithms, because the estimation task consumes some CPU time, all tasks cannot meet their deadlines 100% of the time when the CPU utilization requested by all tasks is 1.0. G achieved by the PES algorithms and TTF are 99.9%.

4.1.5.2 Average Ratio for Meeting Deadlines

As shown in Figure 4-12, the average ratios for meeting deadlines achieved by all algorithms are the same when the CPU utilization requested by all tasks is below 1.0. When the CPU utilization requested by all tasks is 1.0, the average ratios for meeting deadlines achieved by the proposed algorithms are 0.1% lower than those achieved by RMS because of the same reasons discussed in Section 4.1.5.1.

Figure 4-12 Comparison of Average Ratios for Meeting Deadlines Achieved by PES-A, PES-B, TTF and RMS on a Non-overloaded System when Task Periods are Equal to One Another

4.1.5.3 Useful CPU Utilization Achieved

As shown in Figure 4-13, the useful CPU utilizations achieved by all algorithms are the same when the CPU utilization requested by all tasks is below 1.0. When the CPU utilization requested by all tasks is 1.0, the useful utilization achieved by the proposed algorithms are 0.1% lower than those achieved by RMS because of the same reasons discussed in Section 4.1.5.1.

Figure 4-13 Comparison of Useful CPU Utilizations Achieved by PES-A, PES-B, TTF and RMS on a Non-overloaded System when Task Periods are Equal to One Another

4.1.5.4 Inter-task Fairness Metric

Figure 4-14 Comparison of Inter-task Fairness Metric Achieved by PES-A, PES-B, TTF and RMS on a Non-overloaded System when Task Periods are Equal to One Another

As shown in Figure 4-14, the inter-task fairness metrics achieved by all algorithms are the same when the CPU utilization requested by all tasks is below 1.0. When the CPU utilization requested by all tasks is 1.0, the inter-task fairness metrics achieved by the

proposed algorithms are from 1% to 2% higher than those achieved by RMS because of the same reasons discussed in Section 4.1.5.1.

4.1.6 Equal Periods for All Tasks in an Overloaded System

Since there are no differences in the average ratio for meeting deadlines and useful CPU utilization achieved by the different algorithms, these performance metrics will not be discussed here. The guaranteed ratios for meeting deadlines and inter-task fairness metrics achieved by the proposed algorithms are higher than those by RMS. They don't require discussion either since the analysis is identical to that in Section 4.1.1.1 and Section 4.1.1.4. The graphs for G , $\bar{A}$, U and f are included in Appendix A, Figure A-7 to Figure A-10.

4.2 CPU Utilizations Requested by All Tasks Inversely Proportional to Periods of Tasks

In this section the relationship between the computation time and the period C_i/T_i is b/T_i^α ($1 \leq i \leq n$, α is an integer, b is a constant). In other words, the longer the period of a task, shorter is its computation time. The ratios of periods of tasks are held between 1 and 2. There are three test cases, each using a different value of α that is fixed at 2, 4 or 6 (See Section 3.5 for the details). For each test case, non-overloaded and overloaded systems are discussed separately. The relative performances of the scheduling algorithms are similar for the other values of α . The graphs for $\alpha = 2$ and $\alpha = 4$ are included in Appendix B.

4.2.1 The Ratio of Task Periods between 1 and 2 in a Non-overloaded System

4.2.1.1 Guaranteed Ratio for Meeting Deadlines

The graphs for G when α is 2 or 4 in a non-overloaded system are included in Appendix B, Figure B-1 and Figure B-4. The following discussion focuses on the case in which α is 6.

Figure 4-15 Comparison of Guaranteed Ratios for Meeting Deadlines Achieved by PES-A, PES-B, TTF and RMS on a Non-overloaded System when $C/T = O(1/T^6)$

As shown in Figure 4-15, the guaranteed ratios for meeting deadlines achieved by PES-A, PES-B and TTF demonstrate improvements of 5.5%, 5.4% and 5.8% over those achieved by RMS when the CPU utilization requested by all tasks is 0.90; of 9.4%, 8% and 7.3% when the CPU utilization requested by all tasks is 0.95; and of 22%, 24.5% and 20.9% when the CPU utilization requested by all tasks is 1.0. When the CPU utilization requested by all tasks is below 0.90, their performances are comparable to those of RMS. The reason for all the proposed algorithms having higher guaranteed ratios for meeting deadlines than RMS are the same as discussed in Section 4.1.1.1. The performance of

PES-A is observed to be close to that of PES-B. Both PES algorithms perform better than TTF.

4.2.1.2 Average Ratio for Meeting Deadlines

As shown in Figure 4-16, the average ratios for meeting deadlines achieved by PES-A, PES-B and TTF show improvements of 3.2%, 3.2% and 2.7%, respectively, over that achieved by RMS when the CPU utilization requested by all tasks is 0.90; of 2.4%, 1.9% and 1.5% when the CPU utilization requested by all tasks is 0.95; and of 9.7%, 10.1% and 6.9% when the CPU utilization requested by all tasks is 1.0. When the CPU utilization requested by all tasks is below 0.90, the performances of all the algorithms are comparable. Among the proposed algorithms, the PES algorithms perform better than TTF when the CPU utilization requested by all tasks is 1.0. The performances of the PES algorithms are comparable to one another. The rationale for such a behavior is similar to that provided in Section 4.2.1.1.

It is noted that when α is increased, the improvements demonstrated by all the proposed algorithms become greater (See Figure 4-16, 4-17 and 4-18). For example, at the CPU utilization requested by all tasks of 1.0, $\bar{A}$ achieved by PES-A, PES-B and TTF are improved by 1.8%, 2% and 0.8%, respectively, when α is 2; by 6.6%, 6.8% and 4.1% when α is 4; and by 9.7%, 10.1% and 6.9% when α is 6. When α is increased, the tasks with shorter periods have extremely large computation times. For RMS, after these tasks finish running, there is almost no time left; consequently, a lot of jobs in the tasks with longer periods miss their deadlines. So only the few tasks with very short periods can obtain high ratios for meeting deadlines. Other tasks have extremely low ratios for meeting

deadlines. When α is increased, this phenomenon becomes more pronounced, and $\bar{A}$ is lowered.

