Stochastic Task Scheduling in Time-Critical Information Delivery Systems

Matthew Britton

B. Eng

Thesis submitted in fulfilment of the requirements for the degree of

Doctor of Philosophy

in

School of Electrical and Electronic Engineering

at

The University of Adelaide

Faculty of Engineering, Computer and Mathematical Sciences

January 2003

Contents

Abstract

In this thesis we present performance analyses of dynamic, stochastic task scheduling policies for a real-time communications system where tasks lose value as they are delayed in the system. The tasks in our communication system are file transfers such as messages, images and documents. Tasks are modelled by a Time-Value Function describing the value a task imparts to the system at a completion time. We propose a system framework with a task execution model whereby tasks may impart some value to the system if they are partially serviced.

We derive analytic expressions for the system value based on queueing-theoretic methods, and use numerical and simulation analysis to gain further insights into system performance. We examine simple task scheduling policies such as First-Come First-Served and Shortest-Job First, and several more complicated priority scheduling policies. We propose scheduling policy-switching criteria to improve the system performance for particular traffic statistics and system configurations. The techniques are applicable to any scheduling or queueing system where some commodity loses value or imposes a lateness penalty as it is delayed in the system. We also examine the effect other task management policies such as pre-emption, prioritisation, service and admission policies have on system performance.

To examine the performance of the system when sustained overloads occur we examine various methods to discard load and stabilise the system. Using our partial task completion model, we examine the effect partial task completion policies have on system performance. We also look at how task discard policies affect the system performance.

Statement

This thesis has been submitted to the Faculty of Engineering, Computer and Mathematical Sciences at the University of Adelaide for examination in respect of the Degree of Doctor of Philosophy.

This thesis contains no material which has been accepted for the award of any other degree or diploma in any university, and to the best of the author's knowledge and belief, the thesis contains no material previously published or written by another person, expect where due reference is made in the text of the thesis.

The author consents to this thesis being made available for photocopying and loan if accepted for the award of the Degree.

Matthew Britton

January 12, 2003

Acknowledgements

The Defence Science and Technology Organisation is gratefully acknowledged for allowing me time to study for this degree. I would like to thank my informal supervisor John Asenstorfer for his ongoing and significant support. I would especially like to thank my supervisors Cheng Chew Lim and Peter Taylor for the considerable amount of time spent guiding me. Also, I would like to add a special thanks to Jason Scholz for input in the early stages of the degree.

Glossary

AI	Artificial Intelligence
AT	Arrival Triggered pre-emption policy
CBS	Custom Broadcast Satellite
CVF	Completion Value Function
DBS	Direct Broadcast Satellite
DoD	Department of Defence (Aus)/Defense (US)
DSTO	Defence Science and Technology Organisation
EDD	Earliest Due Date scheduling policy
EDF	Earliest Deadline First scheduling policy
ER	Eventual Resume pre-emption policy
ERS	Eventual Re-Start pre-emption policy
ET	Environmental Triggered pre-emption policy
GNR	Guaranteed Next Resume pre-emption policy
GNRS	Guaranteed Next Re-Start pre-emption policy
HPF	Highest Precedence First scheduling policy, used by RATS
IRIS	Increased Reward with Increased Service task completion model
FCFS	First-Come First-Served scheduling policy
LCFS	Last-Come First-Served scheduling policy

LJF	Longest Job First task discard policy
LLF	Least Laxity First scheduling policy
LST	Laplace-Stieltjes Transform
LUF	Least Urgent First task discard policy
MLF	Minimal Laxity First, used by RATS
MOD	Mandatory/Optional Decomposition task model
MUF	Most Urgent First scheduling policy
MUSPF	Most Urgency/Service time Product First scheduling policy
MUSQF	Most Urgency/Service time Quotient First scheduling policy
MV	Multiple Version task completion model
MVF	Most-Valuable First scheduling policy
pdf	Probability Density Function
PDF	Probability Distribution Function
PIP	Primary Injection Point
RATS	ReAl-Time Scheduler, developed by DSTO
RMP	Rate-Monotonic Policy
ROS	Random Order of Service scheduling policy
SJF	Shortest Job First scheduling policy
STF	Shortest Task First scheduling policy used by RATS
TBS	Theatre Broadcast System
TIP	Tactical Injection Point
TPE	Task Processing Element
TVF	Time-Value Function

Notation and Nomenclature

a	Task urgency constant
B	Task urgency random variable
d	Task soft deadline constant
$E[V]$	Expectation of task value
$F(s)$	Laplace Transform (LST) of $F_Y(y)$, $F(s) = \int_{-\infty}^{\infty} e^{-st} F_Y(t)\, dt$
$F^*(s)$	Laplace-Stieltjes Transform (LST) of $F_Y(y)$, $F^*(s) = \int_{-\infty}^{\infty} e^{-st} dF_Y(t)$ [1]
$f_Y(y)$	Probability density function of Y
$F_Y(y)$	Probability distribution function of Y
$\Gamma(\cdot)$	Incomplete Gamma function, $\Gamma(a,z) = \int_z^{\infty} t^{a-1} e^{-t} dt$
λ	Task inter-arrival rate
μ	Task service rate
ρ	System utilisation
T	System (sojourn) time, sum of waiting time and service time
$V(\cdot)$	System value function
W	Waiting (queueing) time of a task in the queue

In general, we use upper-case characters to signify random variables, and use lower-case to signify constants

[1] For an introduction to Laplace-Stieltjes Transforms, see the text by Widder [85].

Chapter 1

Introduction

There are many computational systems that transfer some type of commodity whilst attempting to satisfy a range of strict time (and/or resource) constraints. These are called real-time or *time-critical* systems. There are a range of consequences for not satisfying the constraints that depend largely on the application. The commodity that is transferred may be physical, such as the transfer of goods through a manufacturing system, or it may be the transfer of a computational instruction in a computer bus or the transmission of a message in a communications system. When the commodity must utilise some limited *resource* (alternately referred to as processors or machines in the literature) by competing with other commodities there is usually some mechanism to ensure that the flow of commodities are handled in a manner that benefits the system's overall objectives. This mechanism is usually referred to as the *scheduler* and creates *schedules* which are precise sequences or timings of the transfer of commodities.

The timing or ordering of a schedule is determined by the *scheduling policy*—common system objectives are to minimise waiting time or cost. The commodities are usually referred to as *tasks*, processes, or jobs in the context of scheduling, depending on the application. We shall refer to our commodities as tasks throughout the thesis, unless we cite a reference where jobs or processes are the preferred terms. Performance measures, based on system objectives, are usually applied to

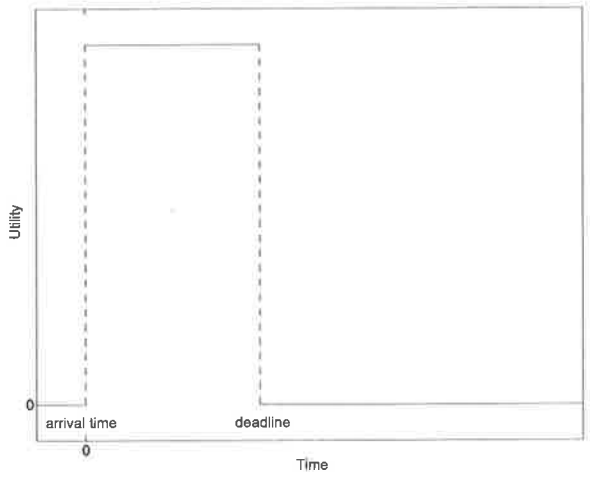

Figure 1.1: Time-Value Function with a Hard Deadline

the schedules that are created to allow performance analysis and comparisons of policies. Examples of performance measures are the number of missed deadlines and the computational complexity of the scheduling algorithm.

When the flow of tasks, or *traffic*, is deterministic, static scheduling is used. Static schedules, as the name suggests, are created once and do not require modification, and are often optimal in the sense that they optimise a performance metric such as mean waiting time (in this example a minimisation). When the traffic is non-deterministic dynamic scheduling may be required, whereby, at a minimum, tasks are serviced based on properties such as size or arrival time. We refer to these properties as task *meta-information*, because the tasks themselves will usually carry the primary information content. Other traffic or system conditions may be taken into account also. For example, changing traffic load may necessitate a change in the current scheduling policy.

Depending on the system objectives, meta-information requirements will vary accordingly. Perhaps the most common example of meta-information is a *hard deadline,* as shown in Figure 1.1, where we show the value in completing a task at any one time. In the hard deadline model, a task is worth maximum value up to an

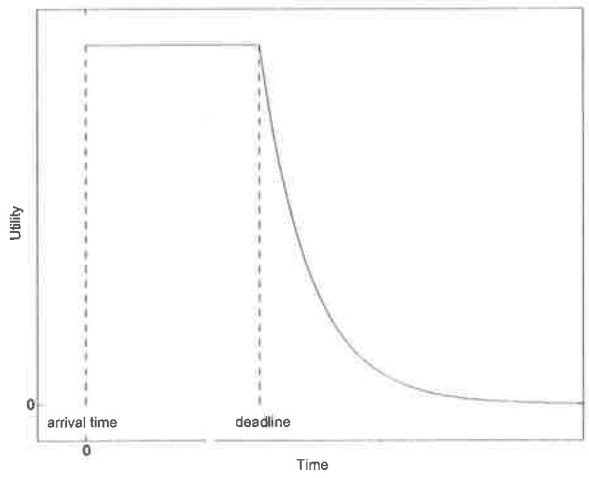

Figure 1.2: Time-Value Function with a Soft Deadline

instant in time, beyond which it has no value. Task deadlines usually map to a real deadline in the physical world, such as a point in time a manufacturing component needs to be delivered to satisfy an order, or when information becomes too old to be used, due to changing circumstances. A common utility to minimise with this approach is the number of missed deadlines.

While the hard deadline model provides some information that may be used to control the timing of the task execution, it does not model the value of the execution timing accurately enough for some applications. The natural extension of the hard deadline concept is the *soft deadline*, where a task has more than two possible values and the value over time may even be continuous. An example of a task execution model using soft deadlines is shown in Figure 1.2. The value of the task over time is called a *time-value function.* This is the model adopted in this thesis. Using the soft deadline model of task value, a natural utility is to maximise the expected task value. This is usually called *time-driven* or *best-effort scheduling* when heuristics are used. The shape of the time-value function is dependent on the application—a smooth transition at the deadline may be an alternative to the discontinuous deadline model shown in Figure 1.2

In this thesis, we work within a time-driven framework to focus on a particular application—a military broadcast satellite system, and many results are applicable widely to the time-driven scheduling community. In this satellite telecommunications system, tasks are discrete units of information such as files, images or messages. The broadcast channel, a radio-frequency resource, is the medium by which tasks traverse the system. The information is created for injection into the system by information *authors* and targeted for a particular *recipient population.* The information is broadcast over a large area to the recipient population. The recipients have quality-of-service requirements and hence define the soft deadlines, so by using one of the utilities given above we can attempt to satisfy the recipients' quality-of-service expectations.

For our system, there are various possible system configurations and assumptions which we explore in detail, such as resource structure, traffic load statistics, task composition and time-value function format. Furthermore, the resource may be divided into sub-channels, each of which may process a task in parallel, and we investigate the effect of this division on task value. As a task is an electronic unit of information, we may compose it in a progressive service format. This means we construct the tasks such that vital information is available separate to the optional portion of the task, and if the system traffic load is excessive we may service only the vital portions of tasks and discard the optional portions, thereby significantly reducing load whilst sacrificing only a small amount of task value. In this thesis, we investigate a number of methods for load mitigation based on partial task service.

The main contribution of this thesis is the maximisation of information value to end-users. We achieve this by attempting to maximise the task value for various system configurations and traffic properties. The results given are based on a number of analytic results, numerical results and simulations. This research is novel in the combined use of partial task service, heuristic scheduling policies and time-critical delivery. As the scheduling problem is (in general) intractable, we use heuristic methods of dynamically swapping between simple scheduling policies according to

system characteristics such as load, rather than attempting to find optimal scheduling policies. Results on optimal scheduling policies for simpler problems are used as a basis for the dynamic heuristics in many cases.

1.1 Contributions of the Research

The author has previously presented results contained in this thesis in a number of fora throughout the candidature. The results have been presented in a number of workshops, organised by the University of Adelaide, the Defence Science and Technology Organisation and the Australian Defence Materiel Organisation [9]. At the time of writing, two journal papers were ready for submission, based on many of the results obtained in the thesis. The first paper [10], in the subject of stochastic task scheduling, described much of the numerical and simulation results described in the thesis, and several initial analytical results. The second paper [8], which primarily focussed on aspects of queueing theory, presents an analysis of task waiting times in simple queueing systems when the Shortest Job First (SJF) queueing discipline (scheduling policy) is used.

The main contributions of this research are:

- Design of a real-time task execution model that gives the system flexibility in the management of the flow of tasks through the system, enabling graceful degradation in information value during periods of high system load and overloads.

- Derivation of analytic results for the time-value of tasks for various system model configurations (for example, serial and parallel scheduling), when the First-Come First-Served (FCFS) scheduling policy is used.

- Examination of the Last-Come First Served (LCFS) and SJF scheduling policy in a queueing-theoretic framework. The analysis includes approximation techniques for finding the Probability Distribution Function (PDF) of task

waiting times in a SJF queueing system, a step necessary in determining the time-value of tasks.

- Design of a stochastic discrete-event simulation program.
- Performance analyses of various system task management policies, including scheduling, task discard and partial-task completion policies.

1.2 Organisation of the Thesis

The thesis is organised as follows. Chapter 2 presents a literature survey of relevant scheduling, queueing and other related disciplines such as operations research and artificial intelligence. In particular, we examine the literature relating to time-value scheduling, as research in this area is closely related to our results.

Chapter 3 describes our problem domain, the assumptions we make and the model of our system. We show how the problem domain naturally leads to a unique research problem, with results that will result in significant performance improvements and are relatively simple to implement.

In Chapter 4 we discuss analytical work on system performance measures when the FCFS scheduling policy is used. As perhaps the most commonly-used scheduling policy, this provides a base-line for comparison with other policies in the thesis, and also as a common basis for comparison with other authors' work in this area.

Chapter 5 presents numerical work when other scheduling policies are used, and also presents simulation results, where we extend the performance analysis much further than is possible for either theoretical or numerical analysis. We introduce many alternative system configurations, scheduling policies and traffic models, and apply task value expectations as the performance measure as used in Chapter 5. The scheduling policies we examine here include LCFS, SJF, Random Order of Service, various heuristic policies that have been proven optimal for static scheduling results and policies examined by other authors such as "Best-Effort Scheduling" from the

work of Locke *et al* [56][41]. Following this, we compare these various scheduling policies with each other and FCFS.

In Chapter 6 we present work on overload scheduling, including task discard and partial task service policies. We compare the performance of each scheme using various assumptions such as traffic statistics and task structure.

Chapter 7 presents discussions on further issues and conclusions we have drawn from the analysis. In this chapter we also discuss the breadth of applicability of the research and define the necessary constraints and features a particular problem domain must have in order for our results to be applied. We conclude this chapter by addressing future research issues.

Chapter 2

Literature Review

This research borrows concepts and builds on results from many areas, including scheduling theory, queueing theory, management science, operations research, computer science and artificial intelligence. Each area of research listed above generally has a set of assumptions that does not allow us to describe and analyse our problem fully, as they are based on problem domains that have different implicit assumptions than ours. For each problem domain, there are often specific system objectives, requirements and performance measures.

Scheduling theory offers a large number of fundamental results that we build upon in this thesis, including many optimality results from static scheduling. Given a brief understanding of the main results in this field it is clear that the scheduling problem discussed in this thesis is intractable, in the sense that it is not possible to derive an optimum scheduling policy which maximises the sum of task value contributions over an infinite planning horizon. The problem is further compounded for the parallel service problem as discussed below. Scheduling theory differs from queueing theory in that it usually facilitates performance analyses using arbitrary traffic sets or static task sets, whereas queueing theory usually allows probabilistic performance measures to be gathered when simple scheduling policies are used. As our application falls somewhere between these disciplines, elements of both will be required to fully describe and analyse our problem.

We borrow concepts freely from management science and operations research for value-based scheduling, where schedules for the flow of commodities are often given real cost benefits or penalties. This concept is useful for our purposes in that we can assign a real benefit from the delivery of tasks in the telecommunications system, unlike the great majority of general scheduling or queueing theory research. By doing this we may define a suitable performance measure.

Computer science and artificial intelligence offer the concept of incomplete processing of tasks, jobs or instructions in order to reduce load and sacrifice quality for the overall benefit of system performance. To the best of the author's knowledge this has not previously been applied to telecommunications. Often in these fields of research, the result of a computation or task is available for a range of given accuracies or qualities, so there is often a logical case for compromising between speed and accuracy. An important part of this literature is the actual methodology for construction of such tasks in order that they may later be segmented and partially serviced. As tasks in our system may be constructed in the same manner, we use similar concepts in our research.

In general, scheduling theory offers performance analyses for a large range of policies, but is usually restricted to small, static task sets. Queueing theory, on the other hand, presents results for a restricted range of scheduling policies (usually First-Come First Served), but is powerful in that it enables probabilistic analyses which provides more insight into each policy than for scheduling theory. To the best of the author's knowledge, time-driven scheduling has not been discussed in queueing-theoretic terms before. We aim to use a hybrid scheduling-theoretic, queueing-theoretic approach in this thesis by examining a large range of policies in a time-driven framework, using probabilistic descriptions of the performance of each policy from the analysis.

2.1 General Scheduling Theory

There are many dimensions to the general scheduling theory literature. We will quickly list each of the distinctions here and expand the main results below. For a more detailed discussion of general scheduling theory, a useful short reference is [79]. One of the most important distinctions in a real-time system is whether the system is static or dynamic, as this greatly affects the chances of finding optimal schedules. For static scheduling, the whole task set is known: no tasks arrive during the execution of the schedule, and hence scheduling the task set is much more simple. This is sometimes referred to as *deterministic* scheduling. For dynamic scheduling, future task arrivals are not known in advance, hence this is sometimes known as *stochastic* scheduling.

Another major distinction in scheduling theory lies in the number of processors or resources used. Optimality results in the realm of multi-processor scheduling are relatively few in the literature.

Perhaps the next most important distinction in general scheduling theory is whether the system includes tasks that are *periodic*, *sporadic* or *aperiodic*. In the literature, periodic tasks are usually described as having deterministic (and constant) inter-arrival times, sporadic tasks repeat but have stochastic arrival times [15], and aperiodic tasks may be described with stochastic or deterministic arrival times.. A sporadic task is therefore usually associated with a physical action such as a radar sweep or a polling computer instruction. Aperiodic tasks, however, arrive once, and this arrival time may or may not be known in advance. The distinction between periodic and sporadic tasks is therefore simply future knowledge about task arrivals. As tasks in our system are discrete units of information which do not need to be sent more than once, we do not consider periodic or sporadic tasks.

Prioritisation is the next distinction we make within the general scheduling theory literature. In some systems, tasks have a fixed priority, while in others task priority may change during the schedule execution.

The final distinction we can make is the performance metric. This is usually based on the problem domain and determines the optimality conditions.

Other minor distinctions include whether the tasks have precedence or release time constraints and whether, in the multiple processor case, the processors are co-located or distributed. A number of distinctions in the literature are irrelevant to our work, including work on shared resource constraints.

Some authors define an overload as a schedule where deadlines are missed, while other authors define an overload in the traditional queueing terminology as processor utilisation overload. We will therefore distinguish between *deadline overload* and *processor overload* from this point forward.

There are a number of fundamental results from static (deterministic) scheduling that are relevant to our work, in that we may use these results as heuristics in our research. A solution to the single-processor problem of minimising the maximum lateness of a static set of non-pre-emptable tasks is given by Jackson's Rule [40], also known as the Earliest Due Date (EDD) algorithm. Where an optimal schedule is one that minimises maximum lateness, for a set of tasks with due dates and processing times the optimal schedule is created by ordering the tasks in non-decreasing due dates. Smith's Rule [78] states that for task weights w_i, deadlines C_i and processing times p_i, the schedule's maximum lateness "penalty" $\sum_i w_i C_i$ is minimised by scheduling the tasks in order of non-decreasing p_i/w_i. Static scheduling research has been conducted with similar assumptions about system performance metric to ours—that is, the system is penalised for each increasingly tardy task according to a time-value function. The major results in best-effort scheduling are examined in detail in Section 2.2.

There are few dynamic (stochastic) scheduling results in the literature. Note that when we use the term *dynamic*, we always refer to systems with no knowledge of future task arrivals. The Earliest-Deadline First (EDF)[1] policy is sometimes called

[1] We note that the concepts of the *due-date* in EDD and the *deadline* in EDF are synonymous, but are simply used in different contexts.

a dynamic policy, but it is dynamic only in the sense that task priority changes over time; not in the sense that there may be future task arrivals. Baruah *et al* [6] conducted important work on a methodology for the comparison of a scheduling policy with a "clairvoyant" scheduler (one that is aware of future arrivals) during deadline overloads. They called this measure the *competitive factor* and found upper bounds based on system configuration. These results showed that without knowledge of future arrivals, the value of computed schedules is limited to a theoretical upper bound. We may use this result to quantify each scheduling policy. The EDF policy is optimal under some dynamic traffic conditions [82].

As some tasks in our system may be periodic, it is necessary to examine the most important results from the literature on periodic task scheduling. A static scheduling system may have sporadic or aperiodic tasks arrivals, and the future arrival times of tasks are known in advance. Liu and Layland [54] proved that a system with a periodic task set, where tasks have a processing time p_i and a period T_i, the task set can be scheduled via the EDF algorithm if the following condition holds:

$$\sum_i p_i/T_i \leq 1. \tag{2.1}$$

The optimality of EDF scheduling with respect to minimising maximum lateness was shown by Dertouzos [26]. The Least-Laxity First (LLF) policy is similar to the EDF policy, except that tasks with the smallest $C_i - p_i$ (the difference between deadline and execution time) are scheduled first. LLF is also optimal in the sense of minimising maximum lateness—as proven by Mok [60]. Liu and Leyland also proved task schedulability by the Rate-Monotonic Algorithm (RMA), which assigns a static priority according to task period. When n tasks have deadlines and constant processing times, the task set is schedulable if $\sum_i^n p_i/T_i \leq n\left(2^{1/n} - 1\right)$ [5]. For a summary of results derived for RMA scheduling, see [74].

Computational complexity is a convenient method in classifying the difficulty of solving a computational problem. It is worthwhile examining various scheduling

problems from this perspective. This gives us a better understanding of the kinds of problems we may find solutions to. A problem is called *NP* if it can be solved in non-deterministic polynomial time. A problem is called *NP-hard* if, in solving the problem in polynomial time, would allow all NP problems to be solved in polynomial time. An example of an NP-hard problem is the travelling salesman problem. A problem which is both NP and NP-hard is called *NP-complete*. When the number of processors is greater than one, many scheduling problems become NP-hard [79]. For example, Ullman [83] proved that static scheduling of n tasks with an arbitrary ordering even with unit processing times is NP-complete, and Mok [60] proved that EDF scheduling was not optimal for the multiprocessor case. However, Coffman and Graham [19] proved that static scheduling on a two-processor system, where tasks have an arbitrary ordering, unit computation times and a deadline, could be solved in polynomial time. Garey and Johnson [35] proved that for the case where the tasks have arbitrary processing times, the problem is NP-complete. They also showed further that for processing times of 1 or 2 units, the problem is NP-complete. It seems that only a restricted range of problems with very specific constraints have polynomial-time solutions, and therefore have any significance. Hence, we must generally use heuristics.

Preemption generally increases the chance of polynomial-time schedulability for a system. However, scheduling a set of pre-emptable tasks on a number of processors whilst minimising the number of late tasks is NP-hard [50].

Tasks may have (i) static (fixed) priorities, as in RMA scheduling; (ii) *time-dependent* dynamic priorities, as in EDF and LLF scheduling or (iii) priority *classes*. A good summary of fixed-priority scheduling can be found in [3]. Examples of other time-dependent dynamic priorities include the Shortest-Job First (SJF) policy, where service or execution time is used to order the tasks. In general, when tasks have dynamic priorities or priority classes, optimality conditions are much more difficult to elicit. For systems with priority classes, each task is scheduled based on which class it belongs to. The value of scheduling tasks from different classes will

usually vary. In some scheduling systems, each class defines how the tasks belonging to it changes priority over time, so that each task has a time-dependent dynamic priority *and* a class, as in [49].

Performance measures are varied in the literature, and depend on the application domain. Results from job-shop scheduling are usually assumed optimal if the schedule that is created minimises maximum lateness or some other closely-related measure. A good summary of job-shop scheduling results is given in [20]. Some algorithms are deemed optimal if, when the schedule does fail to meet a deadline, no other scheduling algorithm can meet the deadline either. Processor utilisation is also used as a performance metric such as in EDF or LLF scheduling. More complex measures such as lateness penalties and time-value are used in scheduling in the presence of deadline overloads, and will be discussed in section 2.2 below, as this is particularly relevant to our research.

2.2 Time-Driven Scheduling

In hard real-time systems, tasks are classified as fully valuable if they arrive before a hard deadline and worthless if they arrive after that time. Soft real time systems classify the value of the task as somewhat reduced after the soft deadline. This is usually called *time-driven* or *best-effort* scheduling. The functions of task value over time are usually called time-value or time-penalty functions. The benefits or penalties can have arbitrary scales, but are most useful when they actually reflect a physical or tangible benefit or penalty, such as a cost or user-perceived quality. Only when this is done can performance measures truly reflect high-level system goals. Because of this, the performance measures that are found in best-effort scheduling are more relevant to this thesis than measures from other areas in the literature.

To the author's knowledge, the earliest known relevant work in this field was performed by McNaughton [58] in 1959. McNaughton's static task model assumed a set of tasks must be scheduled on one or more identical processors, with no task

arrivals after the schedule has begun. A task (i) with a service time a_i incurs a penalty to the system at a constant rate p_i if delayed past its deadline d_i. In general the tasks have different penalty rates. The objective was to minimise the total accumulated penalty until the last task was complete. For the one processor case, where all tasks complete beyond their deadline McNaughton proved that it is optimal to schedule the tasks in order of non-increasing p_i/a_i. We shall refer to this rule as McNaughton's Rule. The result also holds for parallel processing where all the service times a_i are equal and all $d_i = 0$. The rule performs well for parallel processing when these conditions are not met, however, suggesting it may be a useful heuristic for our purposes. Lawler [50] also presented some results for parallel scheduling.

A year after McNaughton published these results, Schild *et al* [72] generalised McNaughton's result to include the case where at least one of the tasks completes before its deadline. The rule described a three-step process where the tasks that complete before their deadline are swapped with other tasks in a systematic fashion via the "Criterion D" inequality. This rule is optimal for the case with one processor. We shall refer to this rule as the Schild's L Rule. A year after this result Schild *et al* produced another paper [73] where the problem was studied with quadratically-increasing and general penalty functions. An optimal scheduling rule was found under the condition (as in McNaughton's case) that all tasks complete after the expiry of their deadlines. To schedule n tasks, $n-1$ matrices are constructed. The first matrix contains every task pair permutation with its associated loss, that is, assuming this pair is serviced first. The task pair permutation with the higher loss is eliminated. The second matrix contains every permutation of the pairs not eliminated in the first matrix together with the next task to be serviced. The process of eliminating task order-of-service permutations with the higher loss is continued until the schedule with the minimum loss is determined. The rule has a computational complexity of $n\,(2^{n-1}-1)$, which is significantly less than a simple exhaustive search of all permutations (complexity $n!$) but is still computationally

infeasible for large n. We refer to this as Schild's Q Rule. In 1965 Fife [33] published a paper which proved the optimality of McNaughton's Rule for the case where tasks arrive via a Poisson arrival process after scheduling has begun, and where the deadlines for all arriving tasks occur at their time of arrival.

