1

Published by the U.S. & Canada 1984
MICRO
U.S. & Canada 1984
100 100 100 100 100 100 100 100 100 100

National Library
of Canada

Bibliothèque nationale
du Canada

Canadian Theses Service

Service des thèses canadiennes

Ottawa, Canada
K1A 0N4

NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Previously copyrighted materials (journal articles, published tests, etc.) are not filmed.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

Les documents qui font déjà l'objet d'un droit d'auteur (articles de revue, tests publiés, etc.) ne sont pas microfilmés.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30.

**Light Traffic Loss of
Random Hard Real-time Tasks
in a Network**

by

David William Craig, B.A.Sc., M.Eng.

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

Ottawa-Carleton Institute for Electrical Engineering
Faculty of Engineering
Department of Systems and Computer Engineering,
Carleton University

June, 1988

© 1988, David William Craig

Permission has been granted to the National Library of Canada to microfilm this thesis and to lend or sell copies of the film.

The author (copyright owner) has reserved other publication rights, and neither the thesis nor extensive extracts from it may be printed or otherwise reproduced without his/her written permission.

L'autorisation a été accordée à la Bibliothèque nationale du Canada de microfilmer cette thèse et de prêter ou de vendre des exemplaires du film.

L'auteur (titulaire du droit d'auteur) se réserve les autres droits de publication; ni la thèse ni de longs extraits de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation écrite.

ISBN 0-315-46230-2

The undersigned hereby recommend to the Faculty of Graduate Studies and Research acceptance of the thesis

*"Light Traffic Loss of
Random Hard Real-time Tasks
in a Network"*

submitted by David William Craig, B.A.Sc., M.Eng., in partial fulfilment of the requirements for the degree of Doctor of Philosophy.

Thesis Supervisor

External Examiner

Chairman,
Department of Systems and
Computer Engineering

Department of Systems and Computer Engineering
Faculty of Engineering
Carleton University
June, 1988

Abstract

The price to be paid for low task loss of hard real-time tasks with random arrival times is light utilization of processors. This thesis deals with light traffic loss of hard real-time tasks both at a single processor and within a homogeneous distributed network of nodes. A general methodology is presented that enables approximate loss prediction at a single node for a Poisson stream of tasks with general computation and laxity density functions under a FCFS, SJF, LLF or EDD pre-emptive run-time scheduling algorithm. Using this methodology, rejected task computation and laxity density functions are derived for each of the scheduling algorithms. The approximation method is extensible to higher traffic levels at a cost of greater computational complexity. Single node results are combined into a homogeneous distributed network model that permits random routing of tasks between nodes and predicts light traffic loss in the network under each of the scheduling algorithms. The network model is extensible to heterogeneous clusters of nodes with different traffic characteristics.

Acknowledgements

I would like to thank my thesis supervisor, Murray Woodside, for all of his guidance and encouragement over the past three years. The support of the Department of National Defence is also appreciated for making it possible for me to complete this degree by granting educational leave. Lastly I want to thank my wife, Elizabeth, who supplied the hope and strength to do this work.

Contents

1	Introduction	1
1.1	Problem Description	1
1.1.1	Soft and Hard Real-time Tasks	2
1.1.2	Soft or Hard Real-time Systems	2
1.1.3	Task Arrival Times	3
1.1.4	Task Parameters	4
1.1.5	Categories of Systems	4
1.2	Scheduling for Distributed Systems	5
1.2.1	Definitions	5
1.2.2	Scheduling Decisions	6
1.3	Thesis Objective and Contributions	8
1.4	Thesis Outline	9
2	Distributed Scheduling Research	11
2.1	General Overview	11
2.2	Uniprocessor and Multiprocessor Scheduling	11
2.3	Periodic Task Scheduling	13
2.4	Dynamic Task Scheduling	14
2.5	Distributed Dynamic Scheduling	15
2.6	Analysis of Scheduling Algorithms	16
3	Light Traffic Performance Model for a Single Processor	19
3.1	Introduction	19
3.2	Low Traffic Analysis	20
3.2.1	Definitions and Terminology	20
3.2.2	General Results for Pairwise Interference	23
3.2.3	FCFS Scheduling Policy	27
3.2.4	LLF Scheduling policy	27
3.2.5	SJF Scheduling Policy	28
3.2.6	EDD Scheduling Policy	29
3.3	Example Calculation	36

3.3.1	Comparison to Simulation	37
3.4	Summary	43
4	Task Rejection Bounds	54
4.1	Introduction	54
4.2	Pre-emptive Scheduling Bounds	55
4.2.1	Acceptance Lower Bound	55
4.2.2	Acceptance Probability for a Restricted Processor	57
4.2.3	Useful and Wasted Busy Period Calculation	58
4.2.4	Validity of Lower Bound Conjecture	61
4.3	Acceptance Upper Bound	62
4.4	Tightness of Acceptance Bounds	64
4.5	Acceptance Bounds and Simulated Results	64
4.6	Summary	66
5	Analysis for Exponential Density Functions	70
5.1	Introduction	70
5.2	Single Node Analytic Solution	71
5.2.1	Comparison to Simulation and Exact Result	71
5.3	Task Loss Bounds	75
5.4	Summary	75
6	Light Traffic Performance for a Homogeneous Network	78
6.1	Introduction	78
6.2	Network Model	80
6.3	Task Flow for a Single Node	82
6.4	Feedback Traffic Solution	83
6.5	Network Prediction and Simulated Results	87
6.6	Summary	101
7	Extensions and Enhancements	102
7.1	Introduction	102
7.2	Double Interval Estimation of Task Loss	103
7.2.1	FCFS Double Interval Estimation	106
7.3	Extension to Heterogeneous Clusters of Networks	109
7.4	Summary	112
8	Conclusion and Summary	113

List of Figures

1.1	Classification of Scheduling Methods for Distributed Computer Systems	7
3.1	A pair of tasks arriving to be scheduled on a idle processor. The second task splits the first task into two parts: an <i>original pre-empted</i> and a <i>remaining pre-empted</i> task.	21
3.2	Computing the pairwise rejection probability for tasks with independence between selected task computation and non-selected task laxity.	26
3.3	Computing the conditional distribution function for non-selected tasks, given the selected task computation	30
3.4	Task selection for the case where $z < w$. Region R indicates CS, LN pairs where tasks are rejected.	32
3.5	Task selection for $DI < DP$ where $z < w$. Region I indicates the area of integration on the (z, w) plane.	34
3.6	Task selection for $DI < DP$ where $z < w$. The complementary values LS and CN are plotted here. Region R indicates (LS, CN) pairs where tasks are rejected.	35
3.7	Task selection for the case where $z < w$. Region I is the region of integration in the (z, w) plane for the complementary values of LS and CN	36
3.8	Predicted density function for computation and collected histogram results from simulation of a single node under the pre-emptive FCFS scheduling algorithm. The load level for these results is 0.20	44
3.9	Predicted density function for computation and collected histogram results from simulation of a single node under the pre-emptive SJF scheduling algorithm. The load level for these results is 0.20	45
3.10	Predicted density function for computation and collected histogram results from simulation of a single node under the pre-emptive LLF scheduling algorithm. The load level for these results is 0.20	46

3.11	Predicted density function for computation and collected histogram results from simulation of a single node under the pre-emptive EDD scheduling algorithm. The load level for these results is 0.20	47
3.12	Predicted density function for laxity and collected histogram results from simulation of a single node under the pre-emptive FCFS scheduling algorithm. The load level for these results is 0.20	48
3.13	Predicted density function for laxity and collected histogram results from simulation of a single node under the pre-emptive SJF scheduling algorithm. The load level for these results is 0.20	49
3.14	Predicted density function for laxity and collected histogram results from simulation of a single node under the pre-emptive LLF scheduling algorithm. The load level for these results is 0.20	50
3.15	Predicted density function for laxity and collected histogram results from simulation of a single node under the pre-emptive EDD scheduling algorithm. The load level for these results is 0.20	51
3.16	Predicted density function for computation and collected histogram results from simulation of a single node under the pre-emptive EDD scheduling algorithm. The load level for these results is 3.0	52
3.17	Predicted density function for laxity and collected histogram results from simulation of a single node under the pre-emptive EDD scheduling algorithm. The load level for these results is 3.0	53
4.1	A restricted processor that schedules only a subset of arriving real-time tasks.	56
4.2	The various possible busy periods for the restricted processor and associated probabilities. The shaded area indicates wasted work done in a busy period.	59
4.3	An input arrival stream arbitrarily separated by even and odd arrival intervals. In order to be eligible under a pre-emptive scheduling policy, all adjacent tasks must be at least pairwise compatible.	63
4.4	Task loss upper and lower bounds for FCFS Pre-emptive scheduling and different P_R . The upper curves have $P_R = 56\%$ while the lower curves have $P_R = 15\%$	67
5.1	A comparison of simulated results and the asymptotic light traffic approximation for the FCFS,LLF,SJF and EDD scheduling policies	74
5.2	Rejection performance plotted as a function of γ/μ for the FCFS, LLF, SJF and EDD scheduling policies. Note how the asymptotic low traffic behavior gives additional insight into relative performance of the policies.	76

6.1	The model of a distributed processing system of N nodes connected by a communication network. Each node may communicate tasks across the network after a random delay of D seconds. A typical node is shown for reference, with both the input and transferred traffic.	80
6.2	A typical node in the distributed processing network and the task flow of input and rejected tasks. A steady state flow of tasks and sufficient nodes to make the feedback traffic Poisson is assumed.	83
6.3	Predicted density function for feedback computation and collected histogram results from simulation of a network of 8 nodes under the pre-emptive FCFS scheduling algorithm and a random task sharing algorithm. The load level for these results is 0.20	93
6.4	Predicted density function for feedback computation and collected histogram results from simulation of a network of 8 nodes under the pre-emptive SJF scheduling algorithm and a random task sharing algorithm. The load level for these results is 0.20	94
6.5	Predicted density function for feedback computation and collected histogram results from simulation of a network of 8 nodes under the pre-emptive LLF scheduling algorithm and a random task sharing algorithm. The load level for these results is 0.20	95
6.6	Predicted density function for feedback computation and collected histogram results from simulation of a network of 8 nodes under the pre-emptive EDD scheduling algorithm and a random task sharing algorithm. The load level for these results is 0.20	96
6.7	Predicted density function for feedback laxity and collected histogram results from simulation of a network of 8 nodes under the pre-emptive FCFS scheduling algorithm and a random task sharing algorithm. The load level for these results is 0.20	97
6.8	Predicted density function for feedback laxity and collected histogram results from simulation of a network of 8 nodes under the pre-emptive SJF scheduling algorithm and a random task sharing algorithm. The load level for these results is 0.20	98
6.9	Predicted density function for feedback laxity and collected histogram results from simulation of a network of 8 nodes under the pre-emptive LLF scheduling algorithm and a random task sharing algorithm. The load level for these results is 0.20	99
6.10	Predicted density function for feedback laxity and collected histogram results from simulation of a network of 8 nodes under the pre-emptive EDD scheduling algorithm and a random task sharing algorithm. The load level for these results is 0.20	100

7.1	The acceptance or rejection events for arriving task A, conditioned on acceptance or rejection of tasks P and NP. The final event space has five events that reject one or more tasks, and seven events that accept task A.	105
7.2	A typical node of cluster 1 in a network of two homogeneous clusters. Total input to the node consists of transferred tasks from cluster 1, transferred tasks from cluster 2, and input tasks to cluster 1.	110

List of Tables•

3.1	Simulation Parameters for Pre-emptive Local Scheduling Algorithms	37
3.2	Analytic and Simulated results for fraction of task loss under the pre-emptive FCFS Local Scheduling Algorithm	39
3.3	Analytic and Simulated results for fraction of task loss under the pre-emptive SJF Local Scheduling Algorithm	40
3.4	Analytic and Simulated results for fraction of task loss under the pre-emptive LLF Local Scheduling Algorithm	41
3.5	Analytic and Simulated results for fraction of task loss under the pre-emptive EDD Local Scheduling Algorithm	42
4.1	Task loss bounds and simulation confidence intervals for task loss for the pre-emptive FCFS Local Scheduling Algorithm	65
4.2	Task loss bounds and simulation confidence intervals for task loss for the pre-emptive SJF Local Scheduling Algorithm	66
4.3	Task loss bounds and simulation confidence intervals for task loss for the pre-emptive LLF Local Scheduling Algorithm	68
4.4	Analytic and Simulated results for fraction of task loss under the pre-emptive EDD Local Scheduling Algorithm	69
5.1	P_R for Poisson arrivals with exponential service and laxity	72
5.2	$f_{LN R}(x)$ for Poisson arrivals with exponential service and laxity	72
5.3	Calculation of $f_{CN R}(x)$ for Poisson arrivals with exponential service and laxity. The equations presented here assume that $\gamma \neq \mu$	72
5.4	Simulation Parameters for Exponential Computation and Laxity Density	73
5.5	$\overline{CS}$ and $\overline{CN A}$ for Poisson arrivals with exponential service and laxity	77
6.1	Equation references required to compute P_R , $f_{CN R}$ and $f_{LN R}$ for general input distributions.	85
6.2	Simulation Parameters for Pre-emptive Local Scheduling Algorithms and simple random task sharing in a Homogeneous Network of 8 Nodes.	87

6.3	Analytic and simulated results for fraction of task loss under the pre-emptive FCFS local scheduling algorithm and simple random task sharing in a Homogeneous Network of 8 Nodes.	88
6.4	Analytic and simulated results for fraction of task loss under the pre-emptive SJF local scheduling algorithm and simple random task sharing in a Homogeneous Network of 8 Nodes.	89
6.5	Analytic and simulated results for fraction of task loss under the pre-emptive LLF local scheduling algorithm and simple random task sharing in a Homogeneous Network of 8 Nodes.	90
6.6	Analytic and simulated results for fraction of task loss under the pre-emptive EDD local scheduling algorithm and simple random task sharing in a Homogeneous Network of 8 Nodes.	91
7.1	FCFS Pre-emptive Scheduling with event probabilities for double interval estimation	107
7.2	Single interval and Double interval estimates for the fraction of task loss under the pre-emptive FCFS Local Scheduling Algorithm along with the exact solution	108

Chapter 1.

Introduction

1.1 Problem Description

Distributed computing systems for real-time tasks have been in use in recent years for a wide number of applications. Examples of such systems include such diverse areas as space or airborne platform management [1], factory process control [2] and robotics [3]. The term *real-time* implies that a value is attached to the time of execution of a task. The correct execution of a task requires that the task be executed within some specified interval, with failure to meet this requirement imposing some sort of penalty. For example, a space platform will go off course if navigational computations are not complete by the time the rockets need to fire or a nuclear reactor will shut down if corrective action is not taken within a specified time.

The term *distributed computing system* implies that computation of tasks is done by a system of processing nodes where communication between nodes incurs some finite delay. Arrivals occur at each processing node and must be completed within the interval specified for each task. Various subcategories of the scheduling problem exist, depending on the type of tasks processed and the characteristics of the processing system. A more detailed description of the various terms used to describe the problem is given in the following sections, as well a statement of the

area of concern for this thesis.

1.1.1 Soft and Hard Real-time Tasks

A task *deadline* defines the interval for completion of a task by specifying the time by which the task must complete. The interval for completion is then the time between the task being ready to run (its arrival) and its deadline. Real-time tasks are generally grouped into two categories: hard real-time and soft real-time. Hard real-time tasks are those which require completion within a specified interval after which the task result becomes valueless. Control of aircraft, spacecraft, and nuclear reactors are examples of systems with hard real-time tasks. If a task cannot be completed in the interval specified, no further work is done on it and the task is considered lost.

In other instances it may be desirable for a task to be completed even if it cannot be completed on time. Soft real-time tasks retain value beyond the time interval specified for execution. A navigational update required for a course correction may provide valuable information even if the result is late. In most cases the value of completing a soft real-time task decreases monotonically as lateness increases until the result becomes worthless. Since a soft real-time deadline is somewhat ill-defined, the term *critical time* has been proposed as an alternative [1].

This thesis is concerned only with hard real-time tasks. Each task is defined by an arrival time, computation requirement and deadline. If a task cannot be completed by its deadline, it is discarded.

1.1.2 Soft or Hard Real-time Systems

Distributed systems are also designated as soft or hard real-time. A system that cannot tolerate *any* late tasks is referred to as a hard real-time system. A hard real-time system must ensure that every task is processed in the time interval

specified and that no tasks are rejected. A soft real-time system is one that can tolerate a certain number of tasks that miss deadlines. Although failure of a task to meet its deadline does not constitute system failure, performance of a soft real-time system will degrade as the number of missed deadlines increase. For hard real-time tasks, the performance of soft real-time systems will be specified by the fraction of arrivals that are lost. For systems with random task arrivals, an average task loss figure is quoted for the entire network.

Systems that must deal with random task arrival times are constrained to be soft real-time systems as it is likely to be impossible to guarantee that no tasks are ever missed. For example, a nuclear reactor can tolerate some predefined level of alarms but is in danger of meltdown as the alarm traffic increases. This thesis is concerned with soft real-time systems and prediction of average task loss for both a single node and a homogeneous network of nodes.

1.1.3 Task Arrival Times

Task arrival times may be grouped into two categories as well: periodic or random arrivals. Arrival times are also termed *task ready times* or times at which a task is ready for execution. Periodic arrivals correspond to tasks that must be done repetitively at specified intervals. Arrival times of periodic tasks are therefore known and completely predictable. The deadline for a periodic task amounts to the requirement that it must be completed before the next invocation. Retrieving items from an assembly line is an example of a hard real-time periodic task. If processing of an item is not complete before the next item rolls off the assembly line, a loss occurs in the system.

Random arrivals do not occur at regular intervals and are therefore not predictable. Arrival patterns are often characterized by some statistic which specifies an average rate of invocation. Poisson arrivals are often assumed in order to make the analysis tractable. For example, interrupts from alarm systems in a manufacturing process occur as a result of random failures and are by nature not predictable. This thesis assumes that task arrival times at processing nodes are

completely stochastic and described by a Poisson stream of arrivals.