Figure 4-16 Comparison of Average Ratios for Meeting Deadlines Achieved by PES-A, PES-B, TTF and RMS on a Non-overloaded System when $C/T = O(1/T^6)$

Figure 4-17 Comparison of Average Ratios for Meeting Deadlines Achieved by PES-A, PES-B, TTF and RMS on a Non-overloaded System when $C/T = O(1/T^2)$

The proposed algorithms differ from RMS in that they don't always favor the tasks with shorter periods. For the PES algorithms, once the tasks with shorter periods achieve the threshold ratio for meeting deadlines, they yield to other tasks. For TTF, all task

periods do not directly influence the task priorities. As a result, the tasks with longer periods have more chances to run under the proposed algorithms than under RMS. Therefore, varying α has little effect on the performances of the PES algorithms and TTF.

Figure 4-18 Comparison of Average Ratios for Meeting Deadlines Achieved by PES-A, PES-B, TTF and RMS on a Non-overloaded System when $C/T = O(1/T^4)$

4.2.1.3 Useful CPU Utilization Achieved

The graphs for U when α is 2 or 4 in a non-overloaded system are included in Appendix B, Figure B-2 and Figure B-5. The following discussion focuses on the case in which α is 6.

As shown in Figure 4-19, the useful CPU utilization achieved for PES-A, PES-B and TTF is 2.9%, 2.8 and 1.5% lower respectively than that achieved by RMS when CPU utilization requested by all tasks is 0.95; 5.5%, 6.3% and 4.7% lower when CPU utilization requested by all tasks is 1.0.

As pointed out earlier, in contrast to RMS, the proposed algorithms do not explicitly favor the tasks with shorter periods. Thus, on one hand the proposed algorithms lead to higher ratios for meeting deadlines for tasks with longer periods than those achieved by

RMS. On the other hand, the ratios for meeting deadlines for tasks with shorter periods are smaller than those achieved with RMS. Since the tasks with shorter periods have very large computation times and the tasks with longer periods have smaller computation times, the CPU utilizations achieved by the proposed algorithms are lower than those by RMS. This trend becomes more obvious when α is increased, since the computation times for the tasks with shorter periods become greater.

Figure 4-19 Comparison of Useful CPU Utilizations Achieved by PES-A, PES-B, TTF and RMS on a Non-overloaded System when $C/T = O(1/T^6)$

4.2.1.4 Inter-task Fairness Metric

The graphs for f when α is 2 or 4 in a non-overloaded system are provided in Appendix B, Figure B-3 and Figure B-6. The following discussion focuses on the case in which α is 6.

As shown in Figure 4-20, the inter-task fairness metrics achieved by PES-A, PES-B and TTF are comparable with that achieved by RMS when the CPU utilization requested by all tasks is 0.90. In comparison to RMS, f achieved by PES-A, PES-B and TTF are improved by 5%, 5% and 4% when the CPU utilization requested by all tasks is 0.95; and

by 11%, 12% and 12% when the CPU utilization requested by all tasks is 1.0. The results for all the proposed algorithms are comparable to one another.

Figure 4-20 Comparison of Inter-task Fairness Metric Achieved by PES-A, PES-B, TTF and RMS on a Non-overloaded System when $C/T = O(1/T^6)$

When the CPU utilization requested by all tasks is higher than 0.85, all the proposed algorithms produce better f in comparison to RMS since they do not explicitly favor tasks with shorter periods. At lower values of CPU utilizations requested by all tasks can meet their deadlines 100% and all algorithms demonstrate the same performance. However, at higher values of CPU utilization requested by all tasks, since all tasks cannot meet deadlines 100% of the time, the disadvantage of RMS becomes apparent.

4.2.2 The Ratio of Task Periods between 1 and 2 in an Overloaded System

The relative performances of the scheduling algorithms are similar to those achieved with the test case described in Section 4.2.1 and will not be discussed any further. The graphs for the guaranteed ratio for meeting deadlines, average ratio for meeting deadlines, useful CPU utilization achieved and inter-task fairness metric are included in Appendix B, Figure B-7 to Figure B-18.

4.3 CPU Utilizations of Tasks Directly Proportional to Periods of Tasks

In this subsection the relationship between the computation time and the period of a task is given by

$$C_i/T_i = b \cdot T_i^\alpha \quad (\alpha = 4 \text{ and } b \text{ is a constant}).$$

In other words, the longer the task periods, the greater are their computation times. The ratios of task periods are held between 1 and 2. The results for the non-overloaded and overloaded systems are presented in the following subsections.

4.3.1 The Ratio of Task Periods between 1 and 2 in a Non-overloaded System

In this test case, the guaranteed ratios for meeting deadlines and inter-task fairness metrics for all the proposed algorithms are better than those achieved by RMS. Among all the proposed algorithms, PES-B performs the best in terms of G . The f values achieved by the proposed algorithms are comparable to one another. The rationale for the behaviors of the algorithms are the same as those presented in Section 4.1.3, and are not repeated. The related graphs are included in Appendix C, Figure C-1 and Figure C-2.

4.3.1.1 Average Ratio for Meeting Deadlines

As shown in Figure 4-21, in comparison to RMS, the average ratios for meeting deadlines achieved by PES-A, PES-B and TTF are 2.1%, 1.8% and 1.7% lower, respectively, when the CPU utilization requested by all tasks is 0.95; 1.8%, 1.4% and 1.5% lower when CPU utilization requested by all tasks is 1.0.