More recently, work has been performed on time-value functions, which in general are bounded functions that describe decreasing benefit or value to a system, as opposed to the approach described above which uses a system of unbounded increasing penalties. In 1985 Jensen *et al* [41] published a paper describing scheduling policies designed to maximise collective value over the long-term using time-value functions. In this paper a task (i) has a time-value function defined as $V_i(t) = K_1 + K_2 t - K_3 t^2 + K_4 \exp(-K_5 t)$, where K_j, $1 \leq j \leq 5$ are positive constants. A critical part of this work was the study of scheduling policies under over-load conditions—that is, not all tasks met their deadlines. Two new scheduling policies and six classical policies were evaluated. The first policy, BEValue1 (to denote *"best effort"*), schedules the task with the highest *value density* V/C, where C is the service time. Any algorithm with this capability is usually called a *value-density scheduling algorithm.* The second policy, BEValue2, begins to schedule tasks using a deadline-ordered sequence, and once the estimated probability of an deadline overload exceeds some threshold, the task with the lowest value density is removed from the schedule. Simulation results showed that: (a) for under-loaded systems the choice of policy was not a critical factor, and (b) that BEValue2 out-performed all other policies as the system became overloaded. Locke extended some of these ideas in his Ph.D thesis [56] and showed via simulations that the policy, which he called the BE policy, out-performed a number of classical policies. In all of this work, however, arbitrary schedules were created with a small number (36) of tasks, some of which were periodic. The tasks were created with stochastic properties (but arbitrarily assigned means) such as service times and inter-arrival times. Even though extensive simulations were run, the policies were analysed only for small data sets and the generality of these results is therefore of concern. Other authors

who have discussed value-density scheduling were Ronen *et al* [71], Morton *et al* [61] (operations research), and Nassehi *et al* [63] [62] (telecommunications).

Chen and Muhlethaler used a similar task model in their papers published in 1991 [13] and 1996 [14], where a static aperiodic task scheduling problem was studied. Their approach was to solve the static time value function scheduling problem on one processor via a decomposition of the task set. The task set is decomposed into several subsets, and within each subset the order is rearranged into an optimal sequence. An algorithm was described which optimally decomposes the set of tasks, and an $O(n^3)$ sub-optimal algorithm was described which sequences the tasks once they are decomposed. The authors use time-value functions with exponential, quadratic and linear shapes.

Moiin's PhD thesis [59] is perhaps the most directly relevant work for this thesis. Moiin used a combined approach of time-driven and partial service scheduling, and analysed heuristic scheduling policies in terms of utility of time-value and completeness/quality-value. We will defer a discussion of this work to Section 2.5.

Another related area of research is scheduling with earliness and lateness penalties. A good review of the work in this field is given in [4]. In some of this research, time-value functions are called "utility models" when applied to the expected utility of retrieving electronic information via resources such as the world-wide web [43].

2.3 Queueing Theory

We are interested in queueing theory in order to draw some more generalised conclusions from our work, which will not be available via scheduling theory. As stated above, scheduling theory offers examples of many interesting policies, but only provides contrived task sets for its performance analysis because of the static nature of most problems. Queueing theory allows us to present the work in a probabilistic framework, but is usually limited to a small number of scheduling policies. A common policy to examine is the First-Come First-Served (FCFS) scheduling policy.

The basic queueing system comprises a traffic process, a queue and a service element. Tasks (or customers) arrive according to the traffic process, are forced to wait in the queue and are serviced according to the demands placed on the service element. If we can describe each stage in the queueing system adequately, queueing theory provides results that will allow us to gain insight into the dynamic nature of such systems. The disadvantage with a queueing theory approach is that detailed analysis of queueing problems are intractable for all but relatively simple models.

Our analysis will be based on finding task value for a queueing system for various queueing disciplines, so we need to be able to analyse other queueing disciplines for comparison. In particular, we are interested in the waiting time of the task, and even the system time, which is the sum of the waiting time and processing time. A derivation of expressions for the waiting time probability distribution function (PDF) for the M/M/1 FCFS queue was first derived by Erlang in 1909 [30], and derived for the M/G/1 queue by Pollaczek [68] and Khintchine [45]. Lindley [53] is also cited in the literature when discussing waiting time distributions. The Last-Come First-Served (LCFS) M/M/1 queue was investigated by Vaulot [84], and Riordan [70] and Wishart [86] examined the M/G/1 LCFS case. Burke [11] derived the waiting time PDF using assumptions of constant service times, a Poisson arrival processes and a random order of service (ROS), that is, the M/D/1 ROS queue. Takács extended this work several years later by looking at general service time distributions and FCFS, LCFS and ROS [80], that is, M/G/1 queues. However, closed-form results were not obtained for the ROS queueing discipline. At around the same time, Kingman [46] and later Durr [28] examined the waiting time moments of the M/G/1 ROS queue. Shanthikumar [75] gave the first analysis of the Shortest-Job First (SJF) queueing discipline for the M/G/1 system, and the results were summarised by Takagi in [81]. We will use many of these authors' results in our research.

Queueing theory, as can be seen above, presents enough results on the waiting times of tasks for different queueing disciplines that we will be able to examine the

scheduling problem in a probabilistic framework.

2.4 Partial Task Service

Tasks may be tangible such as manufactured products traversing a conveyor belt on a factory floor, or they may be intangible such as instructions in a computer or messages in an electronic telecommunication system. However, all tasks have a *composition*—that is, some granularity or detail which is normally hidden to the medium or transferring bearer (and possibly the user). This granularity allows a decomposition of the task, possibly into several independent sub-tasks if the task management system allows this to be done. As the system in this case has the option of scheduling *whole* tasks or sub-tasks, another degree of freedom (and complexity) is added to the schedules. The reason a scheduling system may want to decompose tasks in this manner is to enable a load-reduction scheme whereby less important sub-tasks can be discarded or delayed. Granularity is manifested in many ways because there are many domains in task scheduling.

Systems that are capable of such performance are called *approximate processing*, *imprecise computation* or *design-to-time* systems. There are three methods for facilitating this system behaviour, as described below:

- Mandatory-optional decomposition (MOD) method—Tasks are decomposed into mandatory and optional parts, and optional parts can be ignored in order to reduce load and improve task throughput times. This method is usually referred to as imprecise computation.

- Increased-reward with increased service (IRIS) method—Tasks gain value in a continuous fashion as processing resources are applied, and graceful degradation in task quality may be forced to improve task throughput times. Algorithms in Artificial Intelligence (AI) which exhibit these characteristics are called *anytime algorithms.*

- Multiple version (MV) method—The system may switch between full and approximate versions of tasks. This method is usually called approximate processing or design-to-time scheduling.

Shih *et al* [76][77] looked at the MOD method in the context of static scheduling using deadlines. Chung *et al* [16] looked at the case of periodic task sets. Lim and Zhao [52] implemented a MOD method for dynamic systems that used a task number threshold to switch between full task and mandatory task execution. The system model was a single-server FCFS queue, and results were collected for under-loaded and over-loaded systems. Zhao *et al* [87] then extended this MOD method to reduce the effect of transient overloads in a single-server FCFS queueing system. Task arrival rates were modulated by a normal arrival rate and a higher arrival rate to model the effect of the transient overload. A review of early work in MOD methods is given in [55].

Dey *et al* [27] implemented the IRIS method for a dynamic system, and designed three scheduling algorithms for use in such a system, where task value was set to be proportional to task processing time. The first two policies implemented two-level policies—the first level allocated processing time to tasks, and the second ordered the tasks. The third algorithm was a simple greedy policy. An upper bound of performance for any IRIS policy was derived, and the low-level policy using EDF was shown to give a value close to this limit. The authors presented this work through simulations of the system with varying processor utilisations of 0.5000, 1.000, 2.000 and 4.000, and various other traffic parameters. As described below in Section 2.5, Moiin [59] has examined this area with the use of time-value functions and static schedules. Zilberstein [88] looked at IRIS methods in the field of AI.

The MV method is frequently used in the realm of AI research, where a range of results can be tolerated from an operation. This is not as relevant to our research as the IRIS method. For discussions on the MV method, see [51], [25], [38], [37] and [36].

2.5 Combined Time-Driven and Partial-Service Scheduling

Moiin presented perhaps the most relevant work to this thesis in his PhD thesis [59]. He presented work on single processor and multiple (distributed) processor scheduling algorithms for real-time systems, although only the single processor scheduling analysis is directly relevant. Static scheduling was assumed throughout the analysis. A task execution model was proposed whereby *task value* is defined by the instantaneous product of the time-value and the quality-value function, and our proposed task execution model presented in the following chapter is consistent with this model. The task quality is a function of the amount of processing utilisation the task has been assigned up the point of termination, and the time-value is a function that describes the value of the completion of the task at any point in time. The objective of this work was to maximise the task value associated with a schedule, as in the work of Jensen and Locke [41][42][56] and Chen and Muhlethaler [13][14], described above in Section 2.2. The types of functions presented for both time-value and quality-value functions are linear, quadratic and exponential. Moiin proved that selecting an optimal ordering of such tasks to maximise value is NP-complete by transforming the partition problem, then proposed a number of scheduling algorithms. The algorithms are decomposed into ordering and value optimisation, and each algorithm is characterised by the two-word label describing each stage, as follows:

- Optimal-Optimal—A computationally-intensive (factorial) algorithm that finds the optimal ordering and value-maximisation, and as such is infeasible for most applications.

- Heuristic-Optimal—An exponential algorithm which orders the tasks according to the EDF algorithm [54] and optimises the total value based on a dynamic programming technique.

- Heuristic-Heuristic—A linear algorithm which orders the tasks according to the EDF algorithm and finds the near-maximal total value by solving a system of linear equations in order to assign processing time for each task.

Moiin assessed value based on a set of 50 tasks with loads of 0.7000, 1.000 and 1.200 and comparisons to the Best-Effort, Imprecise and EDF algorithms. The results showed that the Heuristic-Heuristic algorithm has significant advantages over the traditional algorithms, but also significantly under-performs compared to the optimal-optimal and heuristic-optimal algorithms.

Moiin's thesis contains valuable task execution model concepts and scheduling algorithm analyses. The static scheduling algorithms that were presented may serve as useful dynamic algorithms in the future if they are computationally feasible. The work could be extended to dynamic scheduling in simulation quite easily by analysing the steady-state queueing problem. To the author's knowledge, no research exists which addresses time-value scheduling, partial-service and dynamic scheduling, and indeed this was the principal motivation for much of the research presented here.

2.6 Summary

The past work in the literature presented in this chapter shows that there are opportunities for novel research in the area of dynamic, stochastic soft real-time system task scheduling. Results in this area are usually given with the assumption of static task sets. There is also an opportunity to extend this work further to overload mitigation by partial task completion. Research in this area is often presented with dynamic system assumptions, but without soft-real time system assumptions such as the usage of time-value functions. Hence with the combined usage of dynamic traffic, time-value functions, partial task completion and best effort scheduling, a valuable contribution to the literature is possible.

Chapter 3

System Model and Problem Formulation

In this chapter we present a background to the current research, and show how the problem domain motivated the research through identification of current information management shortfalls. We show how this leads to the problem formulation. The system assumptions we must have in order to derive value from the research are then presented. We then design an information management system based on the unique problem characteristics, and present a suite of policies and algorithms to manage the expected system traffic.

3.1 Background

Our research was primarily motivated by a satellite communications application that has been investigated by the Australian Department of Defence (DoD), and more particularly, the Defence Science and Technology Organisation (DSTO).

A broadcast communications system of any type (that is, not only satellite) is distinct from, and is used to complement user-to-user services, and the users of each system have vastly different service expectations. In a user-user communications system, each individual message transmission may be unique, and will usually

have a limited applicability to other users in the field. There must therefore be a high degree of control available to customise each individual message transmission. This may happen dynamically through access to the service provider or the service provider may customise the service based on a static (but previously agreed) customisation. The users of a broadcast communications system, on the other hand, must have a significant common requirement for transmission content for their user requirements to be satisfied. This kind of access is made via a *back-channel*, which is a narrow-band connection in the opposite direction to the main broadcast, and may include dedicated or ad-hoc commercial systems. Often users do have a significant common requirement, and in these circumstances a broadcast system may suffice. If a broadcast system has back-channel access the resultant system conceptually lies somewhere between the extremes of tailored and common communications, with a significant number of users satisfied by a common transmission and a small number with unique requirements. The Direct Broadcast Satellite (DBS) is an example of a broadcast service with back-channel access. It is a commercial system and was originally used for broadcast of television into homes, with users able to control the content via a narrow-band back-channel [17]. DSTO has examined the use of commercial satellite broadcast services for military use since 1995, when the DBS service was gaining popularity. As a result, the Australian DoD is currently fielding a prototype application based on the DBS, known as the Theatre Broadcast System (TBS) [18]. The US DoD has also fielded a system known as the Global Broadcast System (GBS) [57]. We shall refer to all such systems as Custom Broadcast Satellite (CBS) systems, as they all provide back-channel access and have the same basic architecture.

The TBS configuration is shown in Figure 3.1. It consists of a collection of information resources from which is derived the desired transmission. This information is directed to the broadcast centre and transmitted from the Primary Injection Point (PIP) to the broadcast satellite. The satellite broadcasts the information to the users in the field. Users in the field may request information to be transmitted over

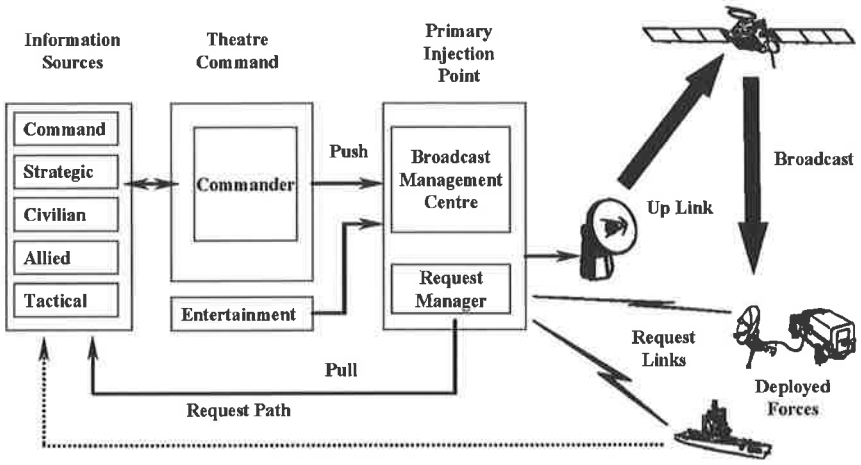

Figure 3.1: Australian Concept of the Theatre Broadcast System (TBS)

the broadcast channel by the narrow-band request links. These requests are received by the Request Manager (RM). The Tactical Injection Point (TIP) represents the injection of information into the system from the information authors. The information that is directed from the information sources to the PIP is controlled by the policies of the Commander.

The DSTO has conducted preliminary research into task scheduling for this application. The report by Blackmore [7] demonstrated how a dynamic programming technique could lead to effective time-value function task scheduling in such an environment. This research assumed that a number of logical channels carried the traffic, and the scheduling algorithm "ReAl Time Scheduling" (RATS) would allocate logical bandwidth to tasks in such a way as to maximise the sum of task value over the defined scheduling time interval. In this model, discrete-value tasks such as messages and files and continuous-value tasks such as video were included. The tasks were attributed with piece-wise linear time-value functions, and the intervals imposed on these functions define the scheduling intervals. In the algorithm, there are n_k $(k \geq 0)$ stages in the dynamic program corresponding to the number of tasks in the queue for each scheduling interval. The scheduling intervals are pe-

riodic and are set by the system administrator. This means that tasks may arrive during an interval but are not scheduled for service until the end of the interval and the subsequent re-evaluation of the task set. The decision taken during each stage of the dynamic program is the amount of bandwidth to allocate to the k^{th} task based on the remaining bandwidth. States at each stage correspond to the remaining bandwidth. Three simulation experiments were conducted for an under-loaded, fully loaded and overloaded system with system utilisations 0.5780, 1.156 and 2.688, respectively. The task arrival process was assumed to be Poisson, the service times were exponentially-distributed and three different task request sources were simulated. Four precedence levels were also simulated. Nine scheduling policies are compared—RATS, First-Come First Served (FCFS), Most-Valuable First (MVF), Shortest Task First (STF), Minimal Laxity First (MLF), Highest Precedence First (HPF), Dynamic HPF, EDF and Preemptive. The Preemptive policy uses priority information so that higher priority traffic can preempt lower priority traffic. The MVF policy is a greedy algorithm that finds the most valuable next-task contribution. For the under-loaded system, performance was similar, with RATS slightly out-performing the other policies. For the fully-loaded system RATS greatly out-performed most of the other policies. For the over-loaded system RATS again out-performed all other policies except STF. Note that RATS does not explicitly address the *ordering* of tasks and hence is not strictly a scheduling algorithm, rather it addresses the bandwidth-allocation problem.

Our research has been motivated by this previous research, and virtually all of the previous assumptions will be used in this thesis. This thesis builds on the previous work by expanding the number of policies, proposing novel policies and using partial-task scheduling to mitigate overloads. However, we limit our work to discrete task scheduling for simplicity, as the physical channel may be segmented into partitions for both discrete and continuous classes of traffic, and each partition may be examined separately. Our work will also concentrate on probabilistic measures of simpler scheduling policies, with simulation experiments used to verify analytic

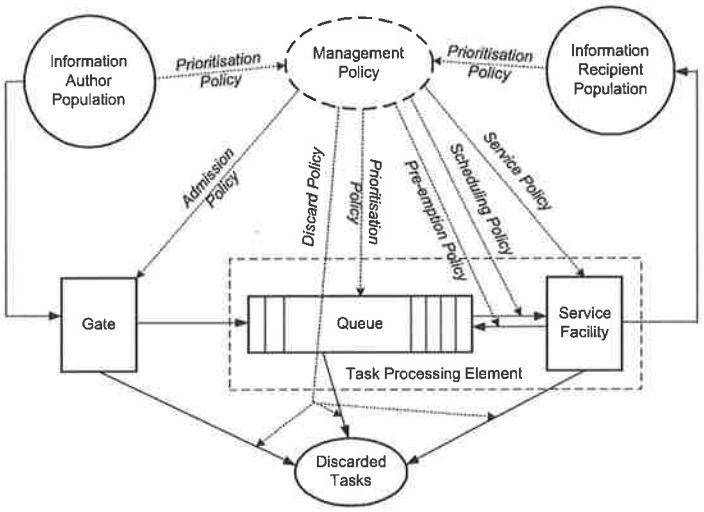

Figure 3.2: System Model

results and extend the analysis to other more complex policies.

3.2 System Model

To build the system model, we first partition the system conceptually into: (i) the information author population, (ii) tasks, (iii) task processing element (TPE), (iv) management policy, and (v) the information recipient population. As shown in Figure 3.2, the information author population produces tasks that are accepted into the TPE.

The management policy acts upon the TPE, which outputs tasks to be sent to the information recipient population. In the following sections we address each part of the system model in detail.

3.2.1 Information Author and Recipient Populations

The information author population includes the users in the field who request information via narrow-band request channels, and the users at the PIP who decide

what information to push to users (see Figure 3.1). We assume that there are r classes of information recipients, each with their own priority c_r.

Both groups of users produce tasks with an assumed value to the target recipients, which we call the user assigned priority, a. We assume that the value assigned by these users agrees with recipients' valuation. In practice this assumption will not strictly be true, however, we assume the resultant approximation is sufficient for our purposes. As prioritisation is performed by the *prioritisation policy* as part of the overall Management Policy suite we defer this discussion to Section 3.2.4 below.

For simplicity we initially assume that the inter-arrival process is Poisson and service times are exponentially-distributed. This assumption for the inter-arrival process is adequate, however in practice the service-time distribution is likely to be long-tailed such as a Pareto distribution [67][65][22] . When we simulate the system we extend analysis to include long-tailed distributions. This will be discussed further in Chapters 4 and 5. Many authors also examine long-range dependence in teletraffic data, which is a property where the current state of the system depends strongly on remote past events. However, we do not examine long-range dependence in this thesis.

3.2.2 Task Model

The task model includes the concepts of urgency, completeness and priority. Urgency is reflected in the time-value function. An example of a time-value is shown in Figure 3.3. The sooner the soft deadline occurs and the faster the function decays past the soft deadline, the more urgent the task is said to be. We then must define two types of urgency—deadline and TVF decay urgency. Hence, users who consider information is valuable initially, but worth little or no value within a short timeframe would regard a task to be urgent. Users who also consider that the value of the task decays soon would regard a task as urgent. It is the users' perception of the urgency of information that defines the time-value functions of the tasks. In the

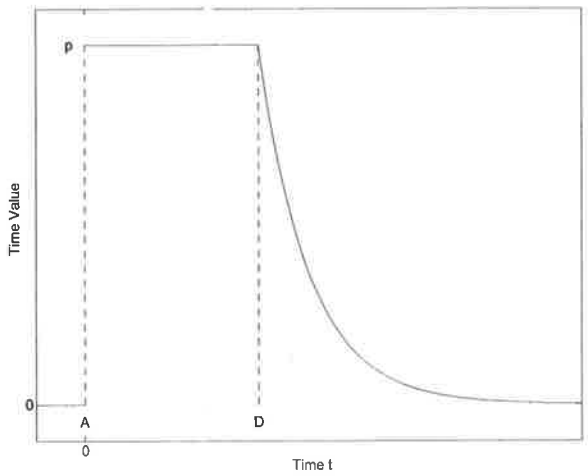

Figure 3.3: Time-Value Function

example shown in Figure 3.3, a task is admitted into the TPE at the arrival time A. From that time the task is worth the maximum value p, its priority, if serviced before the soft deadline which occurs at time D. However, the time-value function is assigned by a user, and a delay is present between the request and the moment the response (the task) arrives at the TPE. If the delay is large the deadline D may occur *before* the arrival time A, in which case the function will have decayed before service may begin. This example is shown in Figure 3.3. The shape of the decaying part of the time-value function is determined by the user, however we will use the common examples of linear, negative exponential and quadratic functions as approximations [59][56][41]. In Figure 3.3 we have shown use of the negative exponential time-value function. We index time t from the arrival time A. The time-value $T_n(t)$ for such a task n is then

$$T_n(t) = \begin{cases} p_n & if \quad A_n + t < D_n \\ p_n e^{-u_n(A_n+t-D_n)} & otherwise, \end{cases}$$

where u_n is the decay (urgency) parameter.

The task model also includes the concept of completeness, in order that we may discard traffic load by partially servicing tasks to increase task throughput. In order

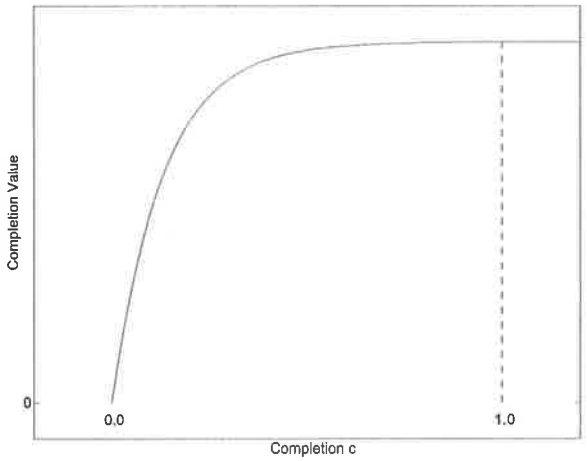

Figure 3.4: Completion-Value Function

for this to occur, we need to understand the relationship between the completeness and the users' perception of information value of the resulting incomplete task. This relationship is given by the completion-value function, an example of which is shown in Figure 3.4. We assume that the value density of each task is skewed towards the start of each task. That is, partially servicing the first part of each task will be proportionally more valuable than servicing the latter parts. Note that we index the function with *completeness* c (which has a maximum value of unity) and not time, as other tasks may be serviced in parallel, reducing the instantaneous proportion of processing time each task receives. When we index the function by time it is with the assumption that all the available processing time is allocated to it, and so does not reflect the expected completion time. For the n^{th} task, the completeness c_n is found by integrating the proportion of processor allocation $R_n(t)$ from its arrival in the system A_n to the time since arrival $A_n + t$, and dividing by the amount of time required to complete the task given the whole proportion of the processing power, B_n:

$$c_n(t) = \frac{1}{B_n} \int_{A_n}^{A_n+t} R_n(\tau)\, d\tau.$$

We then need to map the completion c_n onto the completion-value function $C_n(c)$

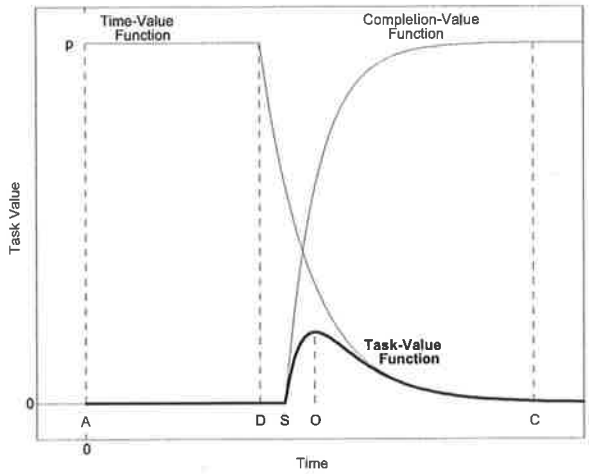

Figure 3.5: Task Value Function Where Service Begins After Soft Deadline

to derive the completion value $C_n(c_n(t))$. Note that this function is also a function of the current time t. The shape of the completion-value function will depend on the type of task being serviced.