In most instances, task arrival times are not exclusively periodic or non-periodic but a mixture of both types. Since schedules for periodic tasks may be drawn up and validated in advance, periodic tasks are usually scheduled first and random arrivals are fitted into an existing schedule on arrival. In some cases, it may be desirable (such as in the instance of rare urgent alarms) to give immediate attention to random arrivals. In this instance, random arrivals should be given priority over periodic tasks.

The processor may also have a background load of non real-time tasks that utilize the processor. Although this thesis assumes exclusively stochastic arrivals, the results may be applied to any situation where random arrivals are given priority over all other types of tasks in the system. Light traffic for random hard real-time tasks does not necessarily imply low utilization of the processor.

1.1.4 Task Parameters

Finally, parameters for real-time tasks are usually specified by task computation and *laxity* parameters. Computation and laxity parameters are either fixed or specified by some probability density function. The term *task laxity* refers to the interval between the task arrival and deadline which is in excess of computation requirement (i.e., $\text{laxity} = \text{deadline} - \text{arrival time} - \text{computation time}$). On arrival, task parameters may be either known or unknown to the scheduling algorithm.¹ For the investigation in this thesis, task parameters are considered to be specified by probability density functions, with the parameter values known at task arrival.

1.1.5 Categories of Systems

¹Tasks which are not data-dependent are more likely to be known exactly at task execution. If the computation depends on input data received then the task computation time will vary according to the degree of processing required.

Systems on which real-time tasks are executed are divided into three groups: single processor, multiprocessor or distributed systems. A multiprocessor system is a group of single processors which are tightly coupled and are often implemented with shared memory. Typically, multiprocessor systems exhibit central control with all tasks allocated and scheduled for execution by a central processor. Task allocation and movement of data between processors is controlled by one processor and the system state is completely known by the controller at all times.

Distributed systems are distinguished from multiprocessor systems by being more loosely coupled and possessing a communication network connecting processors in the system. This network results in delays in communicating between processors and requires some sort of communications protocol. As a result of communication delay, distributed systems are also distinguished by a lack of central control. No one processor can have an exact, up-to-date knowledge of the state of all processors in the network. The trend of most direct digital control systems is now toward decentralized control in order to enhance reliability and increase flexibility [2]. Each processor is responsible for deciding the manner of task execution and interacting with other processors to enhance throughput. Distributed systems are the subject of investigation for this thesis.

1.2 Scheduling for Distributed Systems

1.2.1 Definitions

Scheduling remains an important issue to be solved for single processor, multiprocessor and distributed real-time systems. It is helpful to distinguish terminology used in describing scheduling issues. The following definitions are used in this discussion:

Scheduling: Deals with the general question of both where and when a task should be executed.

Allocation: Deals with the question of where a task should be executed, i.e. on which processor.

Sequencing: Deals with the order of execution of a set of tasks on a given processor.

Run-time dispatching: Deals with the immediate decision of which of a set of tasks to execute next on a given processor.

1.2.2 * Scheduling Decisions

Where task arrival times are random, it is necessary to adjust the schedule dynamically and reallocate tasks. In distributed systems where decentralized control is used, scheduling may occur at several levels. A survey of decentralized scheduling [4] has suggested the classification of schedulers given in Figure 1.1. Scheduling decisions are done either locally or globally for the system. Static global scheduling means that allocation of tasks to processors is done only once and one schedule is all that is required. If global scheduling is dynamic, the schedule for all processors is adjusted as conditions change. A centralized dynamic schedule implies that there is a single decision point for the system. A decentralized dynamic schedule implies that all processors participate in the decision making process.

Distributed systems commonly use decentralized dynamic scheduling as it is difficult to obtain simultaneous state information owing to communication delays. In addition, the time involved in computing and creating a global schedule may be considerable. Decentralized scheduling also increases the robustness of the system by eliminating the dependence on a central location to make all the scheduling decisions.

This thesis examines the problem of dynamic decentralized scheduling where each processor uses a dispatching algorithm to determine which task will be executed locally. Processors cooperate globally in determining a schedule by sharing tasks that cannot be executed locally. Analysis is done for a simple model

Figure 1.1: Classification of Scheduling Methods for Distributed Computer Systems

of load sharing where each processor randomly selects where to send a task that cannot be executed locally.

1.3 Thesis Objective and Contributions

The objective of this thesis is to enable the designer of soft real-time distributed systems with random arrivals and hard real-time tasks to predict the rejection performance, for light traffic levels. Such prediction at present requires extensive simulation, especially for light traffic, where rejection is a rare event. The goal here is to enable the designer to make a rapid approximate analytic calculation of task loss both for a single node as well as a homogeneous network. A low loss rate for hard real-time tasks is likely to be a design requirement for any distributed system. The price to be paid for low loss rate is light traffic at any node in the system. No attempt is made in this thesis to compare the approximate results to existing systems, as data for task loss for such systems is not available. The utility of the approximation of task loss is instead compared to simulated systems.

This thesis presents an analytic model to approximate the rejection performance for a single node and for a homogeneous network of nodes in light traffic. The major contributions of this thesis are as follows:

- A light traffic approximation for a single processor with Poisson arrivals that concentrates on interactions between pairs of tasks and permits the evaluation of task loss under four pre-emptive scheduling algorithms.
- An approximation of rejected task computation and laxity density functions for the four pre-emptive scheduling algorithms studied.
- Closed form solutions for task loss as well as rejected task laxity and computation density functions in the case of exponential computation, exponential laxity density functions. These solutions are given for each of the pre-emptive algorithms studied.

- Evaluation of an upper and lower bound for the task loss rate at a single processor for each of the pre-emptive algorithms.
- An extension of the light traffic approximation to more complex task interactions that permits more accurate prediction of task loss at higher traffic levels.
- A distributed load-sharing task loss performance model of a homogeneous network of nodes with identical light traffic Poisson input streams of hard real-time tasks. This evaluation assumes a network with a large number of processing nodes, so that the total arrival process at each node is Poisson.
- An extension of the network model to include clusters of heterogeneous nodes with different traffic characteristics.

1.4 Thesis Outline

Chapter 1 begins with a general description of terminology used in describing the area of distributed computing. The relevance of each term is then linked to the specific investigation contained in this thesis. A general overview of previous work is given in Chapter 2 which places the specific area of contribution of this thesis and the relevance to other work in the area. Chapter 3 explains the approach taken in this thesis to approximating pre-emptive scheduling and deals specifically with the equations for four pre-emptive run-time algorithms. A comparison to simulated results shows the utility of the approximation in light traffic. Chapter 4 then extends the usefulness of the approximation by deriving upper and lower bounds for task loss under the four pre-emptive run-time algorithms. Chapter 5 deals with the special case of exponential density functions for computation and laxity and presents closed-form analytic solutions to the equations given in Chapter 3. Chapter 6 uses the light traffic approximation for a single node and creates a model for a large network of homogeneous nodes under a simple random task sharing algorithm. Simulated results again show the utility of the result

under light traffic loads. Chapter 7 describes some extensions to the work done that enable a number of the restricting approximations to be relaxed. Finally, Chapter 8 suggests further research to be conducted in this area and summarizes the work in this thesis.

The author's original contribution in this thesis is primarily in describing a method by which pre-emptive scheduling algorithms may be analyzed and incorporated into a network model. Previous attempts using results from queueing theory needed to assume pre-emptive FCFS scheduling and restricted task sharing among nodes to make the analysis tractable. Using N th interval estimation, task loss prediction under the four pre-emptive scheduling techniques may be made as accurate as desired. The ability to predict density functions of computation and laxity for both rejected and transferred tasks is also a significant contribution in the development of distributed network models. Finally, solution of network task loss using the concept of "feedback" tasks is original and a significant extension to the task sharing model in a distributed system.

Chapter 2

Distributed Scheduling Research

2.1 General Overview

Research into distributed scheduling brings together results from operations research as well as from engineering. Dynamic, distributed scheduling of hard real-time tasks is a difficult problem to solve and demands that previous results pertaining to uniprocessor and multiprocessor scheduling be understood. This chapter presents an overview of recent research from both operations research and engineering relevant to this thesis.

2.2 Uniprocessor and Multiprocessor Scheduling

Scheduling hard real-time tasks on a single processor or a multiple processor system usually has as a goal the minimization of task loss due to expired deadlines. The problem of how to order a set of tasks on a processor so as to minimize task loss has been well studied in operations research. The usual assumption is that the number of tasks to be scheduled is fixed at the outset and described by a set of arrival times, computation and laxity requirements. A survey of research up to

1977 on scheduling sets of tasks on a single processor and multiprocessor is given in [5]. At the time of that survey, little work had been done on scheduling real-time tasks. The following summary updates this survey with regard to real-time tasks and places the present research in the perspective of this work.

The degree of difficulty in creating a schedule for a set of tasks depends on the precedence relationships among the tasks and whether or not pre-emption is allowed. For independent tasks, and no pre-emption allowed, the scheduling solution depends on the set of arrival times. If all of the arrival times are identical, the scheduling problem is reduced to ordering the tasks so as to minimize the number of missed deadlines. This problem was solved for the uniprocessor by Moore [6] and later extended by Sidney to include subsets of tasks with separate priorities [7]. The multiprocessor scheduling problem was considered long ago by McNaughton [8], but no optimal algorithm for reducing the number of lost tasks has been presented.

For arrival times that are not identical, special cases have been examined for a single processor to obtain polynomial time algorithms. For the case where each successive arrival has a corresponding later deadline than the proceeding tasks, an $O(n^2)$ algorithm has been presented by Kise [9]. In the general case of arbitrary arrival times, computation and laxity requirements, the ordering of the tasks so as to minimize task loss has been shown to be NP-complete [10].¹

For pre-emptive scheduling and independent tasks (again with arbitrary arrival times, computation and laxity requirements), the problem of finding a schedule to minimize the number of late tasks is also NP-complete. However, finding out whether a pre-emptive schedule exists that schedules *all* tasks with no loss, either for a uniprocessor or multiprocessor, may be done in polynomial time. An $O(n^5)$ algorithm to determine whether such a schedule exists for the uniprocessor and multiprocessor was presented by Martel [12], using a network flow algorithm. An alternative branch and bound formulation for determining

¹A comprehensive discussion of the concept of NP-completeness may be found in [11]. In general, an NP-complete problem in task set scheduling becomes computationally intractable as the number of tasks increases.

whether a schedule exists was presented in [13].

One difficulty in applying the previous results to a set of tasks is that in most engineering applications, the set of tasks to be scheduled is not bounded. For a finite time horizon, the number of tasks may be bounded, but in general the problem of constructing a schedule to minimize task loss is NP-complete. For this reason, research has concentrated on tasks with a predictable arrival structure (periodic tasks) or scheduling algorithms that do not require knowledge of the entire task set (run-time schedulers).

2.3 Periodic Task Scheduling

The basic algorithm to schedule the maximum number of periodic tasks on a single processor was found independently by Serlin [14] and Liu and Leyland [15]. The problem examined by both researchers was to create a pre-emptive schedule for a uniprocessor that allowed maximum utilization of the processor. It was shown that the best schedule is created when tasks with the higher repetition rate are scheduled first. It was also shown in [15] that this type of scheduling leads to an upper bound on processor utilization. Depending on the number of tasks, this utilization may be as low as 70 percent. This algorithm is termed the "rate monotonic" algorithm, since the tasks are scheduled in decreasing order of repetition rate. Gonzalez and Soh [16] have investigated the minimum number of processors which satisfy the deadline requirements of periodic tasks for a non-pre-emptive schedule.

In spite of the proof of optimality of the scheduling algorithm, it has not found wide use in practical systems. Uncertainty in task execution time or task arrival time and resultant instability has led to the use of dynamic scheduling algorithms. A comprehensive treatment of the practical problems associated with the rate monotonic algorithm was given in [17]. Recent work examined the problem of fitting background tasks into a periodic task structure [18].

2.4 Dynamic Task Scheduling

In order to meet the requirement to schedule tasks when the entire task set is unknown, dynamic or *run-time* scheduling is used. This type of scheduling makes a decision at run-time as to which task to schedule, and on what processor, according to the tasks already present. A run-time scheduling algorithm does not take into account future arrivals. As more tasks arrive, the schedule is able to adjust dynamically so as to enable tasks to meet their deadlines. Since decisions are made on incomplete information, dynamic task scheduling will in general create an inferior schedule to that of static (entire task set) scheduling.

The conditions under which a run-time algorithm will find a schedule that will meet all arriving task deadlines (assuming that such a schedule exists for the task set) were explored by Mok [3]. For a uniprocessor, ordering the tasks by earliest due date (EDD) will always find a feasible schedule. Equivalently, the same result could be accomplished by ordering the tasks by least laxity first (LLF). Both of these algorithms will always find a feasible schedule, assuming that a static scheduling algorithm could do so. For the multiprocessor, it was shown that no run-time algorithm exists that will always find a feasible schedule. Without complete knowledge of the task set (including arrival times), it is not possible to make the correct scheduling decision that will allow all tasks to be executed.

In order to address the question of which run-time algorithm to use on a multiprocessor, Jansen [1] investigated the properties of several pre-emptive scheduling algorithms. Jansen assumed that uncertainty existed at scheduling time not only as to future arrivals, but also in the computation and laxity parameters of tasks already present. Jansen studied the performance of well known dispatching algorithms such as shortest job (task) first (SJF), EDD, LLF, first-come-first-served (FCFS) as well as fixed priority scheduling. With the assumption of uncertain task execution time and laxity, Jansen found that two heuristic run-time algorithms (termed BEValue1 and BEValue2) performed better than all the other algorithms tested. These algorithms maximized the expected value at

the completion of the task by computing the probability of a task expiring and adjusting the schedule accordingly.

A study of non-pre-emptive run-time scheduling algorithms has been carried out for the uniprocessor by Woodside and the present author [19]. In this study, task parameters were assumed to be known exactly at the time a scheduling decision as made. For non-pre-emptive algorithms, a dynamic version of the optimal static algorithm of Moore [6] was found to give the best rejection performance.

2.5 Distributed Dynamic Scheduling

Because of the number of applications that require processors to be less centrally located, there has been in recent years a trend to distributed processing systems [2]. Due to the unavoidable communication delay between processors in a distributed network, research into distributed scheduling has concentrated on decentralized control algorithms. The sharing of tasks in a distributed network of tasks is complicated by an uncertainty of the global condition of the network. No central control of the network can obtain simultaneous state information from all processors due to delay across the network. By decentralizing control of the network to all processors, both control and reliability of the network are enhanced.

One method originally proposed by Smith [20] was to have the processors cooperate by a protocol of bidding. An alternate method of control proposed by Stankovic [21] involves utilizing Bayesian decision theory to minimize the error cause by uncertainty of the network state. Stankovic explored the performance of three decentralized control algorithms in [22] and concluded that tuned correctly, decentralized algorithms are stable and considerably improve response time and load balancing.

The most recent work in distributed dynamic scheduling has concentrated on bidding as the method of choice. Ramamrithram and Stankovic [23] refined the bidding mechanism to be a combination of a local scheduling algorithm (termed

the guarantee routine) and a task-sharing mechanism that asked processing nodes to bid on the likelihood that a surplus task would be accepted. Various refinements of the basic algorithm have been explored by Stankovic and others. One idea has been to focus on a select group of nodes (termed focussed addressing) from whom bids are elicited [24,25]. Another extension has been to include tasks with resource requirements [26,27] or precedence constraints [28]. Currently, work is under way to implement all these ideas into an operating system kernel [29].

The bidding mechanism for sharing tasks in a distributed network has been shown to be robust for a wide variety of tasks and system loads. Recent work by Chang [30] has suggested that for low loads (light traffic at each of the processing nodes), simple load sharing approaches would suffice. Kurose [31] has examined a simple distributed load sharing algorithm where each node processes tasks locally according to a pre-emptive FCFS run-time algorithm. If the task cannot be handled locally, it is sent at random to another processor, where it is either accepted or discarded. Tasks are permitted to be transferred only once; if a transferred task cannot be completed at the destination node, it is discarded. This simple algorithm was shown to perform well in very light traffic.

This thesis continues the examination of simple distributed scheduling algorithms for light traffic in distributed systems. In a significant extension to previous work, task loss rate for a distributed network is examined under pre-emptive FCFS, SJF, LLF and EDD local scheduling algorithms and unrestricted random routing of transferred tasks. Although analytical detail is given only for pairwise interaction, Chapter 7 shows how task loss for local nodes may be extended to approach exact analysis. Extension of the network model from homogeneous nodes to heterogeneous clusters of nodes is also given in Chapter 7.

2.6 Analysis of Scheduling Algorithms

The majority of current research involving the performance of scheduling algorithms requires that a simulation of the algorithm be carried out. Little an-

alytic performance prediction of scheduling algorithms is available owing to the intractability of the analysis. Most results carry over from queueing theory research and pertain to uniprocessor and multiprocessor research. In the published queueing results, the scheduling algorithm has normally been assumed to be FCFS.

One of the early results pertaining to hard real-time tasks was published by Gnedenko [32] and dealt with tasks with exponential computation and laxity parameters arriving in a Poisson stream to a multiprocessor. Tasks are sequenced by a central queue according to a FCFS discipline. More recent work by Bacelli [33,34] has extended the analysis of a multiprocessor to allow general density functions for computation and laxity. Analysis of a multiprocessor system yields the percentage of tasks accepted as well as the distribution of task sizes processed. To date, no results have been published pertaining to real-time tasks for other than FCFS service.

To compare simulated distributed algorithm results with analytical results, Stankovic [22] used three very simple models of multiprocessor systems to find a lower bound for performance. Kurose [31] has extended the analysis to a distributed system where there is a fixed common delay (assumed small relative to the total laxity) to send tasks between processors. The load-sharing algorithm studied by Kurose involved a probabilistic selection of which processor to send the task to based on arrival rates (assumed known) to the other processors. Approximate analytic results were then presented for the expected task rejection ratio at each processor.

In an attempt to provide analytical results for distributed systems, this thesis presents a method of analytical prediction for pre-emptive FCFS, SJF, LLF and EDD scheduling algorithms for the asymptotic case of light traffic. By using interactions between pairs of tasks, it is possible to approximate the density functions for rejected task computation and laxity. Using the analytic results for a single node, a model of a homogeneous network is then created that allows the prediction of real-time task loss based on an unrestricted number of task transfers and a random selection of processors. This simple task sharing algorithm is shown to be effective for the case of light traffic at each node in a homogeneous network.