All the proposed algorithms obtain a lower $\bar{A}$ than RMS. Since the tasks with longer periods have very large computation times in this test case, finishing one such job results in many jobs of the tasks with shorter periods being missed. Thus, although the proposed algorithms have higher ratios for meeting deadlines for the tasks with longer periods than

does RMS, they have much lower ratios for meeting deadlines for the tasks with shorter periods. Overall the proposed algorithms result in a lower $\bar{A}$ than RMS.

Figure 4-21 Comparison of Average Ratios for Meeting Deadlines Achieved by PES-A, PES-B, TTF and RMS on a Non-overloaded System when $C/T = O(T^4)$

4.3.1.2 Useful CPU Utilization Achieved

As shown in Figure 4-22, the CPU utilizations achieved by PES-A, PES-B and TTF are comparable with that achieved by RMS when the CPU utilization requested by all tasks is 0.90. They are marginally better than RMS when the CPU utilization requested by all tasks is 0.95. In comparison to RMS, U achieved by PES-A, PES-B and TTF show improvement of 5.5%, 5.4% and 4.3% when the CPU utilization requested by all tasks is 1.0.

All the proposed algorithms produce better results than RMS. PES-B performs slightly better than PES-A and TTF. As discussed in Section 4.3.1.1, the ratio of meeting deadlines for tasks with longer periods achieved by any proposed algorithm is higher than that achieved by RMS. This indicates that for the proposed algorithms, a large number of jobs in tasks with longer periods meet their deadlines in comparison to that for RMS. In

this test case, tasks with longer periods have very large computation times. Thus, although the $\bar{A}$ achieved by the proposed algorithms is lower than that achieved by RMS, the CPU utilizations achieved for the proposed algorithms are still higher than that attained by RMS.

Figure 4-22 Comparison of Useful CPU Utilizations Achieved by PES-A, PES-B, TTF and RMS on a Non-overloaded System when $C/T = O(T^4)$

4.3.2 The Ratio of Periods between 1 and 2 in an Overloaded System

The relative performances of the scheduling algorithms are the same as described in Section 4.3.1. The graphs for the guaranteed ratio for meeting deadlines, average ratio for meeting deadlines, useful CPU utilization achieved and inter-task fairness metric are included in Appendix C, Figure C-3 to C-6.

4.4 Analysis of Overhead

From Chapter 2, it is known that all the proposed algorithms employ an estimation task. This additional task performs computations during each super-period. The ratio of the CPU time consumed by these additional computations and the duration of an

experiment is the overhead associated with a scheduling algorithm. The system parameters for the analysis of the overhead are described in Section 3.6.2.

4.4.1 Overhead for Various CPU Utilization Requested by All Tasks

As shown in Figure 4-23, the overhead is increased for the PES algorithms when the CPU utilization requested by all tasks is increased. The overhead for PES-A varies from 0.0026% to 0.0046%. The overhead for PES-B varies from 0.0036% to 0.0059%. For TTF it remains almost constant when the CPU utilization requested by all tasks is increased. However, the overheads of all the proposed algorithms are small for the workload experimented with.

For the PES algorithms, when the CPU utilization requested by all tasks is increased, more jobs can be estimated to miss their deadlines. As a result, the list storing the indices of these jobs becomes longer. In turn, more time is consumed in searching the list. Consequently the overhead increases with an increase in the value of CPU utilization requested by all tasks.

Figure 4-23 Overhead Introduced by the Proposed Algorithms

As discussed in Section 3.6.1 of Chapter 3, for TTF at the beginning of each super-period, the estimation task computes the ratio for meeting deadlines for each task based on the results of the preceding super-period. Then all tasks are sorted based on their ratios for meeting deadlines. A higher priority is assigned to the task with lower ratio for meeting deadlines. Thus two factors affect the overhead for TTF: the calculation of the ratio for meeting deadlines and the sorting of those ratios for all tasks. Neither of these is affected directly by the CPU utilization requested by all tasks. As a result, the overhead remains constant when the value of CPU utilization requested by all tasks is increased.

4.4.2 Overhead for Various Numbers of Tasks

As shown in Figure 4-24, the overhead for the PES algorithms increases with the number of the tasks. This agrees with the observation made in Section 3.6.1 of Chapter 3. For TTF the overhead seems to be insensitive to the number of the tasks.

Figure 4-24 Overhead Introduced by Different Proposed Algorithms with Various Number of Tasks

4.4.3 Overhead for Various Periods of All Tasks

Figure 4-25 Overhead Introduced by Different Proposed Algorithms with Various Periods of the Tasks

The purpose of this experiment is to check whether the frequency of all tasks sending requests may affect the overhead for a given CPU utilization requested by all tasks. The period multiplication factor in Figure 4-25 is an integer that is used to multiply the computation time and period of each task. With a different multiplication factor task periods are changed but the value of CPU utilization requested by all tasks remains fixed. As expected, for all the proposed algorithms, the overhead decreases with an increase of task periods. This is because increasing task period decreases the frequency of run for the estimation task. Among all the proposed algorithms, TTF has the lowest overhead.

4.4.4 Overhead for Various Super-Periods for the Estimation Task

As shown in Figure 4-26, when the enlargement factor c for super-period increases, the overheads for the PES algorithms remain at almost constant values. The overhead for TTF

is observed to decrease slightly with an increase in c . The overheads for all the proposed algorithms are observed to be small.

Figure 4-26 Overhead Introduced by Different Proposed Algorithms with Various Super-periods

As discussed in Section 3.6.1, an increase in the enlargement factor c leads to an increase in the number of the jobs in a super-period as well as the duration of the super-period. The time consumed by the estimation task increases with the number of jobs in a super-period for the PES algorithms. For the PES algorithms, the increase in the time consumed by estimation task seems to be balanced out by an increase in the duration of the super-period. As a result the overhead does not change significantly with a variation in c .

The computation time for the estimation task used by TTF does not change with the number of jobs in a super-period. As a result when c increases the duration of the super-period increases leading to a decrease in the overhead.