Figure 3.5 shows how the time-value function and completion-value function determine the task value function, shown in bold. In this case, the deadline D occurs after the task arrival time A. The task begins service at S, just after D. As the processing begins, we see that the task value increases until the optimal departure time O, after which time the task value decays. Hence if the service policy assigns $(O - S)$ processing time to the task we will achieve the maximum achievable task value, given the late arrival time. Committing any more processing time will be counter-productive, so any conceivable service policy will have $(O - S)$ as a maximum processing time bound. Figure 3.5 assumes that the task receives all the processing utilisation and is not pre-empted or delayed during service. An example whereby service begins before the deadline D is shown in Figure 3.6. In this case the optimal departure time O is the same as the deadline D. In general,

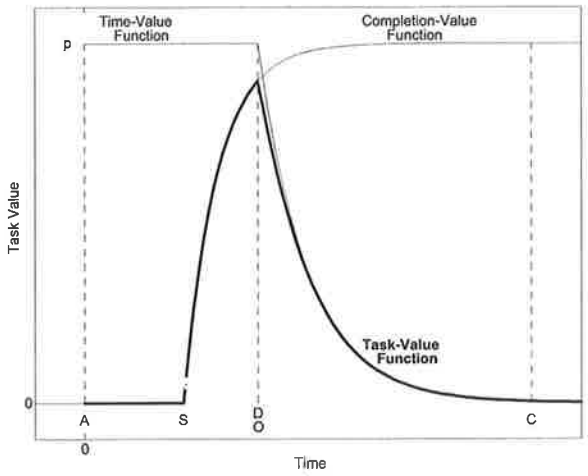

Figure 3.6: Task Value Function Where Service Begins Before Soft Deadline

then, the task value is thus expressed as

$$V_n(t) = \begin{cases} p_n C_n(c_n(t)) & if \quad A_n + t < D_n \\ p_n e^{-u_n(A_n+t-D_n)} C_n(c_n(t)) & otherwise, \end{cases} \tag{3.1}$$

where $c_n(t) = \frac{1}{B_n} \int_{A_n}^{A_n+t} R_n(\tau)\, d\tau$.

3.2.3 Processing Element Model

The task processing element (TPE) contains the queue (the buffer) and the service facility (the processor). It accepts input from (a) the management policy in the form of a suite of policies as discussed in Section 3.2.4, and (b) the information author population in the form of arriving tasks. The TPE outputs tasks to be sent to the intended recipient user population.

Queue

The queue in the TPE is serial and tasks are ordered according to arrival. Any tasks chosen for service are chosen according to a *scheduling policy*, so task position in the queue is not directly important. We also initially assume that the queue

capacity is infinite, that is, no tasks are ever discarded due to queue limitations. As discussed below, all task discards are managed by an explicit *discard policy*, which may operate using information on an assumed maximum queue capacity and will usually operate using other additional information.

Service Facility

When a task is accepted into service it is removed from the queue and arrives at the service facility. The service facility may contain one server or many, but the sum processing power remains constant at all times, as the bandwidth in our radio-frequency satellite communications channel remains constant. The *service policy*, as discussed below, dynamically controls the number of servers and the proportion of processor utilisation (bandwidth) each server may use at any one time.

3.2.4 Management Policy

The management policy is a suite of policies that are used to control the admission of tasks to the TPE, the ordering and service of those tasks and the subsequent reception of those tasks at the recipient population. For every policy we specify an identifier text string which uniquely identifies each policy. The first part of each identifier is a two-letter string used to identify the type of policy, for example **Ad** (admission) or **Sc** (scheduling). Following this is a hyphen and the string that uniquely identifies the specific policy, for example the specific scheduling policy such as First-Come First-Served is denoted **Sc-0**. A management policy is uniquely identified by a set of the policy identifiers. Below we examine this suite of policies in detail.

This notation is conceptually similar to Kendall's notation [44] for queueing systems $A/B/m/K/M$, which describes the inter-arrival (A) and service (B) processes, the number of servers (m), the system's storage capacity (K) and customer population M. By default, the system's storage capacity K and the customer population

M are infinite, and the corresponding labels are usually omitted in this case. An example of Kendall's notation is $M/M/1$, which denotes the queueing system with a simple Poisson arrival process, exponentially-distributed service times, and a single-server. Kendall's notation is insufficient to characterise our Management Policy as it has no labels to describe the scheduling policy (called the queueing discipline) or other policies. It also does not describe our information author and recipient population adequately. Some authors use Kendall's notation to describe scheduling policies, however it is not strictly correct notation to do so.

Admission Policy

Upon departure from the author population, a task must satisfy the *admission policy* that it should be admitted into the queue in the TPE. The admission policy determines this based on the state of the queue and the system load. Initially we assume that the (trivial) admission policy of allowing all tasks into the system is in effect, and that a discard or pre-emption policy discards the tasks if they are deemed not valuable.

Scheduling Policy

The choice of which task to remove from the queue and place in the service facility is made by the *scheduling policy.* Some of the simpler policies such as First-Come First-Served, Last-Come First-Served and Shortest Job First can be analysed theoretically, however other policies will have to be simulated for any comparisons to be made. Many scheduling policies are optimal under certain conditions, such as McNaughton's Rule for deadline scheduling with loss functions [58]. As our system model does not include the same assumptions, these policies will not be optimal, however it will be valuable to compare these policies as we intuitively expect them to out-perform simpler policies. A list of policies from **Sc-0** to **Sc-9** that we shall be examining is given below:

Sc-0 *First-Come, First Served (FCFS)*—The task that arrived first (the earliest) is served next.

Sc-1 *Last-Come, First Served (LCFS)*—The task that arrived last (the latest) is served next.

Sc-2 *Shortest Job First (SJF)*—The task with the shortest service time is served next.

Sc-3 *Most Urgent First (MUF)*—Urgency here is defined by the decaying part of the time-value function (TVF) rather than the deadline. For the linear TVF case, the urgency is assumed to be the gradient. For the negative exponential and quadratic TVF cases, the exponent parameter is used.

Sc-4 *Most Urgency/Service time Quotient First (MUSQF)*—Urgency is defined here as for **Sc-3** above. MUSQF is similar to McNaughton's Rule (see Section 2.2 and [58]) which has been proven to be optimal for a Poisson arrival process by Fife [33]. The difference is that we use the TVF to define urgency rather than the soft deadline. However, we could attempt to schedule using McNaughton's rules based on the soft deadline (for the linear and quadratic TVF cases) or a hard deadline (for the negative exponential TVF case).

Sc-5 *Most Urgency/Service time Product First (MUSPF)*—As for **Sc-4** above, except a product is used.

Sc-6 *Most Valuable First (MVF)*—The task that would provide the most value to the system if serviced next is chosen using this policy. This assumes the task will receive the total processing allocation and will not be pre-empted or delayed in any way.

Sc-7 *Random Order of Service (ROS)*—Tasks are chosen from the queue at random.

Prioritisation Policy

In our system model, the information author attaches an *intended audience value* function a_r to each task which describes the value of the task to each class of customer r. The scheduler must be aware of the number of users in each recipient class, which is described by the function n_r. The recipient value weighting function w_r maps recipient class onto importance. By using the priorities assigned by the information author together with the scheduler's knowledge about the recipient population, the scheduler may then assign the task priority p. We define the no priority case below, and propose a prioritisation policy:

Pt-0 Tasks shall all have an equal priority of unity.

Pt-1 Compose an expression for the non-normalised priority q_n of a task n by

$$q_n = \sum_{i \in r} w_i a_i n_i.$$

Note that we linearly weight the number of customers when determining the priority. We sum the un-normalised priority for each class of recipient before arriving at a combined un-normalised figure of priority. The scheduler then examines all of the tasks in the queue and normalises the priority of each one, as priority has an upper bound of unity. The normalised priority is then given by

$$p_n = \frac{q_n}{\sum_{i \in r} q_i}.$$

The values of p_n then determine the maximum value a task may impart to the system. The scheduling policy may use this priority information in its decision making.

Note that by using **Pt-1** each departure or arrival in the queue will change the normalised priority p of every task in the queue. Computationally this will be undesirable, however, the previously un-normalised priorities q may be used in the

calculation, saving computation time. Note that the intended audience population may be dynamic also. Environmental and operational conditions will lead to changes in the structure of the audience.

Discard Policy

When the system is overloaded, delays in information increase and discards may be considered. Tasks may be discarded or parts of tasks may be discarded. When parts of tasks are discarded, the service policy instead of the discard policy facilitates the discard mechanism. We examine service policies below. When a task is chosen to be discarded by the *discard policy*, any one of the scheduling policies listed above may be used.We examine discard policies (and partial task service policies) in detail in Chapter 6.

Ds-0 No discards use used.

Ds-1-m If the system utilisation $\rho > 1$, set a queue capacity threshold of m tasks. If the queue capacity threshold is exceeded, allow the task incoming into the queue, and discard one task using the Longest Job First (LJF) scheduling policy.

Ds-2-m *Least Urgent First (LUF)*—As for **Ds-1-m** above, except that the task with the lowest urgency is discarded.

Ds-3-m *Least Valuable First (LVF)*—As for **Ds-1-m** above, except that the task with the lowest value (as if it was served next) is discarded.

Pre-emption Policy

Pre-emption is the act of removing a task from active service and inserting a newly-arrived task that is deemed more valuable. We wish to generalise pre-emption to include cases where a trigger may not be the arrival of a new task, which we shall refer to as *Arrival Triggered (AT)* pre-emption. Other triggers are possible, such as

changes to environmental conditions which may affect the system load. We shall refer to this case as an *Environmentally Triggered (ET)* pre-emption. ET pre-emption is only useful under extreme changes in system load, so in general AT pre-emption is more applicable to our domain.

Another complication arises when one considers the possible rewards that are lost once pre-emption takes place. For example, a pre-empted task may have had a very small amount of processing time left which may have resulted in a large reward. It may be optimal in this case to let the task complete and then insert the new task, which is not strictly re-emption.

Pre-emptions are further distinguished by the behaviour of the pre-empted task once the pre-empting task completes—the *resume* policy. When the pre-empting task completes, the pre-empted task may be allowed to resume. This is referred to a *Guaranteed Next Resume (GNR)*. Alternately, the pre-empted task may be discarded. The third option is to place the task back in the queue. The task may either resume from where its execution was interrupted at the time of pre-emption, as in the case of the *Eventual Resume (ER)* policy, or the resource may be forced to begin servicing the task from the beginning again, as in the case of the *Eventual Re-Start (ERS)* policy. For our application, the ER pre-emption will generally be used, as the deployed user receive suites will be able to cache information and resume reception later. In our application, the *Guaranteed Next Re-Start (GNRS)* policy is also a possibility. Further complications arise with consideration of pre-empting pre-empted tasks. We shall ignore the case of multiple pre-emptions for the time being.

In summary, there are eight combinations that uniquely define a pre-emption policy. Either pre-emption is not used, or a pre-emption policy is uniquely defined by its trigger and resume sub-policies. We give the two sub-policies separate identifiers in the pre-emption policy identifier string, as shown below:

Pm-0 No pre-emption

Trigger policy:

Pm-0-r *Arrival Triggered (AT)*

Pm-1-r *Environmentally Triggered (ET)*

Resume policy:

Pm-t-0 *Guaranteed Next Resume (GNR)*

Pm-t-1 *Guaranteed Next Re-Start (GNRS)*

Pm-t-2 *Discard*

Pm-t-3 *Eventual Resume (ER)*

Pm-t-4 *Eventual Re-Start (ERS)*

In the list above, r and t are the Resume and Trigger policies respectively. Using the identifiers we have provided, examples of unique pre-emption policies are **Pm-0** (no pre-emption), **Pm-0-0** (arrival triggered with GNR resume policy) and **Pm-1-4** (environmentally triggered with ERS resume policy).

Service Policy

The service policy determines how much service each task receives once it enters a server. If the system includes multiple servers, it must determine what proportion of the processor utilisation each server has. The service policy must also determine how much processor time to give each task, and hence what quality that task will have. We therefore need one policy to describe the service policy **Sv** for a unique single-server system—a completion policy **C**. For a multiple server system we require two policies to determine a unique service policy **Sv**—the completion policy **C** plus a utilisation policy **U**. We use the following convention:

Given

-utilisation policy **U**

-completion policy **C**

A service policy is uniqely identified by the convention

C for single-server systems

U, C for mutliple-server systems

The first utilisation policy we propose is to provide each server an equal proportion of the total processing utilisation:

U-0-m Servers have equal utilisation

The m servers' proportion of the utilisation may be re-calculated depending on the task priorities, urgencies and service times, for example. The re-calculation of server utilisation proportions are triggered by a task arrival or departure form the service element, the queue or may be triggered by some other event. We provide details of a utilisation policy using task priorities below:

U-1-m For m servers servicing tasks n_m with priority $p(n_m)$, calculate the m^{th} server processor utilisation u_m by:

$$u_m = \frac{p(n_m)}{\sum_{j=\forall m} p(n_j)},$$

at task arrival and departure instants. For empty servers, assume the task priority is zero.

For the policies **U-0** above, when the queue is empty and one server is busy in a system with more than one server, all the processing power of the idle servers is lost until another task arrives or a task completes its execution. We now provide a policy that improves on **U-0** by reducing the number of servers when this situation occurs:

U-2-m Servers have equal utilisation. When the number of tasks in the system m is less than the number of servers, the service element consolidates its processing structure into m servers. This system was first studied in [31] and was also examined in [32].

The service policy also determines the amount of processing time each task receives, and hence the completion of each task. We propose several policies below:

C-0 All tasks complete fully.

C-1 If the system utilisation $\rho > 1$, set all task completion $c = 1/\rho$.

C-2-m-PS-PV If the system utilisation $\rho > 1$, set a queue capacity threshold of m tasks. If the queue capacity threshold is exceeded, tag incoming tasks for partial service. The service time of tagged tasks are reduced to $PS\%$ of the original service time, and the value the task imparts to the system is reduced to $PV\%$ of its original value.

3.3 Problem Formulation

Given a set of input parameters from the two populations, the Problem **P** is to find the suite of policies **M** that maximises the expected task value contributions L over an infinite planning horizon. Each task value is computed by the product of the time-value function and completion-value function at the moment of service termination. The set of input parameters consists of the probability distribution function (PDF) of the task inter-arrival times $A(t)$, the PDF of the task service times $B(t)$, the number of classes in the information recipient population k and the number of deployed users in each class d_r, $0 \leq r < k$:

Problem P

$$\text{Given the input parameters}$$

$$A(t),\ B(t),\ k \text{ and } d_r,\ 0 \leq r < k$$

$$\text{Maximise } L(n) = \lim_{n\to\infty} \frac{\sum_n V_n}{n}$$

$$\text{by finding the best policy set}$$

$$\mathbf{M}=(\mathbf{Ad}, \mathbf{Sc}, \mathbf{Pt}, \mathbf{Ds}, \mathbf{Pm}, \mathbf{Sv})$$

where V_n is the value of the n^{th} task calculated at the point of task termination, as shown above in Equation 3.1.

Given an analysis of the performance of the various policies, it is possible to implement a (sub-optimal) strategy for switching policies based on simple traffic statistics such as average task arrival rate. Each policy would outperform the other given well-defined bounds on the traffic statistics.

In Chapter 4 we look at the performance of the FCFS scheduling policy, that is, the management policy **M=(Ad, Sc-0, Pt, Ds, Pm, Sv)**. We alter the policies other than scheduling, such as increasing the number of servers, and observe the system performance. In Chapter 5 we analyse the performance of the other scheduling policies **Sc-1** to **Sc-09** and again look at combinations of the other policies such as task discard, service and prioritisation. We do this initially through numerical results and later through simulation. The results from these two chapters shows that the scheduling policy **Sc-2** (shortest job first) performs remarkably well under a wide range of traffic and policy conditions.

As most of the results in the following two chapters are general, they are of relevance to many scheduling problems of no particular problem domain. However, the results (such as presented late in Chapter 5) are particularly valuable to the problem domain of broadcast satellite scheduling, as we examine the influence of the information author and recipient populations, overload scheduling and the dynamic assignment of multiple servers.

Chapter 4

The First-Come First-Served Scheduling Policy

In this chapter, we examine the performance of the system when the First-Come First-Served (FCFS) scheduling policy is used. As perhaps the most commonly-used scheduling policy throughout many disciplines, this provides a basis for comparison with other authors' work in this area, and also as a base-line for comparison with other policies in the thesis. As our system model includes task information such as urgency and priority, which the FCFS policy ignores, we do not expect it to perform to the same level as the priority policies discussed in Chapter 5. However, if the improvement of priority policies over FCFS is marginal, then FCFS may be preferable as it has a low computational overhead. We defer this discussion to the end of the following chapter, once we have examined other policies.

In this chapter we derive analytic expressions for the performance of this Management Policy with the assumptions of no pre-emption or priority and a single server. We then examine the case where more than one server is used. The FCFS system is then simulated, even though we also derive analytic expressions. The analytical results are used to validate our simulations. This is important as in proceeding chapters the simulation tool has an important role. In the final section we present

conclusions about the performance of the Management Policy when FCFS is used. In following chapters we will compare the results with other scheduling policies such as Last-Come First-Served, Shortest Job First and other priority policies.

We are primarily concerned with the system value function as defined in Chapter 3, which we take as the sole measure of system performance. The system value function is defined as

$$V_n(t) = \begin{cases} p_n C_n(c_n(t)) & if \quad A_n + t < D_n \\ p_n e^{-u_n(A_n+t-D_n)} C_n(c_n(t)) & otherwise, \end{cases} \tag{4.1}$$

where $c_n(t) = \frac{1}{B_n} \int_{A_n}^{A_n+t} R_n(\tau)\, d\tau$.

The system value function shown above in Equation 4.1 is described in detail in Section 3.2.2, Chapter 3. Essentially, p_n is the priority of the n^{th} task, that is, the maximum contribution it can make to the system. A_n is the arrival time of the task, u_n is the decay (urgency) parameter of the time-value function (TVF), that is, how quickly the task loses any potential contribution to the system upon departure. $c_n(t)$ describes how the task will be processed—its share of the server over time and how complete the task will be before it departs the system. The value of a partial task is described by the completion value function (CVF). The system value is incremented each time a task departs the system, and the magnitude of this increment is defined by the product of the task's TVF at that instant, multiplied by the task's CVF. For individual tasks, the TVF is a simple function of system (sojourn) time, where system time is defined as the sum of the queueing (waiting) time and the service-time. Depending on our motivation, we could define other value functions which index waiting time only or includes task size in order to reward scheduling policies that favour large tasks more than small tasks. The objective is to analyse various queueing models by analysing task value statistics. We seek to maximise the sum of the task value contributions over an infinite planning horizon. To this end, we may use the expectation of a single task's value as the sole performance measure as this is identical to the long-term system value. Consequently, we use the terms "expectation of task value" and "system value" interchangeably in the thesis.

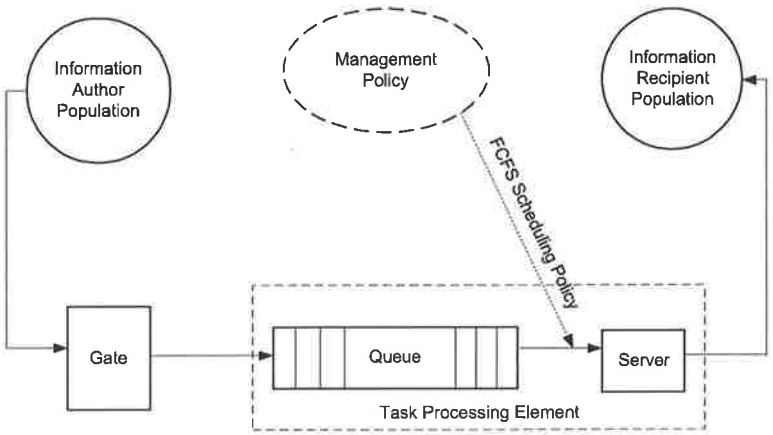

Figure 4.1: Simplified System Model Using Serial Server

4.1 Dynamic Serial Processing Systems

In the first analysis we look at the case where a single server is used. Figure 4.1 shows a simplified view of the system model when a single server is used. In Chapter 3 we presented the general problem formulation as Problem **P**. For the analysis in this section we decompose Problem **P** into Problem **P.1**. Later in the thesis the collection of decomposed problems will together give an insight into the general Problem **P**:

Problem P.1

Given the input parameters

$$A(t),\ B(t),\ k \text{ and } d_r,\ 0 \leq r < k$$

$$\text{Find } L(n) = \lim_{n\to\infty} \frac{\sum_n V_n}{n}$$

by using the policy set

M=(Ad-0, Sc-0, Pt-0, Ds-0, Pm-0, C-0)

In Problem **P.1**, $A(t)$ is the inter-arrival time pdf, $B(t)$ is the service-time pdf, k is the number of recipient classes and $d_r,\ 0 \leq r < k$ is number of users in each class.

V_n is the value of the n^{th} task and M is a string that uniquely defines a management policy. We have defined the notation for the management policy **M** in Chapter 3. In this case, **M** denotes the admission policy where we simply allow all tasks into the system, the FCFS scheduling policy, no preemption, no discard scheme and whole task completion.

4.1.1 Negative Exponential Time-Value Function

Initially we assume a negative exponential TVF, full task completion, Poisson arrival process and exponentially-distributed service times. The Management Policy is therefore uniquely specified by the string **M=(Ad-0, Sc-0, Pt-0, Ds-0, Pm-0, C-0)**. If we assume that every task has an equal urgency, task value is given by

$$\begin{aligned} V(T) &= T(T)\, C(c(T)) \\ &= T(T) \\ &= e^{-aT} \quad a, T \geq 0, \end{aligned} \tag{4.2}$$

where a is the task urgency constant and T is the system time random variable[1]. As stated previously, the performance measure we use is the expectation of task value $E[V]$. The expectation of a function of a random variable is found using the definition

$$E[V]_T = \int_{-\infty}^{\infty} V(t)\, f_T(t)\, dt, \tag{4.3}$$

where $f_T(t)$ is the probability density function (pdf) of the system time and the subscript refers to the fact that the task value function in Equation 4.2 indexes the system time T. For an introduction to the transformation of random variables, see the text by Papoulis [64]. For an M/M/1 system the pdf $f_T(t)$ is given [47] as

$$f_T(y) = \mu(1-\rho)\, e^{-\mu(1-\rho)y} \quad y \geq 0, \tag{4.4}$$

[1] Recalling from our notation that we use uppercase characters to signify a random variable and lower case to signify a constant.

where λ is the mean inter-arrival rate, μ is the mean service rate and ρ is the system utilisation λ/μ. We may set the lower bound of integration in Equation 4.3 to zero. Combining Equations 4.2, 4.3 and 4.4 above we have

$$\begin{aligned} E[V]_T &= \int_0^\infty e^{-at}\mu(1-\rho)e^{-\mu(1-\rho)t}dt \\ E[V]_T &= \frac{\mu(1-\rho)}{a+\mu(1-\rho)}. \end{aligned} \tag{4.5}$$

We now wish to change the objective function to use task waiting time W instead of system time. In the following chapter this derivation will allow us to make a useful comparison between the FCFS scheduling policy and Last-Come First-Served policy. The system value for this case is

$$V(W) = e^{-aW} \quad a, W \geq 0.$$

We know that the waiting time pdf for an M/M/1 queue is given [47] as

$$f_W(y) = (1-\rho)\delta(y) + \lambda(1-\rho)e^{-\mu(1-\rho)y} \quad y \geq 0,$$

where $\delta(w)$ is a unit impulse (Dirac delta function) located at $w = 0$. The expectation for task value for this case is then

$$\begin{aligned} E[V]_W &= \int_0^\infty e^{-aw}\left((1-\rho)\delta(w) + \lambda(1-\rho)e^{-\mu(1-\rho)w}\right)dw \\ E[V]_W &= (1-\rho) + \frac{\lambda(1-\rho)}{a+\mu(1-\rho)}, \end{aligned} \tag{4.6}$$

where we have used the unit impulse sifting property of $\int_{-\infty}^{\infty}\delta(a)f(x)dx = f(a)$.

If we now assume that the task urgency is itself a random variable we can build a more complex model to capture some of the information author and recipient structure. Let us assume that task urgency is negative exponentially-distributed, then

$$V(T) = e^{-BT},$$

where B is the urgency, which in this case is a negative-exponential random variable with the pdf

$$f_B(y) = b_0 e^{-b_0 y} \quad y \geq 0.$$

The expectation of the task value then becomes

$$\begin{aligned}
E[V]_{T,U} &= \int_0^\infty \int_0^\infty e^{-bt} f_{BT}(b,t)\, db\, dt \\
&= \int_0^\infty \int_0^\infty e^{-bt} f_B(b)\, f_T(t)\, db\, dt \\
&= \int_0^\infty \int_0^\infty e^{-bt} b_0 e^{-b_0 b} \mu(1-\rho)\, e^{-\mu(1-\rho)t} db\, dt \\
&= \mu(1-\rho)\, b_0 \int_0^\infty e^{-\mu(1-\rho)t} \left(\int_0^\infty e^{-(t+b_0)b} db \right) dt \\
&= \mu(1-\rho)\, b_0 \int_0^\infty \frac{e^{-\mu(1-\rho)t}}{t+b_0} dt \\
E[V]_{T,U} &= c e^c \Gamma(0,c), \qquad (4.7)
\end{aligned}$$

where $c = \mu(1-\rho)\, b_0$ and $\Gamma(a,z) = \int_z^\infty t^{a-1} e^{-t} dt$ is the Incomplete Gamma function. Various properties of the Incomplete Gamma Function are described in [2] and summarised in Appendix A. The subscript "U" refers to the random urgency of each task. It is useful at this stage to compare $E[V]_{T,U}$ with $E[V]_T$ in order to see the effect of random urgency on the system performance. We can compare the expectations if we set the mean urgency to a, that is, the same as the constant urgency value in the model with objective function as defined in Equation 4.2 above. We have

$$\begin{aligned}
\frac{E[V]_{T,U}}{E[V]_T} &= \frac{\mu(1-\rho)(1/a)\, e^{\mu(1-\rho)(1/a)} \Gamma(0, \mu(1-\rho)(1/a))}{\frac{\mu(1-\rho)}{a+\mu(1-\rho)}} \\
&= \frac{1}{a} e^{\mu(1-\rho)(1/a)} \Gamma(0, \mu(1-\rho)(1/a))\, (a + \mu(1-\rho)).
\end{aligned}$$

The results of this comparison are shown in Table 4.1 . We can see that the random urgency model gives a higher mean value only for a high utilisation and a low urgency, with the constant urgency model having a higher mean value that the random urgency model for the five other cases we have examined.

a	$\rho = 0.9000, \mu = 1.000$	$\rho = 0.5000, \mu = 1.000$	$\rho = 0.1000, \mu = 1.000$
1.000	1.129	0.7951	0.9169
10.00	0.4022	0.3694	0.6080

Table 4.1: The Ratio $E[V]_{T,U}/E[V]_T$ for Different Load and Urgency Parameters.