Simple extensions to the model permit relaxation of the homogeneous network to allow groups of nodes with different processing characteristics.

Chapter 3

Light Traffic Performance Model for a Single Processor

3.1 Introduction

It is clear from [19] that low loss rates will be obtained only in light traffic, and that low utilization is simply the price to be paid for low loss rate when arrival times are unknown. This chapter describes analytic approximations of four pre-emptive run-time scheduling policies applied to a random arrival stream for the asymptotic case of low utilization. A Poisson stream of arrivals of rate λ to the processor will be assumed. The task computation and laxity requirements of a task are assumed to be known on arrival to the processor and described by general density functions. Computation and laxity random variables are assumed to be independent and identically distributed.

The assumption is made in the thesis is that pre-emption occurs at an arrival and tasks are ordered according to a specific policy. "Pre-emption" is used here to mean that the currently executing task is interrupted for the purposes of ordering the tasks and does not imply that another task is always chosen in its place. The current task may be chosen to continue and in effect this means that execution continues as if pre-emption had not occurred.

Task loss occurs if the reordering of tasks causes a deadline to be exceeded. Tasks are eliminated from the queue at the moment it is determined they have become infeasible. A complete characterization of lost tasks (mean rate of loss, execution and laxity distribution) is found for the asymptotic case. Comparison to simulated results indicate the loss rate approximation is quite close to reality even at 20 percent utilization. Execution and laxity distributions for rejected tasks are less sensitive to increases in traffic level, and are close to the actual distribution even at higher utilizations. The contents of this chapter have been abstracted as a separate report [35] and submitted for publication.

3.2 Low Traffic Analysis

The approach for low traffic is to examine the effect of pre-emptive scheduling on a pair of tasks in isolation. It is intuitively clear that if traffic is light enough, most cases of interference will involve only two tasks. This will be shown to be true for the asymptotic case of low utilization in Chapter 6. Therefore the present analysis considers a task which arrives at an idle processor, begins execution, and is later interrupted by a second task. The pre-empted task and the new task are then ordered according to the particular scheduling policy, which selects one of the pair for execution first. The other task may either be completed later or miss its deadline and be rejected from the queue. Task rejection is deemed to take place at the moment of selection. The characteristics of these rejected tasks give a first-order approximation to the stream of lost tasks in light traffic.

3.2.1 Definitions and Terminology

In order to characterize the effect of a scheduling policy on a pair of arrivals, a number of terms need to be defined first. Tasks that arrive to be scheduled will be referred to as *input tasks*. The first input task arrives at an idle processor and is assumed to be scheduled immediately. A second input task arrives at the busy

Figure 3.1: A pair of tasks arriving to be scheduled on a idle processor. The second task splits the first task into two parts: an *original pre-empted* and a *remaining pre-empted* task.

processor and pre-empts the current task. The pre-emptive discipline considered here is, that of *pre-emptive resume*, where the pre-empted task is permitted to resume execution from the point of interruption, assuming that it does not expire in the queue. The second arrival splits the first task into two parts: the *original pre-empted sub-task* and the *remaining pre-empted sub-task*. Figure 3.1 shows the relationship between the various tasks referred to.

The scheduling policy selects either the second input task or the remaining pre-empted task according to one of the following disciplines:

FCFS (First Come, First Served) The pre-empted sub-task is always chosen to be continued regardless of the characteristics of the second input task.

SJF (Shortest Job First) The one with the shorter computation requirement is chosen for execution.

LLF (Least Laxity First) The one with the shorter laxity remaining is chosen for execution.

EDD (Earliest Due Date) The one with the earlier deadline is chosen for execution.

In all cases, the task chosen to execute first is termed the *selected task* and the other the *non-selected task*. If the laxity of the non-selected task is less than the computation requirement of the selected task, the non-selected task will expire and is considered to be discarded.

To define the characteristics of each task class (original pre-empted, remaining pre-empted, second input, selected and non-selected) it is necessary to refer to corresponding density functions. There are three basic quantities of interest: computation, laxity and deadline. A convention will be established by the use of two subscripts on $f(x)$ in the form $f_{uv}(x)$ as required to indicate the random variable:

- Subscript u (The quantity of interest):
 - C: task computation requirement
 - L: task laxity requirement
 - D: task deadline requirement
- Subscript v (Which task or sub-task the quantity belongs to)
 - O: original pre-empted sub-task = task executed up to the moment of pre-emption.
 - P: remaining pre-empted sub-task = task remaining to be completed after pre-emption. (e.g., $f_{CP}(x)$ is the density function of the remaining computation for the pre-empted sub-task at the moment of pre-emption)
 - I: a general input task
 - S: selected task or sub-task
 - N: non-selected task or sub-task

In a similar manner, distribution functions will be indicated by appropriate subscripts on $F(x)$. Where it is necessary to refer to joint density functions, subscript groups will be separated by a comma, as in $f_{uv,wz}(x,y)$. Random variables may be conditioned on the event of non-selected task acceptance (denoted A) task

rejection (denoted R), or on another random variable. Densities conditioned on another random variable will be indicated by the notation $f_{uv|z}(x|y)$. Expectation of a random variable will be indicated by a bar notation (e.g., $\overline{CT}$ is the expected computation requirement for input tasks).

Three definitions will be adopted to refer to the following probabilistic events that may arise:

Sequential Pairwise Interference: A pair of tasks is defined to sequentially pairwise interfere if the inter-arrival time between tasks is less than the computation requirement of the first task. The probability of this event is denoted P_I . This is the situation shown in Figure 3.1.

Sequential Pairwise Rejection: If one of a pair of sequentially pairwise interfering tasks cannot be scheduled on an idle processor under a pre-emptive resume policy due to deadline constraints, that task is defined to be sequentially pairwise rejected. The probability of this event is denoted P_R .

Pre-empted Task Continuation: The scheduling policy selects the pre-empted sub-task to continue without interruption. The probability of this event is denoted P_C (In effect, no pre-emption in the conventional sense has occurred).

For the asymptotic light traffic case, $P_I P_R$ gives the probability of loss (Since other arrivals are ignored, this expression will in general under-estimate the actual loss). The aim of this chapter is to present a method to compute P_I , P_R , P_C and the characteristics of rejected tasks. The density functions derived for rejected tasks include their computation density $f_{CN|R}(x)$ and laxity density $f_{LN|R}(y)$. Deriving the rejected task characteristics will also require analysis of the selection process to find characteristics of selected and non-selected tasks.

3.2.2 General Results for Pairwise Interference

Before dealing with specific scheduling policies, some general results for pairwise interference will be derived first. Of particular interest is P_R , the task rejection probability for each scheduling policy. Also of interest are the characteristics of rejected tasks that result from each policy. This latter quantity determines the possibility of sharing the rejected task across a distributed network where delay is encountered sending the task across the network. If the remaining laxity at rejection is sufficiently large, a task may be sent across the network and scheduled elsewhere. Accordingly, the time of rejection is taken to be the point at which task selection take place in the case of pairwise interference. This section examines how P_R , P_I , $f_{CN|R}(x)$ and $f_{LN|R}(x)$ may be computed in general.

Given the definition of sequential pairwise interference, the probability that one or more tasks will interfere by arriving during the computation time of a general input task is given by:

$$P_I = \int_0^{\infty} f_{CI}(x)(1 - e^{-\lambda x})dx \quad (3.1)$$

The following analysis assumes that only *one* task arrives during the computation time of the first. The accuracy of this light traffic assumption may be estimated by evaluating the probability that n tasks interfere:

$$P_I^n = \int_0^{\infty} \frac{f_{CI}(x)\lambda^n e^{-\lambda x}}{n!} dx \quad (3.2)$$

Provided that P_I^n is small for $n > 1$ is small compared to P_I^1 , the assumption that $P_I \approx P_I^1$ is justified. The above computation underestimates the number of tasks that interfere with the first, as more tasks may arrive if the computation of the first is delayed due to pre-emption. Nevertheless, a pessimistic estimate of the number of tasks may be made, based on the fact that the task execution of an input task cannot be delayed beyond the available laxity. A pessimistic estimate for the average number of tasks that may interfere with an input task is given by $\lambda(CI + LI)$. This thesis assumes that $\lambda(CI + LI) \ll 1$, so that only pairwise interference need be considered in the approximation.

When pairwise interference occurs, the original pre-empted sub-task computation density function representing the work done until an arrival at time t is

given by:

$$f_{CO}(t) = \frac{\lambda e^{-\lambda t}(1 - F_{CI}(t))}{P_I} \quad (3.3)$$

The remaining time to completion for the pre-empted task is given by the residual computation time. It is well known [36] that for Poisson processes, the distribution of an arrival instant over an interval of length CI is uniformly distributed with the residual computation density given by:

$$f_{CP}(x) = \frac{1 - F_{CI}(x)}{CI} \quad (3.4)$$

Since the first arrival is assumed to arrive at an idle processor, it will have begun execution immediately. At the time of arrival of the second task, the laxity of the first task will have remained unchanged. The pre-empted sub-task and the second input task therefore have identical distributions for laxity. Once the scheduling decision is made (however this is done), rejection of the non-selected task depends on how long it must wait for the selected task to be completed. If the waiting time exceeds the non-selected task laxity, the non-selected task is rejected. Each of the task selection policies will be examined separately.

For the case of selection policies where CS and LN are independent (as in the FCFS, LLF and SJF policies) the general results may be carried one step further. The joint density function of CS and LN is (because of independence):

$$f_{CS, LN}(x, y) = f_{CS}(x)f_{LN}(y) \quad (3.5)$$

Figure 3.2 indicates the region R of (CS, LN) pairs which cause tasks to be rejected. Integrating the joint density function over this region gives the pairwise rejection probabilities for the FCFS, SJF and LLF scheduling policies:

$$P_R = \int_0^\infty \int_0^x f_{LN}(y)f_{CS}(x)dydx \quad (3.6)$$

The non-selected task laxity density function given rejection $f_{LN|R}(y)$ is easily derived by integrating the joint density function over R to give:

$$f_{LN|R}(y) = \frac{f_{LN}(y) \int_y^\infty f_{CS}(x)dx}{P_R} \quad (3.7)$$

Figure 3.2: Computing the pairwise rejection probability for tasks with independence between selected task computation and non-selected task laxity.

The non-selected task computation density function given rejection $f_{CN|R}(x)$ depends on whether the scheduling policy uses the computation requirement as part of the selection process. For the FCFS policy, the rejected task computation density function is simply the input task computation density.

To this point the assumption has been made that the pre-empted sub-task can resume processing from the point of interruption. For a distributed system with a pre-emptive resume scheduling policy for rejected tasks, the rejected task computation density function is simply $f_{CN|R}(x)$. This is the assumption used in the simulation work for this thesis.

For rejected tasks that may have to be executed on another processor, another possible discipline is *pre-emptive repeat*, in which the rejected task repeats execution from the beginning of the task. For pre-emptive repeat policies, the final rejected task computation density function $f_{CR}(x)$ depends on the relative likelihood that the non-selected task is either the pre-empted sub-task or the second input task. Since the selection process depends only on the work remaining

and not the work already performed on the pre-empted task, the rejected task computation density is given by:

$$f_{CR}(x) = P_C f_{CN|R}(x) + (1 - P_C) \int_0^x f_{CO}(z) f_{CN|R}(x - z) dz \quad (3.8)$$

This thesis assumes that *pre-emptive resume* is the policy followed for transferred tasks. However, Equation 3.8 may be used to adjust the rejected task computation density function to account for the alternative policy. In any case, what is now required is a description of how each scheduling algorithm selects between the arriving task and the pre-empted sub-task. The following sections describe the selection process and detail how to derive rejected task computation and laxity density functions. In addition, the method to obtain P_R and P_C will be specified for each scheduling algorithm.

3.2.3 FCFS Scheduling Policy

For pairwise interference, the second task is always non-selected and the remaining pre-empted sub-task is always selected to continue. Hence it follows that $P_C = 1$ by definition for this policy. The remaining characteristics follow as:

$$\begin{aligned} f_{CS}(x) &= f_{CP}(x) \\ f_{CN}(x) &= f_{CN|R}(x) = f_{CI}(x) \\ f_{LN}(x) &= f_{LS}(x) = f_{LI}(x) \end{aligned} \quad (3.9)$$

This completely characterizes FCFS scheduling in light traffic.

3.2.4 LLF Scheduling policy

Since LS is the minimum (LN is the maximum) of two independent variables distributed as LI , the distribution functions for LS and LN are given by:

$$\begin{aligned} F_{LS}(x) &= 1 - (1 - F_{LI}(x))^2 \\ F_{LN}(x) &= F_{LI}(x)^2 \end{aligned} \quad (3.10)$$

To obtain the density functions, the above expressions may be differentiated to give:

$$\begin{aligned} f_{LS}(x) &= 2f_{LI}(x)(1 - F_{LI}(x)) \\ f_{LN}(x) &= 2f_{LI}(x)F_{LI}(x) \end{aligned} \quad (3.11)$$

By symmetry, the selected task may be either the input or pre-empted sub-task with equal probability, since the laxity characteristics are indistinguishable. Hence $P_C = 1/2$ and the density functions for selected and non-selected computation are given by:

$$f_{CS}(x) = f_{CN}(x) = f_{CN|R}(x) = \frac{f_{CP}(x) + f_{CI}(x)}{2} \quad (3.12)$$

3.2.5 SJF Scheduling Policy

Density functions for CS and CN are computed from the density and distribution functions of the input and pre-empted sub-task, in a similar manner as for LLF scheduling. The complication here is that CI and CP are not identically distributed since the first task is partially completed at pre-emption. Hence the rejected task computation density depends on the selection process, since CI and CN are *not* independent for the SJF scheduling policy. In order to find which tasks are rejected as a consequence of the selection process, a conditional density of CN given CS must be found.

Following a similar procedure to the previous section, the distribution functions for CS and CN are given by:

$$\begin{aligned} F_{CS}(x) &= 1 - (1 - F_{CI}(x))(1 - F_{CP}(x)) \\ F_{CN}(x) &= F_{CI}(x)F_{CP}(x) \end{aligned} \quad (3.13)$$

Differentiating the distribution functions for CS and CN gives:

$$\begin{aligned} f_{CS}(x) &= f_{CP}(x)(1 - F_{CI}(x)) + f_{CI}(x)(1 - F_{CP}(x)) \\ f_{CN}(x) &= f_{CI}(x)F_{CP}(x) + f_{CP}(x)F_{CI}(x) \end{aligned} \quad (3.14)$$

Since the selection policy is not dependent on laxity, it follows that $f_{LS}(x) = f_{LN}(x) = f_{LN|R}(x) = f_{LI}(x)$. Computing the rejected task computation characteristic is more involved and requires that the relationship between CS and CN be determined. Since CS and LN are independent, Equation 3.6 may be used to calculate P_R . Integrating over the rejected task region R gives the conditional density function for $CS|R$:

$$f_{CS|R}(x) = \frac{f_{CS}(x) \int_0^x f_{LN}(y) dy}{P_R} \quad (3.15)$$

The conditional density function of $CN|CS$ must now be derived. Consider the task selection process between the pre-empted and input task. Since both these tasks are independent, the joint density function for CS and CP is given by $f_{CI,CP}(x, y) = f_{CI}(x)f_{CP}(y)$. The probability that the pre-empted sub-task is selected for service is computed by integrating over all (CI, CP) pairs where $CP \leq CI$:

$$P_C = \int_0^\infty \int_0^x f_{CI,CP}(x, y) dy dx \quad (3.16)$$

Let $CS = z_1$ as shown in Figure 3.3. The point (z_1, z_1) shown in Figure 3.3 corresponds to selecting z_1 as the minimum of CI and CP . Since CS is the minimum of CI and CP , the permissible values for CN (the maximum of CS and CP) is constrained to lie along one of two lines in the (CI, CP) subspace as shown in Figure 3.3. Let $CN = z$ where $z \geq z_1$. The two points (z, z_1) and (z_1, z) shown in Figure 3.3 are the two possible locations of CN in the (CI, CP) subspace. The conditional distribution function for $CN|CS$ is found by integrating over the subspace of (CI, CP) and normalizing:

$$F_{CN|CS}(z|z_1) = \frac{\int_{z_1}^z f_{CI,CP}(z_1, y) dy + \int_{z_1}^z f_{CI,CP}(x, z_1) dx}{\int_{z_1}^\infty f_{CI,CP}(z_1, y) dy + \int_{z_1}^\infty f_{CI,CP}(x, z_1) dx} \quad (3.17)$$

The density function for $CN|R$ may be computed using equations 3.15 and 3.17:

$$F_{CN|R}(x) = \int_0^x F_{CN|CS}(x|y) f_{CS|R}(y) dy \quad (3.18)$$

3.2.6 EDD Scheduling Policy

Figure 3.3: Computing the conditional distribution function for non-selected tasks, given the selected task computation

EDD scheduling involves both the laxity and computation density functions of the input task and the pre-empted sub-task in the selection process. Since CS and LN are not independent, the general method of Section 3.2.2 outlined earlier cannot be applied. To compute deadline density functions for input and pre-empted tasks, we note $DI = CI + LI$ and $DP = CP + LP$. These two deadlines are independent and distributed according to:

$$\begin{aligned} f_{DI}(z) &= \int_0^z f_{CI}(z-x)f_{LI}(x)dx \\ f_{DP}(w) &= \int_0^w f_{CP}(w-x)f_{LP}(x)dx \end{aligned} \quad (3.19)$$

The joint density functions of deadlines and either laxity or computation are given by the following expressions. Note that the joint density functions are zero outside the given range:

$$\begin{aligned} f_{CI,DI}(x,z) &= f_{CI}(x)f_{LI}(z-x) & 0 \leq x \leq z \\ f_{LI,DI}(y,z) &= f_{LI}(y)f_{CI}(z-y) & 0 \leq y \leq z \end{aligned}$$

$$\begin{aligned}
f_{CP,DP}(x,w) &= f_{CP}(x)f_{LP}(w-x) & 0 \leq x \leq w \\
f_{LP,DP}(y,w) &= f_{LP}(y)f_{CP}(w-y) & 0 \leq y \leq w
\end{aligned} \tag{3.20}$$

The conditional density functions for computation and laxity are computed as:

$$\begin{aligned}
f_{CI|DI}(x|z) &= \frac{f_{CI}(x)f_{LI}(z-x)}{f_{DI}(z)} & 0 \leq x \leq z \\
f_{LI|DI}(y|z) &= \frac{f_{LI}(y)f_{CI}(z-y)}{f_{DI}(z)} & 0 \leq y \leq z \\
f_{CP|DP}(x|w) &= \frac{f_{CP}(x)f_{LP}(w-x)}{f_{DP}(w)} & 0 \leq x \leq w \\
f_{LP|DP}(y|w) &= \frac{f_{LP}(y)f_{CP}(w-y)}{f_{DP}(w)} & 0 \leq y \leq w
\end{aligned} \tag{3.21}$$

These density functions are zero outside of the indicated ranges. Selection of either the pre-empted or input task depends on which of the two deadlines is more immediate. For a specific z and w , the input task computation and pre-empted task laxity (or pre-empted task computation and input task laxity) are independent and hence the joint conditional density function for CS and LN may be evaluated. The unconditioned density functions are found by integrating over the appropriate (z, w) pairs for each case. There are two cases, depending on which task is selected for service. The two cases are then combined to give the final result.