4.4.5 Discussion on Overhead

The analysis of the overheads leads to the following conclusions for the experiments presented in this thesis.

- The value of CPU utilization requested by all tasks does not significantly affect the overhead introduced by the proposed algorithms. Compared to other factors, such as the number of tasks, the effect of the varying value of CPU utilization requested by all tasks on overhead is small.
- The number of tasks in the system has different effects on the overhead for different proposed algorithms. For PES-A and PES-B, the overhead increases with the number of tasks. For TTF, the overhead seems to be fairly insensitive to the number of tasks.
- When the periods of all tasks are decreased at the same value of CPU utilization requested by all tasks, the overhead for all the proposed algorithms is increased.
- When the super-period is extended to include more jobs, the overheads for the PES algorithms do not change significantly, but the overhead for TTF decreases slightly.
- The overheads introduced by PES-A and PES-B are small and acceptable for the workloads experimented with. For example, when the number of the tasks is 50 and the CPU utilization requested by all tasks is 0.9, the overhead is 0.02% for PES-A and 0.06% for PES-B respectively. The overhead introduced by TTF is much lower than the overhead introduced by the PES algorithms for the workload experimented with.

4.5 Summary of Results and Insights into System Behavior

The summary of experimental results and the insights developed from a complete analysis of the experiments described in this chapter are presented in this section.

- Guaranteed ratio for meeting deadlines: For most experiments the guaranteed ratios for meeting deadlines achieved by all the proposed algorithms are higher than that achieved by RMS when the CPU utilization requested by all tasks is above 0.9. Except in three cases, PES-B has the highest guaranteed ratio for meeting deadlines. In the first case the CPU utilizations requested by all tasks are equal with $1 < T_w / T_l < 2$ in a non-overloaded system (See Section 4.1.1.1). The second case is the same as the first case except that the system is overloaded with the CPU utilization requested by all tasks is 1.35 or higher (See Section 4.1.2.1). In the third case the CPU utilizations requested by all tasks are equal with tasks of equal periods running on an overloaded system (See Section 4.1.6).
- Average ratio for meeting deadlines: For most experiments the average ratios for meeting deadlines achieved by all the proposed algorithms are higher than that achieved by RMS when CPU utilization requested by all tasks is above 0.9. This observation does not hold in three cases. In the first case the CPU utilization requested by each task is same, and the periods of all tasks are equal. When the CPU utilizations requested by all tasks is equal to 1.0 on such a system, the $\bar{A}$ values for the proposed algorithms are lower than that for RMS. At other levels of CPU utilizations requested by all tasks, the $\bar{A}$ values achieved by the proposed algorithms are the same as those achieved by RMS (See Section 4.1.5.2). In the second case, $C_i / T_i = b \cdot T_i^4$. At all CPU utilizations requested by all tasks, the $\bar{A}$ values achieved by the proposed algorithms are lower than those achieved by RMS (See Section 4.3.1.1 and Section 4.3.2). The third case concerns a non-overloaded system in which the CPU utilizations requested by all tasks are equal and satisfies the inequality: $1 < T_w / T_l < 2$. Here, the $\bar{A}$ value

achieved by TTF is lower than that achieved by RMS (See Section 4.1.1.2). Among all the proposed algorithms, PES-B has the best average ratio for meeting deadlines. In some test cases, however the results for PES-A are comparable to those of PES-B.

- Useful CPU utilization achieved: All the proposed algorithms achieve a higher CPU utilization than RMS when CPU utilization requested by all tasks is above 0.9, except in three cases. In the first case the CPU utilization requested by all tasks are equal, and $T_1 = T_2 \dots = T_n$. On such a system when the CPU utilization requested by all tasks is equal to 1.0, the CPU utilizations achieved by the proposed algorithms are lower than that achieved by RMS. At other levels of CPU utilizations requested by all tasks, the CPU utilizations achieved by the proposed algorithms are the same as those achieved by RMS (See Section 4.1.5.3). In the second case $C_i/T_i = b/T_i^\alpha$, for all the values of CPU utilization requested by all tasks, the CPU utilizations achieved by the proposed algorithms are lower than that achieved by RMS (See Section 4.2.1.3, Section 4.2.2). The third case focuses on a non-overloaded system in which the CPU utilizations requested by all tasks are equal and satisfies the inequality $1 < T_n/T_1 < 2$. In this case, the useful CPU utilization achieved by TTF is lower than that achieved by RMS (See Section 4.1.1.3). In most cases, PES-B has the best useful CPU utilization achieved among all the proposed algorithms. In some test cases, however, the U achieved by PES-A is comparable to that achieved by PES-B.
- Inter-task fairness metric: All the proposed algorithms are fairer than RMS when CPU utilization requested by all tasks is above 0.9. An exception is the case in which the CPU utilization requested by all tasks for all tasks is 1.0, the CPU utilizations requested by all tasks are equal, and $T_1 = T_2 \dots = T_n$ (See Section 4.1.5.4). In most

cases, the results for the proposed algorithms are comparable. However, TTF is fairer than the PES algorithms in two cases. In the first case, the system is not overloaded, the CPU utilizations requested by all tasks are equal and $1 < T_n / T_1 < 2$ is satisfied (See Section 4.1.1.4). In the second case, the system is overloaded, the CPU utilization requested by all tasks are equal and $T_1 = T_2 \dots = T_n$ (See Section 4.1.6).

4.6 Discussion

Based on the preceding analysis, in a non-overloaded system, when CPU utilization requested by all tasks is below 0.9, the G , $\bar{A}$, U and f obtained by the proposed algorithms are comparable to those for RMS. At higher values of CPU utilizations requested by all tasks, in most cases all the proposed algorithms have better G , $\bar{A}$, U and f than RMS. The improvement in $\bar{A}$ and U achieved by the proposed algorithms are lower than the improvement in G and f . The results indicate that in comparison with RMS, the PES algorithms can improve G up to 30% (See Figure 4-1 for example); $\bar{A}$ up to 10% (See Figure 4-16 for example); U up to 5% (See Figure 4-22 for example); and f up to 20% (See Figure 4-4 for example).