The next level of complexity arises when we consider soft deadlines. As the complexity of our analysis rises, the model becomes a better representation of the real system. Let us assume that each task has a random urgency and a constant soft deadline, then

$$V(T) = \begin{cases} 1 & 0 \leq T \leq d \\ e^{-B(T-d)} & T > d, \end{cases}$$

where d is the soft deadline and is constant and non-negative. Task urgency and system time have the pdfs:

$$\begin{aligned} f_B(y) &= b_0 e^{-b_0 y} \quad y \geq 0 \\ f_T(t) &= \mu(1-\rho) e^{-\mu(1-\rho)t} \quad t \geq 0. \end{aligned}$$

The expectation then is given by

$$E[V]_{T,U,C} = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} V(t) f_{BT}(y,t)\, dy\, dt,$$

where the subscript "C" refers to the constant soft deadline of each task. We split the integral over t into the regions prior to and following the soft deadline d so that

$$E[V]_{T,U,C} = \int_0^{\infty} \int_0^{d} V(t) f_B(y) f_T(t)\, dy\, dt + \int_0^{\infty} \int_d^{\infty} V(t) f_B(y) f_T(t)\, dy\, dt.$$

Making the necessary substitutions we obtain

$$\begin{aligned} E[V]_{T,U,C} &= \int_0^d \mu(1-\rho)\, e^{-\mu(1-\rho)t} \left\{ \int_0^\infty b_0 e^{-b_0 y} dy \right\} dt \\ &\quad + \int_d^\infty \mu(1-\rho)\, e^{-\mu(1-\rho)t} \left\{ \int_0^\infty e^{-y(t-d)} b_0 e^{-b_0 y} dy \right\} dt \\ &= \int_0^d \mu(1-\rho)\, e^{-\mu(1-\rho)t} ds \\ &\quad + b_0 \int_d^\infty \mu(1-\rho)\, e^{-\mu(1-\rho)t} \left\{ \int_0^\infty e^{-(t-d+b_0)y} dy \right\} dt \\ &= \int_0^d \mu(1-\rho)\, e^{-\mu(1-\rho)t} dt + b \int_d^\infty \frac{\mu(1-\rho)\, e^{-\mu(1-\rho)t}}{t-d+b_0} dt \\ E[V]_{T,U,C} &= 1 - e^{-b_0 cd}\left(1 + ce^c \Gamma(0,c)\right). \end{aligned} \tag{4.8}$$

Note that we have used the relation $\int_d^\infty \frac{e^{-ax}}{b+x} dx = e^{ab}\Gamma(0, a(b+d))$ in the last step and used $c = \mu(1-\rho)\, b_0$ as before. We note also that $E[V]_{T,U,C}\Big|_{d=0} = E[V]_{T,U}$ as we would expect.

4.1.2 Linearly-Decaying Time-Value Function Task Model

If we have a linearly-decreasing TVF we can express the system value as

$$V(T) = \begin{cases} 1 - aT & 0 \le T < \frac{1}{a} \\ 0 & \frac{1}{a} \le T. \end{cases} \tag{4.9}$$

If we assume that the urgency is a constant then the equivalent expression for the expectation of task value is

$$\begin{aligned} E[V]_{T,lin} &= \int_0^{\frac{1}{a}} V(t)\, f_T(t)\, dt \\ &= \int_0^{\frac{1}{a}} (1 - \omega t)\, \mu(1-\rho)\, e^{-\mu(1-\rho)t} dt \\ &= \int_0^{\frac{1}{a}} \mu(1-\rho)\, e^{-\mu(1-\rho)t} dt - a\mu(1-\rho) \int_0^{\frac{1}{a}} t e^{-\mu(1-\rho)t} dt \\ &= 1 + \frac{a\left(e^{-\frac{\mu}{a}(1-\rho)} - 1\right)}{\mu(1-\rho)}, \end{aligned} \tag{4.10}$$

where the subscript "lin" refers to the linear TVF.

4.1.3 Parabolic Decaying Time-Value Function Task Model

$$V(T) = \begin{cases} 1 - aT^2 & 0 \leq T \leq \frac{1}{\sqrt{a}} \\ 0 & T > \frac{1}{\sqrt{a}}. \end{cases}$$

Again, assuming urgency is a constant we can derive

$$\begin{aligned} E[V]_{T,pbl} &= \int_0^{\frac{1}{\sqrt{a}}} V(t) f_T(t) dt \\ &= \int_0^{\frac{1}{\sqrt{a}}} \left(1 - at^2\right) \mu (1-\rho) e^{-\mu(1-\rho)t} dt \\ E[V]_{T,pbl} &= \frac{2e^{\frac{1}{\sqrt{a}}\mu(-1+\rho)} \sqrt{a}\mu (1-\rho)}{\mu^2 (-1+\rho)^2} - \frac{2e^{-\frac{1}{\sqrt{a}}\mu(1-\rho)} \left(\sqrt{a}\mu\rho - a\right)}{\mu^2 (-1+\rho)^2} \\ &\quad + \frac{\mu^2 - 2\mu^2\rho + \mu^2\rho^2 - 2a}{\mu^2 (-1+\rho)^2}, \end{aligned} \tag{4.11}$$

where the subscript "pbl" refers to the parabolic-decaying TVF.

4.2 Summary

A summary of this section's results are shown in Table 4.2. We have examined and solved the Problem **P.1** using a number of assumptions such as a Poisson arrival process, exponentially distributed service times and various model complexities for the stochastic task properties of urgency, priority and deadlines. The Problem **P.1** is decomposed from the general Problem **P**. Given further analysis of other decomposed problems we will attempt to build a significant insight into the general Problem **P**.

4.3 Dynamic Parallel Processing Systems

As the scheduled resource is bandwidth on a communications system, we may segment the bandwidth any way we choose, for instance to service parallel tasks. When we service parallel tasks, each task's service time increases as each task must share the limited bandwidth resource. However, it is not clear what effect this

Objective function	**Expectation of task value**	**Eqn. No.**
$V(T) = e^{-aT}$	$\frac{\mu(1-\rho)}{a+\mu(1-\rho)}$	4.5
$V(W) = e^{-aW}$	$(1-\rho) + \frac{\lambda(1-\rho)}{a+\mu(1-\rho)}$	4.6
$V(T) = e^{-BT}$	$ce^{c}\Gamma(0,c)$ where $c = \mu(1-\rho)b_0$	4.7
$V(T) = \begin{cases} 1 & 0 \le t \le d \\ e^{-B(T-d)} & d < t \end{cases}$	$1 - e^{-b_0 cd}(1 + ce^{c}\Gamma(0,c))$ where $c = \mu(1-\rho)b_0$	4.8
$V(T) = \begin{cases} 1-aT & 0 \le T < \frac{1}{a} \\ 0 & \frac{1}{a} \le T \end{cases}$	$1 + \frac{a\left(e^{-\frac{\mu}{a}(1-\rho)} - 1\right)}{\mu(1-\rho)}$	4.10
$V(T) = \begin{cases} 1-aT^2 & 0 \le T \le \frac{1}{\sqrt{a}} \\ 0 & T > \frac{1}{\sqrt{a}} \end{cases}$	$\frac{2e^{\frac{1}{\sqrt{a}}\mu(-1+\rho)}\sqrt{a}\mu(1-\rho)}{\mu^2(-1+\rho)^2} - \frac{2e^{-\frac{1}{\sqrt{a}}\mu(1-\rho)}(\sqrt{a}\mu\rho - a)}{\mu^2(-1+\rho)^2} + \frac{\mu^2 - 2\mu^2\rho + \mu^2\rho^2 - 2a}{\mu^2(-1+\rho)^2}$	4.11

Table 4.2: Summary of Results for Single-Server System

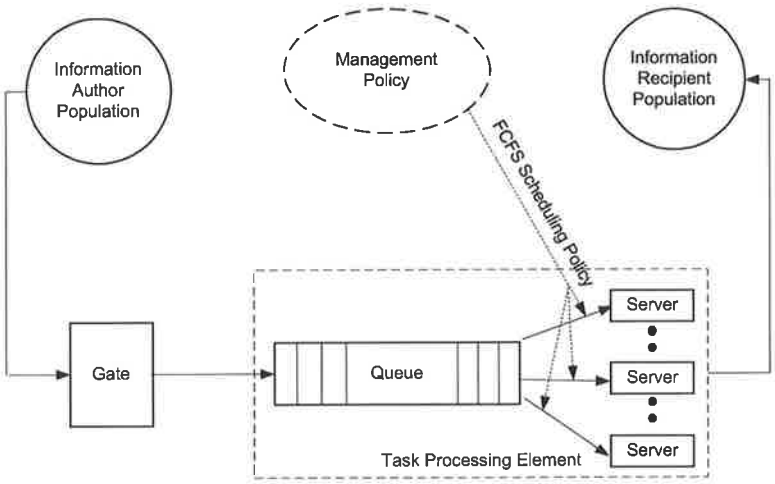

Figure 4.2: Simplified System Model Using Parallel Servers

would have on the system performance, as the tasks will wait (queue) for less time, counteracting the increased service time. Splitting the bandwidth may increase or reduce the system value. In Appendix C we prove that static, two-task serial scheduling out-performs parallel scheduling. In this section we examine the effect on system performance for the FCFS case. In proceeding chapters we will examine parallel processing when other scheduling policies are used. Initially, we look at the case where we have multiple servers. A simplified system model when parallel processing is used is shown in Figure 4.2. We decompose the Problem **P** into Problem **P.2** for the specific parallel processing problem:

Problem P.2

Given the input parameters

$$A(t),\ B(t),\ k \text{ and } d_r,\ 0 \leq r < k$$

$$\text{Find } L(n) = \lim_{n \to \infty} \frac{\sum_n V_n}{n}$$

by using the policy set

M=(Ad-0, Sc-0, Pt-0, Ds-0, Pm-0, (U-0-2, C-0)).

The parameters in Problem **P.2** have the same definition as for the serial-processing system in the previous section. Note that we have assumed that all tasks complete fully and the two servers have an equal and constant magnitude of half the total bandwidth resource. In this case the Management Policy **M** denotes a two-server and whole task service model by the label (**U-0-2, C-0**).

We now examine the two server M/M/2 system. As we need to compare the M/M/2 model with our single-server M/M/1 model, we need to halve the service rate on each of the two servers as we must conserve processing power (bandwidth). We begin with the derivation of the waiting time density. We keep the derivation in this section to the case of the negative exponential TVF. The waiting (queuing) time probability distribution function (PDF) is given by Kleinrock [47, p. 258] as

$$F_W(y)_2 = 1 - \tfrac{2\rho^2}{1+\rho} e^{-\mu(1-\rho)y} \quad y \geq 0,$$

where the subscript refers to the number of servers. In our case the mean service rate is $\mu/2$ as tasks take twice as long to process in each half-speed server in the two-server model. The waiting time pdf can be expressed as:

$$f_W(y)_2 = \tfrac{2\rho^2\mu(1-\rho)}{1+\rho} e^{-\mu(1-\rho)y} + \alpha\delta(y) \quad y \geq 0,$$

where $\alpha\delta(y)$ is the integration constant and $\delta(y)$ is the unit impulse (Dirac delta function) located at the $y = 0$. We have

$$\begin{aligned} \int_0^\infty f_W(w)_2\, dw &= \frac{2\rho^2}{1+\rho} + \alpha = 1 \\ \alpha &= 1 - \frac{2\rho^2}{1+\rho} \\ f_W(y)_2 &= \tfrac{2\rho^2\mu(1-\rho)}{1+\rho} e^{-\mu(1-\rho)y} + \left(1 - \tfrac{2\rho^2}{1+\rho}\right)\delta(y) \quad y \geq 0. \end{aligned} \tag{4.12}$$

We know that the mean waiting time in an M/M/2 queue is given by

$$W = \frac{2\rho^2}{\mu(1+\rho)(1-\rho)}. \tag{4.13}$$

We now check that Equation 4.12 satisfies this property:

$$\begin{aligned} W &= \int_0^\infty w f_W(w)_2 \, dw \\ &= \int_0^\infty w \left(\frac{2\rho^2\mu(1-\rho)}{1+\rho} e^{-\mu(1-\rho)w} + \left(1 - \frac{2\rho^2}{1+\rho}\right)\delta(w) \right) dw \\ &= \frac{2\rho^2\mu(1-\rho)}{1+\rho} \int_0^\infty w e^{-\mu(1-\rho)w} dw \\ &= \frac{2\rho^2\mu(1-\rho)}{1+\rho} \frac{1}{2\mu^2(1-\rho)^2} \\ &= \frac{2\rho^2}{\mu(1-\rho)(1+\rho)} \text{ as expected from Equation 4.13.} \end{aligned}$$

If we decide to use task waiting time W instead of system time in the objective function, then the task value is expressed as

$$V(W) = e^{-aW} \quad a, W \geq 0,$$

and the expectation is given by

$$\begin{aligned} E[V]_{W,2,\mu/2} &= \int_0^\infty e^{-aw} \left(\frac{2\rho^2\mu(1-\rho)}{1+\rho} e^{-\mu(1-\rho)w} + \left(1 - \frac{2\rho^2}{1+\rho}\right)\delta(w) \right) dw \\ &= \frac{2\rho^2\mu(1-\rho)}{1+\rho} \int_0^\infty e^{-(a+2\lambda(1-\rho))w} dw + \left(1 - \frac{2\rho^2}{1+\rho}\right) \int_0^\infty e^{-aw}\delta(w)\, dw \\ &= 1 + \frac{2\rho^2\mu(1-\rho)}{(1+\rho)(a+\mu(1-\rho))} - \frac{2\rho^2}{1+\rho}. \end{aligned} \tag{4.14}$$

We now wish to derive the system (sojourn) time pdf. As the system time pdf is a sum of the waiting time density and the service time density (requiring a convolution of pdfs in the time domain), we choose to multiply the Laplace Transform expressions in the s-domain of each in order to derive the Laplace Transform of the system time density. We derive the Laplace Transform of the waiting time density $W(s)$ first:

$$W(s) = \mathcal{L}\{f_W(w)_2\} = \frac{2\rho^2\mu(1-\rho)}{1+\rho} \left(\frac{1}{s+\mu(1-\rho)} \right) + \left(1 - \frac{2\rho^2}{1+\rho}\right).$$

The service time density is given by

$$f_H(y)_2 = \frac{\mu}{2} e^{-\frac{\mu}{2}y} \; y \geq 0,$$

and the corresponding Laplace Transform is

$$H(s) = \mathcal{L}\{f_H(h)_2\} = \frac{\mu/2}{s+\mu/2}.$$

Using the LT property of $S(s) = H(s)W(s)$, where $S(s)$ is the Laplace Transform of the system time density, we have

$$S(s) = \frac{2\rho^2\mu(1-\rho)}{1+\rho}\left(\frac{\mu/2}{s+\mu/2}\right)\left(\frac{1}{s+\mu(1-\rho)}\right) + \left(\frac{\mu/2}{s+\mu/2}\right)\left(1-\frac{2\rho^2}{1+\rho}\right). \tag{4.15}$$

Using a partial fraction decomposition the first term in Equation 4.15 reduces to

$$F(s) = \left(\frac{B_{11}}{s+\mu/2}\right) + \left(\frac{B_{21}}{s+\mu(1-\rho)}\right),$$

and so

$$F(s) = \left(\frac{1}{\mu(1-\rho)-\mu/2}\right)\left(\frac{1}{s+\mu/2}\right) - \left(\frac{1}{\mu(1-\rho)-\mu/2}\right)\left(\frac{1}{s+\mu(1-\rho)}\right)$$

and

$$\begin{aligned} S(s) &= \frac{2\rho^2\mu(1-\rho)}{(1+\rho)}\left(\frac{1}{2\rho-1}\right)\left(\frac{1}{s+\mu(1-\rho)} - \frac{1}{s+\mu/2}\right) \\ &\quad + \left(\frac{\mu/2}{s+\mu/2}\right)\left(1-\frac{2\rho^2}{1+\rho}\right). \end{aligned} \tag{4.16}$$

The inverse Laplace Transformation of Equation 4.16 is

$$f_T(t)_2 = \frac{2\rho^2\mu(1-\rho)}{(1+\rho)}\left(\frac{1}{2\rho-1}\right)\left(e^{-\mu(1-\rho)t} - e^{-\frac{\mu}{2}t}\right) + \mu\left(1-\frac{2\rho^2}{1+\rho}\right)e^{-\frac{\mu}{2}t}. \tag{4.17}$$

We know that the mean system time in an M/M/1 queue is given by

$$T = \frac{2}{\mu(1+\rho)(1-\rho)}. \tag{4.18}$$

Using Equation 4.17 above we check that we may derive the same mean system time from the pdf. The mean system time is given by

$$\begin{aligned}
T &= \int_0^\infty t\left(\frac{2\rho^2\mu(1-\rho)}{(1+\rho)}\left(\frac{1}{2\rho-1}\right)\left(e^{-\mu(1-\rho)t}-e^{-\mu t}\right)+\mu\left(1-\frac{2\rho^2}{1+\rho}\right)e^{-\frac{\mu}{2}t}\right)dt \\
&= \frac{2\rho^2\mu(1-\rho)}{1+\rho}\left(\frac{1}{2\rho-1}\right)\int_0^\infty t\left(e^{-\mu(1-\rho)t}-e^{-\frac{\mu}{2}t}\right)dt \\
&\quad +\left(1-\frac{2\rho^2}{1+\rho}\right)\int_0^\infty t\left(\frac{\mu}{2}\right)e^{-\frac{\mu}{2}t}dt \\
&= \frac{2\rho^2\mu(1-\rho)}{1+\rho}\left(\frac{1}{2\rho-1}\right)\left(\frac{1}{\mu^2(1-\rho)^2}-\frac{1}{\mu^2}\right)+\left(1-\frac{2\rho^2}{1+\rho}\right)\frac{2}{\mu} \\
&= \frac{2\rho^2}{\mu(1+\rho)(1-\rho)}\left(\frac{1-4(1-\rho)^2}{2\rho-1}\right)+\left(\frac{2(1+\rho)(1-\rho)}{\mu(1+\rho)(1-\rho)}-\frac{4\rho^2(1-\rho)}{\mu(1+\rho)(1-\rho)}\right) \\
&= \frac{\rho^2(3-2\rho)+(1+\rho)(1-\rho)-2\rho^2(1-\rho)}{\frac{\mu}{2}(1+\rho)(1-\rho)} \\
&= \frac{2}{\mu(1+\rho)(1-\rho)} \text{ as expected from Equation 4.18.}
\end{aligned}$$

If we assume the simplest model where urgency is a constant the expectation of the task value is

$$\begin{aligned}
E[V]_{T,2,\mu/2} &= \int_0^\infty V(t)\,f_T(t)\,dt \\
&= \int_0^\infty e^{-at}\left(\begin{array}{c}\frac{2\rho^2\mu(1-\rho)}{(1+\rho)}\left(\frac{1}{2\rho-1}\right)\left(e^{-\mu(1-\rho)t}-e^{-\mu t}\right) \\ +\mu\left(1-\frac{2\rho^2}{1+\rho}\right)e^{-\mu t}\end{array}\right)dt \\
&= \frac{2\rho^2\mu(1-\rho)}{1+\rho}\left(\frac{1}{2\rho-1}\right)\left(\int_0^\infty e^{-(a+\mu(1-\rho))t}dt-\int_0^\infty e^{-(a+\mu)t}dt\right) \\
&\quad +\mu\left(1-\frac{2\rho^2}{1+\rho}\right)\int_0^\infty e^{-\left(a+\frac{\mu}{2}\right)t}dt \\
&= \frac{2\rho^2\mu(1-\rho)}{1+\rho}\left(\frac{1}{2\rho-1}\right)\left(\frac{1}{a+\mu(1-\rho)}-\frac{1}{a+\mu}\right) \\
&\quad +\frac{\mu}{2}\left(1-\frac{2\rho^2}{1+\rho}\right)\frac{1}{a+\mu/2}, \qquad (4.19)
\end{aligned}$$

where the subscript "μ" again refers to the fact the service rate for each server is μ.

Objective function	**Expectation of task value**	**Eqn. No.**
$V(w) = e^{-aW}$	$1 + \dfrac{2\rho^2\mu(1-\rho)}{(1+\rho)(a+\mu(1-\rho))} - \dfrac{2\rho^2}{1+\rho}$	4.14
$V(t) = e^{-aT}$	$\dfrac{2\rho^2\mu(1-\rho)}{1+\rho}\left(\dfrac{1}{2\rho-1}\right)\left(\dfrac{1}{a+\mu(1-\rho)} - \dfrac{1}{a+\left(\frac{\mu}{2}\right)}\right) + \left(\dfrac{\mu}{2}\right)\left(1 - \dfrac{2\rho^2}{1+\rho}\right)\dfrac{1}{a+\left(\frac{\mu}{2}\right)}$	4.19

Table 4.3: Summary of Results for a 2-Server System, where $\rho = \lambda/\mu$ and the Service Rate $= \mu/2$ for Each Server

a	$\rho = 0.9000, \mu = 1.000$	$\rho = 0.5000, \mu = 1.000$	$\rho = 0.1000, \mu = 1.000$
1.000	0.8308	0.8333	0.9242
10.00	0.7507	0.6970	0.6970

Table 4.4: The Ratio $E[V]_{T,2,\mu/2}/E[V]_T$ for Different Loads and Urgency Parameters.

We can now compare the expectation for the random urgency single-server and two-server cases. We have

$$\begin{aligned}\frac{E[V]_{T,2,\mu/2}}{E[V]_T} &= \frac{a+\mu(1-\rho)}{\mu(1-\rho)}\frac{2\rho^2\mu(1-\rho)}{1+\rho}\left(\frac{1}{2\rho-1}\right)\left(\frac{1}{a+\mu(1-\rho)} - \frac{1}{a+\left(\frac{\mu}{2}\right)}\right) \\ &\quad \frac{a+\mu(1-\rho)}{\mu(1-\rho)}\left(\frac{\mu}{2}\right)\left(1 - \frac{2\rho^2}{1+\rho}\right)\frac{1}{a+\left(\frac{\mu}{2}\right)} \\ &= \frac{(1+\rho)\mu + (1+2\rho)a}{(1+\rho)(\mu+2a)}.\end{aligned}$$

The results of this comparison for various values of urgency and utilisation are shown in Table 4.4 . We observe that the single-server model always out-performs the two-server model—an observation we have also made for static, two-task scheduling with a general processor-sharing policy, as shown in Appendix C. A problem in the two-server system is the idle time in one server when there is only one task in the system, delaying this task unnecessarily. A solution to this problem is the adoption of a processor-sharing model, the idle-server processing power is added to

the collective processor power when few tasks are in the system. This would mean there is effectively one server when one task is in the system. The ratio of mean waiting time in a two-server system to the one-server system is (the ratio of the service times is of course $\frac{1}{2}$)

$$\begin{aligned} \frac{W_p}{W_s} &= \frac{\rho^2}{\mu(1+\rho)(1-\rho)} \frac{1-\rho}{\rho/\mu} \\ &= \frac{\rho}{1+\rho}. \end{aligned} \tag{4.20}$$

From Equation 4.20 we can see that, on average, tasks in a serial system wait less than for the two-server case.

We have solved Problem **P.2** using a number of assumptions such as a Poisson arrival process, exponentially-distributed service times, and various model complexities for the stochastic task properties of urgency, priority and deadlines. We have assumed full task completion and two servers at all times.

4.4 Simulation Results and Comparisons with Analytical Results

We now present simulation analyses of the systems examined in Sections 4.1 and 4.3. Details of the simulation are given in Appendix B. We present this section to verify that the simulation produces valid results. In the following chapter we will use the simulation for performance analysis as analytic solutions are not available in some instances.

Our simulation is a stochastic discrete-time simulation. It is stochastic in the sense that traffic properties are random, and it is discrete-event in that the system experiences "jumps" at discrete points in time.

As we wished to analyse the system in equilibrium (that is, under steady-state condition), we needed to address both the transient start-up problem and determine the number of samples to collect once the system had reached equilibrium.

Pawlikowski [66] gives an extensive summary of work related to determining when equilibrium is reached.

We determined when the system was in equilibrium by observing the stability of consecutive values of the mean queue length. We used $q(n)$, the mean queue length after the arrival of the n^{th} task, and smoothed this mean over a window of 100 samples to give

$$I(n) = \begin{cases} \sum_{i=0}^{n} q(i) / (n+1) & if \quad 0 \leq n < 99 \\ \sum_{i=n-99}^{n} q(i) /100 & if \quad n \geq 99. \end{cases}$$

We maintained a vector of 100 consecutive instances of the smoothed mean of the queue length $H = [I(n), I(n-1), \ldots, I(n-99)]$. When $\min(H) + \epsilon < \max(H)$, we assumed that the queue was in equilibrium, where ϵ is the tolerance which we initially set at 10^{-5}. The next decision we made was the number of samples to use once the system was in equilibrium. For this we decided to collect enough post-transient samples such that they were 95% of the total number of samples from start-up.

The first experiment was an analysis of the task model including a constant urgency and immediate soft deadline upon entry to the queue. The mean inter-arrival time was set to $1.000s$ and the mean urgency was set to 1.000. Table 4.5 shows how the theoretical task value (calculated from the expression $E[V]_T$ given in Equation 4.5 above) and simulated mean task value compared. Each simulation was replicated 12 times to give a more accurate estimate of the mean value and allow confidence interval statistics to be analysed. In the simulation tables, we have presented the expected value, the simulated value and the upper and lower 95% confidence intervals. We began by simulating the model with constant urgency and comparing the results with the expression $E[V]_T$, calculated from Equation 4.5 above. The results are shown in Table 4.5 . The table shows that the simulated mean is close to the expected value, and that the 95% confidence interval widths are narrow. The confidence intervals all contain the expected value. The second experiment we considered was a simulation of the system with random urgency and

Utilisation	$E[V]_T$	**Sim. Mean**	**95% Confidence Interval**
0.2000	0.8000	0.8000	0.8000 – 0.8000
0.3000	0.7000	0.7000	0.6999 – 0.7000
0.4000	0.6000	0.6000	0.6000 – 0.6001
0.5000	0.5000	0.4999	0.4998 – 0.5000
0.6000	0.4000	0.4001	0.4000 – 0.4002
0.7000	0.3000	0.2999	0.2998 – 0.3001
0.8000	0.2000	0.1999	0.1998 – 0.2001
0.9000	0.1000	0.09996	0.09972 – 0.1002
0.9500	0.05000	0.04989	0.04954 – 0.05025

Table 4.5: Comparison Between Expected System Value $E[V]_T$ and Simulated Task Value $\overline{V}_T$ for Serial-Server System

constant soft deadline. A comparison of the results with the expression $E[V]_{T,U,C}$ calculated from Equation 4.8 is given in Table 4.6. We used a soft deadline of $1.000s$ and otherwise the same parameters as the previous example . The results show that the simulation results are again accurate and have narrow confidence intervals.

We present results for the two-server, random urgency case in Table 4.7, where we have used the Expression $E[V]_{T,2,\mu/2}$ calculated from Equation 4.19 above . As we can see, the simulation and analytical results are relatively close. The analytical mean is within the 95% confidence intervals for all utilisations. As expected, the confidence intervals are widest at the high utilisations.

The final experiment we present, and one for which we have no analytical results, is that for the random urgency and random soft deadline model. The results are shown in Table 4.8 . We use this model to compare the FCFS scheduling policy to other policies in the proceeding chapters.