For selection of the input task, we must have $DI < DP$. The joint conditional density function for $CI|DI$ and $LP|DP$ for $0 \leq x \leq z$ and $0 \leq y \leq w$ uses the fact that DI and DP are independent:

$$f_{CS, LN|DI, DP, DI < DP}(x, y|z, w) = f_{CI|DI}(x|z)f_{LP|DP}(y|w) \tag{3.22}$$

Substituting the previous expression for conditional densities gives the joint density function of CS and LN :

$$f_{CS, LN|DI, DP, DI < DP}(x, y|z, w) = \frac{f_{CI}(x)f_{LI}(z-x)f_{LP}(y)f_{CP}(w-y)}{f_{DI}(z)f_{DP}(w)} \tag{3.23}$$

Figure 3.4 shows the region R for the (CS, LN) pairs resulting in task rejection. The probability of task rejection is given by integrating the joint density function over R :

Figure 3.4: Task selection for the case where $z < w$. Region R indicates CS, LN pairs where tasks are rejected.

$$P_{R|DI < DP}(z, w) = \int_0^z \int_0^x f_{CS, LN|DI, DP, DI < DP}(x, y|z, w) dy dx \quad (3.24)$$

Substituting the expression for joint density from Equation 3.23, the conditional rejection probability for this case becomes:

$$P_{R|DI < DP}(z, w) = \frac{\int_0^z \int_0^x f_{CI}(x) f_{LI}(z-x) f_{LP}(y) f_{CP}(z-y) dy dx}{f_{DI}(z) f_{DP}(w)} \quad (3.25)$$

The rejection probability for $DI < DP$ is given by integrating over all (z, w) pairs where $z < w$. Since the two tasks are independent, the joint density function $f_{DI, DP}(z, w) = f_{DI}(z) f_{DP}(w)$. The rejection probability for $DI < DP$ is given by:

$$P_{R|DI < DP} = \int_0^\infty \int_0^w P_{R|DI < DP}(z, w) f_{DI, DP}(z, w) dz dw \quad (3.26)$$

Finally, substituting for $P_{R|DI < DP}(z, w)$ from Equation 3.25 this becomes:

$$P_{R|DI < DP} = \int_0^\infty \int_0^w \int_0^z \int_0^x f_{CI}(x) f_{LI}(z-x) f_{LP}(y) f_{CP}(w-y) dy dx dz dw \quad (3.27)$$

This is an integral involving known functions which in general will have to be evaluated numerically.

Derivation for the case where $DP < DI$ is symmetrical in w and z , with $z > w$ and the rejection probability for this case is given by:

$$P_{R|DP < DI} = \int_0^\infty \int_0^w \int_0^z \int_0^x f_{CP}(x) f_{LP}(z-x) f_{LI}(y) f_{CI}(w-y) dy dx dz dw \quad (3.28)$$

The two cases must be combined to give the rejection probability for all (z, w) pairs. The rejection probability for the EDD scheduling policy is given by:

$$P_R = P_{R|DI < DP} + P_{R|DP < DI} \quad (3.29)$$

The probability that the selected task will be the pre-empted subtask is simply the probability that $DP < DI$, and hence $P(DP < DI) = P_C$. The probability for $DP < DI$ is computed from integration over all (DP, DI) pairs where $DP < DI$:

$$P(DP < DI) = \int_0^\infty \int_0^z f_{DI}(z) f_{DP}(w) dw dz \quad (3.30)$$

EDD Rejected Task Characteristics

Rejected task characteristics may be derived from marginal integration over region R shown in Figure 3.4. The same approach is used as in the case of computing rejection probability: the cases of selecting either the input or pre-empted task will be computed separately and then combined to give the total marginal density.

Consider marginal integration of rejected task laxity over the rejected region indicated in Figure 3.4. The region of integration for the case of $DI < DP$ is for $z < w$ and $0 \leq y \leq z$. To translate these conditions to the appropriate area in the (z, w) plane, consider Figure 3.5. The unconditioning of the marginal density function over z and w must be done over region I indicated in Figure 3.5. The two cases are combined in a similar manner as follows:

$$\begin{aligned} f_{LN|R, DI < DP}(y) &= \frac{\int_y^\infty \int_y^w \int_y^z f_{CI}(x) f_{LI}(z-x) f_{LP}(y) f_{CP}(w-y) dx dz dw}{P_R} \\ f_{LN|DP < DI}(y) &= \frac{\int_y^\infty \int_y^w \int_y^z f_{CP}(x) f_{LP}(z-x) f_{LI}(y) f_{CI}(w-y) dx dz dw}{P_R} \\ f_{LN|R}(y) &= f_{LN|R, DI < DP}(y) + f_{LN|R, DP < DI}(y) \end{aligned} \quad (3.31)$$

Figure 3.5: Task selection for $DI < DP$ where $z < w$. Region I indicates the area of integration on the (z, w) plane.

To derive the density function for $CN|R$, it is necessary to recall the relationship between C, L and D . For a given deadline $D = z$ and laxity $L = y$, the computation requirement is given by $C = z - y$. The previous method for deriving $f_{LN|R}(y)$ can be followed to obtain $f_{CN|R}(x)$. Figure 3.6 shows the same plot as Figure 3.4, but with the complementary values of $LS|DI < DP$ and $CN|DI < DP$ graphed instead. The joint density function for $LS|DI < DP$ and $CN|DI < DP$ is given by:

$$f_{LS,CN|DI,DP,DI < DP}(x, y|z, w) = \frac{f_{CI}(z - x)f_{LI}(x)f_{LP}(w - y)f_{CP}(y)}{f_{DI}(z)f_{DP}(w)} \quad (3.32)$$

Region R in Figure 3.6 again represents the fraction of rejected tasks, as it did in Figure 3.4, since the same task pairs are being indicated. The region of integration I in the (z, w) plane for $z < w$ is indicated in Figure 3.7 and is similar for $w < z$. As before, the two cases are combined according to:

$$f_{CN|R,DI < DP}(y) = \frac{\int_y^\infty \int_{w-y}^w \int_0^{z-w+y} f_{CI}(z - x)f_{LI}(x)f_{LP}(w - y)f_{CP}(y)dx dz dw}{P_R}$$

Figure 3.6: Task selection for $DI < DP$, where $z < w$. The complementary values LS and CN are plotted here. Region R indicates (LS, CN) pairs where tasks are rejected.

Figure 3.7: Task selection for the case where $z < w$. Region I is the region of integration in the (z, w) plane for the complementary values of LS and CN .

$$f_{CN|R, DP < DI}(y) = \frac{\int_y^\infty \int_{w-y}^w \int_0^{x-w+y} f_{CP}(z-x) f_{LP}(x) f_{LI}(w-y) f_{CI}(y) dx dz dw}{P_R}$$

$$f_{CN|R}(y) = f_{CN|R, DI < DP}(y) + f_{CN|R, DP < DI}(y) \quad (3.33)$$

3.3 Example Calculation

In general, numerical integration and differentiation is needed to evaluate probabilities P_I , P_R and P_C as well as $f_{CN|R}(x)$ and $f_{LN|R}(y)$. Analytical results for the special case of exponential density functions for both computation and laxity will be presented in Chapter 5. This section illustrates the utility of the method outlined above by considering as an example the task parameters shown in Table 3.1. Task computation times here are relatively long (1 sec) with laxity times of the same order of magnitude. Of interest is the ratio of computation to laxity and traffic intensity (ratio of inter-arrival time to computation). Similar results

<i>Parameter</i>	<i>Value</i>
Computation Density Function	Truncated Normal
Mean Computation	1.0 sec/task
Standard Deviation	0.5 sec
Laxity Density Function	Uniform
Laxity Range	0.0 to 1.0 sec
Traffic Intensity	0.02 to 0.20
Simulation Run Length	5000 tasks
Number of simulation runs averaged	80

Table 3.1: Simulation Parameters for Pre-emptive Local Scheduling Algorithms

would be obtained for any task parameters giving the same ratio of computation to laxity and traffic intensity.

A simulation of the various scheduling techniques was performed in order to compare the approximate analytic prediction with simulated performance. Histograms of rejected task computation and laxity were collected for each task arrival rate as well as task loss performance. Average task loss for a number of nodes with identical arrival rates was computed for a range of traffic intensity. Confidence intervals for task loss performance for each arrival rate were computed using the Student-t distribution for a series of simulation runs. The simulation parameters are also shown in Table 3.1.

3.3.1 Comparison to Simulation

Using the method outlined in the previous sections, task loss predictions for pre-emptive FCFS, SJF, LLF and EDD scheduling were done using numerical integration and differentiation. To generate the following figures, the input computation and laxity figures were sampled at a fixed interval. A fourth order differentiation and integration technique were then used to generate rejection

probabilities as well as rejected task laxity and computation densities.

The results are shown in Tables 3.2 to 3.5 as well as Figures 3.8 to 3.15. For the particular set of parameters considered here, the least task loss performance was achieved under EDD scheduling while the worst performance was given by FCFS scheduling. This result was observed to hold for various density functions for task parameters, including the exponential densities studied in Chapter 5. Task rejection performance of both the SJF and LLF pre-emptive scheduling algorithms depend on the relative ratio of mean computation to mean laxity of input tasks. In this example, SJF pre-emptive scheduling gave better task loss performance than LLF scheduling. A more comprehensive comparison and discussion is given in Chapter 5 for the case of exponential computation and laxity.

As can be observed from Tables 3.2 to 3.5, the accuracy of the predicted task loss degrades as traffic level increases. For low traffic intensity (up to approximately 10 percent utilization) the predicted results lie within the 90 percent confidence interval of simulated results. Above this level, predicted results lie below the actual traffic loss. As the traffic level increases, arriving tasks will interact with more than one task and in general simulated task loss will be greater than that predicted by pairwise interaction. Chapter 7 will outline how greater accuracy may be achieved at higher traffic levels.

Rejected task computation densities computed according to the previous sections and collected histogram data from simulations are shown in Figures 3.8 to 3.11. Histogram data for 20 percent traffic intensity is shown as this is likely have the most deviation from predicted results. As expected, the rejected task computation density for FCFS is identical to input computation density, as neither of these scheduling techniques is computation dependent. Rejected tasks for both the EDD and SJF scheduling algorithms tended to be biased to larger task sizes. In all cases, the simulation histograms agree closely with predicted results. Although not shown, histograms for lower traffic densities show similar agreement.

Similarly, rejected task laxity densities and histogram results are shown in Figures 3.12 to 3.15. Rejected task laxity is always shorter than input task laxity for each of the pre-emptive algorithms. Again, histogram data and predicted

Task Traffic Intensity	Analytic Result Predicted Fraction Task Loss	Simulated Result Lower 90 Percent Confidence Int.	Simulated Result Upper 90 percent Confidence Int.
0.02	0.0114	0.0111	0.0120
0.04	0.0225	0.0222	0.0231
0.06	0.0334	0.0334	0.0342
0.08	0.0440	0.0440	0.0451
0.10	0.0544	0.0540	0.0555
0.12	0.0644	0.0652	0.0662
0.14	0.0743	0.0750	0.0760
0.16	0.0839	0.0852	0.0878
0.18	0.0933	0.0950	0.0967
0.20	0.1024	0.1050	0.1071

Table 3.2: Analytic and Simulated results for fraction of task loss under the pre-emptive FCFS Local Scheduling Algorithm

Task Traffic Intensity	Analytic Result Predicted Fraction Task Loss	Simulated Result Lower 90 Percent Confidence Int.	Simulated Result Upper 90 percent Confidence Int.
0.02	0.0100	0.0098	0.0103
0.04	0.0198	0.0191	0.0200
0.06	0.0294	0.0288	0.0301
0.08	0.0387	0.0394	0.0402
0.10	0.0478	0.0475	0.0489
0.12	0.0566	0.0571	0.0586
0.14	0.0653	0.0660	0.0673
0.16	0.0737	0.0753	0.0771
0.18	0.0820	0.0847	0.0865
0.20	0.0900	0.0918	0.0937

Table 3.3: Analytic and Simulated results for fraction of task loss under the pre-emptive SJF Local Scheduling Algorithm

Task Traffic Intensity	Analytic Result Predicted Fraction Task Loss	Simulated Result Lower 90 Percent Confidence Int.	Simulated Result Upper 90 percent Confidence Int.
0.02	0.0119	0.0115	0.0118
0.04	0.0235	0.0234	0.0243
0.06	0.0349	0.0348	0.0359
0.08	0.0459	0.0459	0.0470
0.10	0.0567	0.0571	0.0588
0.12	0.0673	0.0692	0.0700
0.14	0.0775	0.0799	0.0815
0.16	0.0876	0.0921	0.0930
0.18	0.0974	0.1025	0.1035
0.20	0.1069	0.1129	0.1150

Table 3.4: Analytic and Simulated results for fraction of task loss under the pre-emptive LLF Local Scheduling Algorithm

Task Traffic Intensity	Analytic Result Predicted Fraction Task Loss	Simulated Result Lower 90 Percent Confidence Int.	Simulated Result Upper 90 percent Confidence Int.
0.02	0.0097	0.0088	0.0095
0.04	0.0192	0.0183	0.0190
0.06	0.0284	0.0275	0.0284
0.08	0.0374	0.0363	0.0376
0.10	0.0462	0.0448	0.0461
0.12	0.0548	0.0535	0.0555
0.14	0.0632	0.0626	0.0644
0.16	0.0713	0.0717	0.0732
0.18	0.0793	0.0796	0.0816
0.20	0.0871	0.0882	0.0894

Table 3.5: Analytic and Simulated results for fraction of task loss under the pre-emptive EDD Local Scheduling Algorithm

results show close agreement, even at a traffic intensity of 20 percent.

As a final example of how the approximate rejected task density functions deviate from actual density functions under high traffic load, a simulation of a single node under EDD pre-emptive scheduling was carried out at a traffic intensity of 3.0. The density functions for computation and latency and collected histograms at this load level are shown in Figures 3.16 and 3.17. Even at this high load level, the histogram results closely track predicted density functions.

3.4 Summary

This chapter has presented a means of approximating light traffic performance for four basic pre-emptive scheduling policies: FCFS, SJF, LLF and EDD. The motivation for approximating the performance of these algorithms is the widespread use of hard real-time systems and the lack of analytical results in this area. The approximation was compared with simulated results for these algorithms and shown to be useful over a range of traffic levels. Rejected task characteristics predicted from low traffic performance were also shown to give a good indication of the type of results to be expected and approximated the true density functions closely, even at high traffic levels.

Figure 3.8: Predicted density function for computation and collected histogram results from simulation of a single node under the pre-emptive FCFS scheduling algorithm. The load level for these results is 0.20

Figure 3.9: Predicted density function for computation and collected histogram results from simulation of a single node under the pre-emptive SJF scheduling algorithm. The load level for these results is 0.20

Figure 3.10: Predicted density function for computation and collected histogram results from simulation of a single node under the pre-emptive LLF scheduling algorithm. The load level for these results is 0.20

Figure 3.11: Predicted density function for computation and collected histogram results from simulation of a single node under the pre-emptive EDD scheduling algorithm. The load level for these results is 0.20

Figure 3.12: Predicted density function for laxity and collected histogram results from simulation of a single node under the pre-emptive FCFS scheduling algorithm. The load level for these results is 0.20

Figure 3.13: Predicted density function for laxity and collected histogram results from simulation of a single node under the pre-emptive SJF scheduling algorithm. The load level for these results is 0.20

Figure 3.14: Predicted density function for laxity and collected histogram results from simulation of a single node under the pre-emptive LLF scheduling algorithm. The load level for these results is 0.20

Figure 3.15: Predicted density function for laxity and collected histogram results from simulation of a single node under the pre-emptive EDD scheduling algorithm. The load level for these results is 0.20

Figure 3.16: Predicted density function for computation and collected histogram results from simulation of a single node under the pre-emptive EDD scheduling algorithm. The load level for these results is 3.0

Figure 3.17: Predicted density function for laxity and collected histogram results from simulation of a single node under the pre-emptive EDD scheduling algorithm. The load level for these results is 3.0

Chapter 4

Task Rejection Bounds.

4.1 Introduction

Approximate task loss prediction given by Chapter 3 may not meet a need to be conservative and guarantee the task loss rate. The next best thing to an exact analytic solution (available only for FCFS) is a set of upper and lower bounds for the rejection rate. This Chapter partly meets that goal by deriving bounds for task loss. The pairwise interaction approximation of Chapter 3 is extended to an exact lower bound on rejection. A second conjectured upper bound is developed that is both highly plausible and supported by simulated results. These bounds are shown to be tight in the limit of low traffic and Poisson arrivals. Using the results from Chapter 3, the bounds are shown to be easily computed for any general input computation and laxity distributions. The example studied in Chapter 3 will be analyzed to show how useful these bounds are in guaranteeing performance. Exact analytic solutions for exponential computation and laxity density functions will be given in Chapter 5. These bounds enable the real-time systems developer to guarantee rejection performance within a given range for low-traffic hard real-time traffic applications.