In an overloaded system, the G , $\bar{A}$, U and f achieved by all the proposed algorithms are significantly higher than those achieved by RMS, except for the special cases mentioned earlier. Compared to RMS, the PES algorithms can improve G up to 60% (See Figure 4-5 for example); improve $\bar{A}$ up to 50% (See Figure B-16 for example); improve U up to 10% (See Figure C-5 for example); improve f up to 50% (See Figure A-6 for example).

Among all the proposed algorithms, TTF is a better algorithm in terms of inter-task fairness metric. In most cases, PES-B is a better algorithm than the others in terms of guaranteed ratio for meeting deadlines, average ratio for meeting deadlines and useful

CPU utilization achieved. In some cases, however, PES-A and PES-B demonstrate comparable guaranteed ratios for meeting deadlines, average ratios for meeting deadlines, and useful CPU utilizations achieved.

System overhead can be affected by the value of CPU utilization requested by all tasks, the number of tasks, and the periods of all tasks and the length of the super-period. However, in the test cases described, the overheads for all the proposed algorithms are observed to be small. Furthermore, TTF has a lower overhead than the PES algorithms. For example, when the number of the tasks is 50 and the CPU utilization requested by all tasks is 0.9, the overhead is 0.02% for PES-A, 0.06% for PES-B, and 0.00018% for TTF.

Chapter 5 Conclusions

5.1 Summary

In this thesis we propose three algorithms for firm real-time system. Four performance indices are used to evaluate system performance: guaranteed ratio for meeting deadlines, average ratio for meeting deadlines, inter-task fairness metric and useful CPU utilization achieved. The performances of the proposed algorithms are compared with that of RMS for various combinations of workload and system parameters.

Our experiments revealed a number of shortcomings for the well known RMS technique.

First, RMS cannot always produce a reasonable guaranteed ratio for meeting deadlines. In the experiment with overloaded systems, the guaranteed ratio for meeting deadlines could even be 0 (See Figure 4-5 for example).

Second, RMS can lead to low average ratios for meeting deadlines. The tasks with shorter periods may use much more CPU time than those with longer periods even if they cannot finish before their deadlines or have already achieved a current ratio for meeting deadlines equal to or higher than the threshold ratio for meeting deadlines. The tasks with longer periods, therefore, have fewer chances to run. Hence, the average ratio for meeting deadlines of the system can be low at higher values of CPU utilization requested.

Third, RMS is not fair to all tasks in the system. While tasks with shorter periods may be able to meet deadlines 100% of the time, all the jobs in tasks with longer periods can miss their deadlines. This behavior downgrades inter-task fairness.

Finally, RMS can give rise to a low useful CPU utilization, since RMS assigns priorities to tasks only according to their periods. In many cases, even if there is no possibility of meeting the deadlines, the jobs from these tasks are still assigned higher priorities, preempting other jobs from tasks with longer periods. Therefore, a large amount of CPU time may be wasted. In some cases, however, RMS produces higher useful CPU utilization achieved or a higher average ratio for meeting deadlines (See Figure 4-19 and Figure 4-21 for example).

In this thesis, three algorithms are proposed to try to overcome the disadvantage of RMS. These algorithms are the Pre-Execution Estimation Based Scheduling (PES) Algorithms and the TTF algorithm. In the PES algorithms, an extra task called estimation is added to the system and assigned the highest priority. This task is assigned a period, called a super-period, which is much longer than the period of any other task in the system. The estimation task is called at the start of every super-period. The purpose of adding the estimation task is to determine at the beginning of every super-period whether a job has sufficient time to meet its deadline and whether the estimated ratio for meeting deadlines of a task is equal to or higher than the threshold ratio for meeting deadlines. If the job doesn't meet the two conditions, its index is stored in a list of lower priority jobs.

Three groups of priorities are defined according to the list from estimation and the current ratio for meeting deadlines of each task. Before any job is dispatched to the CPU it is assigned an appropriate priority. A job is assigned a priority value from the first group if its index is not in the list. It is assigned a priority value from the second group if its index is in the list but the task's current ratio for meeting deadlines is lower than the threshold ratio for meeting deadlines. It is assigned a priority value from the third group otherwise.

Within each group of priority, the priority values assigned to jobs depend on the task periods. The higher the period of a task, the lower is the priority.

In general, for PES algorithms priorities do not depend only on the lengths of their periods. The priority of a task may change from one job to another. Two versions of PES algorithms, PES-A and PES-B, are experimented with. PES-B uses a more precise estimation technique than PES-A.

In the Tardy Task First (TTF) algorithm, the estimation task is similarly added to the system and assigned the highest priority. In TTF, the estimation task computes the ratio for meeting deadlines of each task in the previous super-period. Based on this information a priority is assigned to each task in the current super-period: lower the ratio for meeting deadlines of a task, higher is its priority.

At the start of its use, since no tasks have run, the ratio for meeting deadlines for each task is 0. Each task is, therefore, assigned the same priority. TTF always favors the tasks that have lower ratio for meeting deadlines; hence, the priorities of all tasks may be changed at the beginning of each super-period. However, the priorities of all tasks remain fixed throughout a given super-period.

A large number of experiments were conducted to compare the performances of the four strategies. The results of these experiments are discussed in the next section.

5.2 Insights into System Performance

A number of experiments were conducted for a range of workload parameters. The insights gained into system behavior and performances are summarized.

- RMS is known to be an unfair strategy. An important observation made from the experimental results is that, in almost all the experiments the inter-task fairness metric

achieved by the three proposed strategies is superior to that achieved by RMS. Thus, the three strategies can lead to an effective solution on systems in which fairness is an important concern.