Utilisation	$E[V]_{T,U,C}$	**Sim. Mean**	**95% Confidence Interval**
0.2000	0.9968	0.9968	0.9968 – 0.9968
0.3000	0.9756	0.9756	0.9755 – 0.9756
0.4000	0.9269	0.9269	0.9269 – 0.9270
0.5000	0.8515	0.8515	0.8514 – 0.8516
0.6000	0.7522	0.7521	0.7520 – 0.7522
0.7000	0.6300	0.6300	0.6298 – 0.6302
0.8000	0.4823	0.4825	0.4822 – 0.4827
0.9000	0.2972	0.2973	0.2966 – 0.2981
0.9500	0.1786	0.1786	0.1777 – 0.1795

Table 4.6: Comparison Between Expected System Value $E[V]_{T,U,C}$ and Simulated Task Value $\overline{V}_{T,U,C}$ for Serial-Server System

Utilisation	$E[V]_{T,2,\mu/2}$	**Sim. Mean**	**95% Confidence Interval**
0.3000	0.5990	0.5991	0.5990 – 0.5991
0.4000	0.5048	0.5048	0.5047 – 0.5048
0.5000	0.4167	0.4167	0.4166 – 0.4167
0.6000	0.3318	0.3318	0.3318 – 0.3319
0.7000	0.2485	0.2486	0.2485 – 0.2487
0.8000	0.1658	0.1659	0.1657 – 0.1660
0.9000	0.08308	0.08327	0.08310 – 0.08345
0.9500	0.04160	0.04158	0.04124 – 0.04191

Table 4.7: Comparison Between Expected System Value $E[V]_{T,2,\mu/2}$ and Simulated Task Value $\overline{V}_{T,2}$ for Two-Server System

Utilisation	Simulation $\bar{V}_{T,U,D}$	95% Confidence Interval
0.5000	0.7982	0.7981 – 0.7982
0.6000	0.7104	0.7104 – 0.7105
0.7000	0.6023	0.6021 – 0.6025
0.8000	0.4682	0.4680 – 0.4684
0.9000	0.2930	0.2926 – 0.2934
0.9500	0.1775	0.1767 – 0.1782

Table 4.8: Simulated System Value $\overline{V}_{T,U,D}$ for Single-Server System, Random Urgency and Random Soft Deadline Model

4.5 Conclusions

In this chapter, we have derived analytical results for the value of tasks for various system model assumptions. We have verified the validity of the simulation of the system for various models for the case of FCFS, both serial and parallel scheduling, shown in Tables 4.2 and 4.3. By decomposing the general Problem **P** into two Problems **P.1** and **P.2** (which we have solved), we have started to build an insight into the general Problem **P** of finding an optimal policy **M** from a set of policies as proposed in Chapter 3.

We have also executed simulations of other models described in this chapter such as $\overline{V}_W$ and $\overline{V}_{T,U}$; they also have similar accuracies but are not presented here for the sake of brevity. The 95% confidence intervals from Tables 4.5, 4.6, 4.7 and 4.8 show that the simulation is sufficiently accurate and will be a reliable estimate of the expected value in Chapters 5 and 6 when analytical results are not available. The accuracy of the simulation is dependent on the processor utilisation, and we have considered utilisations to 0.9500 only. For lower utilisations, the simulation results have so little variance that the bounds on the confidence intervals are indistinguishable using four significant figures. We have not considered higher utilisations as

simulations must be executed for exceptionally long periods of time. We leave the analysis of systems with high utilisations to the consideration of overload processing in following chapters.

In Chapter 3 we presented a system framework use in the field of value-based scheduling, and in the present chapter we have derived analytical results of performance analysis by using this framework. The work presented in the thesis to this point builds upon other work in literature in static value-based scheduling by analysing the problem from a dynamic and stochastic point of view. Previously, researchers have used static, contrived data-sets to present results. We have generalised the problem by deriving analytical results. In the present chapter we have only examined the simple FCFS scheduling policy; in the following chapters we extend this analysis to more complex scheduling policies and we begin to examine system behaviour during processor overloads when load discard policies are used.

Chapter 5

Last-Come First-Served and Other Scheduling Policies

In Chapter 3 we presented the system model and stated the problem in formal language. In Chapter 4 we gained an insight into the system behaviour when the First-Come First-Served (FCFS) scheduling policy was used. We examined this for the serial- and parallel-server cases. As FCFS is the most common scheduling policy, its analysis gives us a base-line from which to compare other scheduling policies. As many authors examine FCFS in a similar context, it also allows us to compare the present work with other authors' work in similar areas.

In this chapter we present numerical and simulation results for scheduling policies other than FCFS. In particular, we present numerical results for the Last-Come First-Served (LCFS) and Shortest Job First (SJF) scheduling policies and simulation results for a number of other scheduling policies as listed in Section 3.2.4. It is important to look at policies that schedule based on service time, urgency and other task properties as they are likely to make more valuable decisions than simpler policies. This is because they discriminate between more and less valuable tasks than, for example, the FCFS which uses only arrival time to make its decision. In this chapter we compare the value of these more complex policies against each other

and the FCFS policy. Given the analysis of this group of scheduling policies, we will be in a position to solve the Problem **P**, which is to find the best policy M from a set of given policies, as shown in Chapter 3.

5.1 The Last-Come First-Served Scheduling Policy

In this section, we examine the waiting time probability density function (pdf) of the M/M/1 queue when the LCFS scheduling policy is used. We introduce numerical techniques, and begin to rely more heavily on simulation results, as some expressions are not suitable for deriving expectations of task value. Riordan [70] shows that the Laplace Stieltjes Transform of the waiting time probability distribution function (PDF) for an M/M/1 queue when LCFS scheduling is used is

$$F^*(s) = \frac{\lambda + \mu + s - \sqrt{(\lambda + \mu + s)^2 + 4\lambda\mu}}{2s}. \tag{5.1}$$

To find the inverse transform of the expression given on the right-hand side of Equation 5.1, we use Riordan's result from [70]. Riordan uses the inverse pair 556.1,

$$(p^2 + x^2)^{1/2} - p \Leftrightarrow \frac{x}{g} I_1(xg)$$

from [12, p. 59], where

$$I_1(z) = \sum_{m=0}^{\infty} \frac{(0.5z)^{1+2m}}{m!\Gamma(2+m)}$$

is the modified Bessel function of the first kind. The inverted expression is now

$$F_W(w)_{LCFS} = \int_0^w \frac{e^{-(\lambda+\mu)x} I_1\left(2x\sqrt{\lambda\mu}\right)}{x\sqrt{\rho}} dx,$$

and the corresponding pdf is

$$f_w(w)_{LCFS} = \frac{e^{-(1+\rho)\mu w} I_1\left(2\mu w\sqrt{\rho}\right)\sqrt{\rho}}{w} + \alpha\delta(w). \tag{5.2}$$

Here we have included an integration constant with a unit impulse (Dirac delta function) at the origin, which defines the probability of not waiting for service. That is, it is the probability of finding the queue empty and proceeding directly to the server (equal to $1-\rho$, the system utilisation).

As we have discussed in Chapter 4, we may decompose the Problem **P** into more specific problems, and aggregate later on to provide an insight into Problem **P**. The present problem is to examine the system performance by deriving the system value function when the LCFS scheduling policy is used:

Problem P.3

$$\text{Given the input parameters}$$

$$A(t),\ B(t),\ k \text{ and } d_r,\ 0 \leq r < k$$

$$\text{Find } L(n) = \lim_{n\to\infty} \frac{\sum_n V_n}{n}$$

$$\text{by using the policy set}$$

$$\textbf{M=(Ad-0, Sc-1, Pt-0, Ds-0, Pm-0, C-0)}$$

In Problem **P.3**, $A(t)$ is the inter-arrival time pdf, $B(t)$ is the service-time pdf, k is the number of recipient classes and $d_r,\ 0 \leq r < k$ is number of users in each class. V_n is the value of the n^{th} task and **M** is a string that uniquely defines a management policy. We have defined the notation for the management policy **M** in Chapter 3. In this case, **M** denotes the admission policy of simply allowing all tasks into the system, the LCFS scheduling policy **Sc-1**, no preemption, no discard scheme and whole task completion. For brevity we have restricted this analysis to the serial server case and assumed that all tasks complete fully. As we do not have an expression for the expectation of task value, we examine the expectation of task value for the model where the objective function uses task waiting time W instead of system time. We can then compare the waiting-time objective function model of LCFS with the FCFS model. As previously discussed, $L(n)$ is equivalent to the

Utilisation	Numerical $E[V]_{W,LCFS}$	Simulation $\overline{V}_{W,LCFS}$	95% Confidence Interval
0.6000	0.7190	0.7191	0.7190 – 0.7191
0.7000	0.6398	0.6398	0.6397 – 0.6399
0.8000	0.5566	0.5566	0.5565 – 0.5567
0.9000	0.4704	0.4703	0.4700 – 0.4705
0.9500	0.4265	0.4263	0.4260 – 0.4267

Table 5.1: Comparison Between System Value Calculated Numerically $E[V]_{W,LCFS}$ and the Simulated System Value $\overline{V}_{W,LCFS}$ for the Serial-Server LCFS System

expectation of task value. The value of an individual task is

$$V(W) = e^{-aW} \quad a, W \geq 0.$$

The expectation for task value for this case is then

$$\begin{aligned} E[V]_{W,LCFS} &= \int_0^\infty e^{-aw} f_W(w)\, dw \\ &= \int_0^\infty e^{-aw} \frac{e^{-(1+\rho)\mu w} I_1\left(2\mu w\sqrt{\rho}\right)\sqrt{\rho}}{w} dw. \end{aligned} \tag{5.3}$$

The numerical evaluation of the expression given in Equation 5.3 was performed by writing a collection of routines in the Matlab[1] programming language. The Matlab function BESSELI calculates $I_1(z)$, and the numerical integration function QUAD8 performs the integration necessary in Equation 5.3. The calculation of Equation 5.3 for various utilisations is shown in Table 5.1, together with simulation values .

As we can see from Table 5.1, the numerical means all fit within the 95% confidence intervals. The difference between the numerical and simulated mean is small enough that the numerical result is effectively verified, as we have previously verified the accuracy of our simulation in Chapter 4.

[1] Version 6.1.0.450 Release 12.1

a	$\lambda = 0.9000, \mu = 1.000$	$\lambda = 0.5000, \mu = 1.000$	$\lambda = 0.1000, \mu = 1.000$
1.000	2.476	1.057	1.000
5.000	2.040	1.079	1.001
10.00	1.667	1.059	1.001

Table 5.2: The Ratio $E[V]_{W,LCFS}/E[V]_{W,FCFS}$ for Different Loads and Urgency Parameters.

We have described our simulation procedure briefly in Chapter 4, Section 4.4 and in detail in Appendix B. We use 12 simulation runs to derive the 95% confidence intervals on the system value. To determine the number of samples to use, we observe the mean queue length at each instant a task arrived in the queue, and when 100 consecutive values are within a tolerance of 10^{-5} we assume the system is in equilibrium. We then begin to collect system value statistics, and terminate each simulation when pre-equilibrium task arrivals account for 5% of the total number of task arrivals.

From Equation 5.3 above we may compare $E[V]_{W,LCFS}$ with the expectation of task value when FCFS is used, $E[V]_{W,FCFS}$, using Equation 4.6. We show in Table 5.2 how the ratio $E[V]_{W,LCFS}/E[V]_{W,FCFS}$ changes for variations in the urgency constant a and the utilisation $\rho = \lambda/\mu$. As can be seen, the LCFS policy greatly out-performs FCFS at high utilisations, and this difference increases as mean urgency is reduced. At low utilisations the difference between the policies is below one percent in terms of task value.

The final simulation for LCFS we present is for the model with random urgency, random soft deadline and where the task value function is indexed by the system time. The results are shown in Table 5.3 . These results are used to compare with other scheduling policies in Section 5.3 below.

In this section, we have analytically solved Problem **P.3** only for the case where the system value objective function uses the task waiting time, and not the task system time. We have shown that the LCFS scheduling policy greatly out-performs

Utilisation	Simulation $\overline{V}_{T,U,D,LCFS}$	95% Confidence Interval
0.5000	0.8153	0.8153 – 0.8154
0.6000	0.7502	0.7502 – 0.7503
0.7000	0.6811	0.6811 – 0.6812
0.8000	0.6096	0.6095 – 0.6096
0.9000	0.5364	0.5362 – 0.5366
0.9500	0.4996	0.4994 – 0.4998

Table 5.3: Simulated System Value $\overline{V}_{T,U,D,LCFS}$ for the Serial-Server LCFS System

FCFS at high utilisations for this simple task execution model. We have therefore used simulation to give us insight into the remainder of Problem **P.3**. That is, to examine the problem when other objective functions are used, and random soft deadlines and urgency is used. We leave comparisons of other scheduling policies with FCFS and LCFS to the remainder of this chapter.

5.2 The Shortest Job First Scheduling Policy

In this section, we begin to use more sophisticated mathematical queueing-theoretic techniques to examine the Shortest Job First (SJF) scheduling policy. In particular, we derive analytic expressions for the waiting time pdf of the M/M/1 queue when the SJF scheduling policy is used. As in Section 5.1, we also use simulation to extend the analysis of the performance of the system. The analysis provided below gives a framework for the derivation of expressions of task value. We expect that the SJF scheduling policy will perform well, as there are examples in queueing theory where the SJF policy is optimal when waiting time cost penalties are used. See Kleinrock [48, Table 3.1] for a summary of results relating to scheduling policies that are dependant on service time.

For the M/G/1 queue, that is, with a general service time distribution, and with

the SJF scheduling policy, the Laplace-Stieltjes Transform of the waiting time PDF for a task with service time x is given by Takagi [81, Eqn. 2.64, p. 304][2] as:

$$W^*(s;x)_{STN} = \frac{1-\rho}{s}\phi + \frac{\lambda}{s}(1 - B(x))\left\{1 - B^-[\phi;x]\right\}, \tag{5.4}$$

where:

$$\begin{aligned}
\phi &= s + \lambda B(x)\left(1 - \Theta^+(s;x)\right) && (5.5)\\
\Theta^+(s;x) &= B^+[\phi;x] && (5.6)\\
B^+[s;x] &= \frac{\int_0^x e^{-st} dB(t)}{B(x)} && (5.7)\\
B^-[s;x] &= \frac{\int_x^\infty e^{-st} dB(t)}{1 - B(x)}, && (5.8)
\end{aligned}$$

In Equation 5.6 above, $\Theta^+(s;x)$ is Takac's expression for the Laplace-Stieltjes Transform of the PDF of the length of a busy period (see [75] for details), and $B(x)$ is the service time PDF.

We use Takagi's expressions to derive our expression for the waiting time PDF of a task in an M/M/1 queue using the SJF scheduling policy. In the following section we provide details on how to derive the pdf by numerical inversion of the Laplace-Stieltjes Transform given in Equation 5.4. In an M/M/1 queue we have exponentially-distributed service times, giving a service time PDF $B(x) = 1 - e^{-\mu x}$.

[2]Note that there is a typographical error in Takagi's text [81]. He gives the expression $W^*(s;x)_{STN} = \frac{1-\rho}{s}\phi + \frac{\lambda}{s}\{1 - B^-[\phi;x]\}$.

Under this condition, Equations 5.5, 5.6, 5.7 and 5.8 can be written respectively as

$$\phi = s + \lambda\left(1 - e^{-\mu x}\right)\left(1 - \Theta^{+}(s;x)\right) \tag{5.9}$$

$$\Theta^{+}(s;x) = B^{+}\left[s + \lambda\left(1 - e^{-\mu x}\right)\left(1 - \Theta^{+}(s;x)\right);x\right] \tag{5.10}$$

$$\begin{aligned} B^{+}[s;x] &= \frac{\int_0^x e^{-st}\mu e^{-\mu t}dt}{1 - e^{-\mu x}} \\ &= \frac{\mu\left(1 - e^{-(s+\mu)x}\right)}{(s+\mu)\left(1 - e^{-\mu x}\right)} \end{aligned} \tag{5.11}$$

$$\begin{aligned} B^{-}[s;x] &= \frac{\int_x^\infty e^{-st}\mu e^{-\mu t}dt}{e^{-\mu x}} \\ &= \frac{\mu e^{-sx}}{s+\mu}. \end{aligned} \tag{5.12}$$

Substituting Equation 5.10 into Equation 5.11 above yields

$$\begin{aligned} B^{+}[\phi;x] &= \frac{\mu\left(1 - e^{-\left(s+\lambda\left(1-e^{-\mu x}\right)\left(1-\Theta^{+}(s;x)\right)+\mu\right)x}\right)}{\left(s + \lambda\left(1 - e^{-\mu x}\right)\left(1 - \Theta^{+}(s;x)\right) + \mu\right)\left(1 - e^{-\mu x}\right)} \\ &= \Theta^{+}(s;x), \end{aligned} \tag{5.13}$$

so that by rearrangement we get

$$\Theta^{+}(s;x) = \frac{x(s+\mu) + Y(s) + \lambda x\left(1 - e^{-\mu x}\right)}{\lambda x\left(1 - e^{-\mu x}\right)}, \tag{5.14}$$

where $Y(s)$ is a solution to

$$-Z^2 + \left(\lambda x e^{-\mu x} - \lambda x - xs - \mu x\right)Z - \lambda\mu x^2 + e^{Z}\lambda\mu x^2 = 0. \tag{5.15}$$

Substituting Equation 5.14 into Equation 5.9 we have

$$\begin{aligned} \phi &= s + \lambda\left(1 - e^{-\mu x}\right)\left(1 - \frac{x(s+\mu) + Y(s) + \lambda x\left(1 - e^{-\mu x}\right)}{\lambda x\left(1 - e^{-\mu x}\right)}\right) \\ &= \frac{-x\mu - Y(s)}{x}. \end{aligned} \tag{5.16}$$

When we substitute Equation 5.16 into $B^{-}[\phi;x]$, then from Equation 5.12 we have

$$\begin{aligned} B^{-}[\phi;x] &= \frac{\mu e^{-\left(\frac{-x\mu - Y(s)}{x}\right)x}}{\frac{-x\mu - Y(s)}{x} + \mu} \\ &= -\frac{x\mu}{Y(s)}e^{x\mu + Y(s)}. \end{aligned} \tag{5.17}$$

Substituting Equations 5.17 and 5.16 into Equation 5.4 we have:

$$\begin{aligned} W^*(s;x)_{SJF} &= \frac{(1-\rho)}{s}\left(\frac{-x\mu - Y(s)}{x}\right) + \frac{\lambda}{s}e^{-\mu x}\left\{1 + \frac{x\mu}{Y(s)}e^{x\mu+Y(s)}\right\} \\ &= -\mu(1-\rho)\left(\frac{1}{s}\right) - (1-\rho)\left(\frac{Y(s)}{xs}\right) \\ &\quad +\lambda e^{-\mu x}\left(\frac{1}{s}\right) + \lambda x\mu\left(\frac{e^{Y(s)}}{sY(s)}\right). \end{aligned} \tag{5.18}$$

We are particularly interested in finding an expression for the pdf of the waiting time of a task in the queue, in order that we may derive an expression for the task value (other methods also exist that do not require finding a pdf). To derive this, we must first invert the expression given in Equation 5.18. However, this expression includes $Y(s)$, which is implicitly defined by Equation 5.15. If we assume that s is a real variable, the inversion will be relatively simple, however the accuracy of the inversion will not generally be acceptable. If we assume that s is complex, the result will be accurate, however the inversion will be difficult. Hence, finding the solution to Equation 5.15 will be difficult when we consider s as a complex variable. Another difficulty may arise when we must determine which solution is valid. We define a valid root as one which gives a valid task waiting time pdf. We assume that an invalid root will be identified by either creating a negative valued waiting time pdf or by violating the unity integration property of pdfs.

5.2.1 Laplace-Stieltjes Transform Inversion of $W^*(s;x)_{SJF}$

Given a LST of the waiting time PDF, we need to use numerical inversion techniques to derive the waiting time pdf. If we take the inverse LT of $W^*(s;x)_{SJF}$ we will get the pdf of the waiting time given a service time x, $f_W(w;x)_{SJF}$:

$$\begin{aligned} f_W(w;x)_{SJF} &= L^{-1}\{W^*(s;x)_{SJF}\} \\ &= -\mu(1-\rho)u(w) - (1-\rho)L^{-1}\left\{\frac{Y(s)}{xs}\right\} + \lambda e^{-\mu x}u(w) \\ &\quad +\lambda x\mu L^{-1}\left\{\frac{e^{Y(s)}}{sY(s)}\right\}, \end{aligned} \tag{5.19}$$

where $u(w)$ is the unit step function at $w = 0$.

As we need to evaluate $Y(s)$ numerically, so too we will need to evaluate the two inverse LTs in Equation 5.19 numerically. The final expression for the waiting time pdfs is given by

$$\begin{aligned} f_W(w)_{SJF} &= \int_0^\infty f_W(w;x)_{SJF} B(x)\, dx \\ &= \int_0^\infty (1-e^{-\mu x}) \left(\begin{array}{c} -\mu(1-\rho) - (1-\rho) L^{-1}\left\{\frac{Y(s)}{xs}\right\} \\ +\lambda e^{-\mu x} + \lambda x \mu L^{-1}\left\{\frac{e^{Y(s)}}{sY(s)}\right\} \end{array} \right) dx. \end{aligned} \quad (5.20)$$

In order to invert the LST of the waiting time PDF, we must first evaluate the roots of the polynomial-exponential given in Equation 5.15 which we shall refer to as $A(Z)$. We have

$$A(Z) = -Z^2 + \left(\lambda x e^{-\mu x} - \lambda x - xs - \mu x\right) Z - \lambda \mu x^2 + \lambda \mu x^2 e^Z,$$

and so

$$A'(Z) = -2Z + \left(\lambda x e^{-\mu x} - \lambda x - xs - \mu x\right) + \lambda \mu x^2 e^Z, \quad (5.21)$$

and

$$A''(Z) = -2 + \lambda \mu x^2 e^Z. \quad (5.22)$$

Numerical Inversion Using Real s

The waiting time pdf expression given in Equation 5.19 must be inverted in order to proceed any further towards our goal of deriving the expectation of task value for the various models. Once Equation 5.19 is numerically inverted we can then use numerical integration to derive $f_W(w)_{SJF}$ from Equation 5.20 and take the convolution with the service time pdf in the time domain to derive $f_T(t)_{SJF}$, the system time pdf.

From Equation 5.19 we can see that inversion considering s to be a complex variable will be difficult, as we need to find a root of $Y(s)$ that will be a complex

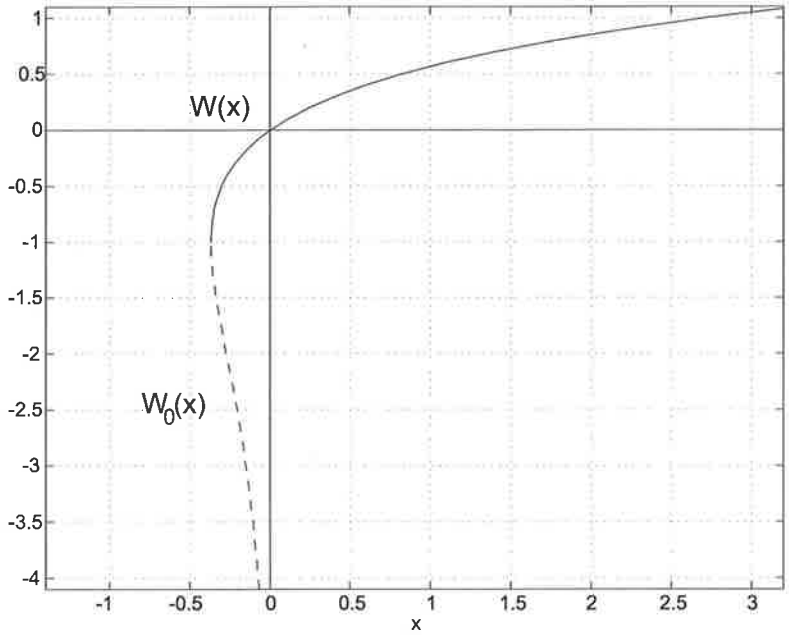

Figure 5.1: The Lambert W function

function. As a first step, we examine the utility of inversion using real values of s. First, however, we must understand the behaviour of the function $A(Z)$. Finding roots to this function gives us an expression for the LST of the length of the busy period PDF $\Theta^+(s;x)$, and therefore by Equation 5.20 an expression for the waiting time PDF itself .

Finding the Stationary Points of A(Z) If we can find the stationary points of $A(Z)$, we can understand how the solution to $A(Z) = 0$ behaves, and hence derive our task waiting time pdf. $A(Z)$ has a quadratic (with a negative squared coefficient) component and an exponential component, so in general, as $Z \to -\infty$ the quadratic component will dominate, leading $A(Z) \to -\infty$. As $Z \to \infty$, the exponential component will dominate leading $A(Z) \to \infty$. The behaviour of the function around zero is complicated, so later we analyse the number of roots and which of these are valid. We first note that $A(Z)$ always has a zero at $Z = 0$, we

also notice that for large (real) negative s, $A(Z) \approx -Z^2 - xsZ$ and therefore a zero exists at $Z \approx -xs$.

In the following discussion we show that $A(Z)$ has two stationary points. Denote the two stationary points of $A(Z)$ by $Z_{SL} \leq Z_{SR}$ (where the subscripts refer to left and right respectively). From Equation 5.21 the stationary points occur at

$$0 = de^Z - bZ + c,$$

where

$$c = \lambda x e^{-\mu x} - \lambda x - xs - \mu x,\ b = 2 \text{ and } d = \lambda \mu x^2,$$

so that we have

$$\begin{aligned} de^Z &= bZ - c \\ de^{Z+c/b} &= b\left(Z - c/b\right) e^{c/b} \\ -\frac{d}{b} e^{c/b} &= \left(\frac{c}{b} - Z\right) e^{(c/b - Z)}. \end{aligned} \tag{5.23}$$

We may express the solution to Equation 5.23 in terms of the Lambert W function [21]. We can say that

$$Z_{SL,SR} = -W\left(-\frac{de^{c/b}}{b}\right) + \frac{c}{b}, \tag{5.24}$$

where

$$c = \lambda x e^{-\mu x} - \lambda x - xs - \mu x,\ b = 2 \text{ and } d = \lambda \mu x^2. \tag{5.25}$$

Expressing our analysis in terms of the Lambert W function is useful as numeric routines for its calculation are freely available. The Lambert function has two branches, known as the principal or $W_p(.)$ branch (usually referred to simply as the $W(.)$ branch), and the $W_0(.)$ or secondary branch, as shown in Figure 5.1. This explains how the two stationary points in Equation 5.25 can be derived from the one function. Depending on the value of the argument, there may be one or two real solutions as observed on the figure. If the argument y satisfies $-1/e \leq y \leq 0$, there

are 2 real negative solutions, and if $y > 0$ there is one real positive solution. The function has complex solutions for $y \leq -1/e$. As we expect to find two stationary points, we also expect the argument y to satisfy $y \geq -1/e$. When we examine the argument of $W_p(.)$ in Equation 5.25, we can see that it will always remain negative (of course, always assuming that s is real). If we can prove it is also limited to $y \geq -1/e$ then we know there will always be two real solutions, and $A(Z)$ will always have two stationary points. The condition for 2 real solutions is therefore[3]

$$-\frac{1}{2}\lambda\mu x^2 e^{(\lambda x e^{-\mu x} - \lambda x - xs - \mu x)/2} \geq -1/e.$$

Solving for s we have

$$s \geq -\frac{-\lambda x + \lambda x e^{\mu x} + \mu x e^{\mu x} + 2e^{\mu x}\log\frac{2}{\lambda\mu x^2} - 2e^{\mu x}}{x e^{\mu x}}.$$

There are no possible values for any of the parameters which contradict the inequality. This means that for real s, $A(Z)$ will always have two real stationary points. As we can see from Equation 5.24, the behaviour of the stationary points is described by the Lambert W function. One stationary point must follow the principal branch of the Lambert W function whilst the other must follow the secondary branch. We can test which stationary point follows the principal branch as solutions on this branch satisfy $W_p(y) \geq -1$, whereas solutions on the secondary branch satisfy $W_0(y) \leq -1$. As the condition $Z_{SL} \leq Z_{SR}$ will always hold by definition and we know that $W_0(y) \leq W_p(y)$, then

$$\begin{aligned} -W_0(y) &\geq -W_p(y) \\ -W_0\left(-\frac{de^{c/b}}{b}\right) + \frac{c}{b} &\geq -W_p\left(-\frac{de^{c/b}}{b}\right) + \frac{c}{b} \end{aligned}$$

which we compare with $Z_{SR} \geq Z_{SL}$, proving that the stationary point Z_{SL} is given by the principal branch of the Lambert W function, whilst the stationary point Z_{SR} is given by the secondary branch. Thus

$$Z_{SL} = -W_p\left(-\frac{de^{c/b}}{b}\right) + \frac{c}{b} \tag{5.26}$$

[3]Note that this approach is equivalent to ensuring that $A'(\log(2/\lambda\mu x^2)) > 0$

and

$$Z_{SR} = -W_0\left(-\frac{de^{c/b}}{b}\right) + \frac{c}{b}. \tag{5.27}$$

As there are numerical routines available to evaluate the solution to the Lambert W function, this enables us to calculate the stationary points.