The restriction to low-traffic hard real-time applications need not imply that the processor is under-utilized. For example, this result may be useful for

systems where aperiodic tasks can be given priority over other tasks in the system. An example of this type of system is a processor running non-real-time background tasks that must also respond to urgent real-time alarm conditions. In such applications, the rate of real-time traffic may be very low, but it may be imperative to be able to guarantee that the probability of not responding to such alarms within a given deadline is very small.

4.2 Pre-emptive Scheduling Bounds

The pre-emptive scheduling algorithms considered for bounding are the same as for Chapter 3. The scheduler knows the execution time and laxity (and hence the deadline) at the time of arrival of a task. Note that pre-emptive FCFS scheduling schedules the tasks in the same order as non-pre-emptive scheduling but allows for the rejection of a task at the moment of arrival if its deadline will not be met. For real-time purposes, early detection of an inability to meet a deadline is critical in order to implement contingency plans to deal with the failure. Tasks that cannot be completed are considered to be rejected as soon as the scheduling failure is detected.

4.2.1 Acceptance Lower Bound

A lower bound on acceptance (upper bound on rejection) may be conjectured by considering a processor with a restricted scheduler. Chapter 3 assumed that each arrival arrived either at an idle processor or a busy processor with only one task. In either case, an arrival was always eligible to be scheduled. This section models a processor that schedules arrivals only under a limited set of circumstances. Figure 4.1 shows such a processor with a Poisson arrival stream of real-time tasks which are either blocked or accepted for scheduling. It may only accept tasks for scheduling under the following conditions:

Figure 4.1: A restricted processor that schedules only a subset of arriving real-time tasks.

- The processor is idle, or
- The arriving task is the first to arrive to a busy processor (i.e., the second task in a busy period).

All tasks arriving to the processor in other circumstances are considered blocked and automatically rejected. By allowing such a limited set of circumstances for scheduling of tasks, the rejection of tasks is conjectured to be higher than if all tasks were scheduled unconditionally. Computing the rejection probability for the restricted processor then forms an upper bound on scheduling performance for the unrestricted processor.

4.2.2 Acceptance Probability for a Restricted Processor

The advantage of considering the restricted processor is that the acceptance probability for arriving tasks under these conditions may be calculated. From flow conservation, the accepted fraction of tasks a of a Poisson input stream of tasks λ is computed from:

$$a\lambda = \frac{\text{Prob}(\text{computing a task that will eventually be completed})}{\text{average computation of completed tasks}} \quad (4.1)$$

To determine the probability that the processor is computing a task that will eventually complete, it is necessary to draw a distinction between *useful* and *wasted* work. When a task is pre-empted on the restricted processor, the scheduling algorithm must choose to either continue the current task or commence execution of the arriving task. If the current task is pre-empted and then discarded due to scheduling constraints, the work already performed on that task is *wasted* in the sense that it does not count toward work done on completed tasks. The only scheduling algorithm exempt from this possibility is the FCFS algorithm, which always chooses to continue with the pre-empted task. The following definitions are used to distinguish useful and wasted work:

B_U : Average useful work done in the busy period of a restricted processor.

B_W : Average wasted work done in the busy period of a restricted processor.

Using these definitions, the probability the restricted processor is doing useful work is computed as:

$$\text{Prob}(\text{computing tasks that will eventually be completed}) = \frac{B_U}{B_U + B_W + \frac{1}{\lambda}} \quad (4.2)$$

The average computation length of completed tasks depends on the number of tasks completed in a busy period and the amount of useful busy work. We define the following probabilities for the restricted processor.

P_I : The probability that one or more tasks will arrive during the execution period of a task.

P_R : Given that two tasks are scheduled under a pre-emptive discipline, the probability that one of the tasks will be rejected.

P_C : Given that two tasks are scheduled under a pre-emptive discipline, the probability that the first task will be selected to continue.

Using these definitions, the average completed task length for the restricted processor is given by:

$$\text{average computation of completed tasks} = \frac{B_U}{1 + P_I(1 - P_R)} \quad (4.3)$$

Combining 4.2 and 4.3, the lower bound to acceptance a_L given by acceptance to a restricted processor is therefore given by:

$$a_L = \frac{1 + P_I(1 - P_R)}{1 + \lambda B_U + \lambda B_W} \quad (4.4)$$

4.2.3 Useful and Wasted Busy Period Calculation

In order to compute the average busy period for a restricted processor, it is necessary to consider the various cases that might arise and the probability

Figure 4.2: The various possible busy periods for the restricted processor and associated probabilities. The shaded area indicates wasted work done in a busy period.

associated with each case. Figure 4.2 details the four cases that might arise, noting the various task and sub-task lengths that arise from the selection process.

Following a similar notation to that introduced in Chapter 3, the average values of various random variables associated with computation length will be required. One additional random variable not previously defined will be introduced here:

$\hat{CQ}$ Computation requirement of a task that is not pre-empted before completion.

The average useful busy work done in a busy period may be expressed as a sum of the four cases detailed in Figure 4.2, using mean values of the computation random variables:

$$B_U = \overline{CQ}(1 - P_I) + (\overline{CO} + \overline{CS})P_IP_RP_C + \overline{CS}P_IP_R(1 - P_C) + (\overline{CO} + \overline{CS} + \overline{CN|A})P_I(1 - P_R) \quad (4.5)$$

Similarly, wasted work done in a busy period corresponds to only one of the cases and is calculated as:

$$B_W = \overline{CO}P_IP_R(1 - P_C) \quad (4.6)$$

The probabilities for P_I , P_R , P_C as well as density functions for CO , CN , CP and $CN|R$ were derived separately for each scheduling discipline in Chapter 3. The remaining density function may be derived from the following information.

The density function for CQ is given by:

$$f_{CQ}(x) = \frac{e^{-\lambda x} f_{CI}(x)}{1 - P_I} \quad (4.7)$$

While Chapter 3 gave expressions for the density function of $CN|R$, the complementary value of $\overline{CN|A}$ may be computed according to:

$$\overline{CN|A} = 2\overline{CN} - \overline{CN|R} \quad (4.8)$$

Explicit expressions for $f_{CS}(x)$ and $f_{CN}(x)$ are found in Chapter 3 for all pre-emptive disciplines except EDD. An explicit evaluation is included here in the

notation defined in Chapter 3 for completeness:

$$f_{CS}(x) = f_{CI}(x) + f_{CP}(x) - f_{CP}(x) \int_x^\infty f_L(z-x) F_{DI}(z) dz - f_{CI}(x) \int_x^\infty f_L(z-x) F_{DP}(z) dz \quad (4.9)$$

$$f_{CN}(x) = f_{CP}(x) \int_x^\infty f_L(z-x) F_{DI}(z) dz + f_{CI}(x) \int_x^\infty f_L(z-x) F_{DP}(z) dz \quad (4.10)$$

Using the above expressions we can compute mean values for the above computation parameters, and then compute B_U, B_W and hence a lower acceptance bound a_L for general computation and laxity input density functions.

4.2.4 Validity of Lower Bound Conjecture

As stated, the scheduling restrictions described for a processor do not prove that the rate of acceptance for tasks will be less than for an equivalent unrestricted processor. In order to prove the conjecture that scheduling restrictions lead to a lower bound on the number of tasks rejected, it is necessary to be able to compare the two processing models directly.

In the case of FCFS pre-emptive service, the comparison is reasonably simple. The two processors behave identically until the arrival of a third task to the busy period. The restricted processor automatically rejects the third task, while the unrestricted processor accepts the third and subsequent task into the queue. Thus the unrestricted processor will always have a longer busy period and all work completed is by definition *useful* work. The average work done per task is the same in both cases, and by equation 4.1 and 4.2 the acceptance of the unrestricted processor will always be greater. Hence for FCFS service at least, the restricted processor acceptance is provably a lower bound.

For other type of pre-emptive service, comparing the restricted and unrestricted case is not as easy. The arrival of the third task may imply an increase

in useful busy work for the unrestricted processor, or it may not. For example, allowing the third task to be scheduled may mean that the first two tasks are eventually lost due to deadline constraints and never be completed. While the average work done per completed task for the unrestricted processor is likely to be less than that of the restricted processor for the pre-emptive disciplines studied, this is difficult to show without knowing the complete analytic solution. A proof of an acceptance lower bound for the other pre-emptive disciplines involves bounding the useful work done in a busy period for each processor as well as bounding the average work completed per task. Neither of these bounds have been proven for non-FCFS pre-emptive service.

Nevertheless, extensive simulation work supports the usefulness of the conjectured lower bound for acceptance for the restricted processor to be useful. The bound appears to hold even in cases where perverse real-time scheduling is done (e.g., latest deadline first at pre-emption). The conjecture that automatically rejecting tasks instead of permitting them to remain in the queue leads to a lower bound for acceptance is an intuitive and reasonable one. The advantage of being able to compute such a bound, as well as the support of simulated results, are compelling reasons for its use.

4.3 Acceptance Upper Bound

An upper bound to acceptance (lower bound to rejection) may be derived from considering the minimum scheduling requirement of a stream of input tasks. For a task and its predecessor or successor to eventually be completed by a run-time scheduling algorithm, it is necessary to be able to schedule both tasks in isolation. An upper bound to acceptance considers all tasks in the input stream and considers how many tasks can be eliminated strictly as a result of comparison with nearest neighbors. Any tasks that are separated by successive arrival intervals must be pairwise compatible (able to be scheduled in isolation) by a run-time algorithm to be eligible for eventual completion.

Arrival intervals are divided into even and odd

Figure 4.3: An input arrival stream arbitrarily separated by even and odd arrival intervals. In order to be eligible under a pre-emptive scheduling policy, all adjacent tasks must be at least pairwise compatible.

Consider the arrival stream shown in Figure 4.3. Arrival intervals are arbitrarily labeled even and odd in order to group arrivals into pairs. Consider the even intervals first. Each even interval pair has a probability P_I of interfering (i.e. the second task pre-empting the execution of the first) when considered in isolation. For an arrival pair separated by an even arrival interval, with probability $P_I P_R$ the pair has one arrival that must be eliminated as a candidate for completion as a result of pairwise comparison. Regardless of whatever other scheduling may take place, both arrivals of the pair can never be completed.

Now consider the arrival pairs formed by the odd intervals. As a result of considering the even intervals first, not all tasks remain in the original input stream. It is important not to "double count" tasks eliminated under even interval consideration. In general, every even interval comparison that eliminates a task also invalidates an odd interval comparison, since the odd pair is no longer separated by a single interval.¹ Comparing the remaining odd interval pairs gives a probability of $P_I P_R (1 - P_I P_R)$ that an odd arrival pair has one arrival eliminated as a candidate for completion.

Since even and odd intervals have equal probability, the total rejection

¹If successive even intervals eliminate tasks, this one for one invalidation may not strictly hold. However, in order to reject the fewest tasks, we consider every even interval that eliminates a task to also invalidate an odd interval comparison.

probability based on arrivals separated by successive arrival intervals is given by $P_I P_R / 2 + P_I P_R (1 - P_I P_R) / 2$. Note that this considers pairs of arrivals in isolation, and that further scheduling consideration will eliminate more arrivals. However, this preliminary elimination of arrivals from consideration gives an upper bound on acceptance given by:

$$a_U = 1 - P_I P_R (1 - \frac{P_I P_R}{2}) \quad (4.11)$$

4.4 Tightness of Acceptance Bounds

Both the lower and upper acceptance bound can be shown to become tight in the limit of low traffic as $\lambda \rightarrow 0$. In the limit as $\lambda \rightarrow 0$, $P_I \rightarrow \lambda \overline{CI}$ and the two limits can be shown to converge to the same expression to first order in λ .

Consider the upper bound for acceptance given by Equation 4.11. In the limit as $\lambda \rightarrow 0$, $a_U \rightarrow 1 - \lambda \overline{CI} P_R$. Now consider the lower bound as given by Equation 4.4. As $\lambda \rightarrow 0$, $B_U \rightarrow \overline{CI}$ and $B_W \rightarrow 0$ to first order in λ . Expanding Equation 4.4 to first order terms in λ gives:

$$a_L \rightarrow (1 + \lambda \overline{CI} (1 - P_R)) (1 - \lambda \overline{CI}) \quad (4.12)$$

Simplifying Equation 4.12 to first order terms in λ , we have $a_L \rightarrow 1 - \lambda \overline{CI}$ and the two limits become identical to first order in λ . Hence in the limit of low traffic, both the upper and lower bounds merge to become tight.

4.5 Acceptance Bounds and Simulated Results

To show the tightness of both acceptance bounds, calculations were done for the example studied in Chapter 3. As before, the general calculation requires that numerical integration techniques be used. The special case of exponential computation and laxity density functions has a simple closed form and will be presented in Chapter 5. Tables 4.1 to 4.4 summarize the upper and lower bounds

computed. The simulation results from Chapter 3 are repeated here for ease of comparison.

Task Traffic Intensity	Analytic Lower Bound Task Loss	Simulation Lower 90 Percent Confidence Int.	Simulation Upper 90 percent Confidence Int.	Analytic Upper Bound Task Loss
0.02	0.0113	0.0111	0.0120	0.0116
0.04	0.0220	0.0222	0.0231	0.0233
0.06	0.0323	0.0334	0.0342	0.0350
0.08	0.0421	0.0440	0.0451	0.0467
0.10	0.0514	0.0540	0.0555	0.0585
0.12	0.0603	0.0652	0.0662	0.0701
0.14	0.0688	0.0750	0.0760	0.0817
0.16	0.0767	0.0852	0.0878	0.0932
0.18	0.0846	0.0950	0.0967	0.1046
0.20	0.0919	0.1050	0.1071	0.1158

Table 4.1: Task loss bounds and simulation confidence intervals for task loss for the pre-emptive FCFS Local Scheduling Algorithm

For all the pre-emptive scheduling algorithms the 90-percent confidence intervals for simulation results overlap the upper and lower acceptance bounds. As traffic increases, the two bounds diverge and are not very effective above a 20 percent traffic intensity. Below 10 percent load, the bounds guarantee a rejection rate within ± 10 percent of the midpoint between them. Other computation and laxity density functions, as well as other pre-emptive scheduling policies were simulated, compared to bounds, and gave similar results.

The rate of divergence of the two bounds depends not only on the traffic intensity but also on P_R . The upper bound diverges rapidly in cases where P_R is small. In these cases, the upper bound is overly pessimistic about the probability of rejection. To give an example, consider the previous example with FCFS

Task Traffic Intensity	Analytic Lower Bound Task Loss	Simulation Lower 90 Percent Confidence Int.	Simulation Upper 90 percent Confidence Int.	Analytic Upper Bound Task Loss
0.02	0.0099	0.0098	0.0103	0.0102
0.04	0.0194	0.0191	0.0200	0.0206
0.06	0.0285	0.0288	0.0301	0.0311
0.08	0.0372	0.0394	0.0402	0.0417
0.10	0.0455	0.0475	0.0489	0.0522
0.12	0.0534	0.0571	0.0586	0.0628
0.14	0.0610	0.0660	0.0673	0.0734
0.16	0.0682	0.0753	0.0771	0.0840
0.18	0.0753	0.0847	0.0865	0.0944
0.20	0.0819	0.0918	0.0937	0.1049

Table 4.2: Task loss bounds and simulation confidence intervals for task loss for the pre-emptive SJF Local Scheduling Algorithm

scheduling. Figure 4.4 plots the upper and lower bounds for the previous example as well as an example with much lower P_R . For the information tabulated in Table 4.1, P_R is 56% while for the second curve in Figure 4.4, P_R has been lowered to 15% by extending the laxity uniform density function to include the range 0 to 4 seconds. Note how the two bounds diverge more quickly for the second curve with smaller P_R .

4.6 Summary

This chapter presented a method of computing bounds for task rejection probability using pre-emptive scheduling policies. The motivation behind this work was to enable a systems developer to guarantee performance when only a pe-

Figure 4.4: Task loss upper and lower bounds for FCFS Pre-emptive scheduling and different P_R . The upper curves have $P_R = 56\%$ while the lower curves have $P_R = 15\%$.

Task Traffic Intensity	Analytic Lower Bound Task Loss	Simulation Lower 90 Percent Confidence Int.	Simulation Upper 90 percent Confidence Int.	Analytic Upper Bound Task Loss
0.02	0.0117	0.0115	0.0118	0.0121
0.04	0.0230	0.0234	0.0243	0.0244
0.06	0.0337	0.0348	0.0359	0.0366
0.08	0.0438	0.0459	0.0470	0.0490
0.10	0.0535	0.0571	0.0588	0.0614
0.12	0.0627	0.0692	0.0700	0.0737
0.14	0.0715	0.0799	0.0815	0.0859
0.16	0.0799	0.0921	0.0930	0.0980
0.18	0.0879	0.1025	0.1035	0.1101
0.20	0.0955	0.1129	0.1150	0.1220

Table 4.3: Task loss bounds and simulation confidence intervals for task loss for the pre-emptive LLF Local Scheduling Algorithm

riodic real-time tasks were present. It was noted that most performance guarantees existed for periodic tasks and aperiodic tasks were given lower priority. Analysis of aperiodic task performance has been previously handled either by simulation or limited to FCFS queueing theory results.

This chapter introduced the notion of wasted work within a busy period in one of the bounds and showed that for non-FCFS pre-emptive policies, non-productive work formed part of the busy period. Using a model of a processor restricted to scheduling at most two tasks, a lower bound for acceptance was conjectured based on an exact calculation of the busy period. An upper bound for acceptance was based on the notion of minimal pairwise acceptance of tasks by a run-time policy. Both of these bounds were shown to converge and become tight in the limit of low traffic. The utility of both of these bounds was shown for a specific example of Poisson arrivals of tasks with exponential computation and

Task Traffic Intensity	Analytic Lower Bound Task Loss	Simulation Lower 90 Percent Confidence Int.	Simulation Upper 90 percent Confidence Int.	Analytic Upper Bound Task Loss
0.02	0.0096	0.0088	0.0095	0.0099
0.04	0.0188	0.0183	0.0190	0.0200
0.06	0.0276	0.0275	0.0284	0.0303
0.08	0.0360	0.0363	0.0376	0.0407
0.10	0.0441	0.0448	0.0461	0.0511
0.12	0.0518	0.0535	0.0555	0.0615
0.14	0.0592	0.0626	0.0644	0.0720
0.16	0.0663	0.0717	0.0732	0.0824
0.18	0.0730	0.0796	0.0816	0.0928
0.20	0.0795	0.0882	0.0894	0.0103

Table 4.4: Analytic and Simulated results for fraction of task loss under the pre-emptive EDD Local Scheduling Algorithm

laxity requirements.