- In a non-overloaded system, when CPU utilization requested is below 0.9, the proposed algorithms are comparable with RMS. In most cases, with a higher CPU utilization requested by all tasks, all proposed algorithms demonstrate a better performance in comparison to RMS. The average ratios for meeting deadlines and useful CPU utilizations achieved by all proposed algorithms are slightly higher than those achieved by RMS. The guaranteed ratios for meeting deadlines and inter-task fairness metric achieved by all proposed algorithms are significantly higher than those achieved by RMS. For example, in comparison to RMS, PES algorithms can improve the guaranteed ratio for meeting deadlines by as much as 30% (See Figure 4-1 for example), the average ratio for meeting deadlines by as much as 10% (See Figure 4-16 for example), the useful CPU utilization achieved by as much as 5% (See Figure 4-22 for example), and the inter-task fairness metric by as much as 20% (See Figure 4-4 for example).
- In an overloaded system, the performance indices achieved by all the proposed algorithms are, in most cases, significantly higher than those achieved by RMS. For example, in comparison to RMS, the PES algorithms can improve the guaranteed ratio for meeting deadlines by up to 60% (See Figure 4-5 for example), the average ratio for meeting deadlines by up to 50% (See Figure B-16 for example), the useful CPU utilization achieved by up to 10% (See Figure C-5 for example), and inter-task fairness metric by up to 50% (See Figure A-6 for example).

- In most cases, TTF is better than the other algorithms in terms of the inter-task fairness metric. In most cases, PES-B is a better algorithm than the others in terms of guaranteed ratio for meeting deadline, average ratio for meeting deadline and useful CPU utilization achieved. In some cases, though, PES-A and PES-B are comparable in terms of guaranteed ratio for meeting deadlines, average ratio for meeting deadlines and useful CPU utilization achieved.
- The overheads for all proposed algorithms are observed to be small for the synthetic workload used in this thesis. The overhead for TTF is observed to be lower than the PES algorithms. PES-A that uses a less accurate method for assigning task priorities incurs a lower overhead in comparison to PES-B.

5.3 Future Work

The threshold ratio for meeting deadlines can affect priorities of the tasks; consequently, it can affect system performance. In this thesis it is assumed that all tasks have the same threshold ratio for meeting deadlines. How the threshold ratio for meeting deadlines affects system performance and how to determine exactly the appropriate threshold ratio for meeting deadlines warrant further investigation.

This thesis focused on a single processor-based system. In the future, more research work should be done on firm/soft real time systems implemented on multiple processors. We have considered only periodic tasks in this thesis. Extension of the algorithms for handling system with both periodic and aperiodic tasks seems to be an interesting direction for further research.

References

- [1] A.K. Atlas and A. Bestavros, "Statistical Rate Monotonic Scheduling," in *Proceedings of the 19th IEEE Real-Time Systems Symposium*, Madrid, Spain, Dec. 1998, pp. 123-132.
- [2] A.K. Atlas and A. Bestavros, "Slack Stealing Job Admission Control," Technical Report BUCS-TR-98-009, Boston University, Computer Science Department, 1998.
- [3] A.K. Atlas, *Statistical Rate Monotonic Scheduling: Quality of Service through Management of Variability in Real-time Systems*, PhD thesis, Boston University, June 1999.
- [4] A. Atlas and A. Bestavros, "Multiplexing VBR Traffic Flows with Guaranteed Application-level QoS Using Statistical Rate Monotonic Scheduling," in *Proceedings of the 8th International Conference on Computer Communications and Networks*, Boston, Massachusetts, Oct 1999, pp. 282-287.
- [5] A. Atlas and A. Bestavros, "Design and Implementation of Statistical Rate Monotonic Scheduling in KURT Linux," in *Proceedings of the 20th IEEE Real-Time Systems Symposium*, Phoenix, AZ, USA, Dec 1999, pp. 272-276.
- [6] N.C. Audsley, "Applying New Scheduling Theory to Static Priority Pre-emptive Scheduling," *Software Engineering Journal*, Vol. 8, No. 5, Sept. 1993, pp. 284-292.
- [7] N.C. Audsley, A. Burns, R. I. Davis, K.W. Tindell and A.J. Wellings, "Fixed Priority Pre-emptive Scheduling: An Historical Perspective," *Real-Time Systems (Special Issue on the History of Real-Time Systems)*, Vol. 8, No. 2, 1995, pp. 173-

198.

- [8] H. Aydin, R. Melhem and D. Mosse, "Optimal Reward-Based Scheduling of Periodic Real-Time Tasks," *IEEE Transactions on Computers*, Vol.50, No. 2, 2001, pp. 111-130.
- [9] G. Bernet, A. Burns and A. Llamosi, "Weakly Hard Real-Time Systems," *IEEE Transactions on Computers*, Vol.50, No. 4, 2001, pp. 308-321.
- [10] A.A. Bertossi and A. Fusiello, "Rate Monotonic Scheduling for Hard-Real-Time Systems," *European Journal of Operational Research*, Vol. 96, 1997, pp. 429-443.
- [11] G.C. Buttazzo, *Hard Real-time Computing Systems: predictable scheduling algorithms and applications*, Kluwer Academic Publishers, Boston. 1997.
- [12] M.K. Gardner and J.W. Liu, "Analyzing Stochastic Fixed-Priority Real-Time Systems," in *Proceedings of Tools and Algorithms for the Construction and Analysis of Systems/European Joint Conferences on Theory and Practice of Software'99 (TACAS/ETAPS'99)*, Amsterdam, Netherlands, Mar 1999, pp. 44-58.
- [13] M.K. Gardner and J.W.S. Liu, "Performance of Algorithms for Scheduling Real-Time Systems with Overrun and Overhead," in *Proceedings of the 11th Euromicro Conference on Real-Time Systems*, University of York, York, England, Jun 1999, pp. 287-296.
- [14] W.A. Halang, *Real Time Computing*, Springer-Verlag, 1994, pp. 762.
- [15] R. Jain, *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation and Modelling*, John Wiley & Sons, Inc. 1991.