Finding the Root $Y(s)$ In this section we analyse how to find the roots of $Y(s)$ for real values of s, allowing us to derive numerical results for the expression for the waiting time pdf $f_W(w;x)_{SJF}$ given in Equation 5.19 above. $Y(s)$ is an expression that appears in our derivation of the waiting time busy period expression given in Equation 5.14. We must solve for $Y(s)$ (a solution to a negative exponential quadratic $A(Z)$) to derive the pdf of waiting time for that tasks and hence derive the system value.

Clearly, $A''(Z)$ (from Equation 5.22 above) always has one root, as it is a shifted exponential that must cross $A(Z) = 0$ at some point by the intermediate value theorem. The intermediate value states that if f is a function which is continuous on the closed interval $[a, b]$. Suppose that d is a real number between $f(a)$ and $f(b)$, then there exists c in $[a, b]$ such that $f(c) = d$. Hence $A(Z)$ always has one inflection point and $A'(Z)$ always has one stationary point. The inflection point is found at $Z = \log(2/\lambda\mu x^2)$. If $\lambda\mu x > 2$ the inflection point occurs at some negative Z, for $\lambda\mu x = 2$ it occurs at $Z = 0$, and if $\lambda\mu x < 2$ it occurs at some positive value for Z. $A'(Z)$ is clearly the sum of a linear component with a negative slope and an exponential component, with limits of ∞ as $Z \to -\infty$ and $Z \to \infty$. As such, depending on the parameters, it may has no zeros, a repeated zero if it just touches the line $A(Z) = 0$, or it may cross the line $A(Z) = 0$ and therefore have 2 zeros. We enumerate the cases in a systematic manner in Table 5.4 .

In the previous section, we determined that $A(Z)$ must always have two stationary points, and so case (i) will never occur. Using these cases we may now develop sub-cases to offer a further insight into the behaviour of $A(Z)$. For case (i),

Case	Condition	Observation
(i)	$A'(\log(2/\lambda\mu x^2)) > 0$	$A'(Z)$ never crosses Z-axis
(ii)	$A'(\log(2/\lambda\mu x^2)) = 0$	$A'(Z)$ touches Z-axis
(iii)	$A'(\log(2/\lambda\mu x^2)) < 0$	$A'(Z)$ crosses Z-axis

Table 5.4: The Three Cases of $Y(s)$

$A'(Z) > 0\ \forall Z$, so that $A(Z)$ will only have one root. This root may be located in three domains depending in the sign of $A(\log(2/\lambda\mu x))$. For case (ii), there are three sub-cases depending on the sign of $A(\log(2/\lambda\mu x))$. For case (iii), there may be 1 root, 1 repeated root or 3 roots, some of which may be repeated. We enumerate all the sub-cases in Table 5.5 .

As $A(Z)$ can have three roots, we wish to know when they will occur, and if possible discriminate between them to find the valid root, of which we assume there will only be one. As we have previously stated, we hope that an invalid root will be identified by creating a negative valued waiting time pdf or by violating the unity integration property of pdfs. An example of case (iii)(c) is shown is Figure 5.2 . We can see that one stationary point is negative, indicating the secondary branch condition, whilst the other crosses the vertical axis at some point, indicating the principal branch condition.

In order to see whether both cases (ii) and (iii) occur (we know case (i) does not occur), we examine the expression $A'(\log(2/\lambda\mu x^2))$, and in particular the stationary point $A'(Z) = 0$. If the stationary point always occurs at negative values of Z, then case (ii) (a repeated root) never occurs. We have

$$A'(\log(2/\lambda\mu x^2)) = -2\log(2/\lambda\mu x^2) + x\lambda e^{-\mu x} - x\lambda - xs - \mu x + 2.$$

We can see that for case (ii) to exist (that is, when $A'(\log(2/\lambda\mu x^2)) = 0$) we require

$$s = -\frac{1}{x}\left(2\log\frac{2}{\lambda\mu x^2} - \lambda x e^{-\mu x} + \lambda x + \mu x - 2\right).$$

Case	Sub-case	Sub-case Condition	Observation
(i)	(a)	$A(\log(2/\lambda\mu x)) < 0$	$A(Z)$ has one root at $Z > \log(2/\lambda\mu x)$
	(b)	$A(\log(2/\lambda\mu x)) = 0$	$A(Z)$ has one root at $Z = \log(2/\lambda\mu x)$
	(c)	$A(\log(2/\lambda\mu x)) > 0$	$A(Z)$ has one root at $Z < \log(2/\lambda\mu x)$
(ii)	(a)	$A(\log(2/\lambda\mu x)) < 0$	$A(Z)$ has one root at $Z > \log(2/\lambda\mu x)$
	(b)	$A(\log(2/\lambda\mu x)) = 0$	$A(Z)$ has one repeated root at $Z = \log(2/\lambda\mu x)$
	(c)	$A(\log(2/\lambda\mu x)) > 0$	$A(Z)$ has one root at $Z < \log(2/\lambda\mu x)$
(iii)	(a)	$A(Z_0) < 0$, $A(Z_1) < 0$	$A(Z)$ has one root at $Z > Z_1$
	(b)	$A(Z_0) = 0$, $A(Z_1) < 0$	$A(Z)$ has one repeated root at $Z = Z_0$, and another at $Z > Z_1$
	(c)	$A(Z_0) > 0$, $A(Z_1) < 0$	$A(Z)$ has roots at $Z < Z_0$, $Z_0 < Z < Z_1$ and $Z > Z_1$
	(d)	$A(Z_0) > 0$, $A(Z_1) = 0$	$A(Z)$ has one repeated root at $Z = Z_1$, and another at $Z < Z_0$
	(e)	$A(Z_0) > 0$, $A(Z_1) > 0$	$A(Z)$ has one root at $Z < Z_0$

Table 5.5: The Subcases of $Y(s)$

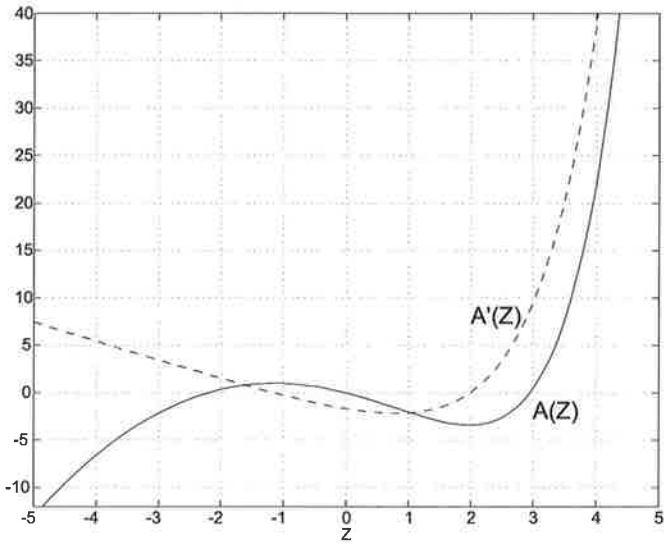

Figure 5.2: Example of $A(Z)$ case (iii)(c), using s=1.000, x=1.000, $\mu = 1.000$ and $\lambda = 0.9000$

which involves no contradiction. Therefore case (ii) can occur as a special case, meaning that a repeated root can also occur.

Approximation of the Root $Y(s)$ We have discovered much about the nature of $Y(s)$, and we have seen that finding an exact expression for the root $Y(s)$ is problematic. An alternative is to find an approximation of the root $Y(s)$ so that we may yield an expression that could be analytically inverted. This would allow us to derive a time-domain expression for the task waiting time pdf using Equation 5.18. To do this, we begin by examining how $A(Z)$ behaves as s varies. We first note that $A(Z)$ always has a root at negative Z. We have also derived an expression for determining the two stationary points using Equations 5.26 and 5.27. As $s \to \infty$, the stationary point Z_{SL} occurs at large negative values of Z. Given this, the $\lambda\mu x^2 e^Z$ term in $A(Z)$ may be disregarded in our approximation. We signify the quadratic approximation of $A(Z)$ by $\hat{A}(Z)$, and the corresponding root by $\hat{Y}(s)$. If we wish to approximate further, we may consider that the linear coefficient is approximated

by $-xs$ as the s term dominates all others. Additionally, we may consider that the $-\lambda\mu x^2$ term is negligible also. With the quadratic approximation, we have

$$\hat{A}(Z) = -Z^2 + \left(\lambda x e^{-\mu x} - \lambda x - xs - \mu x\right) Z - \lambda\mu x^2,$$

so that an approximation for the root is

$$\hat{Y}(s) = \frac{1}{2}(\alpha - xs) - \frac{1}{2}\sqrt{(\alpha - xs)^2 - \gamma},$$

where $\alpha = \lambda x e^{-\mu x} - \lambda x - \mu x$ and $\gamma = 4\lambda\mu x^2$, for the case of the left-hand root. It seems likely that this expression for $\hat{Y}(s)$ could be transformed into an expression suitable for inversion via a Bessel function property similar to Gradshteyn's Bessel function property 6.623 [39] (also given by Erdélyi as property 4.14.5 [29]). If we can do this, we have an analytic expression for the approximation of the root $\hat{Y}(s)$ inversion $\hat{y}(t)$, and we can use it to derive an approximation $\hat{f}_W(w)_{SJF}$ for the task waiting time pdf given in Equation 5.20. From this, we may derive the expectation of system value when SJF scheduling is used.

Inversion Techniques for Real s We chose to use the Laguerre-series method of numerical inversion [1] to examine the viability of inversion using real values of s. Given a LT $F(s)$ of a function $f(t)$ the Laguerre-series representation is

$$f(t) = \sum_0^\infty q_n l_n(t),\ t \geq 0,$$

where

$$\begin{aligned} l_n(t) &= e^{-t/2} L_n(t),\ t \geq 0 \\ L_n(t) &= \sum_{k=0}^{n} \binom{n}{k} \frac{(-t)^k}{k!},\ t \geq 0 \\ Q(z) &\equiv \sum_{n=0}^{\infty} q_n \left(\frac{2s-1}{2s+1}\right)^n = \left(s + \frac{1}{2}\right) F(s), \end{aligned} \tag{5.28}$$

where L_n are the Laguerre polynomials, l_n are the Laguerre functions, q_n are the Laguerre coefficients and $Q(z)$ is the Laguerre generating function. For our purposes we must truncate the number of Laguerre coefficient to limit the number of

computations. We choose to truncate the number of Laguerre coefficients to m. Following this, we choose m points at which we evaluate $F(s)$ $\{s_n, n \leq m\}$ and evaluate the m expressions on the right-hand side of Equation 5.28 above. We then compose m polynomials from the left-hand side of Equation 5.28 and solve a set of linear simultaneous equations to evaluate the Laguerre coefficients $\{q_n, n \leq m\}$. The critical decisions are to choose: (a) the number of points at which to evaluate the LT, and (b) the evaluation points themselves.

We use the example function $F(s) = (1+s)^{-1} \Longleftrightarrow f(t) = e^{-t}$ to introduce the Laguerre-series method of numerical inversion. In this example, we have chosen $m = 30$ and

$$s_n = \left\{10^{-6}, 2.000 \times 10^{-6}, 5.000 \times 10^{-6}, ..., 2.000 \times 10^{3}, 5.000 \times 10^{3}, 10^{4}\right\}.$$

The results are shown in Table 5.6 . As we can see, the largest inversion errors are observed in the interval $t = \{10^{-1}, 10^{2}\}$. It would be simple to correct this inversion error given that we know the time-domain function is monotonic decreasing. In this case we could transform all the values from $t \geq 10.00$ to zero. When we experimented with various choices for $\{s_n, n \leq m\}$, we found that adding extra samples sometimes worsened and sometimes improved the inversion accuracy, and the mechanism by which it did this was not obvious. Clearly some research is needed to determine how many evaluation points are needed and how the evaluation points are found. D'Amore *et al* have studied the real inversion problem extensively by examining Fourier series methods, Laguerre-series methods and n integral representations [23][24]. Our findings support the concerns expressed by these authors that real inversion is an ill-posed problem, requiring extra steps in the analysis. D'Amore *et al* examine various techniques that can be applied to such problems to improve the quality of the inversion.

t	$f(t)$ exact	$f(t) - \widehat{f}(t)$ by real s inversion, using 30 coefficients	$f(t) - \widehat{f}(t)$ using Week's Method, using 10 coefficients
10^{-6}	9.999×10^{-1}	3.331×10^{-16}	$< 2.225 \times 10^{-308}$
10^{-5}	6.738×10^{-3}	$< 2.225 \times 10^{-308}$	$< 2.225 \times 10^{-308}$
10^{-4}	9.990×10^{-1}	-2.220×10^{-16}	$< 2.225 \times 10^{-308}$
10^{-3}	9.900×10^{-1}	1.776×10^{-15}	$< 2.225 \times 10^{-308}$
10^{-2}	9.048×10^{-1}	-2.017×10^{-12}	$< 2.225 \times 10^{-308}$
10^{-1}	3.679×10^{-1}	1.875×10^{-6}	$< 2.225 \times 10^{-308}$
10^{0}	4.540×10^{-5}	-9.149×10^{-2}	1.871×10^{-14}
10^{2}	3.720×10^{-44}	-1.082×10^{-3}	1.454×10^{-43}
10^{3}	$< 2.225 \times 10^{-308}$	-5.849×10^{-164}	$< 2.225 \times 10^{-308}$
10^{4}	$< 2.225 \times 10^{-308}$	$< 2.225 \times 10^{-308}$	$< 2.225 \times 10^{-308}$

Table 5.6: Numeric Inversion of the Function $F(s) = (1+s)^{-1}$ by Real Inversion and by Week's Method.

Inversion using Complex s

As stated above, the problem with inverting Equation 5.19 is that we need to find the solution to the complex function $Y(s)$. We provide the following examination of inverting a simple function using complex values of s to show the accuracy we would gain if we could use complex values of s.

One of the most accurate methods of inverting LTs where s is assumed to be complex is Week's method [34], which is based on the Laguerre method [1]. We have chosen a simple example function to invert: $F(s) = (1+s)^{-1} \Longleftrightarrow f(t) = e^{-t}$. The results of the inversion are shown in Table 5.6, where N is the number of coefficients used. As we can see, the Week's method gives superior performance over real inversion in terms of accuracy. For most values of t, the inverted value using Week's method was within our computational tolerance (2.225×10^{-308}) of the real value.

Clearly, the numerical inversion of a LST using only real-values of s can be subject to large errors and is unreliable. For an inversion that is much more accurate, one must use complex values of s. A popular method of finding the complex roots of a general function is Muller's method [69]. Given a complex value of s, let us first decompose s into its real and imaginary components by designating $s_1 \equiv Real\{s\}$ and $s_2 \equiv Imag\{s\}$, so that $A(Z)$ may be re-written as

$$\begin{aligned} A(Z) &= -Z^2 + \left(\lambda x e^{-\mu x} - \lambda x - x\left(s_1 + s_2 i\right) - \mu x\right) Z - \lambda \mu x^2 + \lambda \mu x^2 e^Z \\ &= -Z^2 + \left(\lambda x e^{-\mu x} - \lambda x - x s_1 - \mu x\right) Z - \lambda \mu x^2 + \lambda \mu x^2 e^Z + x s_2 Z i. \end{aligned}$$

Given that Week's numerical inversion method chooses evaluation points of s itself, we must have a reliable, automatic method for evaluating which of the three roots is valid. We assume only the valid root will give a valid task waiting time pdf. Unfortunately, the solution to the above equation is non-trivial, as we shall see below. We begin by representing the solution to $A(Z) = 0$ as an intersection of two surface contours. As the solution to $A(Z) = 0$ may be decomposed into $Real\{A(Z)\} = 0$ and $Imag\{A(Z)\} = 0$, the solution to $A(Z) = 0$ lies in the intersection of the two

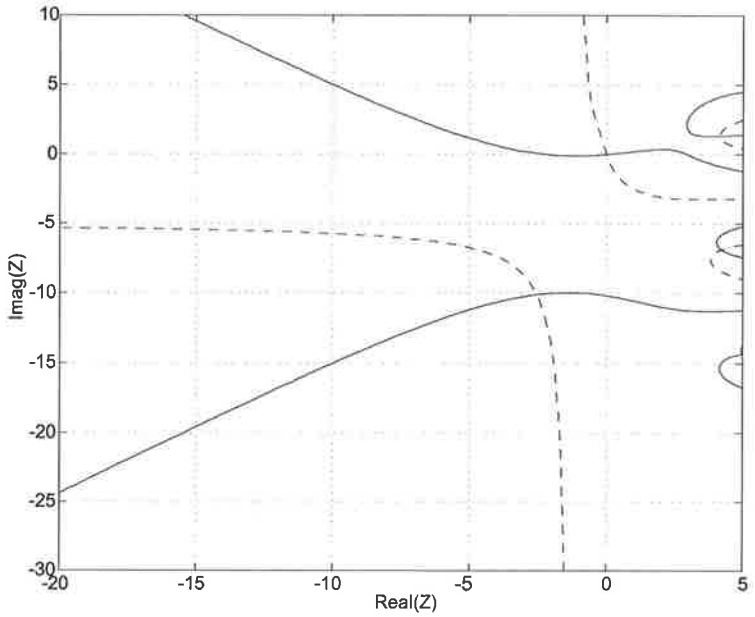

Figure 5.3: Contour plot of Real$\{A\} = 0$ (solid line) and Imag$\{A\} = 0$ (dashed line) using $s = 1.000 + 10.00i$, $x = 1.000$, $\mu = 1.000$ and $\lambda = 0.9000$

contour lines of each surface contour $Real\{A(Z)\}$ and $Imag\{A(Z)\}$ at zero. For example, lets us choose $s = 1.000 + 10.00i$, $x = 1.000$, $\mu = 1.000$ and $\lambda = 0.9000$. A plot of both surface contours about zero for these example parameters is shown in Figure 5.3. The solution to $A(Z) = 0$ is the intersection of the solid and dashed lines.

As we have seen, the behaviour of the roots for the complex $A(Z)$ case is far more difficult than for the real s case. For example, from Figure 5.3 it can be seen that two roots are distinct from the remainder of the roots on the extreme right-hand side of the figure which form a complex intersection pattern. The intersection pattern is far more complicated for other example parameter values, and no rule for determining the valid root is obvious after an initial analysis. However, this is an area of research that may yield results if studied further, enabling us to find the complex root, and consequently enabling the inversion of the waiting time LST by

Utilisation	$\overline{V}_{T,U,D,SJF}$	95% Confidence Interval
0.6000	0.7742	0.7741 – 0.7742
0.7000	0.7132	0.7131 – 0.7133
0.8000	0.6495	0.6494 – 0.6496
0.9000	0.5839	0.5838 – 0.5841
0.9500	0.5504	0.5503 – 0.5506

Table 5.7: Simulated System Value $\overline{V}_{T,U,D,SJF}$ using the Shortest Job First Scheduling Policy and Different System Utilisation Parameters

Week's method. This will allow us to derive numerical results for the expectation of the task value when the SJF scheduling policy is used.

5.2.2 Simulation Results

As we have produced no analytical or numerical results for the SJF scheduling policy, we rely upon simulation to provide insight into the behaviour of the mean task value. Results for the system model with random urgency and random soft deadline are shown in Table 5.7 . We use this table for comparison with other policies in the proceeding section.

5.3 Simulation of Other Scheduling Policies and Comparisons

As we have see throughout the thesis up this point, it has become increasingly difficult to derive analytic or even numerical results as the task execution complexity rises or the scheduling policy is anything other than FCFS or LCFS. Consequently, we must utilise simulation to gain an insight into the performance of other policies.

Table 5.8 shows a comparison of mean task values for the system model with

Utilisation	*FCFS*	*LCFS*	*SJF*	*MUF*	*MUSQF*	*MVF*
0.5000	0.7982	0.8153	0.8317	0.8164	0.8300	0.8216
0.6000	0.7104	0.7502	0.7742	0.7445	0.7689	0.7626
0.7000	0.6023	0.6811	0.7132	0.6613	0.7010	0.7015
0.8000	0.4682	0.6096	0.6495	0.5653	0.6258	0.6395
0.9000	0.2930	0.5364	0.5839	0.4539	0.5414	0.5774
0.9500	0.1775	0.4996	0.5504	0.3902	0.4943	0.5465

Table 5.8: System Value $\overline{V}_{T,U,D,SJF}$ for Different Scheduling Policies and System Utilisation Parameters

random urgency and random soft deadline, using the system time to index the task value function. As can be seen, the FCFS scheduling policy performs poorly, as at the high loads most tasks have waited a long time with their TVF consequently of little value. LCFS and Most-Urgent Service time Quotient First (MUSQF) perform well, however the SJF and Most-Valuable First (MVF) policies out-perform them by around 10% in terms of task value at the highest utilisation. Most-Urgent First (MUF) performs poorly also, better than FCFS but worse than LCFS and MUSQF.

As the difference between the policies seems to be relatively constant for all utilisations, for our second comparison we set the utilisation constant at 0.9500 and vary the mean urgency. The results are shown in Table 5.9 . It can be seen that the SJF policy in fact is not the best policy to use for all levels of urgency, and that the MUSQF and MVF policies do out-perform SJF at other values of urgency. MUSQF out-performs SJF marginally at mean urgency 0.1000, while MVF out-performs SJF at mean urgency 10.00 and 100.0. The MUF policy was the best policy for mean urgency 10^{-2}. Given this variation, we require an over-arching policy that switches the scheduling policy based on the mean task urgency. Another noteworthy point from Table 5.9 is that the FCFS policy performs even worse at the high urgency level compared with other policies. The variance in the system value in simulation runs is extremely small. For example, the 99% confidence interval widths on the SJF

Mean Urgency	*FCFS*	*LCFS*	*SJF*	*MUF*	*MUSQF*	*MVF*
10^{-2}	0.8646	0.9379	0.9694	0.9713	0.9789	0.9314
10^{-1}	0.4976	0.7854	0.8453	0.7938	0.8466	0.7976
10^{0}	0.1775	0.4996	0.5504	0.3902	0.4943	0.5465
10^{1}	0.07356	0.2913	0.3207	0.2022	0.2696	0.3409
10^{2}	0.05353	0.2311	0.2538	0.1723	0.2235	0.2750

Table 5.9: System Value $\overline{V}_{T,U,D,SJF}$ for Different Scheduling Policies and Mean Task Urgency Parameters

results in Table 5.9 are all less than 3.000×10^{-4}. Because the confidence intervals are so tight we can make comparisons between policies with a high certainty, when simulation is used.

From the analysis presented in the tables in this section, it is clear that a simple scheduling policy-switching mechanism would greatly improve the task value, and hence the value of information as perceived by end-users of the system. From the results shown in Table 5.9, this switching policy **Sw** would describe how three scheduling policies **Sc** would be used to obtain the best value:

Switching Policy Sw:

$$\mathbf{Sc} = \begin{cases} \textbf{Sc-4}\ (MUSQF) & urgency \leq 0.1000 \\ \textbf{Sc-2}\ (SJF) & 0.1000 < urgency < 10.00 \\ \textbf{Sc-6}\ (MVF) & urgency \geq 10.00 \end{cases}$$

5.4 Simulation Using Other Traffic Models

In this section we analyse the behaviour of the system when non-Poisson task service times are evident. It is common in the literature for task inter-arrival times to be exponentially distributed, and we follow this convention also. However, in practice

Mean Urgency	$FCFS$	$LCFS$	SJF	MUF	$MUSQF$	MVF
10^{-2}	0.9446	0.9556	0.9639	0.9612	0.9648	0.9499
10^{-1}	0.7905	0.8501	0.8550	0.8364	0.8552	0.8461
10^{0}	0.6010	0.6941	0.6851	0.6544	0.6758	0.6954
10^{1}	0.4650	0.5500	0.5357	0.5096	0.5210	0.5561
10^{2}	0.4074	0.4831	0.4691	0.4498	0.4474	0.4929

Table 5.10: System Value $\overline{V}_{T,U,D,SJF}$ for Different Scheduling Policies and Mean Task Urgency Parameters, using Pareto-Distributed Service Times

the service-time distribution is likely to be heavy-tailed such as a Pareto distribution [67][65][22]. The Pareto distribution has the form

$$F(x) = P[X \leq x] = 1 - (a/x)^b,$$

where b is the Pareto shape parameter.

Heavy-tailed distributions have been found in ethernets, world-wide web, TELNET and FTP traffic. We may examine the effect of Pareto-distributed task sizes by using our simulation. We assume a Pareto shape parameter of $b = 1.150$. The value of $b = 1.150$ (or a similar value) is used by many authors, and is for example analysed in [22]. As the probability distribution function for the $b = 1.150$ case was extremely long-tailed, the simulation did not self-terminate because the system value did not stabilise. We therefore terminated the simulation at 30 million task arrivals for each simulation. Once again, we used a mean critical time of $1.000s$, a mean service time of $0.9500s$, a mean inter-arrival time of $1.000s$ and we used the $\overline{V}_{T,U,D,SJF}$ system model. The results are shown in Table 5.10 . The system value was substantially higher than for the equivalent system when exponentially distributed task sizes are used. At the highest utilisation (10^2), the system value was between 20% and 35% higher than for the system results shown in Table 5.9. The mean task service time

was substantially smaller than expected[4]. We may still make confident comparisons between scheduling policies, however. The maximum 95% confidence interval width from Table 5.10 was 0.001 (FCFS), and as most differences between the simulated system values are greater than this interval, we may in general confidently compare scheduling policies. The FCFS scheduling policy again performs poorly at all mean task urgency levels. As with exponential service times that we examined in Section 5.3, the MUSQF policy performs best at the lowest mean task urgency level we have examined 0.01. At mean task urgency 0.1 the MUSQF and SJF policies perform equally well[5]. At the higher urgency levels the MVF policy performs best. The most obvious difference between the Pareto results and the exponential results in Table 5.9 are that for the Pareto case the SJF policy does not perform as well as for the exponential case.