Coupled with the results from Chapter 3, the performance of run-time pre-emptive scheduling policies on a single processor has been characterized, including the rejection probability and the characteristics of rejected tasks. One of the assumptions made in Chapter 3 was that task rejection took place at the moment of scheduling, when a task first becomes infeasible. The motivation for not permitting the task to remain in the queue was that the possibility existed that the task could be scheduled elsewhere. This possibility depends on the amount of remaining rejected task laxity as well as the delay characteristics of the distributed network. Chapter 6 will approximate distributed network behavior using low traffic performance results for a single processor.

Chapter 5

Analysis for Exponential Density Functions

5.1 Introduction

Analytic results for task loss, characteristics of rejected tasks and task loss bounds in general require numerical techniques for the solution of the single node. The case of exponential density functions for both computation and laxity is appealing for several reasons:

- Simple, closed form solutions exist for all four pre-emptive scheduling algorithms for all of the equations presented in Chapter 3 and Chapter 4.
- The exponential distribution gives a wide variation in task parameters and clearly shows the different performance of the four pre-emptive scheduling techniques.
- The analytic form of solution gives additional insight into the comparative advantage of the pre-emptive scheduling techniques as a function of the task computation to task laxity.
- The exact solution for task loss is available in the literature for non-pre-emptive FCFS scheduling [32], which is identical to pre-emptive FCFS

scheduling task loss.

5.2 Single Node Analytic Solution

The Poisson stream of arrivals at rate λ has exponential task laxity and density functions, given in the notation of Chapter 3 as follows:

$$\begin{aligned} f_{CI}(x) &= \mu e^{-\mu x} \\ f_{LI}(x) &= \gamma e^{-\gamma x} \end{aligned}$$

The memoryless property of the exponential density function makes computation of the pre-empted computation density function trivial. When the second input task arrives, the pre-empted computation density function $f_{CP}(x)$ is equal to $f_{CI}(x)$. Since the pre-empted sub-task and second input task have identical density functions for both computation and laxity in this special case, $P_C = 0.5$ for all scheduling algorithms except FCFS where $P_C = 1$. The probability of interference of a second task for the light traffic approximation is given for all four pre-emptive scheduling algorithms as:

$$P_I = \frac{\lambda}{\mu + \lambda} \quad (5.1)$$

Table 5.1 summarizes the calculation of P_R for each of the four pre-emptive scheduling policies. Table 5.2 and Table 5.3 show the analytic solution for rejected task laxity and computation respectively. (The form of the solution for rejected task computation shown in Table 5.3 assumes that $\gamma \neq \mu$; a simpler form exists when this is true). The predicted task loss approximation is then given by $P_I P_R$ for each of the pre-emptive algorithms.

5.2.1 Comparison to Simulation and Exact Result

To gain an appreciation for the relative performance of each of the algorithms, a simulation of the four pre-emptive policies was done for the same range

service	P_R
FCFS	$\frac{\gamma}{\mu + \gamma}$
SJF	$\frac{\gamma}{2\mu + \gamma}$
LLF	$\frac{2\gamma^2}{(\mu + \gamma)(\mu + 2\gamma)}$
EDD	$\frac{\gamma^2}{(\mu + \gamma)^2}$

Table 5.1: P_R for Poisson arrivals with exponential service and laxity

service	Rejected Task Laxity
FCFS	$(\mu + \gamma)e^{-(\mu + \gamma)x}$
SJF	$(2\mu + \gamma)e^{-(2\mu + \gamma)x}$
LLF	$\frac{(\mu + \gamma)(\mu + 2\gamma)}{\gamma}(e^{-(\mu + \gamma)x} - e^{-(\mu + 2\gamma)x})$
EDD	$(\mu + \gamma)e^{-(\mu + \gamma)x}$

Table 5.2: $f_{LN|R}(x)$ for Poisson arrivals with exponential service and laxity

service	Rejected Task Computation
FCFS	$\mu e^{-\mu x}$
SJF	$(\frac{2\mu(2\mu + \gamma)}{\gamma} - \frac{2\mu^2(2\mu + \gamma)}{\gamma(\mu + \gamma)})e^{-\mu x} - \frac{2\mu(2\mu + \gamma)}{\gamma}e^{-2\mu x} + \frac{2\mu^2(2\mu + \gamma)}{\gamma(\mu + \gamma)}e^{-(2\mu + \gamma)x}$
LLF	$\mu e^{-\mu x}$
EDD	$\frac{\mu^2(\mu + \gamma)^2}{\gamma - \mu}((\frac{1}{\mu(\mu + \gamma)} - \frac{1}{\gamma(\mu + \gamma)})e^{-\mu x} - \frac{1}{\mu(\mu + \gamma)}e^{-2\mu x} + \frac{1}{\gamma^2}e^{-2\gamma x})$

Table 5.3: Calculation of $f_{CN|R}(x)$ for Poisson arrivals with exponential service and laxity. The equations presented here assume that $\gamma \neq \mu$.

<i>Parameter</i>	<i>Value</i>
Mean Computation	1.0 sec/task
Mean Laxity	1.0 sec/task
Traffic Intensity	0.02 to 0.20
Simulation Run Length per node	5000 tasks
Number of simulation runs averaged	80

Table 5.4: Simulation Parameters for Exponential Computation and Laxity Density

of traffic intensities examined in Chapter 3. The simulation parameters are listed in Table 5.4. Statistics were collected for task loss and 90 percent confidence intervals were computed for each traffic level intensity. (Although not shown here, collected histogram data for rejected task computation and laxity showed the same close agreement to the predicted densities as in Chapter 3.)

Figure 5.1 plots the simulated results and compares the light traffic approximation for each of the pre-emptive scheduling algorithms over the range of traffic levels tested. For the particular choice of parameters ($\gamma = \mu$) the performance of the LLF and SJF pre-emptive algorithms are predicted to be identical. While true in the asymptotic case of light traffic, Figure 5.1 shows that the task loss performance of the SJF and LLF algorithms diverge as traffic intensity increases. As expected, the exact result for FCFS pre-emptive scheduling passes through the centre of the confidence intervals for FCFS service. While diverging from the actual result as traffic level increases, the approximate result does give an accurate indication of the relative performance of each algorithm.

The present analysis permits many additional insights into hard real-time scheduling. For example, the "best" scheduler (the one with minimum rejection rate) can be evaluated for light traffic. Figure 5.2 is a good example of the power of the analytic method to evaluate various scheduling policies. In most cases, evaluation by simulation only gives the scheduling method for a particular set of

Figure 5.1: A comparison of simulated results and the asymptotic light traffic approximation for the FCFS, LLF, SJF and EDD scheduling policies

task parameters. As task parameters change, the scheduling method giving the least task loss may change.

Figure 5.2 plots rejection probability as a function of the dimensionless quantity γ/μ . As can be observed, rejection probability increases in a different manner for each of the four scheduling methods as the ratio of computation to laxity increases. For the case of exponential density functions for both computation and laxity, the following conclusions are possible:

- FCFS scheduling for arriving tasks gives the highest probability of rejection, regardless of the ratio of computation to laxity.
- EDD scheduling for arriving tasks gives the lowest probability of rejection, regardless of the ratio of computation to laxity.
- SJF scheduling gives a lower rejection probability than LLF scheduling whenever mean computation exceeds mean laxity. The reverse is true if mean laxity exceeds mean computation.
- As the ratio of computation to laxity increases, the difference in rejection probability between the various methods also increases.

5.3 Task Loss Bounds

Analytic solutions also exist for pre-emptive scheduling bounds as described in Chapter 4. For all algorithms, $\overline{CO} = \overline{CQ} = 1/(\mu + \lambda)$. Table 5.5 gives the expression for $\overline{CS}$ and $\overline{CN|A}$ for each of the four pre-emptive scheduling algorithms. These results, together with the results from Section 5.2 enable the upper and lower rejection bounds for each algorithm to be expressed in closed form.

5.4 Summary

Figure 5.2: Rejection performance plotted as a function of γ/μ for the FCFS, LLF, SJF and EDD scheduling policies. Note how the asymptotic low traffic behavior gives additional insight into relative performance of the policies.

service	$\overline{CS}$	$\overline{CN A}$
FCFS	$\frac{1}{\mu}$	$\frac{1}{\mu}$
SJF	$\frac{1}{2\mu}$	$\frac{3\mu^2+4\mu\gamma+\gamma^2}{\mu(2\mu+\gamma)(\mu+\gamma)}$
LLF	$\frac{1}{\mu}$	$\frac{1}{\mu}$
EDD	$\frac{2\mu^2+2\mu\gamma+\gamma^2}{2\mu(\mu+\gamma)^2}$	$\frac{2\mu^2(\mu+\gamma)}{(\mu^2+2\mu\gamma)} \left(\frac{1}{\mu^2} - \frac{1}{2(\mu+\gamma)^2} \right)$

Table 5.5: $\overline{CS}$ and $\overline{CN|A}$ for Poisson arrivals with exponential service and laxity

This chapter has presented analytic solutions for the special case of tasks with exponential computation and laxity density functions. The closed form solutions presented gave additional insight into pre-emptive scheduling behavior and permit quick solution to the task loss behavior of the pre-emptive scheduling algorithms studied, without the need for numerical integration.

Chapter 6

Light Traffic Performance for a Homogeneous Network

6.1 Introduction

Recently Kurose [31] has shown that simple load sharing techniques, at least for the case of computational resources, can be nearly as effective as more complex or optimum scheduling strategies. The selection of an execution node is made on the basis of a simple local workload threshold and average workloads elsewhere. Although the decision is made dynamically (as tasks arrive), the scheduling choice does not require knowledge of the state of the network and hence the algorithm is termed *quasi-dynamic*. Kurose assumed a limitation of one transfer per task and FCFS (first-come, first served) scheduling was assumed at each node. In related research, other simple non-pre-emptive scheduling algorithms have been simulated and compared for distributed systems by the present authors [19].

Support for simple load sharing strategies also comes from research by Eager, [37] who compared various load sharing policies for non real-time systems. Eager showed that extremely simple adaptive load sharing policies, which collect very small amounts of system state information, can yield dramatic performance improvements. This thesis assumes random task sharing, which does not collect

any state information from the network. For non real-time tasks, random load sharing strategies suffer from instability; no matter what the average load, the system eventually enters a state where the processors in a network are devoting all their time to transferring tasks and none of their time to processing them. This instability does not occur with real-time tasks and distributed networks, as finite task laxity and transfer delay impose a limit on the number of times a task may be transferred. The question of performance improvement possible using small amounts of state information in homogeneous real-time systems remains open.

Following the suggestion in [31], this chapter investigates the performance of a simple task sharing strategy for several simple pre-emptive scheduling strategies. It develops the light traffic performance characteristic for a network of homogeneous nodes, each with identical traffic characteristics. The network connection is assumed to allow transfer of tasks between processors with a given delay distribution. As an extension to [31], nodes are permitted to transfer tasks as often as the available laxity and network delay permits, rather than discarding the task after it has been transferred once. The pre-emptive scheduling algorithms studied are the same as in Chapters 3 and 5.

As will be shown by a simulation example, the network rejection characteristics can be modeled quite well using a light traffic approximation, even for traffic intensity as high as 20 percent at each node. Transferred task computation and laxity density functions are modeled satisfactorily by this light traffic approximation. Input tasks may have general computation and laxity density functions, although the arrivals at each node are assumed to be Poisson. The result is applicable to systems with large numbers of homogeneous nodes with identical input traffic characteristics. Results from this chapter enable the analytic performance prediction of networks that would otherwise require extensive simulation time. This is particularly true in the case of light traffic, where task rejection is a rare event.

6.2 Network Model

The model of a distributed computer network used in this chapter is shown in Figure 6.1. The network consists of N nodes connected by some communications network. A task can be transferred from one node to another in a random time D described by a density function $f_D(x)$, including all necessary setup, communication and transfer processing delays to transfer a task from one node to another. Task transfer may include the necessary state information to enable a pre-empted task to restart (pre-emptive resume) or the task may be required to execute again from the beginning (pre-emptive repeat). We assume that a task submitted at any node in the network may be transferred and executed at any other node in the network.

Each node has an external Poisson stream of hard real-time tasks arriving at rate λ . All tasks have identical computation and laxity distributions. Each task is independent and is assumed to require only computational resources. The random computation requirement of each task C_E and laxity requirement L_E are described by density functions $f_{C_E}(x)$ and $f_{L_E}(x)$ respectively. The hard deadline for task completion is the arrival instant plus $(C_E + L_E)$. Computation and laxity requirements are known on arrival at each node.

The system attempts to schedule all arriving tasks within the deadline specified by each one. If a task cannot be executed by its deadline, it is considered lost and discarded from the system. Arriving tasks are first scheduled at the local node, according to a pre-emptive scheduling discipline. The disciplines considered in this chapter are pre-emptive FCFS, SJF, LLF and EDD. If an arriving task cannot be executed locally, the task-sharing scheme attempts to transfer the task to meet the deadline specified. Tasks are not discarded until the delay in sending a task from one node to another has resulted in an expired deadline. Tasks with expired deadlines are detected on arrival at a node or within the communications network. In real-time terminology, this system is defined as a *soft* real time system in that the loss of a task does not imply system failure.

The load sharing strategy considered here is similar to [31] and does not

Figure 6.1: The model of a distributed processing system of N nodes connected by a communication network. Each node may communicate tasks across the network after a random delay of D seconds. A typical node is shown for reference, with both the input and transferred traffic.

utilize the state of the network when making a decision as to where to send a task. The load sharing scheme is as follows:

- Each node make a decision on arrival of a task whether to accept the arriving task, send the arriving task to another node, or accept the arriving task and send other previously accepted tasks to another node. Tasks are transferred if and only if accepting a task would result in a deadline being missed locally.
- If a task must be transferred, the decision as to where it should be sent is random, based on an equal probability for any other processor in the network. No record is kept of the past history of transfers. A task is eligible to be transferred between processors any number of times, providing that at each transfer, there remains some positive laxity for task execution.

* Note that a task that initially arrives to a node always has positive laxity and hence a rejected task is always transferred to another node at least once. Total traffic to a single node therefore consists of initial input traffic and transferred tasks. In the following section, we examine a typical node in the network and develop equations of task flow that characterize task loss.

The following analysis is based on homogeneity of the network. One node is analyzed on the assumption that:

1. All processing nodes have identical input traffic characteristics.
2. The task stream to each node is Poisson. It is well known from renewal theory that the merging of many non-Poisson streams tends toward a Poisson process, so the merged transfer process into a representative node will tend to be Poisson in a large homogeneous system, thus satisfying our assumption.

6.3 Task Flow for a Single Node

Consider a typical node on the network, as shown in Figure 6.2. Task flow into the node consists of input traffic λ and a fraction of the input traffic $\alpha\lambda$ that

is transferred from other nodes on the network. The transferred traffic from other nodes is referred to as *feedback* traffic, since in general the transferred tasks may have originated from any node, including the local one. In a steady state condition, the feedback traffic will have computation and laxity requirements described by density functions $f_{CF}(x)$ and $f_{LF}(x)$ respectively. On the assumption that the interarrival-time distribution of feedback tasks is Poisson, the total input traffic into a node is Poisson at a rate $(1 + \alpha)\lambda$ with a computation and laxity density function given by:

$$f_{CI}(x) = \frac{1}{1 + \alpha} f_{CE}(x) + \frac{\alpha}{1 + \alpha} f_{CF}(x) \quad (6.1)$$

$$f_{LI}(x) = \frac{1}{1 + \alpha} f_{LE}(x) + \frac{\alpha}{1 + \alpha} f_{LF}(x) \quad (6.2)$$

The method presented in Chapter 3 will be used to estimate the fraction of input traffic rejected locally for the case of low traffic intensity. In the notation of Chapter 3, the flow of rejected tasks out of a typical processor in steady state is given by $P_I P_R (1 + \alpha)\lambda$, with P_I and P_R computed according to the local scheduling method utilized.

6.4 Feedback Traffic Solution

Each rejected task at a node enters the network and is subject to a delay determined by the network characteristics. Tasks whose deadlines expire during transmission across the network are said to "expire in the network", even though task expiry is not detected until arrival at another processing node. These tasks are the only jobs which are finally lost and never processed successfully. Note that transferred tasks arriving at a node may include tasks that may have been transferred several times.

The transferred stream of tasks from other nodes (termed the "feedback stream", since it may include tasks that originated locally) that have not expired

2 of/de 2

Resolution Test Chart (ANSI #2) - 100% Contrast

MicroD

Resolution Test Chart (ANSI #2) - 100% Contrast

Figure 6.2: A typical node in the distributed processing network and the task flow of input and rejected tasks. A steady state flow of tasks and sufficient nodes to make the feedback traffic Poisson is assumed.

in the network is shown in Figure 6.2. The probability that a transferred task will not expire in the network is given by:

$$P_N = \int_0^\infty \int_0^y f_D(x) f_{LN|R}(y) dx dy \quad (6.3)$$

The "feedback" stream thus has a steady state rate of $P_I P_R P_N (1 + \alpha) \lambda$. From flow consideration around the network, it is possible to express the feedback traffic in terms of the rejected tasks:

$$\alpha \lambda = P_I P_R P_N (1 + \alpha) \lambda \quad (6.4)$$

$$\alpha = \frac{P_I P_R P_N}{1 - P_I P_R P_N} \quad (6.5)$$

The laxity characteristic of feedback tasks may be computed from the density functions of rejected task laxity and network delay. The feedback task laxity is the difference between rejected task laxity and network delay, conditioned on the feedback laxity being positive. The distribution function for feedback task laxity is given by:

$$F_{LF}(z) = \frac{\int_0^\infty \int_z^{z+y} f_{LN|R}(y) f_D(x) dy dx}{P_N} \quad (6.6)$$

Computation characteristics of feedback tasks are identical to rejected tasks, i.e. $f_{CF}(x) = f_{CN|R}(x)$.