- [16] M.H.Klein, T.Ralya, B.Pollak, R.Obenza and M.G.Harbour, *A Practitioner's Handbook for Real-Time Analysis: Guide to Rate Monotonic Analysis for Real-Time Systems*, Kluwer Academic Publishers, Dordrecht, the Netherlands, 1993.
- [17] P.A. Laplante, *Real-Time Systems Design and Analysis: An Engineer's Handbook*, IEEE Computer Society Press, 1992.
- [18] S. Lauzac, R. Melhem and D. Mosse, "Comparison of Global and Partitioning Schemes for Scheduling Rate Monotonic Tasks on a Multiprocessor," in *Proceedings of the 10th Euromicro Workshop on Real-Time Systems*, Berlin, Germany, Jun 1998, pp. 188-195.
- [19] S. Lauzac, R. Melhem and D. Mosse, "An Efficient RMS Admission Control and its Application to Multiprocessor Scheduling," in *Proceedings of the 12th International Parallel Processing Symposium*, Orlando, USA, Apr 1998, pp. 511-518.
- [20] J.P. Lehoczky, L.Sha and J.K. Strosnider, "Enhanced aperiodic responsiveness in hard real-time environments," in *Proceedings of the 8th IEEE Real-Time Systems Symposium*, San Jose, California, USA, Dec 1987, pp. 261-270.
- [21] J. P. Lehoczky and L. Sha, Y. Ding, "The Rate Monotonic Scheduling Algorithm: Exact Characterization And Average Case Behavior," in *Proceedings of the IEEE Real-Time Systems Symposium - 1989*, Santa Monica, California, USA, Dec 1989, pp. 166-171.
- [22] J.P. Lehoczky, "Fixed Priority Scheduling of Periodic Task Sets with Arbitrary Deadlines," in *Proceedings of the 11th Real-Time Systems Symposium*, Lake Buena Vista, Florida, USA, Dec 1990, pp. 201-209.

- [23] J. P. Lehoczky and Sandra Ramos-Thuel, "An Optimal Algorithm for Scheduling Soft-Aperiodic Tasks in Fixed-Priority Preemptive Systems," in *Proceedings of the Real-Time Systems Symposium - 1992*, Phoenix, Arizona, USA, Dec 1992, pp. 110-123.
- [24] J.Y.T. Leung and J. Whitehead, "On the Complexity of Fixed-Priority Scheduling of Periodic, Real-Time Tasks," *Performance Evaluation*, Vol. 2, No. 4, 1982, pp. 237-250.
- [25] C.L. Liu and J.W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment," *Journal of the Association for Computing Machinery*, Vol. 20, No. 1, 1973, pp. 46-61.
- [26] Y. Oh and S. Son, "Allocating Fixed-Priority Periodic Tasks on Multiprocessor Systems," *Real-Time Systems Journal*, Vol. 9, 1995, pp. 207-239.
- [27] M. Spuri and G. Buttazzo, "Efficient aperiodic service under earliest deadline scheduling," in *Proceedings of the IEEE Real-Time Systems Symposium*, San Juan, Puerto Rico, Dec 1994, pp. 2-11.
- [28] M. Spuri and G. Buttazzo, "Scheduling aperiodic tasks in dynamic priority systems," *Journal of Real-Time Systems*, Vol. 10, No. 2, May 1996, pp. 179-210.
- [29] L. Sha, R. Rajkumar and J.P. Lehoczky, "Priority inheritance protocols: An approach to real-time synchronization," *IEEE Transactions on Computers*, Vol. 39, No. 9, Sept 1990, pp. 1175-1185.
- [30] B. Sprunt, L. Sha, and J. Lehoczky, "Aperiodic task scheduling for hard-real-time systems," in *Journal of Real-Time Systems*, Vol. 1, Jul 1989, pp. 27-60.

- [31] J.A. Stankovic, M. Spuri, K. Ramamritham and G.C. Buttazzo, *Deadline Scheduling For Real-Time Systems: EDF and Related Algorithms*, Kluwer Academic Publishers, 1998.
- [32] J. K. Strosnider, J. P. Lehoczky and L. Sha, "The deferrable-server algorithm for enhancing aperiodic responsiveness in hard real-time environments," *IEEE Transactions on Computers*, Vol. 4, No. 1, Jan 1995, pp. 73-91
- [33] A.S. Tanenbaum, *Modern Operating Systems*, Prentice-Hall, Inc. 1992.
- [34] A.S. Tanenbaum, *Operating Systems: Design and Implementation*, 2nd edition, Prentice Hall, 1997
- [35] T.-S. Tia, Z.Deng, M. Shankar and M. Storch, "Probabilistic Performance Guarantee for Real-Time Tasks with Varying Computation Times," in *Proceedings, Real-Time Technology and Applications Symposium*, Chicago, Illinois, USA, May. 1995, pp. 164-173.
- [36] K.W. Tindell, "An Extendible Approach for Analyzing Fixed Priority Hard Real-Time Tasks," *Real Time Systems*, 1994, Vol. 6, No. 2, pp. 133-151.
- [37] Tornado Training Workshop, <http://www.windriver.com>
- [38] X. Zhang, "Tool for Task Allocation and Scheduling in Hard Real-time Systems," M.Sc. Thesis, Dept. of Systems and Computer Engineering, Carleton University, 1998.