5.5 Conclusions

In this chapter we have introduced numerical techniques to evaluate task value expectations for the LCFS and SJF scheduling policies, and compared LCFS with the FCFS policy. We have also used simulation to analyse mean task values for a number of other scheduling policies. We have previously verified the accuracy of our simulation in Chapter 4 and in Table 5.1, Section 5.1 for the case of LCFS. We have established that for a constant utilisation, the ordering of the scheduling

[4]For example, in one experiment, we looked at the running mean service time sample mean for the cases of shape parameter $b = 1.150$, $b = 1.200$ and $b = 1.500$. As the shape parameter increases, the tail of the distribution becomes less pronounced. We terminated the simulation when 200 million tasks arrivals had occurred. For the $b = 1.1500$ case, the final sample mean was 0.8902. For the $b = 1.200$ case the sample mean was 0.9193. For the $b = 1.500$ case, the final sample mean was 0.9500. As can be seen, for the extremely heavy-tailed system with $b = 1.1500$, the sample mean was significantly less than the expected mean of 0.9500.

[5]The 95% confidence interval on the system value for MUSQ is $0.8549 - 0.8555$, and for SJF is $0.8548 - 0.8552$, so we can not confidently distinguish their performance at the mean task urgency level of 0.1000.

policy is consistent from lowest to highest task value, but for varying urgency and a constant utilisation this ordering varies. The best policies with increasing levels of urgency are Most Urgent First (MUF), Most Urgent-Service Time Quotient First (MUSQF), Shortest Job First (SJF) and the Most Valuable First (MVF) scheduling policy. This evaluation was performed for five levels of mean task urgency. A study using a greater number of levels of urgency would provide tight bounds on urgency where a particular scheduling policy performed best out of the set of six.

In a deployed system the extension of these results would allow the design of a simple policy-switching system. The best scheduling policy would be chosen using an estimate of the task urgency and other parameters, and a simple scheduling policy switching mechanism would improve the task value over the long term. This would be valuable to system designers attempting to increase the value of information transmitted over networks.

Chapter 6

Overload Scheduling

We have previously discussed highly loaded, but not over-loaded systems. For these under-loaded systems, we have examined how we may switch scheduling policies based on task value, and in general we have seen that it is valuable to switch between the Shortest-Job-First (SJF), Most-Valuable-First (MVF) and Most-Urgent Service time Quotient First (MUSQF). In a real deployed system, overloads occur, and as buffers grow and waiting times become larger, information value is greatly diminished. An overload may be sustained or of short duration. In this chapter we look at how we may manage the system traffic when a sustained overload occurs. In a deployed system, this may occur when a military conflict begins. By applying various discard policies we seek to stabilise the queue and determine which discard schemes provide the best task value. Without some form of load discard in an overloaded system, the queue length and task waiting times grow, and when tasks are eventually serviced their Time Value Functions are greatly diminished, and the corresponding Task Value Functions are also greatly diminished. We must accept the loss of some tasks to keep the queue length and task waiting times from growing unbounded. We reserve this chapter to a discussion of the efficacy of such load discard measures.

We define an overload here as *processor overload*, as distinct from *deadline overload*, following the convention described in Section 2.1. That is, the system utili-

sation is greater than unity, and if we do not discard load the system will become unstable. This will mean for example that the queue length will grow unbounded, possibly causing system failure by task buffer overflow, and at the very least task value will diminish as delays grow. There are several ways we may manage the flow of information during a sustained overload. Discard policies were discussed above Chapter 3, Section 3.2.4. In Section 6.1 below we examine the effect of discarding whole tasks when the system is overloaded. In Section 6.2 we then consider the effect of partially servicing tasks, given that they are composed of a number of discrete units. Further, for partial service, we assume that the tasks have been composed into a number of sub-units, where servicing each sub-unit results in an increased value for the task. Not all sub-units must be serviced, and in general the most valuable sub-units are completed first, giving a greater value than the proportion of time spent on a partially-serviced task. As described in Chapter 2, Section 2.4, this is known as the Increased Reward with Increased Service (IRIS) model of task completion.

In this chapter we do not attempt to use an analytical approach to gain an insight into the system value. Rather, we utilise simulation to examine the system value behaviour. We must modify our previous simulations in order to analyse overload scheduling. When we analysed underload system scheduling, we used the stability of the queue length as our criterion for determining when the system reached equilibrium. As we are now limiting the queue length to small values, and sometimes even eliminating the queue altogether, we need another criterion for determining when equilibrium occurs. We chose to use the mean task value stability as our criterion. To do this, we observe the mean task value at each instant a task arrived in the queue, and when 100 consecutive values are within a tolerance of 10^{-5} we assume the system is in equilibrium. We then begin to collect system value statistics, and terminate each simulation when pre-equilibrium task arrivals account for 5% of the total number of task arrivals. The simulation is repeated to give 12 simulation runs in order that we may derive 95% confidence intervals for system value. All

Maximum Queue Length	Longest Task Discard Policy	Least Urgent Task Discard Policy	Least Valuable Task Discard Policy
0	0.7789	0.7789	0.7789
1	0.7349	0.6530	0.7626
2	0.7075	0.5946	0.7076
5	0.5142	0.5238	0.5747
10	0.5218	0.5016	0.5202
20	0.5142	0.4998	0.5137

Table 6.1: System Value $\overline{V}_{T,U,D,SJF}$ using Task Discard, with Parameters: Urgency=1.000, μ=1/1.100, ρ=1.100 and mean critical time=1.000s

the simulation results presented in this chapter use the mean task value stability technique. The variance of the results is comparable with results we calculated previously in Chapters 4 and 5. That is, the variance between our simulation runs is small enough that we may confidently make comparisons between the simulated values.

6.1 Task Discard Policies

As we show in Table 6.1 , we chose to examine a number of task discard schemes, given that we limited the queue to a certain number of tasks. We chose to study policies which discard based on task length, urgency and value. We have used the SJF scheduling policy as a basis for the study. As we limited the queue capacity to several tasks and discarded any further arrivals, we found that the mean task value (even including the discarded tasks which contribute no value) increases substantially. The example used only a slightly overloaded system, and yet the increase in task value was substantial. The results show that the choice of discard policy does not affect the system value, as the queue is removed, and the system thereby

Maximum Queue Length	Longest Task Discard Policy	Least Urgent Task Discard Policy	Least Valuable Task Discard Policy
0	0.8082	0.8082	0.8082
1	0.7728	0.7728	0.7728
2	0.7370	0.7371	0.7370
5	0.6445	0.6445	0.6445
10	0.5777	0.5778	0.5778
20	0.5535	0.5535	0.5534
∞	0.5504	0.5504	0.5504

Table 6.2: System Value $\overline{V}_{T,U,D,SJF}$ using Task Discard, with Parameters: Urgency=1.000, μ=1/0.9500, ρ=0.9500 and mean critical time=1.000s

requires no discriminative discard policy. The maximum system value in Table 6.1 (0.7789) is significantly (20%) higher than the best mean task value achieved in the under-loaded ($\rho = 0.9500$) case with no discards, 0.5504, as shown in Table 5.9.

The significant improvement on mean task value by discarding tasks for only a slightly over-loaded system suggests that we may also impart system value gains for under-loaded systems. It is obvious why we must discard load in an overloaded system, but we may also apply the same reasoning about increases in system value to under-loaded systems. If we are prepared to accept the loss of some low-value tasks which prevent the timely service other more valuable tasks, then the system value should increase over the long term. For example, a task with a large service time but a low time-value may be serviced at the expense of a number of low service time, high time-value tasks. If we discarded the current task, we would lose a small amount of value from the current task but gain a much larger amount of value from the tasks that follow. We have examined the use of task discard for under-loaded system, and have presented the results in Table 6.2 . The equivalent result for the under-loaded case where discards are not used are found in Table 5.7. The mean

Maximum Queue Length	Longest Task Discard Policy	Least Urgent Task Discard Policy	Least Valuable Task Discard Policy
0	0.6410	0.6410	0.6410
1	0.5953	0.5953	0.5955
2	0.5559	0.4468	0.6017
5	0.4728	0.3713	0.4997
10	0.4629	0.3715	0.4712
20	0.4620	0.3716	0.4659

Table 6.3: System Value $\overline{V}_{T,U,D,SJF}$ using Task Discard, with Parameters: Urgency=1.000, μ=0.5000, ρ=2.000 and mean critical time=1.000s

task value for this case is 0.5504. We found that a substantially higher system value was achieved for under-loaded systems by discarding tasks, based on a policy of limiting the queue length, as is shown from Table 6.2. Table 6.2 also shows that for this under-loaded system, the choice of discard scheme is not important, even when the queue is limited to several tasks. As expected, the best system value shown in Table 6.2 is slightly higher than for the over-loaded case, as we are discarding fewer tasks. These results prove that low time-value tasks were indeed preventing high time-value tasks from receiving timely service, and discarding these low time-value tasks was the correct decision. We note, however, that in under-loaded systems that do not use decreasing Time-Value Functions, such as simple hard deadline systems, task discards may not be as effective as when used in our system. This is because tasks do not lose value as they wait in the queue as in our system.

To examine highly over-loaded systems, we set a utilisation of 2.000, and have presented the results in Table 6.3 . We have been careful in comparing the values from systems with different service time statistics. We held the inter-arrival rate unaltered, so we observed the same number of task arrivals in all the models we examined. However, as we raised the mean service time of tasks, we assumed that the

tasks were still worth equal value. In reality, longer tasks may be worth more than shorter tasks, and this could be factored into the task execution objective function by, for example, multiplying task value functions with service times. Again, we see that significant gains in task value may be made by task discard schemes even for the very highly overloaded case. It is still possible to achieve a mean task value that is higher than for the slightly under-loaded case with no discard ($\rho = 0.9500$), 0.5504. Again, the highest value (0.6410) was achieved by removing the queue altogether.

The clear conclusion that can be drawn from this section is that task discards are an extremely valuable and simple policy to use, not only for overloaded systems but also for under-loaded systems. In the following section we compare the performance of task discard schemes with partial task service schemes.

6.2 Partial Task Service Policies

In this section we investigate the use of partial task discard policies to reduce load under overload conditions. To partially service a task, we choose the Mandatory-Optional Decomposition (MOD) method as described in Chapter 2, Section 2.4. In the MOD method, tasks are decomposed into mandatory and optional parts, and optional parts may be ignored in order to reduce load and improve task throughput times. We chose to set a threshold queue length. Whilst the queue length threshold was exceeded, all arriving tasks were tagged so that only their mandatory part is serviced. Whilst the threshold was not exceeded, all arriving tasks were tagged for normal service. It was expected that this approach would reduce the size of the queue enough so that arriving tasks waited only a small time and were hence worth more value. Obviously we must speculate on how much the mandatory part is worth to the system as compared with the optional part. We must also speculate on how large the mandatory part is compared to the optional part. This depends on the type of information presented. A text document may be decomposed into an executive summary and remainder of the document, for example.

Threshold Queue Length	5%/10% Task Value Policy	5%/20% Task Value Policy	5%/50% Task Value Policy
0	0.09971	0.1994	0.4986
1	0.4436	0.4728	0.5606
2	0.4550	0.4769	0.5423
5	0.4549	0.4664	0.5011
10	0.4521	0.4604	0.4856
20	0.4518	0.4601	0.4845

Table 6.4: System Value $\overline{V}_{T,U,D,SJF}$ using Partial Task Service, with Parameters: Urgency=1.000, μ=1/1.100, ρ=1.100 and mean critical time=1.000s

We assumed that the mandatory part was only 5% of the total size, and that this was worth 10% of the total task value. We refer to this partial task example as the 5%/10% MOD model. This models the unequal information value density within a task that may be separated for our purposes. Table 6.4 shows how the $\overline{V}_{T,U,D,SJF}$ model behaves for these assumptions and various queue length thresholds, given a system utilisation of 1.100. We were surprised to find that, for the 1.100 utilisation level, task discard enables approximately 30% more system value than for partial task service, based on a comparison between Tables 6.1 and 6.4. We initially guessed that the partial task service method would be competitive with the task discard method. When we altered our assumption of the partial task value and instead used a 5%/20% model, we found there was a slight improvement in system value, but the system still did not deliver significant value. These results are also presented in Table 6.4. Even when small partial tasks are worth an unrealistically large amount, the partial task discard method still delivers significantly less value than for the task discard scheme. The far-right column in Table 6.4 shows this for the case of the 5%/50% model. In practice the 5%/50% would rarely exist, as it suggests an unrealistically high density of value in the mandatory part of each the task.

Threshold Queue Length	50%/55% Task Value Policy	50%/65% Task Value Policy	50%/75% Task Value Policy
0	0.4419	0.5222	0.6026
1	0.4899	0.5256	0.5613
2	0.4852	0.5130	0.5407
5	0.4712	0.4875	0.5038
10	0.4656	0.4780	0.4908
20	0.4651	0.4775	0.4899

Table 6.5: System Value $\overline{V}_{T,U,D,SJF}$ using Partial Task Service, with Parameters: Urgency=1.000, μ=1/1.1000, ρ=1.100 and mean critical time=1.000s

The policy is functioning as designed in the sense that it decreases the otherwise unbounded queue. This reduces the amount of time tasks wait until service, and therefore the tasks' Time-Value Function decay was reduced, providing more system value. In fact, partial service-tagged tasks are worth a maximum of 0.1000 for the 5%/10% model and 0.2000 for the 5%/20% model. However, we must conclude that the loss in task value out-weighs the benefits of decreasing task waiting times.

Partial task discard initially seems to give little benefit for the systems we have examined, and even reduces the mean task value for our set of assumptions based on the construction of the sub-tasks. If we had more granularity in partial task size, we may be able to increase the performance of the partial task service policy. We could do this by providing more optional task parts, or we could increase the size of the one optional part. In the next experiment, we increased the size of the optional part of the task. Table 6.5 shows the results for 50%/55%, 50%/65% and 50%/75% models. All models were still realistic, in that none of the models required an unrealistic amount of value density in the mandatory part of each the task. Table 6.5 shows that with this model, substantially less value was still delivered than for the task discard scheme. In our final experiment we examined the 75%/90% model.

Threshold Queue Length	75%/90% Task Value Policy, using Mean Critical Time of 1.000s	75%/90% Task Value Policy, using Mean Critical Time of 10.00s	75%/90% Task Value Policy, using Mean Critical Time of 20.00s
0	0.5700	0.8081	0.8463
1	0.5449	0.8128	0.8592
2	0.5321	0.8077	0.8521
5	0.5071	0.8056	0.8470
10	0.4981	0.8078	0.8462
20	0.4975	0.8081	0.8463

Table 6.6: System Value $\overline{V}_{T,U,D,SJF}$ using Partial Task Service, with Parameters: Urgency=1.000, μ=1/1.100 and ρ=1.100

These results are shown in Table 6.6 . Even at this high task value, the system still failed to deliver a system value approaching that for the task discard scheme.

We have seen that it has been generally more valuable to completely remove the queue or limit it to several tasks only. This is because the task model we have assumed throughout this chapter has included a mean critical time of 1.000s. We can see the effect on the best queue length threshold by extending the mean critical time to 10.00s and 20.00s in Table 6.6. For the case where tasks have extended critical times, the best queue length threshold is no longer 0, as the tasks may wait longer in the system before their Time-Value Function decays, reducing the task's value.

We conclude that for partial task service schemes, the system value reduction outweighs the benefit of reducing waiting time for each task. Task discard schemes, in this initial assessment, seem to give a much greater system value than partial task service schemes.

6.3 Conclusions

For under-loaded systems significant mean task value gains may be achieved by task discard. The choice of discard policy is not important. When the system becomes over-loaded the choice of discard policy becomes important, with Least Task Value the best choice for our particular system example. We note, however, that in under-loaded systems that do not use decreasing Time-Value Functions, such as simple hard deadline systems, task discards may not be as effective as when used in our system.

For slightly over-loaded systems ($\rho = 1.100$) the discard policies using task models 5%/10%, 5%/20% (and even the unrealistic 5%/50% model) contribute less than 20% of the value when using whole task discard schemes. When we looked at models with a larger mandatory component we found a significant improvement, however this still delivered far less value than for the task discard scheme. For example, the 50%/75% model produced approximately 10% more system value than the 5%/20% model. The 75%/90% model performed similarly to the 50%/75% model. Task value reduction greatly outweighed the benefit of reduced waiting time.

We conclude that either a task discard scheme be used or that it be combined with some kind of partial task service scheme. Task discards are clearly valuable and simple to use, not only for overloaded systems but also for under-loaded systems. Their use can increase the system value above that even for under-loaded systems.

Chapter 7

Discussion and Conclusions

Information flow management is an important step in ensuring that the value of information to the end user is acceptable, and we have applied various techniques in order to achieve this. Our work was motivated by an application in satellite broadcast communications. We have applied stochastic scheduling analyses within a time-driven scheduling model in order to provide a system framework where the value of information delivered to the end user may be optimised. We have followed other authors' work on time-driven scheduling, where hard deadline models of real-time task execution have been extended to include the concept of soft deadlines. We adopted this framework as it allows a graceful degradation in the value of information in the presence of overload conditions.

We firstly presented a system framework for use in the field of value-based scheduling, which facilitates analysis based on a suite of policies used to maximise the value of information to be managed in the system. We also presented a task execution model that gives the system flexibility in the management of the flow of tasks through the system, which proves more critical in the case of high system load and overloads.

We derived analytical results for the value of tasks for various system model configurations for the First-Come First-Served (FCFS) scheduling policy. This was done for both serial and parallel processing systems. We showed that the serial pro-

cessing system out-performed the parallel-processing system in terms of the system value metric. We then derived numerical results for the Last-Come, First-Served (LCFS) scheduling policy for the case of the simple model with a waiting time objective function. This could easily be extended to include a system time objective function, as is assumed with our other models. We described how this would be done via a numerical convolution in the time domain. We compared the task value expectations for the FCFS and LCFS case and showed that LCFS is a superior scheduling policy.

We have examined the Shortest Job First (SJF) scheduling policy in a queueing-theoretic framework. We limited our discussion to the derivation of the waiting time Probability Distribution Function, a necessary step in deriving the expression for system value. We found that the analytic complexity limited our discussion, hence we presented approximation techniques and possible extensions to our work.

Given that the analytic complexity of our queueing-theoretic approaches has limited our derivation of system value for models other than FCFS and LCFS, we used simulation analysis to examine the performance of the system when other scheduling policies and discard policies were used.

Our simulation procedure produced results with low variance, as shown by the 95% confidence intervals from Table 4.8. We have also verified the accuracy of our simulation by using our analytic results. The simulation has been used to analyse system value for a number of other scheduling policies, such as the Most Urgent-Service Time Quotient First (MUSQF), Shortest Job First (SJF), Most Urgent-Service Time Product First (MUSPF), the Most Valuable First (MVF), Most Urgent First (MUF) and Earliest Critical Time (ECT) policies. We have established that the ordering of scheduling policies from lowest to highest task value is more sensitive to changes in mean task urgency than system utilisation. When we hold the mean task urgency constant and vary the system utilisation, the ordering of the scheduling policy is consistent from lowest to highest task value, with SJF the most valuable policy for our example. However, when we vary the mean task urgency and hold the

system utilisation constant this ordering varies, so we must specify the best policy for each mean task urgency value. The best policies with increasing levels of urgency are MUSQF, SJF and the MVF scheduling policies. This evaluation was performed for five levels of mean task urgency. More studies are needed to provide accurate bounds on the levels of urgency where a particular scheduling policy performs best out of a given set of policies. We have repeated the simulation for various system configurations, including using various levels of utilisation. We have found that, in general, the MUSQF, SJF and MVF policies out-perform FCFS, MUF and LCFS scheduling policies.

We have shown how to extend our results to design a simple policy-switching system for a deployed system. The best scheduling policy would be chosen using an estimate of the task urgency and other parameters.

The work presented in the thesis builds upon other work in literature in static value-based scheduling by analysing the problem from a dynamic and stochastic point of view. Previously, researchers have used static, contrived data-sets to present results. We have generalised the problem by providing a system framework and task model. We have derived analytical results, and have extended this analysis to more complex scheduling policies and an examination of system behaviour during processor overloads when load discard policies are used.

We initially used the assumption of exponentially-distributed task inter-arrival times and service times. However, we then extended our analysis to heavy-tailed task service times, as we are aware that researchers in teletraffic systems have discovered heavy-tailed distributions for service times in TELNET, FTP and ethernet traffic. The results show that the SJF scheduling policy is less valuable compared to MUSQF and MVF, as compared to the case where service times are exponentially distributed. However, SJF still out-performs FCFS, MUF and LCFS in the heavy-tail case.

When the system becomes over-loaded, the choice of discard policy becomes important, with Least Task Value discard the best choice for the parameters we have chosen. Partial task discard initially seems to give some benefit for the systems

we have examined, but is out-performed by task discard policies. We have also determined that for under-loaded systems significant system value gains may be achieved by task discard. For under-loaded systems, the choice of discard policy is not important. Task discards are an extremely valuable and simple technique, not only for overloaded systems but also for under-loaded systems.

We need to understand better the mechanisms behind the users' evaluation of the timing of information. We have attempted to use exponential, linear and quadratic functions of time-value decay in order to model the user's expectations. We foresee that a foundation in information theory and information value theory may provide a framework to more effectively approach this topic in greater detail. A quantitative study of a deployed system may also be conducted to populate this framework with real data. If we are to use partial task service in an effort to mitigate the effects of overload, we must first decompose each task into sub-tasks, and understand how the completion of each maps onto the value of each sub-task. An information-theoretic approach will also provide a better framework for the qualitative and quantitative study of the value of incomplete information.

There are many other opportunities to extend the research presented in this thesis. For example, it may be possible to determine an optimal management policy. In this case, we would assume that a policy is optimal if no other policy can deliver any more system value over an infinite planning horizon. Even understanding what the upper bound may be would prove useful as it would would provide a suitable baseline from which to measure the performance of all other management policies. A simple improvement to the MVN scheduling policy would be to consider pairs of the most valuable-next tasks, that is, by making the MVN policy more cognisant of the future impact of its actions. We foresee that an improved scheduling policy would take into account task service time, urgency, critical times, the state of the queue and the state of the service facility.

We may also extend our queueing-theoretic approach to other consider other task execution models, including the use of alternate objective functions, and possibly

precedence relationships between tasks.

We have ignored pre-emption of tasks throughout the thesis. Our simulation is capable of pre-empting tasks, however, and it seems reasonable to assume that pre-emption could lead to improved system performance. For example, a low value, long service time task may be pre-empted by any number of newly-arriving high value, short service time tasks to increase system value.

Another extension to the research is to look more carefully at the serial versus parallel service issue. We looked at the simple 2-server case and concluded that serial processing always out-performed 2-server processing. However, we also noted that in our model, the service facility was under-utilised when there was a single task in the system. This would have been improved by using a processor-sharing system. We also note that when tasks do not impart system value at the instant of departure, and instead impart system value during their service, parallel processing may be more valuable.

Appendix A

The Incomplete Gamma Function

The incomplete gamma function appears often in the analysis of the task value function for systems using the First-Come, First-Served scheduling policy, as found in Chapter 4. To aid in further analysis, various properties of the Incomplete Gamma Function (IGF) are discussed below in particular relevance to the forms of the expression that are discussed in this thesis. The reference book by Abramawicz *et al* [2] includes a number of useful properties of the IGF. The definition of the IGF is

$$\Gamma(a,z) \stackrel{def}{=} \int_z^\infty t^{a-1}e^{-t}dt, \tag{A.1}$$

which is property 6.5.3 from [2].

Some authors define Equation A.1 with different bounds on the integral such that $\Gamma(a,z)^* \stackrel{def}{=} \int_0^z t^{a-1}e^{-t}dt$ so that $\Gamma(a,z) + \Gamma(a,z)^* = \Gamma(a)$ as we would expect. We find the $\Gamma(0,z)$ form of the function often in the analysis, which is given by the integral

$$\Gamma(0,z) = \int_z^\infty \frac{e^{-t}}{t}dt.$$

A useful identity of the IGF that is also used in Chapter 4 is also found in the

$\Gamma(a,z) = e^{-z}z^a\left(\frac{1}{z+}\frac{1-a}{1+}\frac{1}{z+}\frac{2-a}{1+}\frac{2}{z+}\cdots\right)$	$x>0, \lvert a\rvert<\infty$
$\Gamma(a,z) \sim z^{a-1}e^{-z}\left[1+\frac{a-1}{z}+\frac{(a-1)(a-2)}{z^2}+\cdots\right]$	$z\to\infty, \lvert\arg z\rvert<\frac{3\pi}{2}$
$\Gamma(z+1,z) \sim z^z e^{-z}\left[\sqrt{\frac{z\pi}{2}}+\frac{2}{3}+\sqrt{\frac{2\pi}{24z}}+\cdots\right]$	$z\to\infty, \lvert\arg z\rvert<\frac{\pi}{2}$

Table A.1: Series Approximations and Expansions for the Incomplete Gamma Function

reference book [2, pg. 230]:

$$\int_d^\infty \frac{e^{-ax}}{b+x}dx = e^{ab}\Gamma(0, a(b+d)).$$

A short list of useful series approximation and expansions is given in Table A.1 so that the reader can extend the analysis of Chapter 4 if desired.

Appendix B

Simulation

The simulation code "Bilbo" was written in C++ on a Unix environment during 1998. The code was later ported to the Cygwin development environment, version 1.1.0. Cygwin is a Unix environment (an emulation) for the Windows operating system. This port allowed the simulation code to be executed on any Windows PC with the inclusion of a ".dll" file. The code has since been developed over approximately three and a half years and contains around 5000 lines of code.

The simulation package includes eight ".c" files with seven corresponding ".h" files, compiled into a ".exe" executable of around 600 kB size. Seven of the ".c" files define the tangible objects within the scheduling system. They are the:

- Task object.
- Task management object.
- Population object.
- System object.
- Statistics object.
- Server object.
- Queue object.

Once a simulation begins, a system object is created. As multiple simulations may be performed to examine varying system parameters, queue, server, scheduler,

population and statistics objects are created and destroyed once every simulation. For each system object, task objects are created and enter the queue, and are destroyed once they leave the (or a) server.

During each simulation, the system maintains an internal simulation clock and a knowledge of when the next event occurs. The event may be a task arrival or departure either at the queue or server. The internal simulation time jumps forward to this event and the event is managed by the task management. When an arrival of departure occurs, the population object and scheduler object set another event to replace it.

The statistics object examines 75 properties of the system including mean queue length, task inter-arrival times, task service times, the system utilisation, the stability of the queue, mean task value and the proportion of tasks that miss their soft deadline. Detailed summaries of simulation runs may be produced to display these statistics.

The stochastic properties of the traffic were created by a 32-bit pseudo-random number generator (PRNG). As the simulations are often quite long, this sequence may repeat itself several times. As no better PRNG was available, we used the 32-bit PRNG with this problem in mind, as repeating sequences were not seen as a significant source of error in the simulation results. We used the PRNG to create uniformly-distributed variates on (0,1), and with this used Monte-Carlo sampling on inverse Negative-Exponential and Pareto Cumulative Distribution Functions to create the variates for traffic arrivals, soft deadlines and task service times.

The simulation code is executed by user-set command line switches. There are 26 commands available to tailor the simulation to the user's needs. These are:

- Soft deadline format—format of the probability density function (pdf); either negative exponential, Rayleigh or constant.