At this point, we have equations for $f_{CN|R}(x)$ (rejected task computation), $f_{LN|R}(x)$ (rejected task laxity) and $P_I P_R$ (fraction of input tasks that get rejected by the local node) given in Chapter 3 for any input computation density $f_{CI}(x)$ and any input task laxity $f_{LI}(x)$. Table 6.1 references the appropriate equations in Chapter 3 to use to compute P_R , $f_{CN|R}$ and $f_{LN|R}$ for general input distributions.

The input stream contains tasks that originate at the node, as well as feedback tasks. Hence to compute the rejected task characteristics we require the feedback task characteristics, and vice versa. Since P_I , P_R , and P_N are all dependent on α , Equation 6.5 gives an implicit equation for α that is in general difficult to solve analytically. This set of relationships may be solved iteratively by the following scheme:

1. Set the fraction of feedback tasks $\alpha = 0$.

<i>Scheduling Algorithm</i>	<i>Equation P_R</i>	<i>Equation $f_{CN R}$</i>	<i>Equation $f_{LN R}$</i>
FCFS	3.9,3.6	3.9	3.9,3.7
LLF	3.11,3.12,3.6	3.12	3.11,3.12,3.7
SJF	3.14,3.6	3.14,3.15	3.7
EDD	3.27,3.28,3.29	3.33	3.31

Table 6.1: Equation references required to compute P_R , $f_{CN|R}$ and $f_{LN|R}$ for general input distributions.

2. Adjust the input traffic consisting of external tasks and feedback tasks with computation density $f_{CF}(x)$ and laxity density $f_{LF}(x)$.
3. Solve for P_I, P_R and P_N in terms of the input traffic characteristics $f_{CI}(x)$ and $f_{LI}(x)$ using the appropriate equations in Table 6.1. Compute a feedback laxity density function $f_{LF}(x)$ and computation density function $f_{LF}(x)$ using Equation 6.6 and a feedback fraction α^{new} using Equation 6.5.
4. If α and α^{new} are equal to within some specified tolerance, then the steady state feedback fraction has been found and the iteration stops. Otherwise, step 2 and 3 are repeated until convergence is achieved.

For the wide variety of cases examined, this scheme converged within 3 or 4 iterations. Accuracy of the technique depends upon the accuracy of numerical integration techniques used to solve for P_I, P_R and P_N . A satisfactory result has been achieved using a fixed grid and a modified trapezoidal integration having an error of $O(h^4)$, where h is the grid step utilized.

Once a value for α has been computed, the fraction of tasks lost may be found from steady state flow consideration. The probability of loss P_L is given by:

$$P_L = P_I P_R (1 + \alpha) - \alpha \quad (6.7)$$

6.5 Network Prediction and Simulated Results

To show the utility of the distributed network model derived in the previous sections, a simulation of an 8-node homogeneous network was carried out using the parameters described in Table 6.2. The parameters in Table 6.2 are identical to Table 3.1 in Chapter 3 for input traffic to a single node. In this simulation, processing nodes may transfer tasks across the network according to an exponential delay characteristic. The loss of tasks in the network was recorded and a 90 percent confidence interval for task loss was computed, based on a series of runs. The prediction of task loss based on the model of the previous sections was also computed, using the iterative technique for α . Once a stable value for α was obtained, the density functions for feedback laxity and computation were recorded. During the simulation, histograms were recorded for the task computation and laxity of transferred tasks. Tables 6.3 to 6.6 summarize the results for both analytic prediction and simulation for each of the pre-emptive local scheduling algorithms.

Examining the task loss obtained from simulation, we see that FCFS pre-emptive local scheduling resulted in the greatest average task loss, while the EDD local scheduling algorithm gave the least average loss. Both the SJF and LLF algorithms gave about the same level of task loss performance over the traffic intensity range examined. Comparing the simulated and predicted results, we see that in general the predicted task loss results lie below the results from simulation and converge as $\lambda \rightarrow 0$. Predicted task loss for the EDD local scheduling algorithm converges for low traffic, but lies outside the simulation 90 percent confidence interval for all traffic intensities. An interesting observation is that the LLF task loss prediction diverges much more quickly from simulated results than the other algorithms. At 20% processor utilization, the predicted task loss is about 20 to 25% low compared to simulated results.

In terms of relative performance, the predicted results indicate that the LLF local scheduling algorithm should be almost on a par with the EDD local scheduling algorithm but simulated results indicate that it is in fact far worse.

<i>Parameter</i>	<i>Value</i>
Computation Density Function	Truncated Normal
Mean Computation	1.0 sec/task
Standard Deviation	0.5 sec
Laxity Density Function	Uniform
Laxity Range	0.0 to 1.0 sec
Traffic Intensity	0.02 to 0.20
Network Delay	Exponential
Delay/Task	mean 1.5sec/task
Simulation Run Length	10000 rejected tasks
Number of Nodes	8 nodes
Number of simulation runs averaged	10

Table 6.2: Simulation Parameters for Pre-emptive Local Scheduling Algorithms and simple random task sharing in a Homogeneous Network of 8 Nodes.

This discrepancy is likely due to the assumption of an infinite population of nodes for task transfer for the analytic prediction versus the simulated restriction of 8 nodes. Unlike other algorithms, the task laxity at rejection for the LLF algorithm leaves sufficient time for many transfers between nodes, and task loss prediction is likely to be sensitive to the number of nodes actually available.

To show how the analytic prediction method permits the approximate prediction of feedback density for computation and laxity, the collected histogram data from a traffic intensity of 20% is compared with the predicted densities. Again, as in Chapter 3, only the highest traffic intensity is shown since lower traffic intensities show at least as close an agreement. Figures 6.3 to 6.6 compare feedback computation density functions for FCFS, SJF, LLF and EDD local scheduling algorithms respectively. Similarly, Figures 6.6 to 6.9 compare feedback laxity density functions for FCFS, SJF, LLF and EDD local scheduling algorithms. As in Chapter 3 the predicted distributions closely match actual simulated data.

Traffic Intensity	Predicted Fraction Task Loss	Lower 90 Percent Confidence Int.	Upper 90 percent Confidence Int.
0.02	0.0078	0.0077	0.0084
0.04	0.0158	0.0158	0.0164
0.06	0.0235	0.0244	0.0253
0.08	0.0311	0.0318	0.0330
0.10	0.0385	0.0407	0.0418
0.12	0.0457	0.0481	0.0493
0.14	0.0527	0.0565	0.0575
0.16	0.0599	0.0635	0.0651
0.18	0.0667	0.0716	0.0734
0.20	0.0734	0.0799	0.0812

Table 6.3: Analytic and simulated results for fraction of task loss under the pre-emptive FCFS local scheduling algorithm and simple random task sharing in a Homogeneous Network of 8 Nodes.

Traffic Intensity	Predicted Fraction Task Loss	Lower 90 Percent Confidence Int.	Upper 90 percent Confidence Int.
0.02	0.0069	0.0069	0.0073
0.04	0.0136	0.0143	0.0148
0.06	0.0203	0.0222	0.0230
0.08	0.0268	0.0288	0.0297
0.10	0.0333	0.0364	0.0376
0.12	0.0396	0.0441	0.0454
0.14	0.0458	0.0514	0.0525
0.16	0.0519	0.0580	0.0594
0.18	0.0580	0.0662	0.0670
0.20	0.0639	0.0727	0.0742

Table 6.4: Analytic and simulated results for fraction of task loss under the pre-emptive SJF local scheduling algorithm and simple random task sharing in a Homogeneous Network of 8 Nodes.

Traffic Intensity	Predicted Fraction Task Loss	Lower 90 Percent Confidence Int.	Upper 90 percent Confidence Int.
0.02	0.0061	0.0066	0.0069
0.04	0.0121	0.0135	0.0142
0.06	0.0179	0.0207	0.0214
0.08	0.0235	0.0276	0.0286
0.10	0.0290	0.0349	0.0358
0.12	0.0344	0.0427	0.0435
0.14	0.0396	0.0496	0.0510
0.16	0.0446	0.0574	0.0590
0.18	0.0496	0.0648	0.0667
0.20	0.0543	0.0727	0.0748

Table 6.5: Analytic and simulated results for fraction of task loss under the pre-emptive LLF local scheduling algorithm and simple random task sharing in a Homogeneous Network of 8 Nodes.

Traffic Intensity	Predicted Fraction Task Loss	Lower 90 Percent Confidence Int.	Upper 90 percent Confidence Int.
0.02	0.0059	0.0060	0.0064
0.04	0.0117	0.0123	0.0127
0.06	0.0175	0.0187	0.0194
0.08	0.0231	0.0251	0.0258
0.10	0.0287	0.0313	0.0325
0.12	0.0342	0.0381	0.0393
0.14	0.0395	0.0443	0.0453
0.16	0.0448	0.0509	0.0521
0.18	0.0500	0.0575	0.0585
0.20	0.0550	0.0641	0.0655

Table 6.6: Analytic and simulated results for fraction of task loss under the pre-emptive EDD local scheduling algorithm and simple random task sharing in a Homogeneous Network of 8 Nodes.

Figure 6.3: Predicted density function for feedback computation and collected histogram results from simulation of a network of 8 nodes under the pre-emptive FCFS scheduling algorithm and a random task sharing algorithm. The load level for these results is 0.20

Figure 6.4: Predicted density function for feedback computation and collected histogram results from simulation of a network of 8 nodes under the pre-emptive SJF scheduling algorithm and a random task sharing algorithm. The load level for these results is 0.20

Figure 6.5: Predicted density function for feedback computation and collected histogram results from simulation of a network of 8 nodes under the pre-emptive LLF scheduling algorithm and a random task sharing algorithm. The load level for these results is 0.20

Figure 6.6: Predicted density function for feedback computation and collected histogram results from simulation of a network of 8 nodes under the pre-emptive EDD scheduling algorithm and a random task sharing algorithm. The load level for these results is 0.20

Figure 6.7: Predicted density function for feedback laxity and collected histogram results from simulation of a network of 8 nodes under the pre-emptive FCFS scheduling algorithm and a random task sharing algorithm. The load level for these results is 0.20

Figure 6.8: Predicted density function for feedback laxity and collected histogram results from simulation of a network of 8 nodes under the pre-emptive SJF scheduling algorithm and a random task sharing algorithm. The load level for these results is 0.20

Figure 6.9: Predicted density function for feedback laxity and collected histogram results from simulation of a network of 8 nodes under the pre-emptive LLF scheduling algorithm and a random task sharing algorithm. The load level for these results is 0.20

Figure 6.10: Predicted density function for feedback laxity and collected histogram results from simulation of a network of 8 nodes under the pre-emptive EDD scheduling algorithm and a random task sharing algorithm. The load level for these results is 0.20

6.6 Summary

In this chapter we have studied a method of approximating the performance of a relatively simple task sharing scheme for a network of homogeneous processors in a soft real-time environment. As suggested in [31], simple task sharing schemes often perform as well as more sophisticated algorithms. The technique described in this chapter permit an approximate evaluation of task loss for a variety of preemptive scheduling algorithms and arbitrary input task computation and laxity density functions. The delay characteristic of the network could also be described by an arbitrary density function. The simple task sharing scheme proposed in this paper is an extension to the one proposed in [31], in that a task may be transferred among nodes on the network any number of times.

The limitations of this chapter are the assumption that feedback traffic to be Poisson and the requirement for a homogeneous network of nodes with identical input traffic. The approximate light traffic analysis at each node described in Chapter 3 also limits the accuracy of the result. Nevertheless, in spite of all these limiting assumptions, the agreement of predicted and simulated results indicate the utility of the result. The limiting assumption of Poisson traffic and an infinite number of nodes shows up most clearly in LLF local scheduling, which permits the greatest number of transfers.

Chapter 7

Extensions and Enhancements

7.1 Introduction

The previous chapters have described an approximate method to compute task loss and rejected task characteristics for both a single node and a network of homogeneous nodes. The utility of the approximate method depends greatly on the assumption that the arriving task "sees" only the previous task and either interferes or does not interfere with that task alone. This is the same as the assumption that in low traffic the probability of more than one arrival occurring to the processor is negligible. As has been observed in the previous chapters, it is this assumption that causes the predicted results for task loss to differ from true values since, in general, arriving tasks may find more than one task waiting to be completed.

The attractiveness of the assumption of low traffic intensity is that it obviates the need for complex queueing analysis required for the exact result. In order to extend the range of task loss prediction without introducing prohibitive complexity, some procedure is required to refine the estimate of the task loss at higher traffic levels. This chapter outlines a method by which the estimate of task loss may be refined so as to approach the true value of task loss as closely as desired.

A second possible refinement is to relax the requirement that the network model consist of homogeneous nodes with identical traffic characteristics. The network model may be expanded to include "clusters" of homogeneous sub-networks, each of which has identical node traffic characteristics and where tasks are transferred within and between clusters.

The intention of this chapter is merely to demonstrate that the results of Chapters 3 to 6 may be generalized in these two directions, by outlining the generalization procedure. Full solutions will not be given, except in special cases.

7.2 Double Interval Estimation of Task Loss

As has been mentioned, the analysis of the preceding chapters assumes that each arrival "sees" only the previous arrival and is rejected or accepted solely on pairwise compatibility with that arrival. The obvious way to enhance the approximation is to allow the current arrival to see not only the previous arrival but also the next-to-previous arrival. The approximation of interference with a single previous arrival is defined to be *single interval estimation*. Extending this to the previous two arrivals is defined as *double interval estimation*. Continuing the process, we define *Nth interval estimation* to be allowing the current arrival to interact with the previous N arrivals. As $N \rightarrow \infty$, the approximation approaches exact queuing theory analysis.

For double interval estimation, rejection takes place based on various cases of interaction between the current arrival and the previous two arrivals. For convenience, the following abbreviations are adopted:

A: The current arrival.

P: The arrival immediately previous to the current arrival.

NP: The next-to-previous arrival to the current arrival.

S: The selected arrival between P and NP.

NS: The non-selected arrival between P and NP.

In order to detail all the various event probabilities associated with double interval estimation, it is necessary to extend the notation introduced in Chapter 3. Rather than simply two events for the single interval estimation case, Figure 7.1 details twelve possible events that arise from double interval estimation. This rather rich event space shows how quickly extending the interval estimation process complicates calculation of rejection probability. Figure 7.1 shows the acceptance or rejection of task A, conditioned on the possible acceptance or rejection events associated with task P and NP. The following shorthand is used to describe the event space:

$\text{int}(T_1, T_2)$: the event that tasks T_1 and T_2 interfere.

$\text{int}(T_1, T_2, T_3)$: the event that task T_3 interferes with *both* tasks T_1 and T_2 .

$\text{nonint}(T_1, T_2)$: The event that tasks T_1 and T_2 do *not* interfere.

$\text{nonint}(T_1, T_2, T_3)$: the event that task T_3 does *not* interfere with *either* task T_1 or T_2 .

$\text{acc}(T_1)$: The event that task T_1 is accepted.

$\text{acc}(T_1, T_2)$: The event that both of tasks T_1 and T_2 are accepted.

$\text{acc}(T_1, T_2, T_3)$: The event that all of tasks T_1 , T_2 and T_3 are accepted.

$\text{rej}(T_1, T_2)$: The event that one of tasks T_1 and T_2 is rejected.

$\text{nonrej}(T_1, T_2)$: The event that neither task T_1 or T_2 is rejected.

$\text{rej1}(T_1, T_2, T_3)$: The event that *one* of tasks T_1, T_2 or T_3 is rejected.

$\text{rej2}(T_1, T_2, T_3)$: The event that *two* of tasks T_1, T_2 or T_3 is rejected.

As can be seen from Figure 7.1, there are twelve possible events that can occur as a result of double interval estimation. Of these, seven events lead to acceptance of task A whereas five events lead to at least one rejected task. One

Figure 7.1: The acceptance or rejection events for arriving task A, conditioned on acceptance or rejection of tasks P and NP. The final event space has five events that reject one or more tasks, and seven events that accept task A.

event ($\text{rej2}(A, S, NS)$) actually results in a rejection of two tasks, one of which was previously accepted. For FCFS pre-emptive service, the probability of this event is zero, since once a task has been accepted in FCFS service, it is never subsequently rejected as a consequence of later arrivals. Note how the computation of probabilities for the double interval estimation interval builds on the results from single interval estimation; the density functions for selected and non-selected tasks have already been determined from the results in Chapter 3. Once the rejection events have been defined, the computation and laxity distributions for rejected tasks may be computed as outlined in Chapter 3 and combined according to the probability of each rejection event. The rejection probability per arrival is computed as the sum of the probabilities of the rejection events:

$$\begin{aligned} \text{Prob}(\text{rejection}) = & \text{Prob}(\text{rej}(A, P)) + \text{Prob}(\text{rej}(A, S)) + \text{Prob}(\text{rej1}(A, S) + \\ & \text{Prob}(\text{rej2}(A, S, NS)) + \text{Prob}(\text{rej}(A, NS)) \end{aligned} \quad (7.1)$$

Generalizing to N th interval estimation requires that acceptance or rejection of the arriving task be conditioned on the acceptance and rejection events of the N th - 1 task. For moderate N , the number of events can grow quickly. As $N \rightarrow \infty$, the number of events becomes unbounded. However, it is likely that the rejection probability will converge quickly. As the next section illustrates, double interval estimation gives a substantial improvement over single interval estimation, even at high traffic loads.

7.2.1 FCFS Double Interval Estimation

As an example of the possible improvement using double rather than single interval estimation, the case of FCFS pre-emptive scheduling with exponential computation and laxity was re-examined. As noted in Chapter 5, the exact solution for task loss is known and therefore gives a baseline for comparison. Table 7.1 summarizes the calculation of event probabilities for the acceptance or rejection of the arriving task in the order given in Figure 7.1.