Appendix A Additional Results for Systems with Same CPU Utilizations Requested by Each Task

Figure A-1 Comparison of Average Ratios for Meeting Deadlines Achieved by PES-A, PES-B, TTF and RMS on a Non-overloaded System when $T_n/T_1 \geq 2$

Figure A-2 Comparison of Useful CPU Utilizations Achieved by PES-A, PES-B, TTF and RMS on a Non-overloaded System when $T_n/T_1 \geq 2$

Figure A-3 Comparison of Guaranteed Ratios for Meeting Deadlines Achieved by PES-A, PES-B, TTF and RMS on an Overloaded System when $T_n/T_1 \geq 2$

Figure A-4 Comparison of Average Ratios for Meeting Deadlines Achieved by PES-A, PES-B, TTF and RMS on an Overloaded System when $T_n/T_1 \geq 2$

Figure A-5 Comparison of Useful CPU Utilizations Achieved by PES-A, PES-B, TTF and RMS on an Overloaded System when $T_n/T_1 \geq 2$

Figure A-6 Comparison of Inter-task Fairness Metric Achieved by PES-A, PES-B, TTF and RMS on an Overloaded System when $T_n/T_1 \geq 2$

Figure A-7 Comparison of Guaranteed Ratios for Meeting Deadlines Achieved by PES-A, PES-B, TTF and RMS on an Overloaded System when Task Periods are Equal to One Another

Figure A-8 Comparison of Average Ratios for Meeting Deadlines Achieved by PES-A, PES-B, TTF and RMS on an Overloaded System when Task Periods are Equal to One Another

Figure A-9 Comparison of Useful CPU Utilizations Achieved by PES-A, PES-B, TTF and RMS on an Overloaded System when Task Periods are Equal to One Another

Figure A-10 Comparison of Inter-task Fairness Metric Achieved by PES-A, PES-B, TTF and RMS on an Overloaded System when Task Periods are Equal to One Another

Appendix B Additional Results for Systems with CPU Utilizations of Tasks Inversely Proportional to Task Periods

Figure B-1 Comparison of Guaranteed Ratios for Meeting Deadlines Achieved by PES-A, PES-B, TTF and RMS on a Non-overloaded System when $C/T = O(1/T^2)$

Figure B-2 Comparison of Useful CPU Utilizations Achieved by PES-A, PES-B, TTF and RMS on a Non-overloaded System when $C/T = O(1/T^2)$

Figure B-3 Comparison of Inter-task Fairness Metric Achieved by PES-A, PES-B, TTF and RMS on a Non-overloaded System when $C/T = O(1/T^2)$

Figure B-4 Comparison of Guaranteed Ratios for Meeting Deadlines Achieved by PES-A, PES-B, TTF and RMS on a Non-overloaded System when $C/T = O(1/T^4)$

Figure B-5 Comparison of Useful CPU Utilizations Achieved by PES-A, PES-B, TTF and RMS on a Non-overloaded System when $C/T = O(1/T^4)$

Figure B-6 Comparison of Inter-task Fairness Metric Achieved by PES-A, PES-B, TTF and RMS on a Non-overloaded System when $C/T = O(1/T^4)$

Figure B-7 Comparison of Guaranteed Ratios for Meeting Deadlines Achieved by PES-A, PES-B, TTF and RMS on an Overloaded System when $C/T = O(1/T^2)$

Figure B-8 Comparison of Average Ratios for Meeting Deadlines Achieved by PES-A, PES-B, TTF and RMS on an Overloaded System when $C/T = O(1/T^2)$

Figure B-9 Comparison of Useful CPU Utilizations Achieved by PES-A, PES-B, TTF and RMS on an Overloaded System when $C/T = O(1/T^2)$

Figure B-10 Comparison of Inter-task Fairness Metric Achieved by PES-A, PES-B, TTF and RMS on an Overloaded System when $C/T = O(1/T^2)$

Figure B-11 Comparison of Guaranteed Ratios for Meeting Deadlines Achieved by PES-A, PES-B, TTF and RMS on an Overloaded System when $C/T = O(1/T^4)$

Figure B-12 Comparison of Average Ratios for Meeting Deadlines Achieved by PES-A, PES-B, TTF and RMS on an Overloaded System when $C/T = O(1/T^4)$

Figure B-13 Comparison of Useful CPU Utilizations Achieved by PES-A, PES-B, TTF and RMS on an Overloaded System when $C/T = O(1/T^4)$

Figure B-14 Comparison of Inter-task Fairness Metric Achieved by PES-A, PES-B, TTF and RMS on an Overloaded System when $C/T = O(1/T^4)$

Figure B-15 Comparison of Guaranteed Ratios for Meeting Deadlines Achieved by PES-A, PES-B, TTF and RMS on an Overloaded System when $C/T = O(1/T^6)$

Figure B-16 Comparison of Average Ratios for Meeting Deadlines Achieved by PES-A, PES-B, TTF and RMS on an Overloaded System when $C/T = O(1/T^6)$

Figure B-17 Comparison of Useful CPU Utilizations Achieved by PES-A, PES-B, TTF and RMS on an Overloaded System when $C/T = O(1/T^6)$

Figure B-18 Comparison of Inter-task Fairness Metric Achieved by PES-A, PES-B, TTF and RMS on an Overloaded System when $C/T = O(1/T^6)$

Appendix C Additional Results for Systems with CPU Utilizations of Tasks Directly Proportional to Task Periods

Figure C-1 Comparison of Guaranteed Ratios for Meeting Deadlines Achieved by PES-A, PES-B, TTF and RMS on a Non-overloaded System when $C/T = O(T^4)$

Figure C-2 Comparison of Inter-task Fairness Metric Achieved by PES-A, PES-B, TTF and RMS on a Non-overloaded System when $C/T = O(T^4)$

Figure C-3 Comparison of Guaranteed Ratios for Meeting Deadlines Achieved by PES-A, PES-B, TTF and RMS on an Overloaded System when $C/T = O(T^4)$

Figure C-4 Comparison of Average Ratios for Meeting Deadlines Achieved by PES-A, PES-B, TTF and RMS on an Overloaded System when $C/T = O(T^4)$

Figure C-5 Comparison of Useful CPU Utilizations Achieved by PES-A, PES-B, TTF and RMS on an Overloaded System when $C/T = O(T^4)$

Figure C-6 Comparison of Inter-task Fairness Metric Achieved by PES-A, PES-B, TTF and RMS on an Overloaded System when $C/T = O(T^4)$