- Priority format—Format of the task priority pdf, either negative exponential or constant.

- Discard Initiator—Task discard initiator, either maximum queue length exceeded or random discards.
- Task discard policy—Task discard policy, either (a) least valuable first, (b) least urgent first, (c) smallest task first or (d) partial task discard.
- Equilibrium statistics—Only collect statistics after system is in equilibrium.
- Help page—Displays a help page with all commands.
- Inter-arrival time format—Format of the task inter-arrival time pdf, as for "Soft deadline format" above.
- Limit utilisation—Limit the utilisation if using random discards.
- Soft deadline mean—Mean of the task soft deadlines.
- Inter-arrival time mean—Mean of the task inter-arrival times.
- Maximum queue size—Limit the queue size for discards.
- Service time mean—Mean of the task service time.
- Urgency mean—Mean of the task urgencies.
- Number of tasks—Maximum number of tasks to queue.
- Number of scheduling policies—Number of policies to examine.
- Number of simulation runs—Number of simulation runs to use.
- Number of servers—Number of servers (processors or channels) to use.
- Objective function—System value objective function; either using system time or waiting time.
- Text output—Format of text to send to screen.
- Optimal load—Set the optimal load to reach.
- Pre-emption—Enable pre-emption of tasks based on various policies.
- Standard deviation—Collect standard deviation statistics between simulation runs.
- Service time format—Format of the task service time pdf, as for "Soft deadline format" above.
- Simulation number—Scheduling policy start number to examine.

- Time-value function—Time-benefit function format, either negative exponential, linear (non-negative or general) or parabolic (non-negative or general).
- Urgency format—Format of the task urgency pdf, as for "Soft deadline format" above.

An example (real) simulation execution is shown below. The example is an analysis of the mean system value for various levels of task urgency, given a Shortest-Job First scheduling policy. The output shows mean system value (underlined) and its 95% and 99% confidence intervals:

```
bilbo -sp 2 -np 1 -ms 1.1 -eq 3 -ds 2 -di 2 -nr 12
================================================
Bilbo:  Stochastic scheduling simulation v0.08
written by Matthew Britton as part of PhD
================================================
Simulation began execution at Sun Nov 25 15:56:59 2001
Starting at policy # 2
-Analysing 1 scheduling policies
 Mean interarrival time set to 1
 Mean service time set to 1.1
 Mean critical time set to 1
 Collecting statistics after equilibrium reached,
    using method #3:  Mean queue length stability
 Simulation stopping criterion method set to #1
 Set number of inter-arrival time levels to 1
 Set number of service time levels to 1
 Using the following parameters:
 -simulation #0:  Analysis of mean task value for
    various utilisations and urgencies
-each simulation will end when task value is stable after equilibrium
-servers:  using 1 only
-utilisation levels:  looping through 1 levels
-simulation runs:  averaging 12 runs to arrive at each result
-urgency levels:  looping through 5 levels
-service time levels:  looping through 1 levels(1.1)
-interarrival times:  looping through 1 levels (1)
-discard initiator:  using #2:  Discarding a task once
    queue has more than 1 tasks in it
-discard scheme:  using #2:  Longest job next
-preemption policy:  using #0:  No preemptions
-objective function:  using #0:  Pure time-value
-time-benefit function:  using #0:  Negative-exponential
-service time pdf:  using #0:  Negative exponential density
-interarrival time pdf:  using #0:  Negative exponential density
-critical time pdf:  using #0:  Negative exponential density
-urgency pdf:  using #0:  Negative exponential density
Simulation execution report:
Util OptLd STime IATime Urg  DSch MaxQ  SJN      StDev(BE)
1.1    1    1.1    1    0.01  2    1   0.9931  6.417e-06
    min=0.9931 max=0.9931 95%CI=0.9931-0.9931 99%CI=0.9931-0.9931
1.1    1    1.1    1    0.1   2    1   0.9417  4.43e-05
    min=0.9416 max=0.9417 95%CI=0.9416-0.9417 99%CI=0.9416-0.9417
1.1    1        1.1    1     1    2    1   0.7349 0.0001314
    min=0.7346 max=0.7351 95%CI=0.7348-0.735 99%CI=0.7348-0.735
```

```
1.1    1        1.1    1    10    2    1   0.5075 0.0001883
    min=0.5072 max=0.5079 95%CI=0.5074-0.5076 99%CI=0.5073-0.5077
1.1    1        1.1    1   100    2    1   0.4259 0.0001773
    min=0.4256 max=0.4262 95%CI=0.4258-0.4261 99%CI=0.4258-0.4261
#Simulation took 0 days, 13 hours, 12 minutes and 51.969000 seconds
```

A copy of the simulation executable "bilbo.exe" may be downloaded from the world-wide web at *geocities.com/brittonm1*. The file "cygwin.dll" is also required in order to execute the file, and may be downloaded from *sources.redhat.com/cygwin*.

Appendix C

Optimality of Serial Two-Task Scheduling

Proposition For the case of statically scheduling two tasks, each described by service times and time-value functions that are monotonic decreasing with sojourn time and urgency, and where a task only contributes value to the system at the instant of completion, a serial schedule will always produce more time-value than a parallel schedule.

Proof

We denote the tasks by T_0 and T_1. Both tasks have a positive real urgency u_0 and u_1 associated with their execution which describes the decay in potential value that may be imparted to the system upon task completion. In general, parallel scheduling allows us to segment the processor dynamically so that two real-valued functions $p_0(t)$ and $p_1(t)$ describe the proportion of processor power the tasks receive at any time t.

Serial Execution Case

The choice of which task T_n to schedule first will depend on its service time μ_n and urgency u_n and also the properties of the other task. We denote the time-value function of a task T_n by $f(u_n, \mu_n)$, and the time-value of completion as V_n. If task

T_0 is scheduled first its value will be $V_0 = f(u_0, \mu_0)$ and the value of the other task will be $V_1 = f(u_1, \mu_0 + \mu_1)$. However if the task T_1 is scheduled first the system receives $V_0 = f(u_0, \mu_0 + \mu_1)$ and $V_1 = f(u_1, \mu_1)$. The maximum system value is received by maximising the sums of values for each case. We arbitrarily choose to schedule first the task whose index (subscript) ordinally precedes the other when the order is not important. Therefore,

$$V_s = \begin{cases} f(u_0, \mu_0) + f(u_1, \mu_0 + \mu_1) & if \\ & f(u_0, \mu_0) + f(u_1, \mu_0 + \mu_1) \\ & \geq f(u_0, \mu_0 + \mu_1) + f(u_1, \mu_1) \\ f(u_0, \mu_0 + \mu_1) + f(u_1, \mu_1) & if \\ & f(u_0, \mu_0) + f(u_1, \mu_0 + \mu_1) \\ & < f(u_0, \mu_0 + \mu_1) + f(u_1, \mu_1). \end{cases} \tag{C.1}$$

V_s denotes the system value for the serial processing case. Regardless of the processor share functions $p_n(t)$, the same amount of processor effort must be expended. The functions may be devised such that both tasks complete at the same time or where one may complete before the other. We now divide this proof into two sections to analyse each case.

Parallel Execution Case 1: Tasks Complete at the Same Time

When both tasks complete at the same time, the final execution time will be $t = \mu_0 + \mu_1$ and the system value will be

$$V_{p1} = f(u_0, \mu_0 + \mu_1) + f(u_1, \mu_0 + \mu_1). \tag{C.2}$$

V_{p1} denotes the system value for the special case where both tasks complete at the same time. Because the function f is monotonic decreasing on urgency and sojourn time, we can state that

$$f(u_0, \mu_0) > f(u_0, \mu_0 + \mu_1) \tag{C.3}$$

$$f(u_1, \mu_1) > f(u_1, \mu_0 + \mu_1). \tag{C.4}$$

By making additions to the left- and right-hands of the inqualities we can also state that

$$f(u_0, \mu_0) + f(u_1, \mu_0 + \mu_1) > f(u_0, \mu_0 + \mu_1) + f(u_1, \mu_0 + \mu_1) \tag{C.5}$$

and

$$f(u_0, \mu_0 + \mu_1) + f(u_1, \mu_1) > f(u_0, \mu_0 + \mu_1) + f(u_1, \mu_0 + \mu_1), \tag{C.6}$$

which are the inequalities we require in order to prove the optimality of serial processing over this special parallel processing case. That is,

$$V_s > V_{p1} \tag{C.7}$$

Parallel Execution Case 2: One Task Completes Service Before the Other

Because the amount of processor workload is fixed, the task that completes second must still complete at time $t = \mu_0 + \mu_1$. The other task will complete before this time, but will complete some time after its own service time because it shares the processor during its sojourn with the other task. We use the variables ϵ_0 and ϵ_1 to denote the extra time each task takes to complete its service past the time it would otherwise complete given an unshared processor. The bounds imposed on these variables are then $0 < \epsilon_n < \mu_{\overline{n}}$. The left hand bound in Inequality C.5 is the serial processing case and the right hand bound is the special parallel processing case where tasks complete at the same time. The system value is then described by

$$V_{p2} = \begin{cases} f(u_0, \mu_0 + \epsilon_0) + f(u_1, \mu_0 + \mu_1) & if \quad \text{task } T_0 \text{ completes first} \\ f(u_0, \mu_0 + \mu_1) + f(u_1, \mu_1 + \epsilon_1) & if \quad \text{task } T_1 \text{ completes first.} \end{cases} \tag{C.8}$$

For the case where task T_0 completes first in the parallel case we have the two serial processing cases to compare against: either task T_0 or T_1 may be served first. Let

us examine to the case when the task T_0 is completed first in each case. Hence, the following inequality holds:

$$f(u_0, \mu_0) + f(u_1, \mu_0 + \mu_1) > f(u_0, \mu_0 + \epsilon_0) + f(u_1, \mu_0 + \mu_1). \tag{C.9}$$

Similiarly, for the case where T_1 completes first in each case we can state that

$$f(u_0, \mu_0 + \mu_1) + f(u_1, \mu_1) > f(u_0, \mu_0 + \mu_1) + f(u_1, \mu_1 + \epsilon_1). \tag{C.10}$$

Inequalities C.9 and C.10 give us a comparison between Inequalities C.1 and C.8, proving the optimality of serial vs parallel scheduling for the case where either T_0 or T_1 is completed first in both cases. This completes half the proof for the present case. For the case where task T_0 completes first in the parallel case and T_1 completes first in the serial case we know that a necessary condition for T_1 completing first in the serial case is given in the second line of Inequality C.1 as

$$f(u_0, \mu_0 + \mu_1) + f(u_1, \mu_1) > f(u_0, \mu_0) + f(u_1, \mu_0 + \mu_1), \tag{C.11}$$

because f is monotonic decreasing on μ we can also state that:

$$f(u_0, \mu_0 + \mu_1) + f(u_1, \mu_1) > f(u_0, \mu_0 + \epsilon_0) + f(u_1, \mu_0 + \mu_1), \tag{C.12}$$

which is the inequality we need to prove the optimality of serial processing in this case, by giving a direct comparison between the second line in Inequality C.1 and the first line in Inequality C.8. Similarly for the case where task T_1 completes first on the parallel processor and T_0 completes first on the serial processor we have the serial processing condition given in the first line of Inequality C.1 as

$$f(u_0, \mu_0) + f(u_1, \mu_0 + \mu_1) \geq f(u_0, \mu_0 + \mu_1) + f(u_1, \mu_1). \tag{C.13}$$

Again, because f is monotonic decreasing on μ we can state that

$$f(u_0, \mu_0) + f(u_1, \mu_0 + \mu_1) \geq f(u_0, \mu_0 + \mu_1) + f(u_1, \mu_1 + \epsilon_1), \tag{C.14}$$

which is what we require in order to prove serial processing optimality in this case, by giving a direct comparison between the first line in Inequality C.1 and the second line in Inequality C.8. The four cases studied prove the optimality of serial processing over parallel processing when one task completes before the other. That is:

$$V_s > V_{p2} \tag{C.15}$$

Combining the Inequalities C.7 and C.15 we can now state that serial processing will always outperform parallel processing for the two-task case.

Bibliography

[1] J. Abate, G. L. Choudhury, and W. Whitt. On the Laguerre method for numerically inverting Laplace transforms. *INFORMS Journal on Computing*, 8:413–427, 1996.

[2] M. Abramawicz and I. A. Stegun. *Handbook of Mathematical Functions with Formulas, Graphs and Mathematical Tables.* Dover Publishing, New York, December 1972.

[3] N. C. Audsley, A. Burns, R. I. Davis, K. W. Tindell, and A. J. Wellings. Fixed priority pre-emptive scheduling: An historical perspective. *Real-Time Systems*, 8:173–198, 1995.

[4] K. R. Baker and G. D. Scudder. Sequencing with earliness and tardiness penalties: a review. *Operations Research*, 38(1):22–36, 1990.

[5] K.R. Baker. *Introduction of Sequencing and Scheduling.* Wiley, 1974.

[6] S. Baruah, G. Koren, D. Mao, B. Mishra, A. Raghunathan, L. Rosier, D Shasha, and F. Wang. On the competitiveness of on-line real-time task scheduling. In *Proc. 12th IEEE Real-Time Systems Symp.*, pages 106–115, 1991.

[7] P. A. Blackmore. Information scheduling in a military global broadcast system. In *The IEEE International Conference on Networks (ICON'99), Session 8B*, Brisbane, Australia, September 28 - October 1 1999.

[8] M. Britton. Waiting times in an M/M/1 queue with the shortest-job-first queueing discipline. *INFORMS Journal of Operations Research, to be submitted.*

[9] M. Britton. Stochastic task scheduling in telecommunication systems. In *Multicasting and Scheduling Techniques for a Military Broadcast Environment.* Defence Materiel Organisation, Australian Theatre Broadcast System Project Office, 25th-28th March 2002.

[10] M. Britton. Stochastic task scheduling in telecommunication systems. *INFORMS Journal of Computing*, to be submitted.

[11] P. J. Burke. Equilibrium delay distribution for one channel with constant holding time, Poisson input and random service. *Bell Systems Technical Journal*, 38:1021–1031, July 1959.

[12] G. A. Campbell and R. M. Foster. *Fourier Integrals for Practical Applications.* D. Van Nostrand Co., New York, 1948.

[13] K. Chen. A study on the timeliness property in real-time systems. *Real-Time Systems*, 3:247–273, 1991.

[14] K. Chen and P. Muhlethaler. A scheduling algorithm for tasks described by time value function. *Real-Time Systems*, 10:293–312, 1996.

[15] S. Choi and A. K. Agrawala. Scheduling aperiodic and sporadic tasks in hard real-time systems. Technical Report CS-TR-3794, Institute for Advanced Computer Studies, Department of Computer Science, University of Maryland, 1997.

[16] J-Y. Chung, J. W. S. Jiu, and K-J. Lin. Scheduling periodic jobs that allow imprecise results. *IEEE Trans. Comp*, 39(9):1156–1173, 1990.

[17] H. D. Clausen, H. Linder, and B. Collini-Nocker. Internet over direct broadcast satellites. *IEEE Communications Magazine*, 37(6):146–151, June 1999.

[18] C. Cocks, E. Arbon, T. Burford, G. O'Shea, B. Khuu, P. Blackmore, A. Coutts, and P. Stimson. The theatre broadcast system. Technical Note DSTO-TN-0287, Defence Science and Technology Organisation, July 2000.

[19] E. G. Coffman and R. Graham. Optimal scheduling for two-processor systems. *ACTA Informat.*, 1:200–213, 1972.

[20] R. W. Conway, W. L. Maxwell, and L. W. Miller. *The Theory of Scheduling.* Addison-Wesley, 1967.

[21] R. Corless, G. H. Gonnet, D. E. G. Hare, D. J. Jeffrey, and D. E. Knuth. On the Lambert W function. *Advances in Computational Mathematics*, 5:329–359, 1996.

[22] M. E. Crovella, M. S. Taqqu, and A. Bestavros. Heavy-tailed probability distribution in the world wide web. In *A Practical Guide to Heavy Tails*, chapter 1, pages 3–26. Chapman and Hall, New York, 1998.

[23] L. D'Amore and A. Murli. The real inversion of a Laplace transform function. In *Proceedings of ICIAM '95*, 1995.

[24] L. D'Amore, A. Murli, and M. Rizzardi. Recent developments related to the numerical inversion of a Laplace transform function: The real inversion problem. In *Proceedings of the International Conf. On Inverse Problems in Engineering: Theory and Practice*, 1996.

[25] K. S. Decker, V. R. Lesser, and R. C. Whitehair. Extending a blackboard architecture for approximate processing. *Journal of Real-Time Systems*, 2(1):47–49, 1990.

[26] M. L. Dertouzos. Control robotics: The procedural control of physical processes. In *Proceedings of IFIP Congress*, pages 807–813, Stockholm, Sweden, 1974.

[27] J. K. Dey, J. Kurose, and D. Towsley. On-line scheduling policies for a class of IRIS (Increasing Reward with Increasing Service) real-time tasks. *IEEE Transactions on Computers*, 45(7):802–813, July 1996.

[28] L. Durr. Priority queues with random order of service. *Operations Research*, 19:453–460, 1971.

[29] A. Erdelyi, W. Magnus, F. Oberhettinger, and F. G. Tricomi. *Tables of Integral Transforms*, volume I. McGraw-Hill, New York, 1954.

[30] A. K. Erlang. *The Life and Works of A. K. Erlang.* The Copenhagen Telephone Co., Copenhagen, 1948.

[31] G. Fayolle and R. Iasnogorodski. Two coupled processors: The reduction to a Riemann-Hilbert problem. *Z. Wahrscheinlichkeitsth*, 47:325–351, 1979.

[32] G. Fayolle, P. J. B. King, and I. Mitrani. The solution of certain two-dimensional Markov models. *Adv. Appl. Prob.*, 14:295–308, 1982.

[33] D. W. Fife. Scheduling with random arrivals and linear loss functions. *Management Science*, 11(3):429–437, 1965.

[34] B. S. Garbow, G. Giunta, J. N. Lyness, and A. Murli. Software for an implementation of Week's method for the inverse of the Laplace transform. *ACM TOMS*, pages 163–170, 1988.

[35] M. R. Garey and D. S. Johnson. Complexity results for multiprocessor scheduling under resource constraints. *SIAM Journal of Computing*, 4:397–411, 1975.

[36] A. Garvey, K. Decker, and V. Lesser. A negotion-based interface between a real-time scheduler and a decision-maker. Technical Report 94-08, Department of Computer Science, University of Massachussets, March 17 1994.

[37] A. Garvey, M. Humphrey, and V. Lesser. Task interdependencies in design-to-time real-time scheduling. In *Proceedings of 11th National Conference on AI*, pages 580–585, 1993.

[38] A. Garvey and V. Lesser. A survey of research in deliberative real-time artificial intelligence. Technical Report 93-84, Dep. Computer Science, Uni. Massachussets, Nov 19 1993.

[39] I. S. Gradshteyn and I. M. Ryzhik. *Tables of Integrals, Series and Products: Corrected and Enlarged Edition.* Academic Press, 1980.

[40] J. R. Jackson. Scheduling a production line to minimise maximum tardiness. Research Report 43, Univ. of Calif., Los Angeles, 1955.

[41] E. D. Jensen, C. D. Locke, and H. Tokuda. A time-driven scheduling model for real-time operating systems. In *Proc. 6th IEEE Real-Time Systems Symp.*, pages 112–122, December 1985.

[42] E. D. Jensen, J. D. Northcutt, R. K. Clark, S. E. Shipman, F. D. Reynolds, D. P. Maynard, and K. P. Loepere. Alpha: An operating system for mission-critical integration and operation of large, complex, distributed real-time systems. In *Proc. 1989 Workshop on Mission Critical Operating Systems*, September 1989.

[43] C. Johnson. What's the web worth? The impact of retrieval delays on the value of distributed information. In *Time and the Web*, Staffordshire University, UK, June 1997.

[44] D. G. Kendall. Stochastic processes occurring in the theory of queues and their analysis by the method of the imbedded Markov chain. *Ann. Math. Stat.*, 24:19–53, 1953.

[45] A. Y. Khintchine. Mathematical theory of stationary queues (in Russian). *Matem. Sbornik*, 39(4):73–84, 1932.

[46] J. F. C. Kingman. On queues in which customers are served in random order of service. *Proc. Cambridge Phil. Soc.*, 58:79–81, 1962.

[47] L. Kleinrock. *Queueing Systems*, volume 1: Theory. John Wiley and Sons, New York, 1975.

[48] L. Kleinrock. *Queueing Systems*, volume 2: Computer Applications. John Wiley and Sons, New York, 1976.

[49] L. Kleinrock and Roy P. Finkelstein. Time dependent priority queues. *Operations Research*, 15(1-3):104–116, 1967.

[50] E. L. Lawler. Recent results in the theory of machine scheduling. In A. Bachen et. al., editor, *Mathematical Programming: The State of the Art*, pages 202–233. Springer-Verlag, New York, 1983.

[51] V. R. Lesser, J. Pavlin, and E. H. Durfee. Approximate processing in real-time problem solving. *AI Magazine*, 9(1):49–61, 1988.

[52] C. C. Lim and W. Zhao. Performance analysis of dynamic multitasking imprecise computation system. *IEE Proceedings-E*, 136(5):345–350, September 1991.

[53] D. V. Lindley. The theory of queues with a single server. *Proc. Cambridge Phil. Soc.*, 48:277–289, 1952.

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

[55] J. W. S. Liu, W-K. Shih, A. C-S. Yu, J-Y. Chung, and W. Zhao. Algorithms for scheduling imprecise computations. *IEEE Computers*, 24(5):58–65, 1991.

[56] C. D. Locke. *Best-Effort Decision Making for Real-Time Systems.* PhD thesis, CMU-CS-86-134, Dept. of Computer Science, Carnegie-Mellon University, May 1986.

[57] T. McCormick, P. McCormick, W. Hill, and M. Choffel. Information management for the global broadcast system: Operational considerations and recommendations. In *Military Communications Conference*, pages 523–527. IEEE, 1996.

[58] R. McNaughton. Scheduling with deadlines and loss functions. *Management Science*, 6(1):1–12, 1959.

[59] H. Moiin. *Real-Time Scheduling Algorithms.* PhD thesis, University of California, Dec 12 1992.

[60] A. K. Mok. *Fundamental Design Problems of Distributed Systems for the Hard Real-Time Environment.* PhD thesis, Dept. Electrical Eng. and Comp. Sc., MIT, Cambridge, Mass., May 1983.

[61] T. E. Morton and D. W. Pentico. *Heuristic Scheduling Systems.* John Wiley, New York, 1993.

[62] M. M. Nassehi. Channel access schemes and fiber optic configurations for integrated-services local area networks. Technical Report CSL-TR-87-322, Department of Comp. Sc., Stanford University, March 1987.

[63] M. M. Nassehi and F. A. Tobagi. Transmission scheduling policies and their implementation in integrated-services high-speed local area networks. In *Fifth Annual European Fibre Optic Communications and Local Area Networks Exposition*, pages 185–192, Basel, Switzerland, 1987.

[64] A. Papoulis. *Probability, Random Variables and Stochastic Processes.* McGraw-Hill, Inc., 1991.

[65] K. Park, G. Kim, and M. Crovella. On the relationship between file sizes, transport protocols, and self-similar network traffic. In *Proc. IEEE International Conference on Network Protocols*, pages 171–180, 1996.

[66] K. Pawlikowski. Steady-state simulation of queuing processes: A survey of problems and solutions. *ACM Computing Surveys*, 22(2):123–170, 1990.

[67] V. Paxson and S. Floyd. Wide area traffic: The failure of Poisson modeling. *IEEE/ACM Transactions on Networking*, 4(2):226–244, 1996.

[68] F. Pollaczek. Uber eine aufgabe der wahrscheinlichkeitstheorie. *Mathematische Zeitschrift*, 32:64–100 and 729–750, 1930.

[69] W. H. Press, S. A. Tenkolsky, W. T. Vetterlig, and B. P. Flannery. *Numerical Recipes in C: The Art of Scientific Programming.* Cambridge University Press, 2nd edition, 1992.

[70] J. Riordan. Delays for last-come first-served service and the busy period. *Bell Systems Technical Journal*, 40:785–793, May 1961.

[71] Y. Ronen, D. Mosse, and Martha E. P. Value-density algorithms for the deliberation-scheduling problem. *SIGART Bulletin*, 7(2):41–49, 1996.

[72] A. Schild and I. Fredman. On scheduling tasks with associated linear loss functions. *Management Science*, 7:280–285, 1961.

[73] A. Schild and I. Fredman. Scheduling tasks with deadlines and non-linear loss functions. *Management Science*, 9:73–81, 1962.

[74] L. Sha, R. Rajkumar, and S. Sathaye. Generalized rate-monotonic scheduling theory: A framework for developing real-time systems. *Proceed. IEEE*, V-82(1):68–82, 1994.

[75] J. G. Shanthikumar. Analysis of the control of queues with shortest processing time service discipline. *Journal of the Operations Research Society of Japan*, 23(4):341–352, 1980.

[76] W. K. Shih, J. W. S. Liu, and J. Y. Chung. Fast algorithms for scheduling imprecise computations. In *Proceedings of the IEEE Real-Time Systems Symposium*, pages 12–21, 1989.

[77] W. K. Shih, J. W. S. Liu, and J. Y. Chung. Algorthms for scheduling imprecise computations with timing constraints. *SIAM Journal of Computing*, 20(3):537–552, 1991.

[78] W. E. Smith. Various optimizers for single stage production. *Naval Research Logistics Quarterly*, 3:59–66, 1956.

[79] J. A. Stankovic, M. Spuri, M. Di Natale, and G. C. Buttazzo. Implications of classical scheduling results for real-time systems. *IEEE Computer*, June 1995.

[80] L. Takacs. Delay distributions for one line with Poisson input, general holding times, and various orders of service. *Bell Systems Technical Journal*, 42(2):487–503, March 1962.

[81] H. Takagi. *Queueing Analysis: A Foundation of Performance Evaluation*, volume 1: Vacation and Priority Systems, Part 1. North-Holland, 1991.

[82] D. Towsley and S. S. Panwar. Optimality of the stochastic earliest deadline policy for the G/M/c queue serving customers with deadlines. Technical Report UM-CS-1991-061, Uni. Massachusetts, September 1991.

[83] J. D. Ullman. Polynomial complete scheduling problems. In *Proc. Fourth Symp. Operating System Principles*, pages 96–101, New York, 1973. ACM.

[84] E. Vaulot. Delais d'attende des appels telephoniques dans l'ordre inverse de leur arrivee. *Comptes Rendus Acad. Sci. Paris*, 238:1188–1189, 1954.

[85] D. V. Widder. *An Introduction to Transform Theory.* Princeton University Press, Princeton, NJ, 1971.

[86] D. M. G. Wishart. Queueing systems in which the discipline is last-come, first served. *Operations Research*, 8:591–599, 1960.

[87] W. Zhao, C. C. Lim, J. W. S. Liu, and P. Alexander. In S. Natarajan, editor, *Imprecise and Approximate Computation*, chapter Overload Management by Imprecise Computation. Kluwer Academic Publishers, 1995.

[88] S. Zilberstein. *Operational Rationality through Compilation of Anytime Algorithms*. PhD thesis, University of California at Berkeley, 1993.