<i>Final Event in Figure 7.1</i>	<i>Event Probability</i>
acc(A)	$\frac{\mu^2}{(\mu+\gamma)^2}$
acc(A,P)	$\frac{\mu^2\lambda}{(\mu+\lambda)^2(\mu+\gamma)}$
rej(A,P)	$\frac{\mu\lambda\gamma}{(\mu+\lambda)^2(\mu+\gamma)}$
acc(A)	$\frac{\mu\lambda\gamma}{(\mu+\lambda)^2(\mu+\gamma)}$
acc(A,S)	$\frac{\mu\lambda^2\gamma}{(\mu+\lambda)^2(\mu+\gamma)^2}$
rej(A,S)	$\frac{\lambda^2\gamma^2}{(\mu+\lambda)^2(\mu+\gamma)^2}$
acc(A)	$\frac{\lambda\mu^3}{(\mu+\lambda)^3(\mu+\gamma)}$
acc(A,S,NS)	$\frac{\lambda^2\mu^3}{(\mu+\lambda)^3(\mu+\gamma)^2}$
rej1(A,S,NS)	$\frac{\lambda^2\mu(2\mu\gamma+\gamma^2)}{(\mu+\lambda)^3(\mu+\gamma)^2}$
rej2(A,S,NS)	0
acc(A,NS)	$\frac{\lambda^2\mu^3}{(\mu+\lambda)^3(\mu+\gamma)^2}$
rej(A,NS)	$\frac{\lambda^2\gamma^2\mu}{(\mu+\lambda)^3(\mu+\gamma)^2}$

Table 7.1: FCFS Pre-emptive Scheduling with event probabilities for double interval estimation

Traffic Intensity	Single Interval Task Loss	Double Interval Task Loss	Exact Solution of Task Loss
0.02	0.009723	0.009946	0.009934
0.04	0.019072	0.019771	0.019736
0.06	0.028069	0.029458	0.029409
0.08	0.036734	0.038993	0.038954
0.10	0.045083	0.048366	0.048374
0.12	0.053134	0.057569	0.057670
0.14	0.060903	0.066596	0.066845
0.16	0.068404	0.075444	0.075899
0.18	0.075650	0.084110	0.084835
0.20	0.082655	0.092593	0.093654

Table 7.2: Single interval and Double interval estimates for the fraction of task loss under the pre-emptive FCFS Local Scheduling Algorithm along with the exact solution

The rejection probability per arrival is given by equation 7.1 as

$$\begin{aligned}
 Prob(rejection) = & \frac{\mu\lambda\gamma}{(\mu + \lambda)^2(\mu + \gamma)} + \frac{\lambda^2\gamma^2}{(\mu + \lambda)^2(\mu + \gamma)^2} + \\
 & \frac{\lambda^2\mu(2\mu\gamma + \gamma^2)}{(\mu + \lambda)^2(\mu + \gamma)^3} + \frac{\lambda^2\gamma^2\mu}{(\mu + \lambda)^3(\mu + \gamma)^2} \quad (7.2)
 \end{aligned}$$

The double event rejection probabilities were computed for the case example in Chapter 5 and compared to the single interval estimate. Table 7.2 shows equation 7.2 computed for each traffic level and gives the exact solution [32] as well as the single interval estimate.

As can be seen from Table 7.2, double interval estimation gives a dramatic improvement in the accuracy of the task loss estimate. Even at a traffic load of 0.20, the double interval estimate is within 2% of the actual value. The single interval estimate error is of the order of 12%, showing the magnitude of the improvement possible. Note that the rejection term now contains second order terms

in λ .

7.3 Extension to Heterogeneous Clusters of Networks

One other limiting assumption done in the analysis to date has been the requirement that the network consist of homogeneous nodes with identical traffic characteristics. It is possible to extend the network model to include "clusters" of homogeneous nodes, each of which may have different traffic characteristics and where clusters may have different processing characteristics. The approach in this section is to show the generalization for two clusters, and then outline the procedure for N clusters.

Figure 7.2 shows a typical node of cluster 1 in network of two homogeneous clusters. There are N_1 nodes in cluster 1 and N_2 nodes in cluster 2 where N_1 and N_2 are both large. Each cluster has an arrival rate λ_i and a feedback rate α_i . There is a probability γ_i that the task within a cluster will stay within the homogeneous set and a probability $1 - \gamma_i$ that it will be transferred to nodes of the other cluster. If equal routing probability is initially chosen as the default, then $\gamma_1 = N_1/(N_1 + N_2)$ and $\gamma_2 = N_2/(N_1 + N_2)$.

Each typical node in a cluster is assumed to have some steady state input which consists, in general, of input tasks to the cluster, transferred tasks from other nodes in the same cluster, and transferred tasks from the other cluster. The feedback task flow components are defined to consist of a fraction α of input tasks from the same cluster and Φ of input tasks from the other cluster. Each typical node has probabilities of P_{Ii} , P_{Ri} and P_{Ni} for pairwise interference, pairwise rejection and network retention. All of these quantities can be computed for an input flow with general computation and laxity density functions.

By equating feedback flow from both the local cluster and remote cluster,

Figure 7.2: A typical node of cluster 1 in a network of two homogeneous clusters. Total input to the node consists of transferred tasks from cluster 1, transferred tasks from cluster 2, and input tasks to cluster 1.

we have for nodes in cluster 1:

$$\begin{aligned} \Phi_2 \lambda_2 + \alpha_1 \lambda_1 = & \gamma_1 P_{N1} P_{I1} P_{R1} [(1 + \alpha_1) \lambda_1 + \Phi_2 \lambda_2] + \\ & (1 - \gamma_2) P_{N2} P_{I2} P_{R2} [(1 + \alpha_2) \lambda_2 + \Phi_1 \lambda_1] \end{aligned} \quad (7.3)$$

An identical equation holds for typical nodes in cluster 2:

$$\begin{aligned} \Phi_1 \lambda_1 + \alpha_2 \lambda_2 = & \gamma_2 P_{N2} P_{I2} P_{R2} [(1 + \alpha_2) \lambda_2 + \Phi_1 \lambda_1] + \\ & (1 - \gamma_1) P_{N1} P_{I1} P_{R1} [(1 + \alpha_1) \lambda_1 + \Phi_2 \lambda_2] \end{aligned} \quad (7.4)$$

These two equations implicitly define Φ_1 and Φ_2 in terms of α_1 and α_2 . As in Chapter 6, an iterative procedure can be carried out to solve for Φ_1, Φ_2, α_1 and α_2 .

1. Set the fraction of feedback tasks Φ_1, Φ_2, α_1 and α_2 for both clusters to zero.
2. Solve for P_{Ii}, P_{Ri} and P_{Ni} in terms of the input traffic characteristics of λ_i for cluster i . Compute the fraction of feedback tasks α_i for each cluster.
3. Using Equations 7.3 and 7.4, compute the transferred fraction of tasks Φ_i that is transferred from one cluster to the next.
4. Adjust the input traffic to consist of local tasks and feedback tasks from both the local cluster and the remote cluster according to α_i and Φ_i for each cluster. Compute the new feedback fractions Φ_1, Φ_2, α_1 and α_2 for the revised input traffic. If the feedback fractions do not appreciably change, the steady state feedback has been reached and the iteration is terminated. Otherwise, the iteration repeats at Step 2 until convergence is achieved.

The total task loss in steady state is the task loss from both clusters, which is given by:

$$\begin{aligned} P(\text{rejection}) = & (1 - P_{N1}) P_{I1} P_{R1} [(1 + \alpha_1) \lambda_1 + \Phi_2 \lambda_2] + \\ & (1 - P_{N2}) P_{I2} P_{R2} [(1 + \alpha_2) \lambda_2 + \Phi_1 \lambda_1] \end{aligned} \quad (7.5)$$

This procedure can be generalized to N clusters in a straightforward manner. By equating flow at a typical node at each cluster, there are N equations

relating the α_i cluster feedback fractions to the Φ_i intercluster feedback coefficients. The iteration steps are identical to the above, except that the equations for input flow must now be solved for N clusters.

This solution can now be used to find optimal γ_i probabilities to minimize network task loss. For heterogeneous clusters, equal routing probability between clusters is not likely to achieve minimal task loss. In principle, solving for minimal task loss implies minimizing a non-linear objective function for $\gamma_1, \gamma_2, \dots, \gamma_N$. Using the above solution method, standard techniques can be used to search for the optimal routing strategy.

7.4 Summary

This chapter has presented some extensions to the approximation technique for estimating task loss at a local node and at a network. By using N th order interval estimation, the approximation to task loss at a single node may be improved so as to approach the exact solution as $N \rightarrow \infty$. However, as N increases, the number of possible acceptance or rejection events per arrival grows prohibitively large. As was shown by the analysis of FCFS pre-emptive scheduling, even double interval estimation can lead to a dramatic improvement in the estimation of task loss.

The second important extension outlined in this chapter is the relaxation of the requirement for a strictly homogeneous network. Clusters of nodes with different traffic and processing capability can also be solved, using the iterative technique outlined in Chapter 6.

Chapter 8

Conclusion and Summary

Design of soft real-time systems with hard real-time tasks and random arrivals demands that some measure of task loss performance be available to the designer. A common metric that is used for such systems is the average task loss due to missed deadlines. Prediction of task loss is exact for systems with entirely periodic tasks since the arrival of such tasks is regular and predictable. Task parameters are known at the outset, and a schedule can be created that either schedules all the tasks or some fraction of them.

The situation is much more complicated when task arrival patterns are not predictable. In such instances, a schedule must be drawn up "on the fly" as tasks arrive by some sort of run-time scheduling algorithm. This thesis has examined the situation where the task arrival pattern is entirely stochastic - a Poisson stream of arrivals with known task computation and laxity requirements. A major contribution of this thesis has been to detail a method to approximate task loss for the FCFS, SJF, LLF and EDD pre-emptive scheduling algorithms. Exact task loss prediction is available only for the FCFS algorithm, despite of the better task loss performance of the other algorithms. The approximation to task loss given in Chapter 3 simplified the analysis for light traffic by assuming that loss was dominated by rejection between pairs of tasks. In Chapter 7 this sort of analysis was termed *single interval estimation* since the arriving task was

assumed to interact only with the preceeding arrival. By relaxing this assumption and extending to *Nth interval estimation* (interaction with the proceeding *N* tasks), the solution to task loss and rejected task density functions approached the exact solution. Alternately, Chapter 4 derived an upper and lower bound for task loss at a single node for each of the pre-emptive scheduling algorithms that were sufficient in light traffic.

An equally important contribution of the thesis has been the development and evaluation of a network model, using the approximate results for single node local scheduling algorithms as a basis. The performance of a simple task sharing algorithm that randomly routed tasks from node to node on a distributed network was evaluated in Chapter 6. The initial network model of Chapter 6 required the nodes to be homogeneous and have identical traffic characteristics, Chapter 7 showed how the model could be extended to include clusters of homogeneous nodes in a heterogeneous network. While the task loss predictions diverged from simulated task loss performance as load increased, the model provided a means to evaluate relative performance of local scheduling algorithms and approached the true value of task loss for light traffic.

Many areas remain open to research as an outgrowth of this thesis. While Chapter 7 outlines how double interval estimation may be done, the analytic work remains to be completed for the SJF, LLF and EDD algorithms. The network model enhancements provide a springboard to more research in comparing analytic and simulated results, particularly in different task sharing algorithms. Finally, the analytic assumption of Poisson flow for feedback traffic needs to be investigated to see how many nodes are required in order to validate this assumption.

References

- [1] E. Jensen, C. Locke, and H. Tokuda, "A time driven scheduling model for real-time operating systems," in *Proceedings of the 6th IEEE Real-Time Systems Symposium*, December 1985.
- [2] J. D. Schœffler, "Distributed computer systems for industrial process control," *IEEE Trans. Computers*, vol. 17, pp. 11-18, February 1984.
- [3] A. Mok and M. Dertouzos, "Multiprocessor scheduling in a hard real-time environment," in *Proceedings of the 7th Texas Conference on Computer Systems*, November 1978.
- [4] L. Casey, "Decentralised scheduling," *The Australian Computing Journal*, vol. 13, pp. 58-63, May 1981.
- [5] J. M.J. Gonzales, "Deterministic processor scheduling," *Computing Surveys*, vol. 9, pp. 173-204, September 1977.
- [6] J. Moore, "An n job, one machine sequencing algorithm for minimizing the number of late jobs," *Management Science*, vol. 15, pp. 102-109, September 1968.
- [7] J. Sidney, "An extension of moore's due date algorithm," in *Symposium on the Theory of Scheduling and Its Applications*, (S. Elmaghraby, ed.), (New York), pp. 393-398, Springer-Verlag, 1973.
- [8] R. McNaughton, "Scheduling with deadlines and loss functions," *Management Science*, vol. 6, pp. 1-12, October 1959.

- [9] H. Kise, T. Ibaraki, and H. Mine, "A solvable case of the one-machine scheduling problem with ready and due times," *Operations Research*, vol. 26, pp. 121-126, January 1978.
- [10] M. Garey and D. Johnson, "Scheduling tasks with nonuniform deadlines on two processors," *Journal of the Association of Computing Machinery*, vol. 23, pp. 461-467, July 1976.
- [11] R. Karp, *Complexity of Computer Computations*, ch. Reducibility among Combinatorial Problems, pp. 85-104. Plenum Press, 1972.
- [12] C. Martel, "Preemptive scheduling with release times, deadlines and due times," *Journal of the Association of Computing Machinery*, vol. 29, pp. 812-829, July 1982.
- [13] P. Bratley, M. Florian, and P. Robillard, "Scheduling with earliest start and due date constraints," *Naval Research Logistics Quarterly*, vol. 18, pp. 511-519, December 1971.
- [14] O. Serlin, "Scheduling of time critical processes," in *1972 Spring Joint Computing Conference*, pp. 925-932, AFIPS Press, 1972.
- [15] C. Liu and J. Layland, "Scheduling algorithms for multi-programming in a hard-real-time environment," *JACM*, vol. 20, pp. 46-61, 1973.
- [16] M. Gonzalez and J. Soh, "Periodic job scheduling in a distributed processor system," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 12, pp. 530-535, September 1976.
- [17] L. Sha, J. Lehoczky, and R. Rajkumar, "Solutions for some practical problems in prioritized preemptive scheduling," in *Proceedings of the 7th IEEE Real-Time System Symposium*, December 1986.
- [18] J. Lehoczky, L. Sha, and J. Strosnider, "Enhanced aperiodic effectiveness in hard real-time systems," in *Proceedings of the 8th IEEE Real-Time System Symposium*, December 1987.

- [19] C. M. Woodside and D. W. Craig, "Local non-preemptive scheduling of hard real-time tasks," in *Proceedings of the 8th IEEE Real-Time System Symposium*, December 1987.
- [20] R. Smith, "The contract net protocol: high level communication and control in a distributed problem solver," *IEEE Trans. Computers*, vol. 29, pp. 1104-1113, December 1980.
- [21] J. Stankovic, "The analysis of a decentralized control algorithm for job scheduling utilizing bayesian decision theory," in *Proceedings of the 1981 International Conference on Parallel Processing*, pp. 333-340, 1981.
- [22] J. Stankovic, "Simulations of three adaptive, decentralized controlled, job scheduling algorithms," *Computer Networks*, vol. 8, pp. 199-217, 1984.
- [23] K. Ramamrithram and J. Stankovic, "Dynamic task scheduling in distributed hard real-time systems," *IEEE Software*, vol. 1, no. 3, pp. 65-75, 1984.
- [24] W. Zhao and K. Ramamrithram, "Distributed scheduling using bidding and focussed addressing," in *Proceedings of the 6th IEEE Real-Time System Symposium*, December 1985.
- [25] J. Stankovic, K. Ramamrithram, and S. Cheng, "Evaluation of a flexible task sharing algorithm for distributed hard real-time systems," *IEEE Trans. Computers*, vol. 34, no. 12, pp. 1130-1144, 1985.
- [26] W. Zhao, *A Heuristic Approach to Scheduling Hard Real-Time Tasks with Resource Requirements in Distributed Systems*. PhD thesis, University of Massachusetts, 1986.
- [27] W. Zhao and K. Ramamrithram, "Simple and integrated heuristic algorithms for scheduling tasks with time and resource constraints," *The Journal of Systems and Software*, vol. 7, pp. 195-205, 1987.
- [28] S. Cheng, J. Stankovic, and K. Ramamrithram, "Dynamic scheduling of groups of tasks with precedence constraints in distributed hard real-time sys-

- tems," in *Proceedings of the 7th IEEE Real-Time System Symposium*, December 1986.
- [29] K. Ramamithram and J. Stankovic, "Overview of the spring project," Tech. Rep. 87-54, University of Massachusetts, 1987.
 - [30] H. Chang and M. Livny, "Distributed scheduling under deadline constraints: a comparison of sender-initiated and receiver-initiated approaches," in *Proceedings of the 7th IEEE Real-Time System Symposium*, December 1986.
 - [31] J. F. Kurose, S. Singh, and R. Chipalkatti, "A study of quasi-dynamic load sharing in soft real-time distributed systems," in *Proceedings of the 7th IEEE Real-Time System Symposium*, December 1986.
 - [32] B. Gnedenko and L. Kovalenko, *Elements of Queueing Theory*. Israel Program for Scientific Research, 1968.
 - [33] F. Bacelli, P. Boyer, and G. Hebuterne, "Single server queues with impatient customers," *Adv. Appl. Prob.*, vol. 16, pp. 887-905, 1984.
 - [34] F. Bacelli and G. Hebuterne, "On queues with impatient customers," in *Performance 81*, North Holland Publishing Co., 1981.
 - [35] D. Craig and C. Woodside, "Light traffic performance characteristics of run-time preemptive scheduling algorithms for hard real-time tasks," Tech. Rep. SCE-88-4, Carleton University, March 1988.
 - [36] L. Kleinrock, *Queueing Systems Volume 1: Theory*. Wiley Interscience, 1975.
 - [37] D. L. Eager, E. D. Lazowska, and J. Zahorjan, "Adaptive load sharing in homogeneous distributed systems," *IEEE Trans. Software Eng.*, vol. SE-12, pp. 662-675, May 1986.

END

29/06/89

FIN